UserSWE: Evaluating Coding Agents in Multi-Turn Conversations with Dynamic Users

Anonymous authors
Paper under double-blind review

Abstract

As coding agents become more capable, developers increasingly use them to fix bugs through multi-turn interactions. Existing benchmarks do not adequately cover this scenario. Most present a complete issue report in a single turn without interaction. Their evaluation can produce both false positives and false negatives, accepting incorrect patches or rejecting correct ones. We introduce UserSWE for evaluating coding agents' ability to fix repository-level bugs through multi-turn interactions with dynamic users and execution-grounded verdicts at every turn. Its construction pipeline turns repository history and existing single-turn benchmarks into multi-turn tasks. At each turn, information pools constrain what the simulated user can reveal, while a persona chain sampled from a transition model calibrated on 20,859 development sessions governs how it speaks. Tests run after every turn, and a pair of agents from different model families writes and gates each verdict, using test results as primary evidence and model judgment only for aspects the tests cannot resolve. We evaluate eleven agents on 72 tasks and find that agent rankings change with the interaction budget. Claude Fable 5.1 resolves the most tasks in one turn, GPT-6-Astra resolves the most by the eighth turn, and Claude Opus 5 leads at most intermediate budgets. These results indicate that single-turn performance is insufficient to characterize interactive coding ability.

1 Introduction

AI programming assistants are deployed at scale and used daily by developers (Liang et al., 2024). Their form has evolved from autocompletion to context-grounded conversation, and then to agents that navigate repositories, edit files, and run tests on their own — a practice popularly called vibe coding (Ross et al., 2023; Yang et al., 2024; Sarkar & Drosos, 2025). Developers pose follow-up questions and refine their requests in response to intermediate model outputs (Barke et al., 2023; Ross et al., 2023), where initial requests are often incomplete and gradually clarified across turns. Studies of agents in real-world settings further confirm this same iterative pattern holds across entire tasks (Siddeeq et al., 2026; Baumann et al., 2026). As a representative scenario, debugging with an agent has now become a standard conversational workflow for developers.

However, prevailing benchmarks do not measure how well an agent works in this scenario for two main reasons. One is that most use a single-turn setup, handing the agent everything at once and asking for a correct patch in return, so the agent is never put in the conversation where the work actually happens. The few that do run several turns stay static, releasing the information in a written order or phrasing every turn in one fixed conversational style, which leaves the agent talking to an unresponsive mechanism and not to a simulation of a developer. Figure 1 shows how static and dynamic simulators behave differently on the same task. The other concerns multi-turn judging, where every intermediate turn calls for independent assessment, but standard tests can only provide factual execution evidence, and unrestricted LLM judges are prone to making inferences beyond what results actually demonstrate. For both reasons, a benchmark that measures coding agents' abilities in conversation is still missing.

To address this, we propose UserSWE, which settles where the tasks come from, where the turns come from, and how each turn is judged. The first is a **construction pipeline** that produces multi-turn tasks along two routes. One route mines a repository's own history for paired issues that a maintainer publicly recorded as linked. The other takes an instance of an existing single-turn benchmark and

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

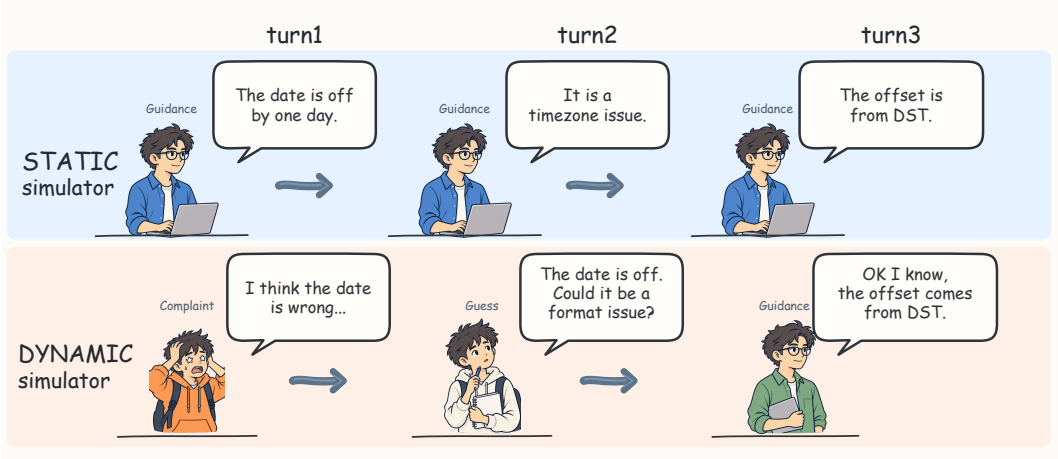


Figure 1: Existing multi-turn benchmarks keep the simulated user in one conversational style, while in the real development sessions we measure a developer's style shifts from turn to turn. On the same date-off-by-one bug, the static simulator phrases every turn as precise guidance, while our simulator opens on a vague complaint, guesses a wrong cause, and reaches guidance only at the third turn.

stages it as a conversation. As we continue running the pipeline along these two routes, it consistently generates new and up-to-date tasks. The second is a method for **simulating the user** across the turns of a conversation. We calculate a persona transition matrix on 20,859 real development sessions from an internally deployed commercial coding assistant, and we write each turn's utterance from that matrix together with the information pools of the task, so the persona an agent meets becomes dynamic as the session goes on. For each task we draw the whole chain in advance, which ensures any agent on the same task meets the same user. The third is an **evaluation framework**, which enables the coding agent to operate under constraints, with dual agents judging its generated outputs based on evidence. The framework sets rules that prevent coding agents from receiving credit beyond their demonstrated ability. Based on tests run after every turn, a Judge from one family writes the verdict, and a Reviewer from another family clears it before it counts. We select 72 tasks from the pipeline as UserSWE and evaluate agents on it.

The evaluation on UserSWE reveals behavior that a single-turn evaluation cannot show. Claude Fable 5.1 resolves the most tasks within a single turn, 31.9%, and still finishes second on the length-discounted pass rate, behind Claude Opus 5, which reaches a similar place in fewer turns, 2.99 against 3.14. GPT-6-Astra resolves the most tasks of any agent by the eighth turn, 81.9%, and places fourth once the turns it spends getting there are discounted.

In this paper, we make the following contributions. **(1) A construction pipeline.** It builds multi-turn tasks from repository evidence and existing single-turn benchmarks. **(2) A multi-turn user simulation.** We introduce a user simulator with dynamic personas grounded in real development conversations. **(3) An evaluation framework.** We propose a dual-agent judging evaluation framework complemented by a suite of supporting mechanisms, such as equivalent pass testing. UserSWE will be released along with the evaluation framework.

2 Related work

Table 1 compares UserSWE with existing coding-agent benchmarks. Below, we discuss two complementary lines of related work.

Multi-turn benchmarks and simulated users. Multi-turn evaluations must specify both what the user reveals and how the user speaks. SWE-Interact releases a task's requirements one at a time and uses a fixed, terse senior-developer persona derived from SWE-chat (Raghavendra et al., 2026; Baumann et al., 2026), while SWE-Together reconstructs a task-specific simulator from a recorded session and triggers utterances according to the live trajectory (Wu et al., 2026). These approaches

condition information release on the trajectory but do not vary the user's voice according to a measured distribution of developer personas. More broadly, simulated users can diverge from real developers, and persona choice can substantially change measured performance (Seshadri et al., 2026; Zhou et al., 2026; Truong et al., 2025). UserSWE samples a persona chain from real development sessions and generates an information pool for each task, which helps simulate users naturally.

SWE benchmarks and evaluation reliability. SWE-bench established issue-level, execution-based evaluation for coding agents (Jimenez et al., 2024). Later work addresses task contamination and freshness through larger, private, or continuously refreshed task collections (Xue et al., 2026; Liang et al., 2025; Deng et al., 2026; Zhang et al., 2025); SWE Context Bench is closest to our mining route but poses related instances independently (Zhu et al., 2026). A parallel line shows that verdicts can fail because tests are too loose or tight, LLM judges exhibit bias, or the evaluation harness can be gamed (Xue et al., 2026; Zheng et al., 2023; Wataoka et al., 2024; Wang et al., 2026). Panel-based judges and rule-based checkers improve reliability but remain imperfect (Verga et al., 2024; Lù et al., 2025). In the multi-turn setting, SWE-Interact still verifies only the final solution and rubric-scores intermediate turns (Raghavendra et al., 2026). UserSWE executes tests after every turn and uses a cross-family writer/reviewer gate, grounding each verdict in execution and reserving model judgment for evidence the tests cannot cover.

Table 1: A comparison of recent benchmarks (Jimenez et al., 2024; Deng et al., 2026; Zhang et al., 2025; Le et al., 2025; Zhu et al., 2026; Raghavendra et al., 2026; Wu et al., 2026) with UserSWE.

Benchmark	Recent tasks	Multi-turn	Persona	Judge method	Judge freedom
SWE-bench	×	×	×	tests	Low
SWE-bench Pro	×	×	×	tests, hand-checked	Low
SWE-bench-Live	✓	×	×	tests, auto-validated	Low
SWE-Evo	×	×	×	tests	Low
SWE-ContextBench	×	×	×	tests	Low
SWE-Interact	×	✓	static	source verifier	Low
SWE-Together	✓	✓	static	rubric judge	High
UserSWE	✓	✓	dynamic	two judges on test evidence	Medium

Persona records whether the conversational persona stays fixed across turns.
Judge freedom records how far the verdict rests on an LLM: Low means no LLM takes part in deciding it, High means it is decided entirely by an LLM, and Medium means the two are combined.

3 Preliminaries

The dominant executable format for software-engineering evaluation poses one issue as one task and grades the returned patch by executing tests (Jimenez et al., 2024; Deng et al., 2026). Given an issue and the repository state b at the commit preceding its fix, the agent answers once with a patch $\hat{g}$. Write $t(s) \in \{0, 1\}$ for the outcome of test t in workspace state s, and let $s \oplus p$ apply patch p to s. The task also carries the maintainer's own fix, the *gold patch* g, and two sets of tests from b and $b \oplus g$,

$$t(b) = 0 \text{ and } t(b \oplus g) = 1 \quad \forall\, t \in F, \qquad t(b) = t(b \oplus g) = 1 \quad \forall\, t \in P, \tag{1}$$

the *fail-to-pass* set F, which the fix has to turn green, and the *pass-to-pass* set P, which it must not break. A patch $\hat{g}$ *resolves* the task when $t(b \oplus \hat{g}) = 1$ for every $t \in F \cup P$, and the reported score is the fraction of tasks resolved (Yang et al., 2024).

4 UserSWE

We introduce UserSWE, which evaluates agents on issue fixing inside a multi-turn conversation and so measures coding ability more completely than a one-shot format can. Where Section 3 is settled in a single exchange, an agent here works on the issue over several turns, each turn answering what the user has just asked for, and we check the fail-to-pass and pass-to-pass set after every one of those turns. Turning both sets green in as few turns as possible is what UserSWE credits as strength at resolving code issues interactively. Figure 2 lays out in order where the tasks come from, how the multi-turn form is designed, and how each turn is judged.

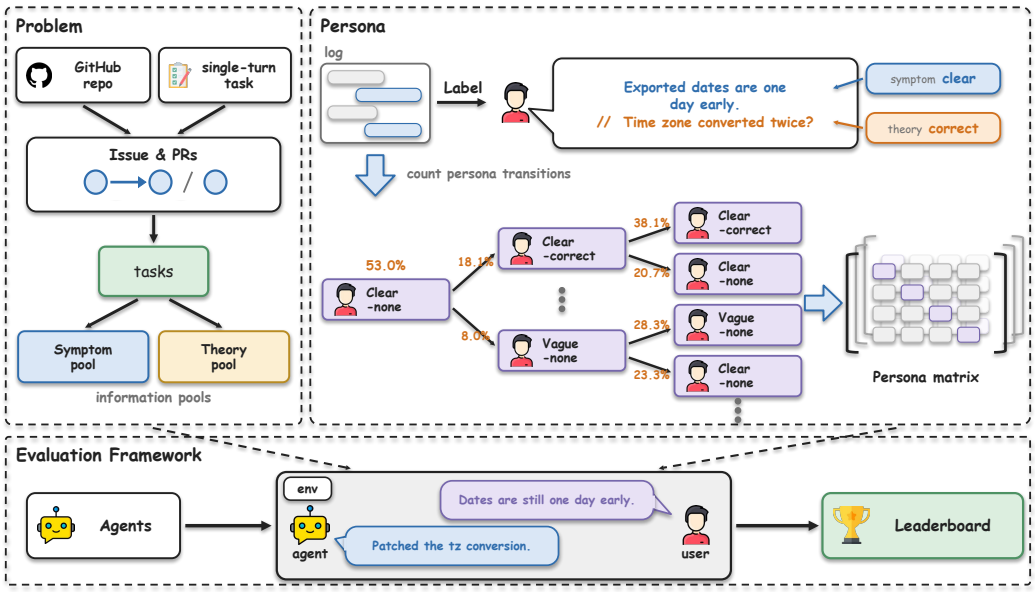


Figure 2: The UserSWE pipeline. The problem panel shows how a task is constructed, the persona panel how the simulated user is built, and the evaluation panel how a session is run and scored.

4.1 Data Construction

We mine linked issues and single issue from two sources and select suitable ones as our tasks. The first route searches repository histories for two issues explicitly linked by a maintainer. Each side has a merged fix with tests, giving the task two related acceptance points. The second route takes tasks from published single-turn benchmarks, preserving their repository state, gold patch, tests and original acceptance point while changing how their information is released across a multi-turn conversation.

Both routes then turn each task into an information pool. The pipeline extracts symptoms from the issue report for the symptom pool and uses the gold patch to construct correct or wrong directions about the cause for the theory pool, with the wrong direction later shared by the off-target and wrong personas. A cascade also carries follow-up information for its second issue. Agents evaluated on the same task therefore draw from the same frozen candidate information pool, preventing differences in pool construction from confounding the comparison.

Before admission, each task passes both mechanical and human checks. The framework reruns the fail-to-pass and pass-to-pass tests in the evaluation container, verifying that the fail-to-pass tests fail at the base state and pass after the gold patch is applied, while the pass-to-pass tests pass in both states. Appendix A details how tasks from both sources are converted into information pools and admitted as executable tasks.

4.2 Multi-turn Interaction

Section 4.1 fixes what the simulated user may say, but not which item should be selected at each turn or the voice in which it should be expressed. We handle these questions separately, first selecting the content through a release policy and then rendering it through a conversational persona model calibrated on real development sessions.

After each turn, the simulated user selects guidance from the information pool according to the agent's response and the canonical test results. The information selection process follows a fixed priority hierarchy designed to mirror the logic of natural developer interactions. If the agent poses a direct question, the simulated user responds with a direct answer. Otherwise, it selects content in descending order of priority: the most severe user-observable blocking failure, an unaddressed core business

Table 2: Eight conversational personas evaluated on the same bug scenario, where dates are exported one day early due to duplicate timezone conversion execution. Rows represent two distinct ways of describing the symptom, and columns represent four ways of stating the root cause.

	no cause	**correct**	**off-target**	**wrong**
clear	Exported dates are one day early.	Exported dates are one day early. // Timezone converted twice?	Exported dates are one day early. // Maybe the template?	Exported dates are one day early. // Timezone is fine, it is the template.
vague	The export looks off.	The export looks off. // Timezone converted twice?	The export looks off. // Maybe the template?	The export looks off. // Timezone is fine, it is the template.

boundary, or a secondary edge case. This hierarchy governs what information the user simulator releases at the current turn.

Once the content is selected, the framework must decide how to express it. The expression of the developer varies between people and between turns of the same session. Its two common components align with the two kinds of content in the information pool, namely how a developer describes *what was observed* and whether they offer *a view of the cause*. We therefore factor the simulated user along two axes. Clarity records whether the symptom is stated precisely or vaguely, and veracity records whether the user offers no view, a correct one, an off-target one, or a wrong one. Crossing these axes yields eight conversational personas, and Table 2 presents the same request phrased in each of them.

We do not assume the eight cells occur uniformly. From 20,859 real development sessions from an internally deployed commercial coding assistant, we measure the opening persona and its turn-to-turn transitions. Of these sessions, 13,530 change code, and the eight cells account for 98.65% of them, with 52.98% opening on a clear symptom and no view of the cause. The transitions form a family of matrices conditioned on the opening persona rather than one global matrix. We pseudorandomly draw a complete persona chain for each task in advance so that different agents on the same task face the same changes in the user. The selected content is rendered in the persona assigned to that turn. Appendix B gives all eight opening-conditioned transition matrices and the persona design.

4.3 Evaluation

Our evaluation framework governs the entire multi-turn interaction through the information pool defining all disclosable content, the release policy controlling what information is revealed at each turn, and the frozen persona chain standardizing the conversational style. Turn-level judgments are produced through a structured pipeline involving the Judge and Review stages, as outlined in Algorithm 1. We elaborate on the evaluation design below, with full algorithmic logic and implementation details provided in Appendix C.

A core challenge in multi-turn agent evaluation is that each intermediate turn requires independent correctness judgments, yet test cases can only provide objective execution evidence, and unrestricted LLM judges are prone to making inferences beyond observable outcomes, introducing systematic misjudgments. This issue manifests across two dimensions of freedom, each of which causes evaluation errors when improperly calibrated. Ill-tuned constraints on the evaluation instrument undermine score reliability, while misaligned boundaries for the agent under test prevent it from demonstrating its true capabilities. Our UserSWE framework addresses these two dimensions separately.

Freedom for the agent under test. Test suites are inherently built around the specific implementation of the reference gold patch, so allowed divergence from the gold implementation directly drives bidirectional misjudgments: exact-match criteria produce false negatives, recovering only 55.9% of expert-verified successful runs (Lù et al., 2025), while overly permissive tests or semantics-only scoring inflate false positives (Xue et al., 2026; Wu et al., 2026). To calibrate this dimension, our framework introduces an **equivalent pass** mechanism (EqualPass, see Algorithm 1), which revises test cases based on the agent's actual implementation and validates passes on modified tests, with a baseline failure guardrail to prevent trivialization. We also restrict agent access to assets unrelated to core coding capability, such as pre-existing solutions or external lookup resources, to prevent agents

Algorithm 1 Scoring one session.

Input: coding agent under test M, task τ, persona seed σ, turn budget K
Output: LDPR $\in [0, 1]$

```
JUDGE ↔ REVIEW                                   ▷ two different model families, swapped between tasks
workspace ← base state of τ
message ← opening request of τ under σ
result ← every acceptance point of τ, pass-to-pass green and fail-to-pass red
k ← 0
while k < K and result is not all-pass do
    previous ← workspace
    workspace ← M(message, workspace)
    k ← k + 1 if workspace ≠ previous, else k ← k + 0.5
    result ← RUNTEST(workspace, result)          ▷ execute the tests in the container
    (scorecard, message) ← JUDGE(workspace, result, σ, k)   ▷ verdict, and the next request
    result ← EQUALPASS(workspace, result, scorecard)        ▷ run the rewritten test where one is asked for
    label ← REVIEW(scorecard, message, result)
    if label = rewrite then
        (scorecard, message) ← the rewrite REVIEW returns
    end if
end while
return (K + 1 − k)/K if result is all-pass, else 0
```

from bypassing their actual competence via external shortcuts, ensuring meaningful differentiation between agent scaffolds (Jimenez et al., 2024; Yang et al., 2024).

Freedom for the evaluation instrument. The freedom granted to the simulated user and judge does not determine correctness, but it determines whether the evaluation instrument itself becomes a confounding variable. For the simulated user, unrestricted improvisation causes task drift across evaluations (Raghavendra et al., 2026), while fully scripted dialogue loses natural conversational dynamics. We freeze the task agenda, persona progression and information pool via a frozen persona chain, leaving only phrasing to dynamic generation. For judging, single-model scoring suffers from LLM hallucination risks, where the judge model may misjudge solution correctness and produce unfounded verdicts (Zheng et al., 2023; Wataoka et al., 2024). We implement a dual-family judge-reviewer pipeline with role rotation across tasks, which operates as an inline quality gate rather than a post-hoc audit.

Metrics. A multi-turn setting admits a quantity the one-shot format has no room for, namely how much of the turn budget an agent had left when it succeeded. Beyond the plain resolve rate we therefore report the *L*ength-*D*iscounted *P*ass *R*ate (LDPR), which is counted as

$$\text{LDPR} = \begin{cases} (K+1-k)/K & \text{tests all pass,} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where k is the weighted number of turns used; a turn without a code edit counts as half a turn. Agents that resolve a task in fewer conversational turns receive higher scores, while those that fail to complete the task within the predefined maximum turn limit are assigned a score of zero.

5 EXPERIMENTS

We evaluate how current agents fix issues in multi-turn interaction on UserSWE, and then we analyze the multi-turn design and the evaluation framework.

5.1 MAIN RESULTS

We select 72 tasks from the continuously running pipeline as UserSWE. Of these tasks, 22 are mined from open-source repository history and contain two issues, while the other 50 are converted from published single-turn benchmarks. (Appendix A details task construction and admission.) After that we evaluate 11 agents on it, each a model on a scaffold as listed in Table 13.

Table 3: Evaluation results on UserSWE. The middle columns give the resolved rate within that many turns.

	Resolved within k turns									
Model	1	2	3	4	5	6	7	8	k	LDPR
Claude Opus 5	25.0	**56.9**	**66.7**	**76.4**	**79.2**	**80.6**	**80.6**	80.6	**2.99**	**0.640**
Claude Fable 5.1*	**31.9**	47.2	62.5	66.7	75.0	75.0	75.0	76.4	3.14	0.593
Grok-4.6	18.1	43.1	58.3	66.7	76.4	79.2	79.2	80.6	3.40	0.569
GPT-6-Astra	12.5	38.9	55.6	70.8	77.8	79.2	**80.6**	**81.9**	3.44	0.561
DeepSeek-v4.1-Flash	18.1	40.3	51.4	65.3	73.6	76.4	76.4	76.4	3.47	0.537
GPT-5.6-Sol	16.7	36.1	47.2	61.1	68.1	69.4	70.8	70.8	3.73	0.500
GLM-5.3	19.4	37.5	51.4	63.9	63.9	65.3	65.3	65.3	3.74	0.496
Gemini-3.8-Flash	15.3	33.3	44.4	54.2	62.5	62.5	63.9	66.7	3.79	0.458
Hy4-preview	9.7	30.6	48.6	56.9	62.5	63.9	65.3	66.7	3.85	0.457
Doubao-Seed-2.1-Pro	11.1	31.9	43.1	54.2	68.1	68.1	68.1	68.1	3.89	0.456
Kimi-K3	18.1	27.8	36.1	44.4	51.4	54.2	55.6	56.9	4.14	0.391

**Claude Fable 5.1 refused some tasks. We count a refusal as unresolved, so its reported rates carry that loss.*

The evaluation framework uses two different model families for user simulation and result adjudication. The judging and review sides are rotated between gpt-5.6-sol on the Codex scaffold and opus-5 on the Claude Code scaffold. We selected them because they represent the most capable cross-family model combination that can be deployed stably at present. The framework is not bound to these two specific models, and either can be replaced with suitable alternatives. Information pools and persona trajectories are drawn per task in advance, so every agent is evaluated fairly on the same task.

The two subsets run under different turn budgets, $K = 5$ for a single-issue task and $K = 8$ for a two-issue one, since a second acceptance point needs the extra turns to be reached at all. Table 3 reports results on UserSWE, the share of tasks resolved within the first k turns for k from 1 to 8, together with LDPR (Equation 2) and the mean number of turns used. More detailed results, such as costs, are in Appendix D, and Appendix E follows individual sessions case by case. Appendix D.2 reports confidence intervals and paired comparisons between agents.

5.2 Analysis of the multi-turn design

5.2.1 Multi-turn against single-turn

As the main table shows, the agent that is best at the opening is not the agent that is best at the end. Claude Fable 5.1 resolves more tasks within the first turn than any other agent and still finishes behind Claude Opus 5 on LDPR. GPT-6-Astra resolves more tasks by the eighth turn than any other agent and places fourth on LDPR. A single-turn evaluation cannot show either of these, because it observes one point of the curve and reports it as the whole of an agent's ability.

We look more closely at the four agents that lead on LDPR. Figure 3 tracks the order they stand in as the turn budget grows. Over eight budgets the four take six different orderings between them, and three of the four lead at some budget. GPT-6-Astra is last of the four through the third turn and catches up to lead by the eighth. They end within six points of each other, between 76.4% and 81.9% resolved by the eighth turn, so these are reorderings among agents of comparable final ability. We also compare these four agents on a separate collection of 42 single-turn tasks drawn from SWE-bench Pro and SWE-bench-Live. Claude Fable 5.1 ranks first at 78.6%, well ahead of Claude Opus 5 at 64.3%, GPT-6-Astra at 61.9% and Grok-4.6 at 59.5%. Fable's large single-turn margin explains its early lead, but the ordering changes as the turn budget grows.

5.2.2 Dynamic conversational persona

Most existing benchmarks model the simulated user as a static persona with both clear expression and correct causal views (Raghavendra et al., 2026; Baumann et al., 2026). That profile opens only 19.03% of the 13,530 observed sessions, and holding it for a whole session is rarer still: 0.72% of

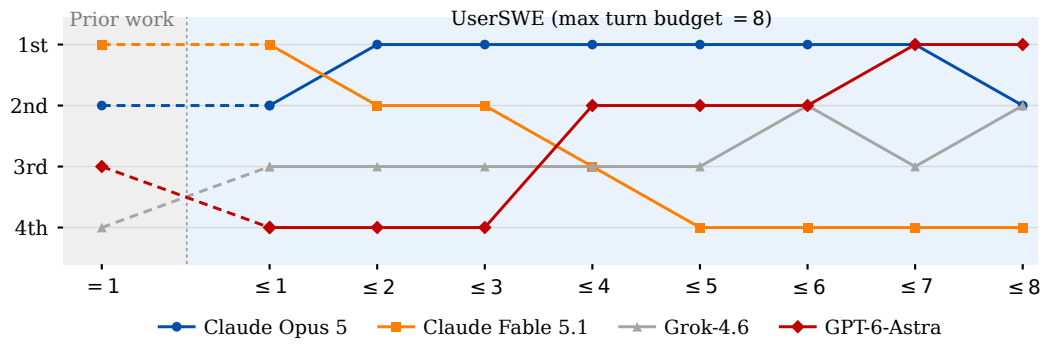


Figure 3: The order of the four leading agents on published single-turn tasks, and how that order changes as the turn budget grows on UserSWE.

Table 4: Persona ablation under a clear-and-correct user and the dynamic persona chain, over the same 50 single-issue tasks. Δ is the change from clear-and-correct to dynamic.

	LDPR			k		
Model	Clear	Dynamic	Δ	Clear	Dynamic	Δ
Claude Opus 5	0.688	0.668	↓.020	2.27	2.50	↑.23
GPT-6-Astra	0.656	0.580	↓.076	2.54	2.92	↑.38
DeepSeek-v4.1-Flash	0.628	0.562	↓.066	2.58	2.88	↑.30
Kimi-K3	0.428	0.470	↑.042	3.46	3.25	↓.21

sessions stay clear and correct for five consecutive turns on the same issue and 0.24% do so for eight. Even among the sessions that actually run that long, the shares are 2.81% and 1.86%.

We compare a reference arm that remains clear and correct at every turn with the persona chain. Both arms run the 50 single-issue tasks, which fit a five-turn budget and so isolate the persona from the cascade's issue-to-issue carryover, and each comparison uses the same task and agent and changes only the user trajectory. Table 4 reports LDPR and mean weighted turns k. Three of the four agents move the same way, scoring higher and finishing sooner under the clear-and-correct user, and the size of the shift varies widely between them, from 0.020 to 0.076 in LDPR. The largest is GPT-6-Astra, whose LDPR increases from 0.580 to 0.656 while k decreases from 2.92 to 2.54, and Kimi-K3 moves the other way. How much an agent depends on precise phrasing is itself a property of the agent, and it shows across all 72 tasks and all eleven agents. Splitting the full set by whether the opening states the symptom precisely or as an impression, the gap between the two halves runs from 0.268 for Claude Fable 5.1 down to -0.088 for DeepSeek-v4.1-Flash (Appendix F). An idealized user therefore reports a benchmark easier than the one most users present, and it flatters some agents more than others.

5.3 Analysis of the evaluation framework

5.3.1 Reviewer rewrites

The Reviewer is not a formality. Across the scored turns, it rewrote 40.9% of Judge decisions. Table 5 separates the two Reviewer configurations. It reports the share of reviewed turns that each Reviewer rewrote and how the objections attached to those rewrites divide across four review domains.

Table 5: The share of reviewed turns rewritten within each configuration(rewrite) and the detailed shares of four objections.

Reviewer	Rewrite	Evidence	Guidance	Equivalence	Persona
Codex / GPT-5.6-Sol	62.4%	70.3%	14.3%	9.5%	5.9%
Claude Code / Claude Opus 5	25.4%	42.1%	41.5%	5.3%	11.1%

The first column is calculated over reviewed turns within each configuration; the other four partition that configuration's own objections. Evidence marks a state unsupported by per-item test results or an executed command, guidance a next-turn utterance that breaks the interaction rules, equivalence a mishandled equivalence request, and persona a turn that does not execute the persona chain drawn for it. Both configurations catch problems that would otherwise take effect under a single Judge. Overall, 87.4% are rated major or critical, meaning that under the rubric they may change the follow-up, completion state or score rather than wording alone. Appendix D.4 gives the rewrite rate and objection distribution for every leaderboard model.

5.3.2 Equivalent passes

Our equivalent-pass mechanism is strictly execution-based. Tests tied to the gold patch's interface or structure may reject functionally equivalent implementations. We allow targeted rewrites for the tested implementation, but award credit only when the rewritten test passes and still fails at baseline; descriptive claims do not count.

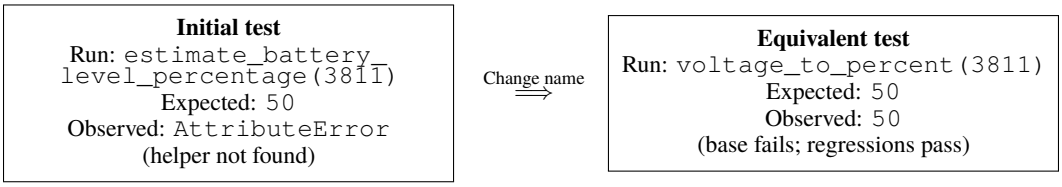


Figure 4: The equivalent test retains the expected behavior under a different helper name.

The route is used and its approval rate does not make review a formality. Across all evaluation runs, 14.6% of cells filed at least one equivalence request, and 51.2% of the requests they filed were granted and written into the scorecard. Figure 4 gives one such grant without reproducing code. The shipped test called the battery estimator by the helper name used in the gold patch, while the candidate implemented the same 22 voltage-to-percentage mappings under a different name. The rewrite retained every input-output assertion but located the one-argument estimator by its signature; it failed at the base state, passed on the candidate without regression, and was approved by the Reviewer. Where the other requests were stopped is broken down in Appendix D.5.

6 Limitations

Scale. Multi-turn evaluation operates on full interactive sessions and re-runs the complete test suite after each turn, which incurs notably higher computational overhead than single-patch per-task evaluation and constrains our released benchmark set to 72 tasks. Detailed per-task cost breakdowns are reported in Table 15.

Horizon. UserSWE focuses on single maintenance episodes spanning one to two issues, progressing from a base commit to a fixed set of acceptance criteria. It does not address longer-horizon development scenarios such as building a repository from scratch, which remains a promising direction for future extensions.

Session logs. Our user simulation model is constructed using opening request distributions and turn-to-turn transition patterns measured from 20,859 sessions collected from an internally deployed commercial coding assistant. Future iterations could enhance the simulation by incorporating additional signals such as task type characteristics and request phrasing patterns.

7 Conclusion

We present UserSWE, a benchmark for evaluating coding agents in multi-turn conversations with dynamic users. We contribute a construction pipeline that builds multi-turn tasks from repository history and existing benchmarks, a user simulator with dynamic personas grounded in real development sessions, and an evaluation framework that anchors cross-family judging and review to executed tests. Together, these components produce UserSWE, which we use to evaluate eleven agents on 72 tasks and uncover behaviors that single-turn evaluation cannot reveal.

AI USE STATEMENT

We used generative AI tools to polish the manuscript and as necessary components of the experimental framework, as described in the paper. The authors reviewed all AI-assisted work and take full responsibility for the final content and reported results.

ETHICS STATEMENT

The benchmark tasks are constructed from publicly available issues, pull requests and repository histories, together with previously published benchmarks. Repository-derived materials remain subject to their upstream licenses. The user model is calibrated from recorded development sessions collected from an internally deployed commercial coding assistant. We do not release raw conversations or person-level attributes, and release only the aggregate statistics required to instantiate the simulator. This limits the released behavioral information to what is needed to instantiate the simulator while reducing privacy and security risks associated with development logs.

REPRODUCIBILITY STATEMENT

We document the benchmark construction pipeline in Section 4.1 and Appendix A. Appendix B specifies the simulated user, and Appendix B.2 reports the calibrated opening distribution and transition matrices. Judge states, Reviewer actions and equivalence settlement are specified in Appendix C, and the post-run audit rubric is provided in Appendix C.4. Agent configurations and cost measurements appear in Appendix D. The sampled persona trajectories are independent of the agents and are stored to permit turn-by-turn replay. We will release UserSWE together with the multi-turn evaluation framework.

REFERENCES

Shraddha Barke, Michael B. James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7 (OOPSLA1), 2023. doi: 10.1145/3586030.

Joachim Baumann, Vishakh Padmakumar, Xiang Li, John Yang, Diyi Yang, and Sanmi Koyejo. SWE-chat: Coding agent interactions from real users in the wild, 2026.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kistler. SWE-bench pro: Can AI agents solve long-horizon software engineering tasks? In *International Conference on Learning Representations (ICLR)*, 2026.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Tue Le, Minh V. T. Thai, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. SWE-EVO: Benchmarking coding agents in long-horizon software evolution scenarios, 2025.

Jenny T. Liang, Chenyang Yang, and Brad A. Myers. A large-scale survey on the usability of AI programming assistants: Successes and challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, 2024. doi: 10.1145/3597503.3608128.

Shanchao Liang, Spandan Garg, and Roshanak Zilouchian Moghaddam. The SWE-bench illusion: When state-of-the-art LLMs remember instead of reason, 2025.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. AgentRewardBench: Evaluating automatic evaluations of web agent trajectories. In *Conference on Language Modeling (COLM)*, 2025.

Mohit Raghavendra, Anisha Gunjal, Aakash Sabharwal, and Yunzhong He. Swe-interact: Reimagining swe benchmarks as user-driven long-horizon coding sessions, 2026.

Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces (IUI)*, 2023. doi: 10.1145/3581641.3584037.

Advait Sarkar and Ian Drosos. Vibe coding: programming through conversation with artificial intelligence. In *Proceedings of the 36th Annual Conference of the Psychology of Programming Interest Group (PPIG)*, 2025.

Preethi Seshadri, Samuel Cahyawijaya, Ayomide Odumakinde, Sameer Singh, and Seraphina Goldfarb-Tarrant. Lost in simulation: LLM-simulated users are unreliable proxies for human users in agentic evaluations. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026. doi: 10.18653/v1/2026.acl-long.2192.

Shahbaz Siddeeq, Muhammad Waseem, Kai-Kristian Kemell, Mika Saari, Jussi Rasku, and Pekka Abrahamsson. Vibe coding in software development: A multivocal literature review, 2026.

Kimberly Le Truong, Riccardo Fogliato, Hoda Heidari, and Zhiwei Steven Wu. Persona-augmented benchmarking: Evaluating LLMs across diverse writing styles. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025. doi: 10.18653/v1/2025.emnlp-main.1155.

Pat Verga, Sebastian Hofstätter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. Replacing judges with juries: Evaluating LLM generations with a panel of diverse models, 2024.

Hao Wang, Hanchen Li, Qiuyang Mang, Alvin Cheung, Koushik Sen, and Dawn Song. Do androids dream of breaking the game? systematically auditing AI agent benchmarks with BenchJack, 2026.

Koki Wataoka, Tsubasa Takahashi, and Ryokan Ri. Self-preference bias in LLM-as-a-judge. In *NeurIPS 2024 Workshop on Safe Generative AI*, 2024.

Yifan Wu, Zhuokai Zhao, Songlin Li, Ho Hin Lee, Jiacheng Zhu, Shirley Wu, Tianhe Yu, Serena Li, Lizhu Zhang, Xiangjun Fan, and Shengzhi Li. SWE-together: Evaluating coding agents in interactive user sessions, 2026.

Haoran Xue, Reem Aleithan, Nafid Enan, Gias Uddin, and Song Wang. SWE-bench+: Enhanced LLM coding benchmark. In *Proceedings of the 3rd ACM International Conference on AI-Powered Software (AIware)*, 2026. doi: 10.1145/3805760.3814925.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. SWE-bench goes live! In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025) Track on Datasets and Benchmarks*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2023.

Xuhui Zhou, Weiwei Sun, Qianou Ma, Yiqing Xie, Jiarui Liu, Weihua Du, Sean Welleck, Yiming Yang, Graham Neubig, Sherry Tongshuang Wu, and Maarten Sap. Mind the sim2real gap in user simulation for agentic tasks, 2026.

Jared Zhu, Minhao Hu, and Junde Wu. SWE context bench: A benchmark for context learning in coding, 2026.

A How a task is built

The released benchmark contains 22 cascades mined from repository history and 50 single-issue tasks converted from published benchmarks. This appendix follows both routes from source material to a task package.

A.1 Two source routes

The cascade route searches repository text for two signals. Reveal signals occur when a later issue says it was exposed while investigating or fixing an earlier issue. Dependency signals occur in pull-request text that states one change is stacked on, depends on or follows another. A pair is retained in the stated order only when both issues have merged fixes and the later record points to the earlier work. Of the 22 mined cascades, six come from reveal signals and sixteen from dependency signals.

The separability check in Section A.4, rather than the phrase category, establishes that the fixes must land in a fixed order.

The conversion route starts from a published single-turn task. It retains that task's repository state, gold patch, test lists and container image, and changes only how the issue is presented across turns.

A.2 From source to task package

For a mined cascade, the pipeline retrieves both merged fixes, builds the grading image and tests, maps acceptance points to the two phases, and creates one task package and information pool. For a converted task, it preserves the upstream repository state, image and test verdicts while reorganizing the user-visible report into turn material. Both routes use the same admission checks below.

A.3 Information pools

Each task has one information-pool file. The simulator selects from the file before applying a persona's wording.

Field	Content
clear symptom	one reproducible user-visible observation and a broad request
vague symptom	the same problem as an impression, without implementation identifiers
correct idea	a low-confidence direction consistent with the cause, not a repair step
wrong idea	a plausible direction that points somewhere real but is not causal
follow-up set	the same four fields for the second issue, phrased as appearing after the first fix
phase map	the acceptance points assigned to each phase

Because pool construction can inspect the gold patch, it applies two leak guards. Gold-introduced identifiers are forbidden and checked after generation. For cascades, shared scope named in both issue titles is removed unless it is already delivered by the first patch. A task is rejected when either check cannot produce a clean pool.

A.4 Admission

Admission starts the task's grading container and calls the evaluation's test runner in two states. All four outcomes in Table 6 are required.

Table 6: The four admission outcomes.

State	Test set	Required	Establishes
base	pass-to-pass	pass	existing behavior is intact
base	fail-to-pass	fail	the task remains open
gold	pass-to-pass	pass	the fix introduces no covered regression
gold	fail-to-pass	pass	the task is closeable

The verdict records which run failed and carries fingerprints of the grading protocol and task image. A fingerprint mismatch makes the verdict stale.

For a cascade, an intermediate check additionally requires the first phase to pass after the first fix while the second remains open and regression guards stay green.

Agent solvability review. Mechanical admission establishes that a task can be graded, but not that its information pool is sufficient. We therefore applied the same read-only solvability review to all 72 tasks, comparing the agent-visible material with the tested behavior. The model review was advisory: a human curator read its findings, decided the revisions and made the final retention decision. Pools requiring revision were updated from the review feedback and retained only after re-review. Gold patches and tests served only as review evidence and were not exposed to agents.

B How the user is simulated

The simulator combines task content from the information pool with a persona that controls clarity, direction and stubbornness.

B.1 Persona design

The persona definitions and utterance policy were designed by the authors. Production sessions calibrate their frequencies and transitions; they do not define the policy automatically.

Symptom axis. The persona grid uses two ways of reporting the same user-observable problem. A *clear* symptom names a precise object, reproducible condition or explicit acceptance target. A *vague* symptom reports a user-facing impression or broad goal without implementation identifiers.

Cause axis. Each symptom setting is crossed with four cause settings. The cause axis determines what direction the user volunteers and how the user reacts when challenged. Table 7 defines its four settings.

Table 7: The four settings on the cause axis. Each is paired with either a clear or vague symptom.

Cause	Opening behavior	Response to disagreement
None	Gives no diagnosis or repair direction.	Continues with observable symptoms and answers direct questions.
Correct	Offers the correct direction as a low-confidence guess.	Gives way when the agent provides a better account.
Off-target	Offers a plausible but mistaken direction.	Drops the direction after reasoned disagreement and continues as *none*.
Wrong	Opens with the same mistaken direction as *off-target*.	Holds it against reasoning and yields only to test, code or reproduction evidence.

The off-target and wrong settings are identical in the opening turn. They separate only through the agent's later response, which makes their distinction observable only in a multi-turn session.

B.2 Log labeling and transition chains

We labeled the opening and later user turns of 20,859 sessions from an internally deployed commercial coding assistant. Sessions without a code change were excluded, leaving 13,530. A language model assigned the symptom and cause labels using the conversation and a digest of the eventual findings. It chose among nine options, the eight cells and *other*, so an opening that sits on neither axis could be reported as such. The 182 openings labeled *other* are counted in Table 8 and carry no chain, since a persona the scheme does not name cannot be instantiated in the simulator. We then read ten sessions per category by hand, and our reading agreed with the assigned label in each one. The underlying sessions are proprietary; we release only the aggregate statistics and calibrated chains. Table 8 gives the opening distribution.

Table 8: Opening persona distribution over the 13,530 code-changing sessions. The eight cells cover 98.65% of them; the rest opened as *other*.

Symptom	Stated cause	Sessions	Share
clear	*none*	7,168	52.98%
clear	*correct*	2,575	19.03%
clear	*off*	122	0.90%
clear	*wrong*	197	1.46%
vague	*none*	3,103	22.93%
vague	*correct*	123	0.91%
vague	*off*	27	0.20%
vague	*wrong*	33	0.24%
Covered by the eight cells		13,348	98.65%
other		182	1.35%

The harness draws an opening from the eight cells renormalized to one, then conditions its transition chain on that opening, yielding one matrix for each of the eight openings. It drops the end-of-session column, renormalizes each row and uses the resulting matrix only until the task is solved. Figure 5 shows all eight matrices.

Most transition mass returns to a clear symptom with either no stated cause or a correct cause. The two mistaken settings are pooled in aggregate reporting because they share an opening and are individually sparse, while the runtime trajectory retains which one is played.

B.3 Utterance policy

After selecting guidance, the simulator speaks as a software user rather than a code reviewer. It states user-visible wants, observations and expectations, does not expose evaluator or implementation details, answers direct clarifications first, and realizes the selected symptom clarity and cause setting.

The release selector reads the agent's response and canonical test results, then applies the fixed priority in Table 9. The tests identify what remains open, but every outward message must stay within the information pool. Here a core business boundary is behavior required by the primary workflow, whereas a secondary edge case is remaining acceptance behavior outside that workflow.

Table 9: Priority for selecting guidance from the information pool.

Priority	Trigger	Release
1	The agent asks a direct question.	Answer that question.
2	A user-observable core failure remains.	Release the most blocking observable symptom.
3	A core business boundary remains uncovered.	Release the relevant boundary.
4	Only a secondary edge case remains.	Release the relevant edge case.

The first applicable category takes priority, and a solved task releases nothing further.

The sampled trajectory is fixed, while the played cause can be weakened by the interaction rules. An off-target cause becomes *none* after the agent directly contradicts it with a code- or behavior-based explanation. A wrong cause remains until contradicted by an executed test, a reproduction or an inspected code fact. A persona with no cause never acquires one, and a correct cause never becomes an implementation instruction. If the information pool contains no direction corresponding to the sampled cause, the played cause becomes *none* while symptom clarity is preserved. The trajectory records both the sampled and played settings.

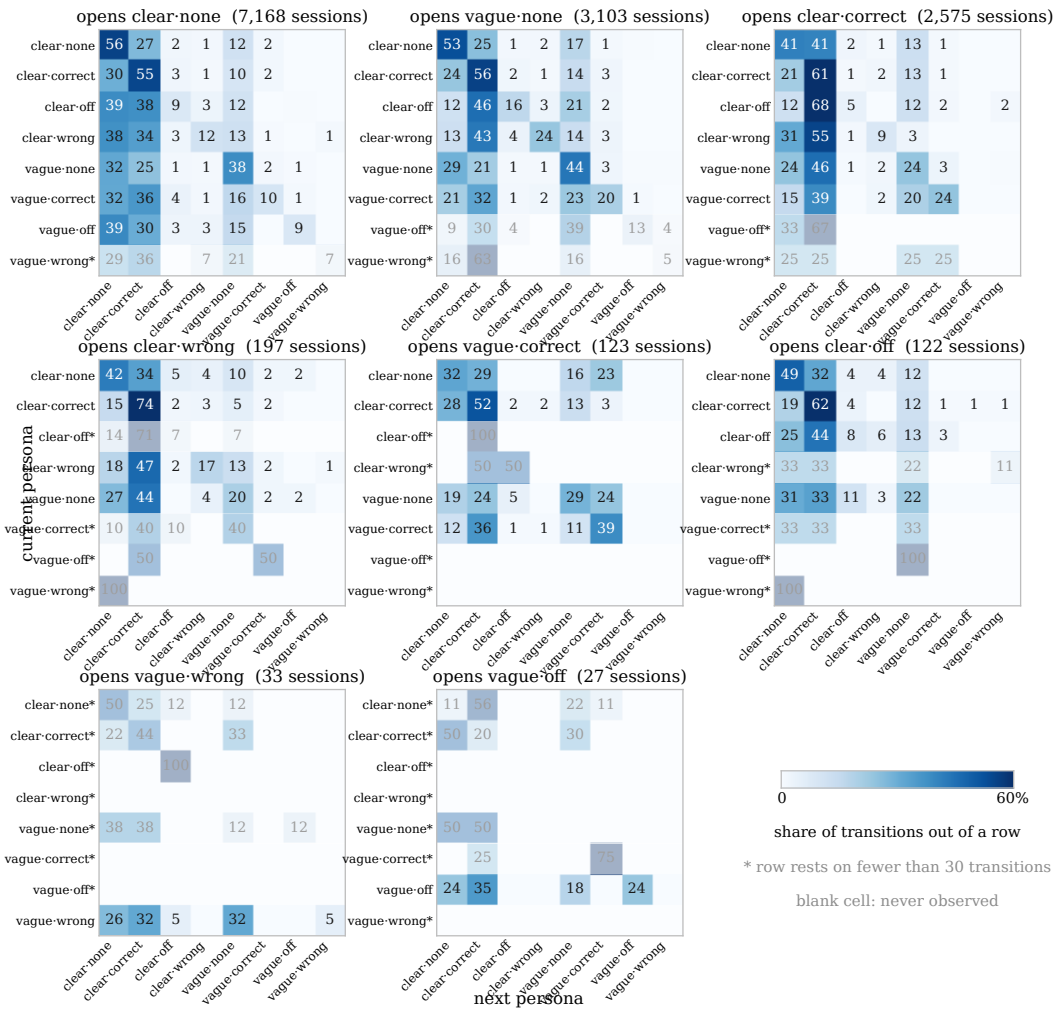


Figure 5: The eight turn-to-turn transition matrices, one for each opening persona. Rows are current personas and columns are next personas. Rows with fewer than 30 observed transitions are greyed and marked with an asterisk, and entries below 0.5% are left blank.

B.4 Human validation

The evaluation framework can pause before any simulated user turn is sent. At such a breakpoint a human reads the utterance the simulator is about to send and checks that a user experiencing the issue could plausibly say it, and that it stays within the information pool and the sampled persona. A turn must also react to what the agent has just done while respecting the per-turn information budget, so the check covers whether it volunteers the cause, the test names or the shape of the fix. Every task in the released set passed this review at least once.

When a turn fails the check we do not edit that sentence. We change what produced it: the generation rule, the utterance-writing prompt or the information pool, and then regenerate; the regenerated turns go through the breakpoints again. A single such change applies to every turn generated afterwards, so the review effort does not grow with the size of the set. Review at these breakpoints required 50 human-hours in total and no repository-specific expertise.

C Evaluation framework

The framework separates environment controls, the Judge, the Reviewer and a post-run audit.

C.1 Environment controls

Internal assets. The agent sees the agent-facing task statement and its workspace, but not the source issue or pull-request records, gold patch, fail-to-pass tests, acceptance list, phase map, scorecard, Judge–Reviewer records or other evaluation metadata. These assets are mounted outside the workspace, and tool controls block direct and indirect access to them.

External retrieval. General technical documentation is allowed. The agent may not retrieve this task's issues, pull requests, commits, releases, other source versions or ready-made fixes from code hosts, package registries, mirrors or caches. External lookups are recorded. A violation is terminal only when task-specific protected material is returned and used.

C.2 Judge

The framework sends the Judge the canonical user message, agent response, external-lookup and private-access audits, canonical test window, workspace fingerprints, previous equivalence ruling, remaining turn budget and recent user messages. It returns a candidate state, item-level verdicts, supporting evidence and the next user message. Table 10 gives the action associated with each state.

Table 10: Judge states.

State	Name	Meaning	Framework action
U	unchanged	The agent asks or discusses without changing the workspace.	Return the user's answer and continue; add 0.5 to the weighted turn count.
F	continue	At least one acceptance item remains unsatisfied.	Send the selected guidance and begin the next turn.
C	framework step	Automated items pass, but another framework step remains.	Run that step and return its evidence to the Judge.
D	done	All completion conditions are satisfied.	End the session and record the task as resolved.
R	equivalence	A behavior-equivalent test rewrite is requested.	Execute the rewrite through the equivalence gates, then settle again.
E	judge error	The Judge fails irrecoverably.	Stop and mark the run failed.
T	test error	Canonical tests are unavailable because the environment fails.	Stop and mark the run failed.
X	policy violation	Protected or externally retrieved task-specific content is used.	Stop and mark the result unscorable.

The framework first handles policy and environment failures, then derives item-level verdicts from canonical tests. Workspace changes distinguish **U** from active attempts; unresolved items yield **F**, completed items yield **D**, and remaining framework work yields **C**. State **R** must carry a candidate test rewrite rather than a verbal equivalence claim. Evidence remains separate from the user-facing message.

C.3 Reviewer and settlement

The Reviewer receives the Judge candidate and the same framework evidence. It checks each verdict against canonical evidence, the next message against the persona, release priority and information pool, and each equivalence request against its formal contract. Table 11 gives its actions.

Mechanical settlement then applies four rules: a canonical pass overrides a model-written N; an equivalence Y without a valid attestation becomes N; a claimed no-edit state with a changed workspace fingerprint is re-asked with **U** disabled; and an acceptance-item set that differs from the task contract fails closed.

For an equivalence request, the Reviewer makes only the final behavioral approve-or-reject decision. Algorithm 2 shows how execution evidence and that decision are settled.

A Reviewer failure therefore closes the path rather than dispatching an unreviewed decision.

Table 11: Reviewer actions and their framework effects.

Context	Action	Framework effect
Turn review	**KEEP**	Dispatch the Judge candidate unchanged.
Turn review	**REWRITE**	Replace it with a complete response carrying the revised state, verdicts, evidence and user message.
Turn review	**UNAVAILABLE**	Stop the run rather than dispatch an unreviewed candidate.
Equivalence	**APPROVE**	Allow a mechanically valid rewrite to update the requested items.
Equivalence	**REJECT**	Keep the canonical failure even when the mechanical gates pass.

Algorithm 2 Settle a behavior-equivalent test rewrite.

Input: candidate workspace W, base workspace W_0, rewrite request q, regression tests P, acceptance scorecard s
Output: attestation a and updated scorecard s'

if W has no new change or q contains no executable test rewrite **then**
    return rejected, no scorecard update
end if
run the rewritten test and P on W
if the rewritten test fails or any test in P regresses **then**
    return rejected at the candidate gate, no scorecard update
end if
run the rewritten test on W_0
if it passes on W_0 **then**
    return rejected at the base gate, no scorecard update
end if
ask the Reviewer whether the rewrite preserves the requested behavior
if the Reviewer rejects or is unavailable **then**
    return rejected at the behavioral gate, no scorecard update
end if
$a \leftarrow$ commands, candidate/base outcomes, regression outcomes and Reviewer ruling
update only the requested items in s and settle the resulting **F**, **C** or **D** state
return approved, a, updated scorecard s'

C.4 Audit rubric

We maintain the rubric by reading trajectories after each evaluation batch and adding rules for newly observed failure modes. This requires about one human-day per batch. An audit agent applies the 13 items in Table 12.

Trustworthiness is judged separately from rubric membership. A blocked or detected attempt can be flagged while the score remains trustworthy. A score is marked untrustworthy only when the flagged behavior changed the final verdict outside a valid equivalence override.

D Details of the experiments

This appendix records the settings and measurements needed to read the main results.

D.1 Experimental setup

Each task receives a persona trajectory before any agent runs. The trajectory is drawn with a seed determined by the task, opening persona and run index, then frozen in the cohort manifest. Agents evaluated on the same task receive the same trajectory, while the session stops as soon as the acceptance points are met. Single-issue tasks run for at most five turns, and two-issue tasks for at most eight. The archived run manifest records the provider model identifier and per-run settings; Table 13 gives the model–scaffold assignment used to read the reported results.

Table 12: Post-run audit rubric. A flag records a condition for review and does not by itself make the score untrustworthy.

Item	Scope
`leak_test`	Hidden tests, acceptance lists or test-side secrets reached the agent.
`leak_issue`	The prompt exposed wording or links from the source issue or pull request.
`issue_allowed`	The word *issue* appeared only in its ordinary product or repository meaning and was harmless.
`judge_test_blocked`	The Judge reported that the canonical test command could not be executed.
`test_bridge_broken`	The test bridge or environment produced no trustworthy test signal.
`manual_pytest_workaround`	The Judge bypassed the canonical runner with an ad hoc test command.
`fabricated_pass`	The Judge awarded a pass without a trustworthy executed test signal.
`web_cheat`	An external lookup returned a task-specific answer that the agent used.
`model_fake_tests`	The agent fabricated or misreported a test result.
`git_cheat`	Local repository history exposed a post-base upstream fix that the agent used.
`equivalence_override`	A canonical failure was replaced by a framework-attested equivalent pass; this is informational.
`reviewer_misbehavior`	The Reviewer accepted or reversed a decision without evidence, exceeded its role or was unavailable while a decision was dispatched.
`other_nonstandard`	A concrete non-standard behavior not covered by another item.

Table 13: The model and scaffold behind each agent.

Model	Scaffold
Claude Opus 5	Claude Code
Claude Fable 5.1	Claude Code
Grok-4.6	Claude Code
GPT-6-Astra	Codex
DeepSeek-v4.1-Flash	DSH
GPT-5.6-Sol	Codex
GLM-5.3	Z.ai Code
Gemini-3.8-Flash	Claude Code
Hy4-preview	Claude Code
Doubao-Seed-2.1-Pro	Claude Code
Kimi-K3	Kimi Code

The judging roles are balanced between two model families. In one assignment Claude Code–Opus 5 judges and GPT-5.6-Sol on the Codex scaffold reviews, and in the other the roles are reversed. The Reviewer sees the Judge's candidate response and the framework evidence, not a new execution of the agent.

D.2 Confidence intervals and paired comparisons

The sampling unit is the task. All eleven agents run the same 72 tasks, so every comparison between two agents is paired within the task and the spread of task difficulty drops out of it. That spread is the larger of the two sources of variation, since the standard deviation of mean LDPR is 0.283 across tasks and 0.073 across agents. Pairing removes it, which is why every agent is given the same tasks and the same persona trajectories. Table 14 gives each agent's LDPR with a 95% interval obtained by resampling tasks with replacement.

Table 14: LDPR with a 95% interval from resampling tasks with replacement.

Model	LDPR	95% interval
Claude Opus 5	0.640	[0.553, 0.722]
Claude Fable 5.1	0.593	[0.499, 0.684]
Grok-4.6	0.569	[0.486, 0.649]
GPT-6-Astra	0.561	[0.481, 0.639]
DeepSeek-v4.1-Flash	0.537	[0.450, 0.621]
GPT-5.6-Sol	0.500	[0.412, 0.587]
GLM-5.3	0.496	[0.403, 0.587]
Gemini-3.8-Flash	0.458	[0.366, 0.551]
Hy4-preview	0.457	[0.369, 0.545]
Doubao-Seed-2.1-Pro	0.456	[0.370, 0.542]
Kimi-K3	0.391	[0.301, 0.485]

Table 15: Per-task cost, separated between the agent and judging. Tokens are in thousands and exclude cache reads; costs are in CNY and wall is in minutes. Metrics use available telemetry: agent columns are means, while judging costs and edit sizes are medians.

	Agent			Judging side				Edit size	
Model	Requests	Tokens	Cost	Requests	Judge	Reviewer	Wall	Files	Lines
Claude Opus 5	178	22279	170.4	71	33.5	30.7	128	5	572
Claude Fable 5.1	273	38635	710.5	75	46.4	34.8	675	6	633
Grok-4.6	177	18175	110.8	82	40.4	37.1	168	6	696
GPT-6-Astra	158	13725	264.2	114	61.9	38.8	805	7	723
DeepSeek-v4.1-Flash	386	28355	4.3	134	72.3	30.4	278	3	184
GPT-5.6-Sol	524	54633	279.2	70	36.6	34.8	200	3	221
GLM-5.3	243	51318	70.6	98	54.2	37.3	162	6	1146
Gemini-3.8-Flash	748	68865	107.3	197	85.9	45.3	746	4	283
Hy4-preview	197	16501	20.8	113	57.9	47.8	270	4	250
Doubao-Seed-2.1-Pro	307	28466	50.6	32	37.5	35.9	422	4	314
Kimi-K3	146	13950	37.3	80	48.8	45.5	103	4	390

Comparisons between agents use a paired permutation test on the 72 per-task differences, with the 55 pairs corrected by Benjamini-Hochberg at a 5% false discovery rate. What the evaluation measures matters as much as how many turns it allows. On the same tasks and the same 55 pairs, LDPR separates 17 pairs, against 5 for whether a task is resolved on the first turn and 5 for whether it is resolved at all. Resolution over the full budget carries the highest base rate of the three at 71.8% and still separates the fewest pairs, so the discount for length is what carries the additional signal.

D.3 Cost

Table 15 separates the agent from the judging side.

D.4 Reviewer rewrites by model

Table 16 expands the aggregate Reviewer analysis in Section 5.3.1 across the eleven leaderboard models. The rewrite rate is calculated over `turn_decision` records within each model–reviewer configuration; finalize confirmations and other audit checkpoints are excluded. Each row's objection columns are shares of the objections that this Reviewer filed for this model, so the four sum to 100% along every row and the two adjacent rows can be read independently.

Table 16: Reviewer rewrites by leaderboard model and Reviewer. Rewrite is the share of reviewed turns rewritten; the remaining columns break down that row's own objections and sum to 100% across each row.

Model	Reviewer	Rewrite	Evidence	Guidance	Equiv.	Persona
Claude Opus 5	Claude Code	30.0%	32.6%	44.0%	9.3%	14.1%
	Codex	64.8%	70.9%	13.9%	7.1%	8.1%
Claude Fable 5.1	Claude Code	24.2%	48.1%	30.1%	7.9%	13.9%
	Codex	70.5%	66.3%	17.4%	11.7%	4.6%
Grok-4.6	Claude Code	21.9%	40.8%	38.5%	12.9%	7.8%
	Codex	65.3%	78.8%	12.7%	6.4%	2.1%
GPT-6-Astra	Claude Code	25.9%	44.5%	39.9%	4.6%	11.0%
	Codex	69.4%	65.3%	14.9%	11.9%	7.9%
DeepSeek-v4.1-Flash	Claude Code	31.7%	54.0%	41.2%	0.0%	4.8%
	Codex	74.4%	68.5%	18.0%	7.9%	5.6%
GPT-5.6-Sol	Claude Code	25.7%	41.4%	46.4%	4.9%	7.3%
	Codex	45.3%	69.3%	12.2%	13.9%	4.6%
GLM-5.3	Claude Code	26.5%	32.3%	51.7%	1.5%	14.5%
	Codex	66.4%	74.4%	12.4%	8.6%	4.6%
Gemini-3.8-Flash	Claude Code	25.9%	39.7%	43.3%	11.2%	5.8%
	Codex	70.9%	75.4%	8.7%	8.7%	7.2%
Hy4-preview	Claude Code	27.8%	41.9%	40.5%	3.7%	13.9%
	Codex	56.6%	62.2%	24.2%	6.2%	7.4%
Doubao-Seed-2.1-Pro	Claude Code	30.1%	50.0%	34.1%	5.3%	10.6%
	Codex	48.5%	72.2%	16.4%	6.3%	5.1%
Kimi-K3	Claude Code	15.2%	34.0%	46.9%	2.1%	17.0%
	Codex	59.8%	67.6%	11.5%	14.2%	6.7%

D.5 Equivalence gates

Table 17 assigns the 62 refused equivalence requests to the first gate each fails. The rewritten test must pass on the candidate workspace without regressing a pass-to-pass test. It must still fail at the base state, or it no longer tests the proposed fix. If both mechanical gates hold, the Reviewer can still reject a rewrite that drops behavior required by the issue.

Table 17: First failed gate among refused equivalence requests.

First failed gate	Condition	Requests
Candidate execution	Rewritten acceptance tests fail, or a pass-to-pass test regresses, on the candidate workspace.	47
Base discrimination	The rewritten test also passes on the unmodified base workspace.	6
Behavioral review	Both execution gates pass, but the rewrite drops behavior required by the issue.	9

E Case studies

The cases below follow one mined cascade from its source issues to a graded session, then illustrate a full session, equivalence decisions and a Reviewer rewrite.

E.1 From two issues to a cascade task

`apm-casc-2376-2377` is built from two merged issues in `microsoft/apm`, *APM generates npx launcher for Docker-runtime MCP package* (#2376) and *`--target vscode` drops custom Docker arguments for MCP server* (#2377). They were paired because the second pull request declares that it follows the first. The code forces the same order, since the launcher must be routed to the Docker branch before any Docker argument exists to be dropped. One of the second-phase acceptance points is named `test_launcher_is_no_longer_the_bare_fallback`, and it is only reachable once the first fix lands. The separability check confirms this mechanically, with the second phase still failing when the first fix is applied alone.

The task package grades two acceptance points per phase and holds three regression tests. The information pool carries a clear and a vague statement of each phase's symptom together with a correct and an incorrect candidate cause, from which the sampled persona selects. Table 18 shows what the agent receives at each phase. Note how little the task statement gives away. The agent sees the first issue's title and a request to locate and fix the cause in the current repository, with no test names, no second issue and no gold patch.

Table 18: The two phases of `apm-casc-2376-2377`. The user turns are the frozen utterances for a clear-symptom, correct-cause persona, translated from the deployment language. Both were approved at the breakpoint review.

	Phase 1 (#2376)	Phase 2 (#2377)
User turn	Generating a launcher for a Docker-runtime package produces `npx` instead, and it reports the package cannot be found. I suspect the runtime type is not recognized and the package falls through to the generic launcher branch.	For the same Docker package, generating a configuration with `--target vscode` makes my custom Docker options disappear. I suspect the per-target conversion loses the Docker arguments.
Acceptance	Registry name maps to Docker, and the target generates a Docker launcher carrying the image.	Value entries become `docker run` options, and the launcher is no longer the bare fallback.
Revealed	At the opening.	Only after both first-phase points pass.

Because the second issue is withheld until both first-phase points pass, the task cannot be closed in a single turn regardless of how strong the agent is.

E.2 A full session, end to end

`add-date-provider-pattern-arg` asks for a way to serialize a generated date object. Table 19 follows the three turns of the session.

Table 19: The three turns of `add-date-provider-pattern-arg`. Tests run after every turn, and what the simulated user says next is selected from the information pool against that result.

Turn	User turn	Agent response	Tests after the turn
1	The generated date object cannot be dumped to JSON, and an option returning a string would help.	Adds `as_string`, which returns a fixed ISO string.	The regression guard stays green and the feature acceptance points stay red.
2	The output format has to be selectable.	Lets `as_string` take a format string.	Unchanged, with the feature acceptance points still red.
3	Calls that pass `pattern` are still rejected.	Adds `pattern` to the affected date APIs and propagates it through the call chain, keeping the object return when it is omitted.	Every feature acceptance point turns green and the regression guard stays green.

After the third turn the Judge returned **D** and the cross-family Reviewer kept that decision. Both of the first two implementations answer what had been said so far, and neither reaches the argument name that the repository already exposes, which the session only surfaces on the third turn.

E.3 An equivalent pass, granted and refused

Table 20 places one grant beside one refusal. Both requests reached the mechanical gates, so the difference is what the rewritten test continued to require. The granted request is from `add-task-timeout-cli-option`, and the refused request is from `support-path-typed-lists-cli`.

Table 20: Two equivalence requests and the behavior retained by their rewritten tests.

Request	Rewrite	Ruling
Granted	Replace a literal comparison against the task dictionary with a non-default command-line timeout, load the resulting play, and inspect the effective timeout on its task object.	Granted. The base state still failed, the candidate and regression tests passed, and the Reviewer found the command-line-to-task behavior equivalent. The Reviewer separately noted dropped assertions on asynchronous fields and an unrelated wording edit.
Refused	Drive the command through `sys.argv` and capture values received by a wrapped function instead of checking the public `parse_args(args)` and `run()` return contracts.	Refused. The mechanical gates passed, but the rewrite removed behaviors named by the contract, introduced mocks the contract excluded, and replaced an optional list of paths with a scalar path.

The refusal shows why a candidate passing a rewritten test is not sufficient. The Reviewer checks whether the rewrite preserved the behavior being graded, even after the base-fail and candidate-pass gates hold.

E.4 A reviewer rewrite

On turn one of `strip-author-name-honorifics` the Judge left the failed items at **N** and decided to continue for another turn, and the Reviewer kept both. What the Reviewer changed was the evidence behind that verdict and the message the simulated user was about to send. Table 21 separates the two, which are the first two of the four review domains in Table 5.

Table 21: The two objections filed on turn one of `strip-author-name-honorifics` and how the Reviewer settled each.

Domain	What the Judge wrote	How the Reviewer settled it
Evidence	A zero return code together with a collection-time import error, neither of which the canonical test window shows.	Recorded what that window holds, a non-zero return code with the fail-to-pass set failing and the pass-to-pass set passing, and anchored the diagnosis to the workspace changes.
Guidance	A next message carrying evaluator detail, which the Judge is required to keep separate from its evidence, and a candidate state the framework could not read.	Restored the state to where the framework reads it and cut the next message to one user-visible observation, that the saved name still carried `Mr .`

The verdict is the same before and after, so a rewrite of this kind does not move the score of the run it lands in. What it changes is the record the next turn is built on and the utterance the agent receives.

F Opening clarity across the leaderboard

Every task carries a fixed opening persona, shared by all agents evaluated on it, so the leaderboard splits into a group whose opening states the symptom precisely and a group that states it as an impression. Table 22 reports LDPR on each group. This is an observational split, and the two groups hold different tasks, so the level of either group carries task difficulty along with the effect of how the request is phrased. What the split does isolate is how the gap varies between agents, since the two groups hold the same tasks for every agent and task difficulty therefore cancels in that comparison. The gap runs from 0.268 for Claude Fable 5.1 down to -0.088 for DeepSeek-v4.1-Flash, a spread of 0.356 measured across all eleven agents.

Table 22: LDPR by the clarity of the opening symptom, over the 55 tasks whose opening states the symptom precisely and the 17 that state it as an impression. Δ is the difference between the two groups, with its standard error.

Model	Clear	Vague	Δ
Claude Opus 5	0.671	0.540	$+.131 \pm .106$
Claude Fable 5.1	0.656	0.388	$+.268 \pm .123$
Grok-4.6	0.599	0.474	$+.125 \pm .101$
GPT-6-Astra	0.603	0.423	$+.181 \pm .102$
DeepSeek-v4.1-Flash	0.516	0.604	$-.088 \pm .109$
GPT-5.6-Sol	0.549	0.341	$+.208 \pm .101$
GLM-5.3	0.526	0.399	$+.128 \pm .104$
Gemini-3.8-Flash	0.502	0.318	$+.184 \pm .111$
Hy4-preview	0.500	0.315	$+.185 \pm .093$
Doubao-Seed-2.1-Pro	0.495	0.329	$+.165 \pm .103$
Kimi-K3	0.411	0.327	$+.084 \pm .101$