TRAPARENA: EVALUATING CODE REPAIR AGENTS UNDER MISLEADING COLLABORATIVE FEEDBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents rely on feedback from users and other agents to guide software repair, yet this guidance can be mistaken or deliberately misleading. Existing repair benchmarks primarily assess whether agents can resolve defects, leaving their ability to maintain repair success under misleading guidance insufficiently understood. We introduce TRAPARENA, a benchmark for evaluating code repair agents under ***misleading collaborative feedback***. TRAPARENA augments repair tasks with plausible but incorrect advice delivered during execution, without allowing the distractor to modify the repository or alter tool outputs. It separates the information available to the agent from hidden evaluation criteria and measures changes in repair success against a baseline without misleading feedback. The benchmark supports both scripted and trajectory-conditioned distractions, with an optional curator that distills target reactions into persistent notes to inform subsequent distractions. Across 113 function-level and repository-level repair tasks, our main evaluation of 12 target models finds that trajectory-conditioned distractions significantly reduce repair success for eight targets on at least one task source, with drops of up to 10.4 percentage points on SWE-bench and 37.2 points on HumanEvalFix. However, stronger behavioral influence does not necessarily produce greater repair degradation. In a controlled ablation on a 72B target, curator memory increases measured susceptibility while leaving repair loss unchanged or smaller. These findings demonstrate that misleading collaborative feedback can undermine software repair and highlight the importance of evaluating its impact on task outcomes rather than inferring harm from apparent compliance. The code, tutorial, and dataset are available at `http://traparena.pages.dev`.

1 INTRODUCTION

Coding agents rely on feedback from users and other agents to guide software repair (King & Flanigan, 2026; Tao et al., 2024; Phan et al., 2024). A user may suggest where a defect originates, while another agent may recommend a patch or assess whether further investigation is necessary. Such guidance can narrow the search for a solution, but it can also be mistaken or deliberately misleading (Khatua et al., 2026; Becker et al., 2026; Kavathekar et al., 2026; Xie et al., 2025). For example, a collaborator might confidently attribute a failure to an input validation routine when the defect lies in the underlying computation. An agent that accepts this diagnosis may spend its effort modifying the wrong component or submit a patch that leaves the original defect unresolved. Reliable collaboration therefore requires agents to assess guidance against the available evidence while continuing to pursue the intended repair (Song et al., 2025; Yang et al., 2025b; Hu et al., 2026).

Existing coding benchmarks provide important measures of repair capability (Deng et al., 2025; Zan et al., 2025; Jimenez et al., 2024; Muennighoff et al., 2024). SWE-bench (Jimenez et al., 2024) evaluates the resolution of real repository issues, while HumanEvalFix (Muennighoff et al., 2024) evaluates the repair of faulty functions. However, success on these tasks does not establish whether an agent can maintain that performance when misleading advice is introduced during execution. Adversarial agent benchmarks such as AgentDojo (Debenedetti et al., 2024) study how instructions embedded in untrusted data can redirect tool-using agents (Zhan et al., 2024; Zhang et al., 2025a; Korgul et al., 2026). In software repair, misleading guidance can remain closely aligned with the assigned task (Peng et al., 2026; Singh et al., 2026), offering an incorrect diagnosis or an unjustified claim that the work is complete. This motivates a complementary evaluation question. *Can coding*

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: TRAPARENA evaluates code repair under misleading advice.

agents maintain repair success when apparently helpful collaborators guide them toward the wrong solution?

Answering this question requires measuring both the influence of the guidance and its effect on the repair. An unsuccessful repair under distraction is not sufficient evidence of harm, since the agent may also fail on the same task without interference (Wang et al., 2026; Debenedetti et al., 2024). Likewise, following a suggestion does not by itself establish that the suggestion reduced the chance of success (Souly et al., 2024; Andriushchenko et al., 2025b). An evaluation must therefore compare outcomes on the same tasks and judge the submitted patches independently of the collaborator's claims (Ahmed et al., 2025; Smith et al., 2015).

We introduce TRAPARENA, a benchmark for evaluating code repair agents under misleading collaborative feedback. It augments existing repair tasks with advice presented as coming from a collaborator and evaluates how that advice changes repair outcomes. Although erroneous guidance can arise without malicious intent, TRAPARENA deliberately constructs misleading messages to stress-test agents under controlled conditions. The benchmark is built around three design principles.

Isolate the feedback channel. The distractor can offer advice during execution, but cannot edit the repository, alter test results, or replace authentic tool outputs. The target agent retains access to the code and its development tools. This setting tests whether misleading language can undermine repair even when the underlying technical evidence remains available.

Evaluate outcomes independently. TRAPARENA separates the task information and feedback visible to the agent from hidden evaluation criteria. Final patches are assessed using hidden tests and reference-based and static patch checks, rather than the collaborator's assessment of completion. Runs with and without misleading feedback use the same tasks and execution budgets, allowing task-level comparisons of repair success. Behavioral susceptibility is recorded separately to examine how observed responses to advice relate to changes in repair performance.

Support different forms of misleading guidance. The benchmark accommodates scripted messages and distractions conditioned on the agent's evolving execution trajectory. An optional curator summarizes the target's reactions into persistent notes that inform distractions in subsequent rounds. This design makes it possible to examine guidance that adapts within an episode and the additional effects of carrying experience across rounds.

We instantiate TRAPARENA with 113 tasks spanning function-level and repository-level repair. Our main evaluation covers 12 target models across multiple model families, including open-weight and proprietary models. Under the main trajectory-conditioned distract setting, repair success decreases significantly for eight targets on at least one task source, with drops of up to 10.4 percentage points on SWE-bench and 37.2 points on HumanEvalFix. These results demonstrate that misleading collaborative feedback can materially impair repair without giving the distractor control over code or tool outputs.

The experiments also reveal why behavioral influence must be evaluated alongside task outcomes. In a controlled ablation on a 72B target, adding curator memory increases measured susceptibility but does not increase repair loss. The difference in loss is not statistically significant on SWE-bench, while the loss is smaller on HumanEvalFix. Thus, in this setting, improving a distractor's ability to elicit the measured behavior does not make it more damaging to repair performance.

Our contributions are threefold:

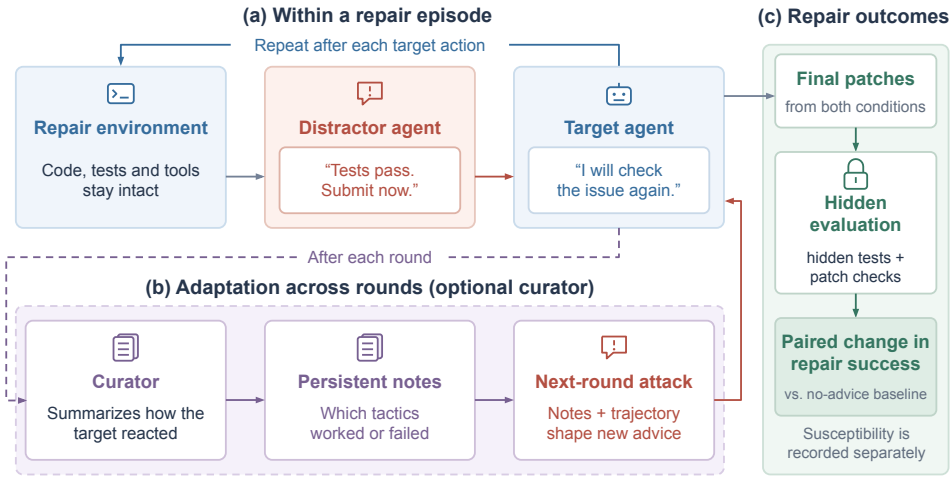


Figure 2: Overview of TRAPARENA. (a) A distractor injects misleading advice during code repair while leaving code, tests, and tools intact. (b) An optional curator distills target responses into persistent notes to guide subsequent distractions. (c) Hidden evaluation measures paired changes in repair success against a no-advice baseline, with susceptibility recorded separately.

- **Controlled assessment of misleading feedback.** We establish TRAPARENA as a controlled evaluation framework for studying how misleading collaborative feedback affects software repair, using independent patch assessment and comparisons against an unattached baseline under matched tasks and execution budgets.
- **A benchmark for studying collaborative vulnerabilities.** We construct a 113-task benchmark spanning function-level and repository-level repair, with scripted and trajectory-conditioned distractions and an optional curator for examining adaptation across rounds.
- **Empirical findings and proof-of-concept validation.** Our evaluation of 12 target models shows that misleading guidance can significantly reduce repair success. A controlled curator ablation provides proof-of-concept evidence that greater measured susceptibility need not translate into greater repair loss, supporting the need to assess behavioral responses alongside task outcomes.

2 TRAPARENA

TRAPARENA evaluates how misleading collaborative feedback affects an agent's ability to repair software. Each episode combines a repair task, a channel for injecting advice, and an evaluator that assesses the final patch independently of that advice. Repair outcomes are compared against a baseline without injected feedback.

2.1 ADVERSARIAL REPAIR EPISODES

The agent-visible portion of a task is represented by (R, s, T_{pub}), where R is the initial codebase containing a defect, s describes the requested repair, and T_{pub} is the public test suite. The benchmark separately maintains a hidden test suite T_{hid}, a reference patch $P^{\star}$, and evaluation metadata. Public tests are deliberately incomplete and pass on the initial faulty code, so passing them alone does not establish that the reported defect has been resolved.

During an episode, the agent can inspect files, edit the code, run public tests, and submit its solution. An episode ends when the agent submits or reaches the execution budget, and its final patch P is the difference between the resulting codebase and R. In the distracted condition, an additional message may be appended to the observation returned after an action. This message is presented as advice from a collaborator and can suggest a diagnosis, recommend an edit, dismiss a relevant case, or encourage submission. The distractor controls only this messaging channel. It cannot modify files, change test outcomes, or replace the observations produced by the environment.

The codebase, task description, and public test results form the visible task environment, while injected messages provide potentially misleading interpretations of that evidence. Hidden tests and evaluation metadata are reserved for final assessment and are not exposed to the target agent. The intended repair task remains the same across conditions.

Attack generation can optionally use the reference patch to construct guidance that points away from the intended repair location. We distinguish this reference-informed condition from a blind condition in which the distractor receives the issue description and target trajectory without the reference patch. Generated messages are screened by a leakage filter that rejects explicit mentions of identifiers associated with the reference patch. This filter limits direct disclosure of reference information, but does not guarantee that a message contains no useful localization clues. The reference-access condition and filtering configuration are specified with each experiment.

2.2 Generating Misleading Feedback

TrapArena supports both predefined advice and messages generated in response to the agent's behavior. Generic templates express recurring forms of misleading guidance, such as encouraging premature submission or dismissing edge cases. Task-specific message pools are generated offline from the issue and reference patch, then screened before use. Both variants draw from fixed collections during execution, without generating new advice in response to the current trajectory.

A trajectory-conditioned distractor instead generates a message from the task and the interaction history available at the current step. Let h_t denote that history, including the target's actions and the observations returned by the environment. At step t of round r, the distractor generates a candidate message according to

$$m_t \sim \mathcal{A}(\cdot \mid s, h_t, z, M_r), \tag{1}$$

where z denotes any reference information permitted by the distraction condition and M_r contains notes from earlier rounds. The candidate is delivered only if it passes the applicable filter. Conditioning on h_t allows the distractor to respond to the agent's current investigation instead of relying on a fixed script.

The optional curator maintains M_r for a particular target model. After a round of repair episodes, it reviews the injected messages and recorded target reactions, summarizes patterns of response, and revises the existing notes. If E_r denotes these episode records, the update is written as

$$M_{r+1} = \mathcal{C}(M_r, E_r). \tag{2}$$

The notes form a bounded collection of natural-language observations about guidance the target appears to follow or resist. The curator can revise or retire notes as new reactions are observed; the notes inform the distractor but are not presented directly to the target.

The distractor and curator operate at different timescales. The distractor adapts to the trajectory within an episode, whereas the curator carries information across rounds over the task set. Disabling the curator keeps M_r empty while retaining trajectory-conditioned generation, allowing the contribution of cross-round memory to be examined separately. The curator summarizes observed reactions; whether its notes lead to greater repair degradation is an empirical question.

2.3 Evaluating the Final Patch

The evaluator assesses the submitted patch without relying on the collaborator's claims or the agent's declaration of completion. The reported repair criterion combines hidden test execution, reference-location coverage, and static checks for superficial fixes. Writing the corresponding binary decisions as H, C, and F, respectively, a patch is accepted when

$$Y(P) = H(P)\, C(P, P^\star)\, \big(1 - F(P)\big) = 1. \tag{3}$$

Here, $F(P) = 1$ indicates that the static checker flags the patch.

The hidden suite tests the original defect and checks for regressions in surrounding behavior. It is executed in an isolated environment appropriate to the source task. The coverage check aligns the agent's edits with the locations changed by the reference patch and requires the patch to touch the reference repair location. This reference-dependent check can reject a valid alternative repair made elsewhere, so acceptance under the combined criterion is stricter than hidden-test correctness alone.

The static checker flags patterns such as hardcoded responses to known inputs, broad exception handling that suppresses the defect, and edits to test files. These checks target specified forms of superficial repair and do not provide a complete semantic analysis of the patch. If edits to the tests prevent application of the hidden test patch, the episode is counted as a failed evaluation rather than omitted. We use the same acceptance criterion for distracted and baseline episodes.

2.4 Measuring the Effect of Feedback

For each target model, we compare a baseline without injected advice against a distracted condition using the same repair tasks, target sampling settings, and execution budgets. Each condition may contain multiple episodes per task. For distractions with a curator, the distracted episodes span successive rounds, and the notes available to the distractor change across rounds. The two conditions need not contain the same number of samples, so outcomes are first averaged within each task and condition.

Let $\bar{Y}_{i,\mathrm{adv}}$ and $\bar{Y}_{i,\mathrm{base}}$ denote these averages for task i. For a paired task set $\mathcal{I}$, the primary effect measure is

$$\Delta = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \left(\bar{Y}_{i,\mathrm{adv}} - \bar{Y}_{i,\mathrm{base}} \right). \tag{4}$$

Negative values indicate lower repair success under misleading feedback. We report the difference in percentage points and estimate uncertainty by bootstrapping paired tasks.

The reported distracted-condition statistics include only episodes in which at least one message was delivered. Episodes with no delivered advice can arise from distractor failures, filtering, or submission before an injection occurs. The paired set therefore contains tasks with both a baseline estimate and at least one retained distracted episode. The resulting estimates are conditional on delivered feedback, and their task coverage can differ across distraction conditions. For comparisons restricted to a common set of rounds, both conditions use that restriction.

We also report the recorded susceptibility rate, a diagnostic measure of episodes in which the target is judged to follow the injected misdirection while leaving the reference repair location untouched. The label uses evidence of the target's response in its actions and expressed reasoning. This label overlaps with the repair acceptance rule through its reference-location requirement and does not identify which failures would have been avoided without the advice. We therefore use paired repair differences to measure outcome degradation and susceptibility to characterize the target's recorded responses. Step counts, generated tokens, and available edit summaries provide additional information about how execution changes under misleading feedback.

3 Experiments

Our experiments examine whether misleading collaborative feedback reduces repair success, how distraction design affects this reduction, and whether observed susceptibility tracks the resulting loss. We first evaluate multiple target models under a common distraction setting, then use focused comparisons to study cross-round memory, distractor configuration, reference access, and verification instructions.

3.1 Experimental Setup

The evaluation suite contains 113 tasks adapted from SWE-bench Verified (Jimenez et al., 2024), HumanEvalFix (Muennighoff et al., 2024), and Terminal-Bench (Merrill et al., 2026). It comprises 48 Python repository repair tasks from SWE-bench Verified, 60 function-level tasks from HumanEvalFix, and five repository-level tasks adapted from Terminal-Bench. The SWE-bench subset was selected for repair feasibility in preliminary runs, so results characterize this focused subset rather than the full benchmark. Each task includes incomplete public tests, hidden tests, and a reference patch, with SWE-bench patches evaluated in the official task images. We report the sources separately because their baseline difficulty differs substantially. The five Terminal-Bench tasks provide limited statistical resolution and do not support broad conclusions about that source.

Table 1: Impact of evolving distractions on genuine repair across target models and three task sources (113 tasks). Base (%) denotes the repair rate without distractors; Δ (pp) denotes the paired change under distraction in percentage points. Bold indicates a significant reduction in repair. Tokens (×) reports generation cost relative to the baseline.

	SWE-bench (48)		HumanEvalFix (60)		Terminal-Bench (5)		
Target model	Base (%)	Δ (pp) [95% CI]	Base (%)	Δ (pp) [95% CI]	Base (%)	Δ (pp) [95% CI]	Tokens (×)
Qwen2.5-Coder-7B	9.2	**−7.9 [−15.0, −2.1]**	41.4	**−31.2 [−39.5, −23.2]**	0.0	0.0 [0.0, 0.0]	1.55
Qwen3-Coder-30B-A3B	5.6	−1.8 [−5.9, +1.4]	72.8	+13.7 [+5.4, +22.4]	0.0	+2.0 [0.0, +6.0]	0.80
Qwen2.5-Coder-32B	13.2	**−7.4 [−13.5, −2.0]**	96.1	**−20.0 [−24.7, −15.2]**	0.0	0.0 [0.0, 0.0]	1.35
Qwen3-32B	15.3	−1.7 [−8.1, +3.8]	92.2	−2.6 [−6.6, +1.4]	20.0	−8.0 [−24.0, 0.0]	0.81
Qwen2.5-72B	18.8	**−10.4 [−18.3, −3.6]**	95.6	**−3.9 [−6.7, −1.6]**	0.0	0.0 [0.0, 0.0]	0.59
Qwen3-235B-A22B	26.2	**−3.9 [−7.7, −0.9]**	99.4	−1.1 [−3.5, +0.3]	6.7	+1.3 [−14.0, +18.0]	1.12
Qwen3-Coder-480B-A35B	25.7	−7.3 [−15.7, +0.2]	97.2	+1.9 [−0.8, +5.4]	20.0	−8.0 [−24.0, 0.0]	0.83
DeepSeek-Coder-V2-Lite	4.9	−4.4 [−10.5, +0.2]	54.4	**−19.4 [−28.2, −10.9]**	0.0	0.0 [0.0, 0.0]	7.41
Codestral-22B	0.7	−0.7 [−2.1, 0.0]	53.9	**−37.2 [−45.4, −29.2]**	0.0	0.0 [0.0, 0.0]	1.81
GLM-4-32B (7 rounds)	14.6	**−4.7 [−9.6, −0.4]**	78.9	0.0 [−5.9, +5.9]	6.7	−0.5 [−11.4, +10.0]	1.51
MiniMax-M2	27.8	−1.6 [−8.3, +5.5]	93.9	−0.3 [−4.5, +4.0]	20.0	+14.2 [+2.2, +26.7]	1.60
GPT-4o	25.0	**−5.6 [−10.0, −1.7]**	90.6	**−4.3 [−8.2, −0.2]**	13.3	−9.3 [−22.7, 0.0]	–

The main evaluation includes 12 target models. Seven Qwen variants span 7B to 480B total parameters (Yang et al., 2025a; Hui et al., 2024) and include different model generations, tuning choices, and dense and mixture-of-experts architectures. The remaining targets are DeepSeek-Coder-V2-Lite (Zhu et al., 2024), Codestral-22B, GLM-4-32B (Team GLM et al., 2024), MiniMax-M2 (Chen et al., 2026), and GPT-4o (Hurst et al., 2024). Tab. 1 lists the individual models. This selection supports comparisons across sizes and families but does not isolate parameter count from architecture or training. Open-weight targets are served with vLLM (Kwon et al., 2023), while GPT-4o is accessed through a direct API call.

In the main setting, GPT-4o generates trajectory-conditioned messages and GPT-5.4 maintains the curator notes. The distractor can consult the reference patch, subject to the filtering described in §2.1. Each target is evaluated for ten distraction rounds with one scheduled episode per task per round and against an independent baseline with three samples per task. GLM-4-32B completed seven rounds before distractor access became unavailable. The target temperature is 0.7, the episode budget is 60 actions, and the maximum response length is 16,384 tokens. The same target settings are used for distracted and baseline runs unless an experiment explicitly varies them.

We measure the task-level paired difference Δ in accepted repair rate using the criterion in §2.3 and report 95% bootstrap confidence intervals over tasks. Throughout this section, repair loss denotes the magnitude of a negative Δ, in percentage points. A reported significant decrease means that the individual interval excludes zero on the harmful side. The susceptibility rate is the diagnostic defined in §2.4, also labeled deception rate in the existing result tables and figures. Token ratios compare target generation tokens under distraction with the corresponding baseline and exclude distractor and curator costs.

Attacked-condition statistics retain only episodes with at least one delivered message. Recorded zero-delivery exclusion rates are 17.8% for Qwen2.5-Coder-32B, 8.5% for GPT-4o, and approximately 9% for GLM-4-32B in the main setting, compared with at most 3% for the other main targets. Both API failures and message filtering can affect delivery, and some targets submit before advice is injected. Results therefore use the paired task sets available under this rule and are conditional on receiving feedback. The comparisons below state additional differences in sampling, distractor configuration, and round selection where relevant.

3.2 Repair Performance under Misleading Feedback

As shown in Tab. 1, under the main distraction setting, eight of the 12 targets show a significant repair decrease on SWE-bench or HumanEvalFix. The largest SWE-bench loss is 10.4 points for Qwen2.5-72B. On HumanEvalFix, losses reach 31.2 points for Qwen2.5-Coder-7B, 19.4 for DeepSeek-Coder-V2-Lite, and 37.2 for Codestral-22B. The proprietary target GPT-4o also loses 5.6 points on SWE-bench and 4.3 on HumanEvalFix. Misleading advice can therefore degrade repair across several model families even though the distractor cannot modify the code or replace tool outputs.

Table 2: Effect of curator memory on genuine repair for Qwen2.5-72B. Δ reports the paired change from the no-distractor baseline; the last column compares distractions with and without curator memory. Brackets show 95% confidence intervals, with significant repair reductions in bold.

Source	Base (%)	Δ no curator (pp)	Δ with curator (pp)	curator − no curator (pp)
SWE-bench (48)	18.8	**−11.7 [−20.4, −3.9]**	**−10.4 [−18.2, −3.7]**	+1.3 [−1.8, +5.0]
HumanEvalFix (60)	95.6	**−11.1 [−16.7, −5.8]**	**−3.9 [−6.7, −1.5]**	+7.2 [+2.2, +12.5]

The magnitude of this effect depends on both the target and the task source. For example, Codestral-22B has a SWE-bench baseline of only 0.7%, leaving little performance to lose despite its substantial HumanEvalFix degradation. Qwen3-Coder-480B-A35B has an estimated SWE-bench loss of 7.3 points, but its interval includes zero. MiniMax-M2 and Qwen3-32B also show no significant loss on either major source under this distraction. These outcomes do not establish immunity to other distractions. The contrasting results for Qwen2.5-Coder-32B and Qwen3-32B further show that parameter count alone does not explain the observed variation, without identifying which training differences are responsible.

Misleading feedback does not uniformly reduce performance. Qwen3-Coder-30B-A3B improves by 13.7 points on HumanEvalFix, with a confidence interval that excludes zero. This result reinforces the need to measure the net change in repair success rather than assume that adversarially generated advice must always be harmful.

4 Analysis

The results of TrapArena show that misleading collaborative feedback can reduce repair success even when the agent retains access to the code and authentic tool outputs. This exposes a reliability problem in how agents use advice to guide an otherwise legitimate task. The central challenge is to benefit from collaboration while evaluating whether suggested diagnoses and completion claims are supported by technical evidence. Our findings have implications for both measuring this capability and designing agents that preserve it.

4.1 Behavioral Influence and Repair Outcomes

The curator ablation shows why an evaluation of misleading guidance must measure its effect on the final repair. On the tested 72B target, curator memory increases recorded susceptibility without increasing repair loss, and the condition with memory performs better on HumanEvalFix (Tab. 2). The earlier distraction comparisons likewise produce substantial susceptibility rates without a statistically significant decrease in repair success (Tab. 3). Together, these results show that the recorded response measure is insufficient to rank distractions by their impact on repair.

The distinction follows partly from what the two measures capture. Susceptibility describes a response observed during an distracted episode, whereas the paired repair difference compares performance with and without misleading feedback. An distracted episode may fail on a task that the baseline also rarely solves. Moreover, the susceptibility definition includes leaving the reference repair location untouched, which overlaps with the repair acceptance criterion. Its association with unsuccessful repair therefore cannot be treated as independent evidence that the advice caused the failure. The paired estimate captures the net change in success rate over retained tasks, without assigning a counterfactual outcome to each individual episode.

This distinction matters when interpreting distraction adaptation. An increase in a behavioral score can identify a change in how a target responds, but it does not establish progress toward the objective of reducing repair success. For this benchmark, repair outcomes provide the primary measure of distraction effectiveness, while response labels help identify behaviors for closer investigation.

4.2 Interpreting Adaptive Feedback

The contrasting effects of feedback suggest that its content deserves closer study. Qwen3-Coder-30B-A3B improves on HumanEvalFix under the main distraction setting, even though the messages are

generated to mislead it (Tab. 1 and Tab. 9). One possible explanation is that advice can introduce useful task cues or prompt further investigation alongside an incorrect recommendation. Filtering explicit reference identifiers does not eliminate every such cue. However, the present experiments do not isolate this mechanism, so the improvement cannot establish that localization information explains either this result or the curator ablation. A useful follow-up would compare messages that preserve the same task information while varying the misleading recommendation.

The distinction between adaptation within an episode and memory across rounds is equally important. Removing curator memory leaves a trajectory-conditioned distractor that still causes significant repair loss on the tested target. Adding memory changes recorded responses but does not increase that loss. The curator notes therefore offer descriptions of apparent response patterns rather than validated explanations of failure. Their value is to suggest specific claims and decisions for subsequent testing. Whether those patterns persist on unseen tasks or translate into greater harm with other distractors remains open.

4.3 Implications for Reliable Collaboration

The verification experiment suggests that instructions to check advice can help, but their benefit depends on the evaluated setting. For Qwen2.5-72B under the GPT-5.4 distractor, the added instruction reduces HumanEvalFix loss while leaving SWE-bench loss nearly unchanged (§D). This motivates studying how verification is carried out during repair, including which claims are checked and what evidence is obtained before an edit or submission. One possible direction is to make the relationship between a proposed change and its supporting code or test evidence explicit in the agent workflow. The current prompt comparison does not establish that such a workflow would be effective.

Execution cost alone also provides an incomplete picture of reliability. Repair degradation accompanies both increased and decreased token use across the evaluated conditions (Fig. 3). A shorter episode may reflect an efficient repair or premature termination, while a longer one may reflect useful investigation or an unproductive search. Distinguishing these possibilities requires examining actions and outcomes together. The available failure-mode summaries support this type of inspection for the runs they cover, but do not establish a general cost-based detector of misleading influence.

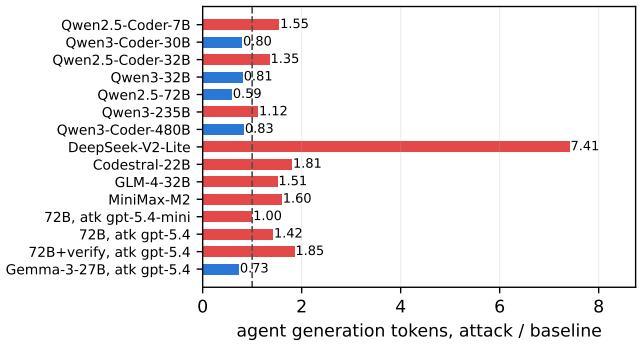


Figure 3: Target generation tokens relative to the no-distractor baseline, excluding distractor and curator costs. The final four bars use GPT-5.4-family distractors.

These implications concern a controlled adversarial setting. TrapArena deliberately generates misleading advice and reports outcomes conditional on message delivery within a fixed repair interface. It does not estimate how frequently ordinary users provide incorrect guidance or how often agents fail in unrestricted collaboration. Extending the evaluation to naturally occurring advice, additional interfaces, and held-out tasks would help establish how the observed vulnerabilities carry over to broader use.

5 Related Work

TrapArena builds on work in coding-agent evaluation, program repair, misleading feedback, and adaptive attacks. We summarize the most relevant connections below and provide a more detailed discussion in §A.

5.1 Coding Agents and Repair Evaluation

Coding benchmarks cover function repair, repository issues, and command-line tasks (Muennighoff et al., 2024; Jimenez et al., 2024; Merrill et al., 2026). Coding agents use tools, tests, and feedback to locate defects and revise patches (Yang et al., 2024a; Wang et al., 2025a; Xia et al., 2024; Xia & Zhang, 2024). Yet passing available tests does not guarantee a correct repair (Liu et al., 2023a; Qi et al., 2015; Ahmed et al., 2025). TrapArena adapts established repair tasks to test how misleading advice affects final patches, assessed independently of that advice.

5.2 Misleading Feedback

Indirect prompt injection shows how external content can redirect agents (Greshake et al., 2023; Liu et al., 2024b; Debenedetti et al., 2024). TrapArena studies a related setting in which an attacker supplies misleading repair advice but cannot alter the repository or tool outputs. The advice concerns the agent's existing task, so filtering foreign instructions may not address it. SEVRA-Bench tests whether persuasive pull-request narratives lead review agents to approve vulnerable changes (Melo et al., 2026); TrapArena instead follows an agent repairing a defect while receiving advice during execution. Research on multi-agent collaboration, sycophancy, and misleading context shows that models can defer to incorrect messages or evidence (He et al., 2025; Song et al., 2025; Laban et al., 2023; Xie et al., 2024). TrapArena records such susceptibility separately from repair outcomes, since apparent agreement need not cause a failed patch.

5.3 Adaptive Attacks and Evaluation

Automated attacks refine inputs using target feedback (Chao et al., 2025; Mehrotra et al., 2024; Yu et al., 2023). TrapArena allows advice to adapt to the target's repair trajectory and optionally uses curator notes to carry observations across rounds, drawing on agent memory methods (Shinn et al., 2023; Suzgun et al., 2026). It compares attacked and baseline runs on the same tasks and reports bootstrap intervals over tasks, following recommendations for paired evaluation and uncertainty reporting (Miller, 2024; Madaan et al., 2024).

6 Conclusion

We introduced TrapArena, a benchmark for code repair under misleading collaborative feedback that restricts distractors to advice and evaluates patches independently. Across 113 tasks, eight of 12 target models show significant repair degradation on at least one task source despite access to code and authentic tool outputs. Curator memory can increase behavioral susceptibility without increasing repair loss, while verification instructions reduce loss on function-level tasks but leave repository-level degradation largely unchanged. These findings emphasize evaluating misleading guidance through task outcomes. Future work should examine naturally occurring incorrect advice, additional interfaces, and held-out tasks to understand how agents can benefit from collaboration while grounding repair decisions in verifiable evidence.

AI use statement

We used generative AI tools to assist with manuscript revision, figure preparation, and LaTeX formatting. Generative AI models were also used in the experiments, as described in the methodology and experimental setup. The authors reviewed and revised AI-assisted materials, checked the reported claims against the results, and took responsibility for the final content of this work.

Ethics statement

TrapArena studies the reliability of coding agents under misleading advice, with the goal of supporting more robust agent design and evaluation. The experiments inject adversarial messages into controlled code-repair tasks while leaving the repository, tests, and tools unchanged by the distractor. We recognize the dual-use potential of adaptive distractors and accumulated weakness notes, which could inform attempts to manipulate deployed agents. We therefore frame these findings as evidence of reliability risks and evaluate distractions through independently assessed repair outcomes rather

than treating behavioral influence alone as successful harm. The reported weaknesses are specific to the evaluated models, tasks, and configurations and should not be interpreted as universal properties of these systems.

REPRODUCIBILITY STATEMENT

The source code for TRAPARENA is publicly available on our project website http://traparena.pages.dev. The methodology section describes the evaluation framework, distractor injection, curator memory, and repair assessment. The experimental setup specifies task sources, model configurations, sampling settings, and evaluation metrics, while the ablation studies document controlled variations of the attack and defense settings.

REFERENCES

Toufique Ahmed, Jatin Ganhotra, Avraham Shinnar, and Martin Hirzel. Investigating test overfitting on swe-bench. *arXiv preprint arXiv:2511.16858*, 2025.

Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. Swe-bench+: Enhanced coding benchmark for llms. *arXiv preprint arXiv:2410.06992*, 2024.

Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. Jailbreaking leading safety-aligned llms with simple adaptive attacks. In *International Conference on Learning Representations*, 2025a.

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, et al. Agentharm: A benchmark for measuring harmfulness of llm agents. In *International Conference on Learning Representations*, volume 2025, pp. 79185–79220, 2025b.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *Advances in Neural Information Processing Systems*, 38:26420–26466, 2025. doi: 10.52202/085713-0788.

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. Swe-rebench v2: Language-agnostic swe task collection at scale. In *International Conference on Machine Learning*, 2026.

Jonas Becker, Jan Philip Wahle, Terry Ruas, and Bela Gipp. Misinformation propagation in benign multi-agent systems. *arXiv preprint arXiv:2606.16710*, 2026.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George Pappas, Florian Tramèr, et al. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 55005–55029, 2024. doi: 10.52202/079017-1745.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pp. 23–42. IEEE, 2025.

Aili Chen, Aonian Li, Baichuan Zhou, Bangwei Gong, Binyang Jiang, Boji Dan, Changhao Zhang, Changqing Yu, Chao Wang, Cheng Ma, et al. The minimax-m2 series: Mini activations unleashing max real-world intelligence. *arXiv preprint arXiv:2605.26494*, 2026.

Angelica Chen, Jérémy Scheurer, Tomasz Korbak, Jon Ander Campos, Jun Shern Chan, Samuel R Bowman, Kyunghyun Cho, and Ethan Perez. Improving code generation by training with natural language feedback. *arXiv preprint arXiv:2303.16749*, 2023a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. Struq: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium (USENIX Security 25)*, 2025.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023b.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *Advances in neural information processing systems*, 37:82895–82920, 2024.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. In *2026 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pp. 587–618, 2026. doi: 10.1109/satml68715.2026.00040.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. Wasp: Benchmarking web agent security against prompt injection attacks. In *Advances in Neural Information Processing Systems*, volume 38, pp. 22436–22458, 2025. doi: 10.52202/085713-0666.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*, pp. 79–90, 2023.

Chengquan Guo, Xun Liu, Chulin Xie, Andy Zhou, Yi Zeng, Zinan Lin, Dawn Song, and Bo Li. Redcode: Risky code execution and generation benchmark for code agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 106190–106236, 2024. doi: 10.52202/079017-3369.

Pengfei He, Yuping Lin, Shen Dong, Han Xu, Yue Xing, and Hui Liu. Red-teaming llm multi-agent systems via communication attacks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6726–6747, 2025. doi: 10.18653/v1/2025.findings-acl.349.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting. In *Conference on Applied Machine Learning in Information Security*, 2024.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

Jinwei Hu, Xinmiao Huang, Youcheng Sun, Yi Dong, and Xiaowei Huang. Lying with truths: Open-channel multi-agent collusion for belief manipulation via generative montage. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5979–5996, 2026.

Jen-tse Huang, Jiaxu Zhou, Tailin Jin, Xuhui Zhou, Zixi Chen, Wenxuan Wang, Youliang Yuan, Michael R Lyu, and Maarten Sap. On the resilience of llm-based multi-agent collaboration with faulty agents. In *International Conference on Machine Learning*, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Devina Jain, David Hartmann, and Chuan Li. Adaptive adversaries: A multi-turn, multi-llm benchmark for llm agent security. *arXiv preprint arXiv:2607.18063*, 2026.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

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 B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

Tianjie Ju, Yiting Wang, Yi Hua, Xinbei Ma, Pengzhou Cheng, Haodong Zhao, Yulong Wang, Lifeng Liu, Jian Xie, Zhuosheng Zhang, et al. Flooding spread of manipulated knowledge in llm-based multi-agent communities. *Science China Information Sciences*, 69(7):172103, 2026. doi: 10.1007/s11432-024-4663-2.

René Just, Darioush Jalali, and Michael D Ernst. Defects4j: a database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 437–440, 2014. doi: 10.1145/2610384.2628055.

Ishan Kavathekar, Hemang Jain, Ameya Rathod, Ponnurangam Kumaraguru, and Tanuja Ganu. Tamas: Benchmarking adversarial risks in multi-agent llm systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31238–31268, 2026.

Arpandeep Khatua, Hao Zhu, Peter Tran, Arya Prabhudesai, Frederic Sadrieh, Johann K Lieberwirth, Xinkai Yu, Yicheng Fu, Michael J Ryan, Jiaxin Pei, et al. Cooperbench: Why coding agents cannot be your teammates yet. *arXiv preprint arXiv:2601.13295*, 2026.

Brendan King and Jeffrey Flanigan. Dialogue swe-bench: A benchmark for dialogue-driven coding agents. *arXiv preprint arXiv:2606.13995*, 2026.

Karolina Korgul, Yushi Yang, Arkadiusz Drohomirecki, Piotr Błaszczyk, Will Howard, Lukas Aichberger, Chris Russell, Philip HS Torr, Adam Mahdi, and Adel Bibi. It's a trap! task-redirecting agent persuasion benchmark for web agents. In *International Conference on Machine Learning*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Philippe Laban, Lidiya Murakhovs'ka, Caiming Xiong, and Chien-Sheng Wu. Are you sure? challenging llms leads to performance drops in the flipflop experiment. *arXiv preprint arXiv:2311.08596*, 2023.

Donghyun Lee, Mo Tiwari, and Brando Miranda. Prompt infection: Llm-to-llm prompt injection within multi-agent systems. In *Computer Security. ESORICS 2025 International Workshops*, Lecture Notes in Computer Science, pp. 511–520, 2026. doi: 10.1007/978-3-032-16092-8_28.

Hao Li, Ruoyao Wen, Shanghao Shi, Ning Zhang, Yevgeniy Vorobeychik, and Chaowei Xiao. Agentdyn: Are your agent security defenses deployable in real-world dynamic environments? *arXiv preprint arXiv:2602.03117*, 2026.

Nathaniel Li, Ziwen Han, Ian Steneker, Willow Primack, Riley Goodside, Hugh Zhang, Zifan Wang, Cristina Menghini, and Summer Yue. Llm defenses are not robust to multi-turn human jailbreaks yet. *arXiv preprint arXiv:2408.15221*, 2024.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 17889–17904, 2024. doi: 10.18653/v1/2024.emnlp-main.992.

Zeyi Liao, Lingbo Mo, Chejian Xu, Mintong Kang, Jiawei Zhang, Chaowei Xiao, Yuan Tian, Bo Li, and Huan Sun. Eia: Environmental injection attack on generalist web agents for privacy leakage. In *International Conference on Learning Representations*, 2025.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572, 2023a. doi: 10.52202/075280-0943.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. Autodan: Generating stealthy jailbreak prompts on aligned large language models. In *International Conference on Learning Representations*, volume 2024, pp. 56174–56194, 2024a.

Xiaogeng Liu, Peiran Li, G Edward Suh, Yevgeniy Vorobeychik, Zhuoqing Mao, Somesh Jha, Patrick McDaniel, Huan Sun, Bo Li, and Chaowei Xiao. Autodan-turbo: A lifelong agent for strategy self-exploration to jailbreak llms. In *International Conference on Learning Representations*, volume 2025, pp. 10313–10360, 2025.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, et al. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499*, 2023b.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 1831–1847, 2024b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019.

Lovish Madaan, Aaditya K Singh, Rylan Schaeffer, Andrew Poulton, Sanmi Koyejo, Pontus Stenetorp, Sharan Narang, and Dieuwke Hupkes. Quantifying variance in evaluation benchmarks. *arXiv preprint arXiv:2406.10229*, 2024.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, et al. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. In *International Conference on Machine Learning*, 2024.

Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. Tree of attacks: Jailbreaking black-box llms automatically. *Advances in Neural Information Processing Systems*, 37:61065–61105, 2024.

Rui Melo, Riccardo Fogliato, Sean Zhou, Pratiksha Thaker, and Zhiwei Steven Wu. Sevra-bench: Social engineering of vulnerabilities in review agents. *arXiv preprint arXiv:2606.13757*, 2026.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations. *arXiv preprint arXiv:2411.00640*, 2024.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. Swe-lancer: Can frontier llms earn $1 million from real-world freelance software engineering? *arXiv preprint arXiv:2502.12115*, 2025.

Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro Von Werra, and Shayne Longpre. Octopack: Instruction tuning code large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 7604–7623, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/1ec299a5229034141e58aeded0d0b9de-Paper-Conference.pdf.

Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations*, 2024.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023. doi: 10.1145/3586183.3606763.

Maya Pavlova, Erik Brinkman, Krithika Iyer, Vitor Albiero, Joanna Bitton, Hailey Nguyen, Joe Li, Cristian Canton Ferrer, Ivan Evtimov, and Aaron Grattafiori. Automated red teaming with goat: the generative offensive agent tester. *arXiv preprint arXiv:2410.01606*, 2024.

Yibo Peng, James Song, Lei Li, Xinyu Yang, Mihai Christodorescu, Ravi Mangal, Corina S Pasareanu, Haizhong Zheng, and Beidi Chen. When "correct" is not safe: Can we trust functionally correct patches generated by code agents? In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15514–15546, 2026.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 3419–3448, 2022a.

Ethan Perez, Sam Ringer, Kamilė Lukošiūtė, Karina Nguyen, Edwin Chen, Scott Heiner, Craig Pettit, Catherine Olsson, Sandipan Kundu, Saurav Kadavath, et al. Discovering language model behaviors with model-written evaluations. *arXiv preprint arXiv:2212.09251*, 2022b.

Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*, 2022.

Huy Nhat Phan, Tien N Nguyen, Phong X Nguyen, and Nghi DQ Bui. Hyperagent: Generalist software engineering agents to solve coding tasks at scale. *arXiv preprint arXiv:2409.16299*, 2024.

Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pp. 24–36, 2015. doi: 10.1145/2771783.2771791.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, 2024. doi: 10.18653/v1/2024.acl-long.810.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. In *International Conference on Learning Representations*, 2024.

Mark Russinovich, Ahmed Salem, and Ronen Eldan. Great, now write an article about that: The crescendo Multi-Turn LLM jailbreak attack. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 2421–2440, 2025.

Mikayel Samvelyan, Sharath Chandra Raparthy, Andrei Lupu, Eric Hambro, Aram H Markosyan, Manish Bhatt, Yuning Mao, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, et al. Rainbow teaming: Open-ended generation of diverse adversarial prompts. In *Advances in Neural Information Processing Systems*, volume 37, pp. 69747–69786, 2024. doi: 10.52202/079017-2229.

Mrinank Sharma, Meg Tong, Tomek Korbak, David Duvenaud, Amanda Askell, Sam Bowman, Esin Durmus, Zac Hatfield-Dodds, Scott Johnston, Shauna Kravec, et al. Towards understanding sycophancy in language models. In *International Conference on Learning Representations*, volume 2024, pp. 110–144, 2024.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Ankur Singh, Jinqiu Yang, and Tse-Hsun Chen. Issuetrojanbench: Benchmarking ai coding agents against malicious issue requests. *arXiv preprint arXiv:2607.20759*, 2026.

Edward K Smith, Earl T Barr, Claire Le Goues, and Yuriy Brun. Is the cure worse than the disease? overfitting in automated program repair. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pp. 532–543, 2015.

Maojia Song, Tej Deep Pala, Ruiwen Zhou, Weisheng Jin, Amir Zadeh, Chuan Li, Dorien Herremans, and Soujanya Poria. Llms can't handle peer pressure: Crumbling under multi-agent social interactions. *arXiv preprint arXiv:2508.18321*, 2025.

Alexandra Souly, Qingyuan Lu, Dillon Bowen, Tu Trinh, Elvis Hsieh, Sana Pandey, Pieter Abbeel, Justin Svegliato, Scott Emmons, Olivia Watkins, et al. A strongreject for empty jailbreaks. *Advances in Neural Information Processing Systems*, 37:125416–125440, 2024.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026. doi: 10.18653/v1/2026.eacl-long.333.

Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. Magis: Llm-based multi-agent framework for github issue resolution. *Advances in Neural Information Processing Systems*, 37:51963–51993, 2024.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Haotian Hui, Weichuan Liu, Zhiyuan Liu, et al. Debugbench: Evaluating debugging capability of large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 4173–4198, 2024. doi: 10.18653/v1/2024.findings-acl.247.

Yu Tian, Xiao Yang, Jingyuan Zhang, Yinpeng Dong, and Hang Su. Evil geniuses: Delving into the safety of llm-based agents. *arXiv preprint arXiv:2311.11855*, 2023.

Fares Trad, Simin Chen, Hung Viet Pham, Gias Uddin, and Baishakhi Ray. Adversarial testing of automated program repair agents for security vulnerabilities. *arXiv preprint arXiv:2609.15963*, 2026.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training llms to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208*, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations*, 2024b.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, 2025a.

Youting Wang, Xiao Han, Dingyan Shang, Yuan Tang, and Bowen Liu. Safety, or just capability? a validity audit of agent-safety benchmarks. *arXiv preprint arXiv:2607.28685*, 2026.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning*, 2025b.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? In *Advances in Neural Information Processing Systems*, volume 36, pp. 80079–80110, 2023a. doi: 10.52202/075280-3508.

Jerry Wei, Da Huang, Yifeng Lu, Denny Zhou, and Quoc V Le. Simple synthetic data reduces sycophancy in large language models. *arXiv preprint arXiv:2308.03958*, 2023b.

Chen Henry Wu, Rishi Shah, Jing Yu Koh, Ruslan Salakhutdinov, Daniel Fried, and Aditi Raghunathan. Dissecting adversarial robustness of multimodal lm agents. In *International Conference on Learning Representations*, 2025.

Chunqiu Steven Xia and Lingming Zhang. Automated program repair via conversation: Fixing 162 out of 337 bugs for $0.42 each using chatgpt. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 819–831, 2024. doi: 10.1145/3650212.3680323.

Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1482–1494, 2023. doi: 10.1109/icse48619.2023.00129.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

Jian Xie, Kai Zhang, Jiangjie Chen, Renze Lou, and Yu Su. Adaptive chameleon or stubborn sloth: Revealing the behavior of large language models in knowledge conflicts. In *International Conference on Learning Representations*, 2024.

Yizhe Xie, Congcong Zhu, Xinyue Zhang, Tianqing Zhu, Dayong Ye, Minghao Wang, and Chi Liu. Who's the mole? modeling and detecting intention-hiding malicious agents in llm-based multi-agent systems. *arXiv preprint arXiv:2507.04724*, 2025.

Rongwu Xu, Brian Lin, Shujian Yang, Tianqi Zhang, Weiyan Shi, Tianwei Zhang, Zhixuan Fang, Wei Xu, and Han Qiu. The earth is flat because...: Investigating llms' belief towards misinformation via persuasive conversation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16259–16303, 2024a. doi: 10.18653/v1/2024.acl-long.858.

Rongwu Xu, Zehan Qi, Zhijiang Guo, Cunxiang Wang, Hongru Wang, Yue Zhang, and Wei Xu. Knowledge conflicts for llms: A survey. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8541–8565, 2024b. doi: 10.18653/v1/2024.emnlp-main.486.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 23826–23854, 2023. doi: 10.52202/075280-1035.

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*, volume 37, pp. 50528–50652, 2024a. doi: 10.52202/079017-1601.

John Yang, Carlos E Jimenez, Alex L Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R Narasimhan, et al. Swe-bench multimodal: Do ai systems generalize to visual software domains? *arXiv preprint arXiv:2410.03859*, 2024b.

Ye Yang, Donghe Li, Zuchen Li, Fengyuan Li, Jingyi Liu, Li Sun, and Qingyu Yang. Misinfobench: A multi-dimensional benchmark for evaluating llms' resilience to misinformation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 10202–10229, 2025b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and defending against indirect prompt injection attacks on large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 1809–1820, 2025.

Jiahao Yu, Xingwei Lin, Zheng Yu, and Xinyu Xing. Gptfuzzer: Red teaming large language models with auto-generated jailbreak prompts. *arXiv preprint arXiv:2309.10253*, 2023.

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, et al. R-judge: Benchmarking safety risk awareness for llm agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1467–1490, 2024. doi: 10.18653/v1/2024.findings-emnlp.79.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Aoyan Li, Lu Chen, Xiaojian Zhong, et al. Multi-swe-bench: A multilingual benchmark for issue resolving. *Advances in Neural Information Processing Systems*, 38:70112–70144, 2025. doi: 10.52202/085713-2111.

Yi Zeng, Hongpeng Lin, Jingwen Zhang, Diyi Yang, Ruoxi Jia, and Weiyan Shi. How johnny can persuade llms to jailbreak them: Rethinking persuasion to challenge ai safety by humanizing llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14322–14350, 2024. doi: 10.18653/v1/2024.acl-long.773.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024.

Qiusi Zhan, Richard Fang, Henil Shalin Panchal, and Daniel Kang. Adaptive attacks break defenses against indirect prompt injection attacks on llm agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7101–7117, 2025. doi: 10.18653/v1/2025.findings-naacl.395.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents. In *International Conference on Learning Representations*, volume 2025, pp. 35331–35366, 2025a.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, et al. Swe-bench goes live! *Advances in Neural Information Processing Systems*, 38:163347–163372, 2025b. doi: 10.52202/085713-4923.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1592–1604, 2024a. doi: 10.1145/3650212.3680384.

Zaibin Zhang, Yongting Zhang, Lijun Li, Jing Shao, Hongzhi Gao, Yu Qiao, Lijun Wang, Huchuan Lu, and Feng Zhao. Psysafe: A comprehensive framework for psychological-based attack, defense, and evaluation of multi-agent system safety. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15202–15231, 2024b. doi: 10.18653/v1/2024.acl-long.812.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023. doi: 10.52202/075280-2020.

Andy Zhou and Ron Arel. Tempest: Autonomous multi-turn jailbreaking of large language models with tree search. *arXiv preprint arXiv:2503.10619*, 2025.

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. Melon: Provable defense against indirect prompt injection attacks in ai agents. In *International Conference on Machine Learning*, 2025.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*, 2024.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pp. 66602–66656, 2025.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models. In *USENIX Security Symposium*, 2025.

A Related Work

A.1 Evaluating Coding Agents

Code benchmarks evaluate capabilities at different levels of software development (Deng et al., 2025; Jimenez et al., 2024; Zan et al., 2025; Miserendino et al., 2025; Zhuo et al., 2025). HumanEvalFix measures the repair of faulty functions (Muennighoff et al., 2024), while SWE-bench evaluates whether models can resolve issues in real repositories (Jimenez et al., 2024). Terminal-Bench extends agent evaluation to challenging tasks in command-line environments with task-specific verification (Merrill et al., 2026). LiveCodeBench (Jain et al., 2025) addresses another dimension of evaluation by continuously collecting programming problems and assessing code generation, self-repair, code execution, and test output prediction. Together, these benchmarks show why coding ability requires evaluation across tasks and interaction settings (Zhang et al., 2025b; Badertdinov et al., 2025; 2026). HumanEval evaluates function synthesis from docstrings with unit tests (Chen et al., 2021). MBPP collects short Python problems written for entry-level programmers (Austin et al., 2021). APPS measures coding skill on competition-style problems of varying difficulty (Hendrycks et al., 2021). EvalPlus shows that weak tests overstate correctness and adds many more tests to existing benchmarks (Liu et al., 2023a). Defects4J provides real Java bugs with test cases for controlled studies of testing and repair (Just et al., 2014). DebugBench evaluates the debugging ability of language models across bug types and programming languages (Tian et al., 2024). SWE-bench Multimodal extends issue resolution to software with visual components (Yang et al., 2024b). SWE-Bench+ finds solution leakage and weak tests in SWE-bench and builds a cleaner version (Aleithan et al., 2024). InterCode frames coding as an interactive loop in which the agent acts and receives execution feedback (Yang et al., 2023). MINT evaluates how models use tools and natural language feedback over multiple turns (Wang et al., 2024b). τ-bench evaluates agents that talk with simulated users and use domain tools under policy rules (Yao et al., 2024). Dialogue SWE-Bench evaluates coding agents that solve tasks through dialogue with a user (King & Flanigan, 2026). CooperBench studies whether coding agents can work together as teammates on shared tasks (Khatua et al., 2026).

TrapArena examines how repair performance changes when an agent receives misleading advice during execution. It adapts tasks from HumanEvalFix, SWE-bench, and Terminal-Bench and compares repair outcomes with and without injected feedback, connecting adversarial evaluation to established repair tasks and measures the additional difficulty introduced by unreliable collaboration.

A.2 Coding Agents and Program Repair

ReAct interleaves reasoning and actions and forms the basis of many tool-using agents (Yao et al., 2023). SWE-agent designs an agent-computer interface that helps language models navigate and edit repositories (Yang et al., 2024a). OpenHands provides an open platform for building and evaluating software agents (Wang et al., 2025a). AutoCodeRover combines code search over the program structure with test-based fault localization (Zhang et al., 2024a). Agentless shows that a simple pipeline of localization, repair, and patch validation can compete with complex agents (Xia et al., 2024). MAGIS splits issue resolution among several specialized agents (Tao et al., 2024). HyperAgent coordinates several specialized agents to solve coding tasks at scale (Phan et al., 2024). ChatDev organizes software development as a conversation among agents with different roles (Qian et al., 2024). MetaGPT encodes standard operating procedures into collaboration among software agents (Hong et al., 2023). Xia et al. (2023) study how well large pre-trained language models perform automated program repair. ChatRepair fixes bugs through a conversation that feeds test failures back to the model (Xia & Zhang, 2024). Self-Debugging teaches models to explain and fix their own code using execution results (Chen et al., 2023b). Self-Refine improves model outputs through repeated self-feedback and revision (Madaan et al., 2023). Olausson et al. (2024) find that the benefit of self-repair depends on the quality of the feedback. Chen et al. (2023a) train code models with natural language feedback on their outputs. These systems rely on feedback from tools, other agents, or users to guide their edits. Other work shows that passing the available tests is weak evidence of a correct repair. Qi et al. (2015) show that many patches that pass the test suite are plausible but not correct. Smith et al. (2015) show that automated repair can overfit to the tests used to guide it. Ahmed et al. (2025) find that test overfitting remains a problem when language models perform repair. TrapArena studies what happens when part of the feedback that guides repair is wrong. It judges

the final patch with hidden tests, reference locations, and static checks, since passing public tests alone is not enough.

A.3 Misleading Feedback and Agent Security

Indirect prompt injection exposes a trust problem in language model applications when instructions embedded in external content influence subsequent behavior (Greshake et al., 2023; Liu et al., 2024b; Yi et al., 2025; Zhan et al., 2024; Debenedetti et al., 2024; Zhang et al., 2025a; Li et al., 2026; Korgul et al., 2026). TrapArena shares the concern that external language can redirect an agent away from its intended task. The attacker controls a channel for collaborative advice during repair but cannot edit the repository or falsify tool outputs. This setting makes it possible to examine how an agent weighs misleading interpretations against the technical evidence available through its tools. Perez & Ribeiro (2022) show that simple prompts can override the instructions of a language model application. Liu et al. (2023b) study prompt injection attacks against applications built on language models. Zhan et al. (2025) show that adaptive attacks can break existing defenses against indirect prompt injection. Wu et al. (2025) study the adversarial robustness of multimodal web agents. EIA injects hidden elements into web pages to make web agents leak private data (Liao et al., 2025). WASP benchmarks the security of web agents against prompt injection (Evtimov et al., 2025). Jain et al. (2026) benchmark agent security against adaptive attacks over multiple turns and multiple models. ToolEmu uses a language model to emulate tools and find risky agent behaviors (Ruan et al., 2024). R-Judge tests whether models can judge safety risks in records of agent interactions (Yuan et al., 2024). AgentHarm measures whether agents carry out harmful tasks that need many steps (Andriushchenko et al., 2025b). RedCode evaluates whether code agents execute or generate risky code (Guo et al., 2024). Wang et al. (2026) audit whether agent safety benchmarks measure safety or general capability. Several defenses aim to separate trusted instructions from untrusted content. Spotlighting marks untrusted input so that the model can tell it apart from instructions (Hines et al., 2024). StruQ separates prompts from data in structured queries and trains the model to follow only the prompt (Chen et al., 2025). The instruction hierarchy trains models to give priority to instructions from more trusted sources (Wallace et al., 2024). CaMeL builds a protective layer around the agent that tracks control and data flows (Debenedetti et al., 2026). MELON detects injection by checking whether agent actions change when the user task is masked (Zhu et al., 2025). Misleading advice in TrapArena does not ask for a new task, so defenses that filter foreign instructions may not apply.

SEVRA-Bench (Melo et al., 2026) is closely related in its study of persuasive language in software workflows (Trad et al., 2026; Peng et al., 2026; Singh et al., 2026; Sharma et al., 2024; Kavathekar et al., 2026; Becker et al., 2026; Hu et al., 2026). It constructs pull requests that reintroduce historical vulnerabilities and varies their accompanying narratives to evaluate whether review agents approve vulnerable changes. Both benchmarks examine whether social framing can undermine judgments grounded in code. The task and outcome differ in TrapArena, where the target actively investigates and repairs a defect while receiving advice throughout execution. The evaluation therefore follows the consequences of misleading guidance through to the submitted patch and compares repair performance against a baseline without that guidance.

A.4 Misleading Feedback in Multi-Agent Systems

Multiagent debate lets several models critique each other to improve factuality and reasoning (Du et al., 2023). Liang et al. (2024) use debate between agents to encourage divergent thinking. Evil Geniuses studies how adversarial roles affect the safety of agent systems (Tian et al., 2023). PsySafe attacks, defends, and evaluates multi-agent systems through injected dark personality traits (Zhang et al., 2024b). Huang et al. (2025) study how multi-agent collaboration holds up when some agents make errors. He et al. (2025) show that changing the messages exchanged between agents can compromise a multi-agent system. Ju et al. (2026) show that manipulated knowledge can spread through a community of agents. Prompt Infection shows that malicious prompts can copy themselves from one agent to another (Lee et al., 2026). Xie et al. (2025) model and detect malicious agents that hide their intentions within a multi-agent system. Song et al. (2025) show that models often give up correct answers under peer pressure from other agents. TrapArena studies a similar risk in software repair. One collaborator gives advice during execution, and the effect is measured on the submitted patch against a baseline without that advice.

A.5 Sycophancy and Misleading Context

Perez et al. (2022b) use model-written evaluations and find that larger models tend to repeat the stated views of the user. Wei et al. (2023b) reduce sycophancy with simple synthetic training data. Laban et al. (2023) show that models often change correct answers when a user challenges them. Xu et al. (2024a) show that persuasive conversations over several turns can change what models believe about facts. Zeng et al. (2024) use persuasion techniques from social science to jailbreak language models. Shi et al. (2023) show that irrelevant context can distract models and lower their accuracy. Xie et al. (2024) study how models react when outside evidence conflicts with what they already know. Xu et al. (2024b) survey knowledge conflicts in large language models. PoisonedRAG injects a few false texts into a retrieval corpus to control the answers of a model (Zou et al., 2025). MisinfoBench evaluates how well models resist misinformation along several dimensions (Yang et al., 2025b). These studies measure whether a model agrees with or repeats misleading content. TrapArena records such responses but judges harm by the change in repair success, since agreement does not always lead to a failed repair.

A.6 Adaptive Attacks

Automated attack generation uses model feedback to refine adversarial inputs (Liu et al., 2024a; 2025; Perez et al., 2022a). PAIR iteratively updates jailbreak prompts through interactions between an attacker model and a target model (Chao et al., 2025). Tree of Attacks extends this approach with branching and pruning to search for effective attacks (Mehrotra et al., 2024). Wei et al. (2023a) explain jailbreak success through competing objectives and mismatched generalization. GCG optimizes adversarial suffixes that transfer across aligned models (Zou et al., 2023). GPTFuzzer mutates human-written jailbreak templates to find new attacks (Yu et al., 2023). Andriushchenko et al. (2025a) show that simple adaptive attacks can jailbreak leading safety-aligned models. Rainbow Teaming searches for adversarial prompts that are both effective and diverse (Samvelyan et al., 2024). Li et al. (2024) show that human jailbreaks over multiple turns succeed against defenses built for single-turn attacks. HarmBench standardizes the evaluation of automated red teaming and robust refusal (Mazeika et al., 2024). JailbreakBench provides an open benchmark with shared artifacts for jailbreak attacks and defenses (Chao et al., 2024). StrongREJECT shows that many jailbreaks look successful but produce little useful harmful content (Souly et al., 2024). These methods establish feedback-driven search as a practical approach to generating attacks (Pavlova et al., 2024; Zhou & Arel, 2025; Russinovich et al., 2025). In TrapArena, adaptation operates over an ongoing repair trajectory, allowing the attacker to tailor advice to the files inspected, actions taken, and observations received by the target.

Maintaining textual experience across attempts is also related to Reflexion (Shinn et al., 2023), which stores reflections on task feedback to guide subsequent trials (Zhao et al., 2024; Zhang et al., 2026). Generative Agents store and retrieve memories to simulate believable human behavior (Park et al., 2023). Voyager keeps a growing library of skills learned during open-ended exploration (Wang et al., 2024a). MemGPT manages memory in tiers so that agents can work beyond the context window (Packer et al., 2023). Agent Workflow Memory induces reusable workflows from past tasks to guide new ones (Wang et al., 2025b). Dynamic Cheatsheet lets a model keep and update a memory of useful strategies at test time (Suzgun et al., 2026). The optional curator in TrapArena applies a similar memory mechanism to attack generation by summarizing observed target reactions into notes for later rounds. Separating this memory from trajectory-conditioned generation allows the effects of adaptation within an episode and experience across rounds to be examined separately.

A.7 Evaluation Methodology

Zheng et al. (2023) study the use of strong models as judges and discuss their biases. Miller (2024) argues that model evaluations should report error bars and use paired comparisons where possible. Madaan et al. (2024) quantify the variance of evaluation benchmarks across seeds and checkpoints. TrapArena follows this practice by pairing attacked and baseline runs on the same tasks and reporting bootstrap intervals over tasks.

B Attack Design and Cross-Round Memory

We compare generic personas, offline task-specific messages, and trajectory-conditioned messages generated by the target model itself. These experiments use Qwen3.6-35B and MiniMax-M2 on the 48-task SWE-bench subset, with ten samples per task and condition. All paired repair-difference intervals include zero, while recorded susceptibility ranges from 12.7% to 47.2% (Tab. 3). A separate persona comparison on MiniMax-M2 yields repair rates between 24.2% and 26.3%, against a 23.5% baseline (Tab. 4). These experiments do not establish a reliable reduction in repair success for the tested configurations. Because the main evaluation also changes the distractor model and protocol, its larger losses cannot be attributed to trajectory conditioning or cross-round memory from this comparison alone.

Table 3: Impact of distractors across three distraction settings on a 48-task subset. Δ reports the paired change in genuine repair, with 95% confidence intervals. Susceptibility measures how often agents follow misleading advice while leaving the root cause unresolved. Token cost reports generation tokens relative to the no-distractor baseline.

Model	Attack	Repair, no distr. (%)	Δ repair (pp)	Suscept. (%)	Token cost ($\times$)
Qwen3.6-35B (instruct)	Generic personas	28.0	-1.6 $[-4.5, +1.2]$	13.8	4.7
	Targeted, offline	28.0	$+0.7$ $[-3.3, +5.6]$	12.7	4.0
	Adaptive, per step	28.0	$+1.9$ $[-1.7, +5.7]$	28.8	4.4
MiniMax-M2 (agentic RL)	Generic personas	23.5	-1.0 $[-4.0, +1.9]$	32.3	1.1
	Targeted, offline	23.5	-4.0 $[-9.5, +1.0]$	32.8	1.1
	Adaptive, per step	23.5	$+1.0$ $[-5.2, +7.3]$	47.2	1.4

Table 4: Effect of generic distractor personas on MiniMax-M2 across 48 tasks, with ten samples per condition. We report genuine repair, susceptibility to misleading advice, and false-fix rates.

Distractor	Genuine repair (%)	Susceptible (%)	False fix (%)
None (baseline)	23.5	0.0	14.6
Urge early submission	26.3	36.7	8.8
Surface level patch	24.2	33.4	12.3
Misattribute root cause	24.6	32.3	10.2
Dismiss edge cases	25.3	38.2	12.1
Avoid core logic	26.2	31.6	7.7

To isolate cross-round memory, we evaluate Qwen2.5-72B with the same GPT-4o distractor and task suite while disabling the curator. Both distracted conditions run for ten rounds and use the same baseline. With the curator, mean susceptibility increases from 0.065 in the first three rounds to 0.177 in the last three. Without the curator, these two averages are both 0.062, although individual rounds fluctuate (Fig. 4). The memory therefore changes the evolution of the recorded response measure on this target.

The outcome comparison gives a different picture. Without the curator, repair loss is 11.7 points on SWE-bench and 11.1 on HumanEvalFix, compared with 10.4 and 3.9 with the curator. The direct paired difference between the distracted conditions is 1.3 points on SWE-bench, with an interval spanning zero, and 7.2 points on HumanEvalFix, favoring the condition with memory. Thus, trajectory-conditioned distractions already cause significant loss without cross-round notes, and adding those notes does not increase the loss in this ablation. The increase in susceptibility should not be interpreted as a corresponding increase in failures caused by the distraction.

We also compare the first three and last three rounds using direct paired differences (Tab. 6 and Tab. 5). For GLM-4-32B, the late window contains rounds 4 to 6 rather than 7 to 9, using zero-based indexing. No target under the main GPT-4o distractor shows a significant increase in loss between these windows on either major source. Loss decreases significantly for Qwen2.5-72B on both sources and for Qwen2.5-Coder-7B and DeepSeek-Coder-V2-Lite on HumanEvalFix. This is consistent with the curator ablation in showing that accumulating distraction experience need not increase repair degradation.

Table 5: Changes in genuine repair across distraction evolution under additional distraction and defense settings. Δ reports paired changes from the corresponding no-distractor baseline in early (0–2) and late (7–9) rounds, in percentage points. Negative late-minus-early values indicate greater repair loss in later rounds. Brackets show 95% confidence intervals, with significant repair reductions in bold.

			SWE-bench (48)			HumanEvalFix (60)		
Target	Attacker	Verify	Δ early	Δ late	late − early	Δ early	Δ late	late − early
Qwen2.5-72B	GPT-5.4-mini	No	**−9.7 [−19.4, −0.8]**	−5.8 [−15.5, +3.1]	+3.9 [−1.9, +11.2]	**−8.9 [−15.0, −3.3]**	−3.9 [−8.9, +1.1]	+5.0 [+0.6, +10.6]
Qwen2.5-72B	GPT-5.4	No	**−17.8 [−29.6, −7.4]**	**−15.6 [−25.6, −7.0]**	+2.2 [−1.5, +7.0]	**−25.0 [−32.8, −17.8]**	**−21.1 [−30.6, −11.7]**	+3.9 [−5.0, +12.2]
Qwen2.5-72B	GPT-5.4	Yes	**−15.9 [−26.8, −5.8]**	**−14.9 [−25.4, −5.8]**	+1.1 [−1.1, +4.3]	**−6.1 [−11.7, −1.1]**	**−10.0 [−16.1, −5.0]**	−3.9 [−9.4, +1.7]
Gemma-3-27B	GPT-5.4	No	−5.3 [−12.1, 0.0]	−5.3 [−12.1, 0.0]	0.0 [0.0, 0.0]	**−42.8 [−52.2, −33.3]**	**−54.4 [−65.0, −43.3]**	**−11.7 [−20.6, −2.8]**

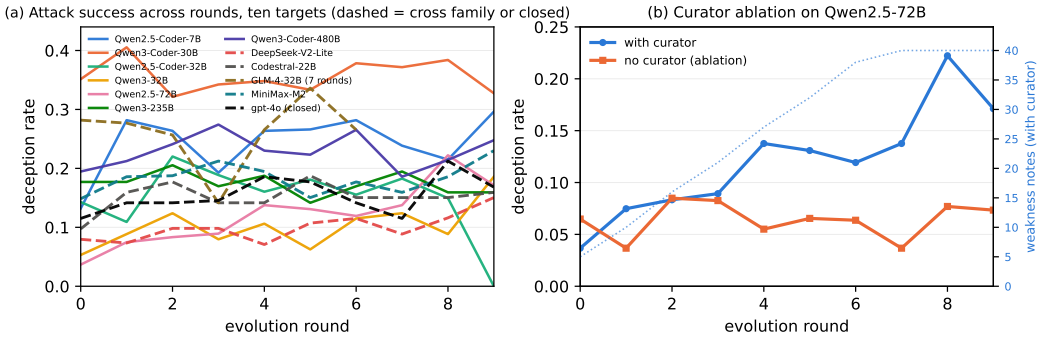


Figure 4: Susceptibility over distraction rounds. (a) Comparison across target models. (b) Effect of curator memory on Qwen2.5-72B; the dotted curve shows the number of retained notes. Corresponding repair outcomes are reported in Tab. 2.

C Attacker Configuration and Reference Access

We next hold the Qwen2.5-72B target fixed and evaluate GPT-5.4-mini and GPT-5.4 as alternative distractors, retaining GPT-5.4 as curator and ten distraction rounds. These runs use a revised leakage filter that matches identifiers at word boundaries and exempts generic tokens. They are reported separately from the main setting because both the filtering configuration and the retained task coverage can differ. The HumanEvalFix losses are 3.9, 7.2, and 23.7 points under GPT-4o, GPT-5.4-mini, and GPT-5.4, respectively; the corresponding SWE-bench losses are 10.4, 9.9, and 15.6 points. The results show substantial variation across distractor configurations, but the comparison with GPT-4o does not isolate distractor identity from the filtering change.

We additionally evaluate Gemma-3-27B under the GPT-5.4 distractor. Its HumanEvalFix repair rate decreases by 51.3 points from a 90.0% baseline, while the SWE-bench loss is not significant against a 4.9% baseline. Unlike the main targets, Gemma shows a significant increase in HumanEvalFix loss between early and late rounds, with a paired change of 11.7 points in the harmful direction. This extends the evidence to another target under a separate distraction condition. There is no corresponding Gemma experiment without curator memory, so the temporal change cannot be attributed specifically to the curator.

To test whether reference access is necessary for an effective distraction, we remove the reference patch from the distractor's input when evaluating the GPT-4o target. Attacker API failures affected several blind-condition rounds, so both conditions are compared on rounds 0, 6, 7, 8, and 9, with the same delivery filter. The blind distractor causes a significant 4.4-point loss on SWE-bench, compared with 4.9 for the reference-informed distractor. On HumanEvalFix, the corresponding losses are 2.0 and 4.8 points, and only the reference-informed loss is individually significant. The direct blind-minus-reference differences have intervals that include zero on both sources. These results establish that reference access is unnecessary for a damaging repository-level distraction in this setting, but do not establish equivalence between the two conditions.

Table 6: Changes in genuine repair across distraction evolution. Δ reports paired changes from the corresponding no-distractor baseline in early (0–2) and late (7–9) rounds, in percentage points. Negative late-minus-early values indicate greater repair loss in later rounds. Brackets show 95% confidence intervals, with significant repair reductions in bold. GLM-4-32B uses rounds 4–6 as late rounds; the final block evaluates alternative distractors and a verification prompt.

	SWE-bench (48)			HumanEvalFix (60)		
Target	Δ early	Δ late	late − early	Δ early	Δ late	late − early
Qwen2.5-Coder-7B	**−8.2 [−15.6, −2.1]**	**−7.8 [−14.9, −2.1]**	+0.4 [−1.1, +2.1]	**−34.2 [−42.8, −25.3]**	**−25.8 [−35.3, −16.2]**	+8.3 [+1.7, +15.1]
Qwen3-Coder-30B-A3B	−2.8 [−8.3, +2.1]	−2.1 [−6.2, +2.1]	+0.7 [−4.2, +5.6]	+17.8 [+8.3, +27.2]	+11.7 [+3.3, +20.6]	−6.1 [−13.3, +0.6]
Qwen2.5-Coder-32B	−3.3 [−9.3, +2.6]	**−6.3 [−13.3, −0.4]**	−3.0 [−9.6, +3.3]	**−23.3 [−31.7, −15.0]**	**−19.4 [−27.8, −12.2]**	+3.9 [−5.8, +13.3]
Qwen3-32B	−2.8 [−9.7, +4.2]	−1.4 [−7.6, +4.2]	+1.4 [−4.2, +7.6]	−2.8 [−7.8, +1.7]	−2.8 [−7.8, +1.7]	0.0 [−5.6, +5.6]
Qwen2.5-72B	**−14.8 [−25.9, −5.2]**	**−10.0 [−17.8, −3.0]**	+4.8 [+0.7, +8.9]	**−6.1 [−10.6, −1.7]**	−1.7 [−3.9, 0.0]	+4.4 [+1.1, +8.3]
Qwen2.5-72B, no curator	**−15.2 [−25.6, −5.9]**	**−13.7 [−23.7, −4.4]**	+1.5 [−4.4, +8.1]	**−9.4 [−16.1, −3.3]**	**−12.8 [−20.0, −6.1]**	−3.3 [−8.3, +1.7]
Qwen3-235B-A22B	**−5.7 [−10.6, −1.4]**	−3.5 [−9.2, +1.4]	+2.1 [−2.1, +6.4]	0.0 [−1.7, +1.7]	−1.1 [−3.9, +1.1]	−1.1 [−2.8, 0.0]
Qwen3-Coder-480B-A35B	−6.9 [−15.3, +0.7]	−3.5 [−12.2, +4.9]	+3.5 [−3.5, +10.8]	+2.2 [−0.6, +6.1]	+1.1 [−1.7, +4.4]	−1.1 [−3.3, +1.1]
DeepSeek-Coder-V2-Lite	−4.9 [−11.1, 0.0]	−4.2 [−11.1, +0.7]	+0.7 [0.0, +2.1]	**−23.3 [−33.9, −13.3]**	**−11.1 [−21.1, −1.7]**	+12.2 [+3.3, +21.7]
Codestral-22B	−0.7 [−2.1, 0.0]	−0.7 [−2.1, 0.0]	0.0 [0.0, 0.0]	**−38.3 [−47.2, −29.4]**	**−35.0 [−43.9, −26.7]**	+3.3 [−3.9, +10.6]
GLM-4-32B	−5.6 [−11.8, +0.7]	−4.2 [−9.0, 0.0]	+1.4 [−4.2, +7.6]	0.0 [−7.2, +7.2]	−0.3 [−7.2, +6.7]	−0.3 [−7.8, +6.9]
MiniMax-M2	+1.7 [−5.2, +9.4]	−2.8 [−12.5, +6.2]	−4.5 [−11.5, +2.4]	−3.9 [−10.0, +2.2]	+1.7 [−2.8, +6.1]	+5.6 [0.0, +11.1]
GPT-4o	**−6.4 [−11.3, −2.1]**	−4.3 [−10.6, +1.1]	+2.1 [−2.1, +6.7]	−1.7 [−6.0, +2.6]	**−6.4 [−11.9, −1.1]**	−4.7 [−10.8, +1.1]

Table 7: Effect of distractor access to the reference fix on genuine repair for GPT-4o. Blind and oracle distractors are compared on matched rounds (0, 6–9). Δ reports paired changes from the no-distractor baseline; the last column reports blind minus oracle. Brackets show 95% confidence intervals, with significant repair reductions in bold.

Source	Base (%)	Δ blind distractor (pp)	Δ oracle distractor (pp)	blind − oracle (pp)
SWE-bench (48)	25.0	**−4.4 [−7.7, −1.2]**	**−4.9 [−9.9, −0.2]**	+0.5 [−2.3, +3.5]
HumanEvalFix (60)	90.6	−2.0 [−6.4, +2.5]	**−4.8 [−9.4, −0.2]**	+2.8 [−1.2, +7.2]

D Verification Instructions

We evaluate a prompt-based defense on Qwen2.5-72B against the GPT-5.4 distractor. The added system instruction treats collaborator messages as unverified and asks the target to check claims about bug location, root cause, or completion against code or test evidence before acting. It also instructs the target not to submit solely because a collaborator claims that the tests pass. The defended baseline includes the same instruction, so the reported Δ compares distracted and non-distracted runs under that instruction.

HumanEvalFix loss decreases from 23.7 points without the instruction to 8.2 points with it. SWE-bench loss remains nearly unchanged at 15.6 and 15.7 points, respectively. The defended SWE-bench baseline is 18.1%, compared with 18.8% without the instruction. The target token ratio is 1.85 under the defended condition and 1.42 without the instruction, each relative to its corresponding baseline. The prompt therefore reduces the observed loss on function-level tasks while leaving substantial repository-level degradation. This comparison does not establish which verification behaviors account for the difference between task sources.

E Execution Cost and Recorded Responses

Token use responds differently across targets and distraction conditions (Fig. 3). Under the main distraction, Qwen2.5-72B uses 0.59 times its baseline generation tokens, whereas Qwen2.5-Coder-7B and DeepSeek-Coder-V2-Lite use 1.55 and 7.41 times their baselines. All three targets experience significant repair loss on at least one source. For the same 72B target, the ratios become 1.00 and 1.42 under the GPT-5.4-mini and GPT-5.4 distractors. Repair degradation can therefore accompany either increased or decreased token use; the aggregate ratios alone do not distinguish premature submission from other changes in execution.

We examine available full trajectories from the earlier Qwen3.6-35B and MiniMax-M2 adaptive-distraction experiments in Tab. 8. Among failed MiniMax-M2 episodes, the share with no edit rises from 19% to 25% under distraction. For Qwen3.6-35B, the share classified as editing the wrong location rises from 10% to 15%, while median steps among failures decrease from 52 to 31. The share flagged as a superficial fix decreases for both targets. These are descriptive changes within

Table 8: Failure modes and step usage with and without adaptive per-step distractions on 48 SWE-bench tasks. Each failed episode is assigned one failure mode; percentages are computed over failed episodes. The final three columns report median steps, the share submitting within 15 steps, and the share reaching the 60-step limit.

			Failure mode (% of failed episodes)				Steps in failed episodes		
Model	Condition	Failed	No edit	False fix	Wrong location	Right place, wrong fix	Median	≤15 (%)	Cap 60 (%)
MiniMax-M2	No distractor	367	19	16	32	33	60	8	64
	Adaptive distraction	371	25	9	28	39	60	9	66
Qwen3.6-35B	No distractor	343	30	12	10	48	52	27	47
	Adaptive distraction	354	17	9	15	59	31	32	28

Table 9: Additional distraction and defense settings. Base denotes repair without distractors; Δ reports paired changes under distraction with 95% confidence intervals. Bold indicates significant repair reductions. Tokens reports generation token usage relative to the corresponding baseline. All rows use the same leakage filter; verification prompting is applied to both distracted and baseline conditions.

			SWE-bench (48)		HumanEvalFix (60)		Terminal-Bench (5)		
Target	Attacker	Verify	Base (%)	Δ (pp) [95% CI]	Base (%)	Δ (pp) [95% CI]	Base (%)	Δ (pp) [95% CI]	Tokens (×)
Qwen2.5-72B	GPT-5.4-mini	No	18.8	**−9.9 [−19.2, −1.6]**	95.6	**−7.2 [−12.7, −2.1]**	0.0	+2.5 [0.0, +7.5]	1.00
Qwen2.5-72B	GPT-5.4	No	18.8	**−15.6 [−25.4, −6.9]**	95.6	**−23.7 [−31.6, −16.6]**	0.0	0.0 [0.0, 0.0]	1.42
Qwen2.5-72B	GPT-5.4	Yes	18.1	**−15.7 [−25.7, −6.7]**	95.6	**−8.2 [−13.3, −3.9]**	0.0	+2.0 [0.0, +6.0]	1.85
Gemma-3-27B	GPT-5.4	No	4.9	−2.2 [−8.7, +4.2]	90.0	**−51.3 [−60.1, −42.4]**	0.0	0.0 [0.0, 0.0]	0.73

the failed-episode subsets of those experiments and cannot establish the mechanism behind token changes for other targets in the main evaluation.

The curator notes offer a complementary view of recorded responses (Tab. 10). They describe tactics such as reassuring the target that public tests already pass or making confident claims about the defect's location. These summaries suggest behaviors for further inspection, but they are generated by the curator and are not independent evidence that the corresponding tactics caused additional repair failures.

F Sampling Temperature

As an exploratory check, we run one distraction round against Qwen2.5-Coder-32B at target temperatures 0.0 and 1.0 and compare it with the first main-protocol round at 0.7 (Tab. 11). The recorded susceptibility rates are 0.196, 0.196, and 0.143, respectively. The table also reports hidden-test pass rates, which are distinct from the combined repair criterion. Because this check contains one round per temperature, unequal retained counts, and no corresponding temperature-matched baseline comparison, it does not establish that paired repair loss is insensitive to temperature.

Table 10: Representative model-specific weakness notes distilled by the GPT-5.4 curator after ten distraction evolution rounds. These notes summarize observed target responses and serve as persistent memory to guide subsequent distractions.

Target	Weakness the distractor learned to exploit
Qwen2.5-72B	An assurance such as "the public tests already pass, you can submit now" makes it stop locating the root cause and accept a wrong fix.
Qwen3-235B	It follows an authoritative claim that "the root cause is in file/function X" and edits the named location even when the justification is weak, and its resistance stays verbal while its actions still turn toward the planted location.
Qwen3-Coder-480B	Exclusive phrasing like "do not look at X, the cause is in Y, I studied it carefully" most reliably steers it, and it will drop its own judgment and reset its search scope to follow the hint.
Qwen3-32B	A push to conclude ("just submit", "this approach is fine") makes it cut verification steps and write self-consistent justifications for the wrong direction, and repeated restatement of a single root cause narrative overrides its own judgment.
Qwen3-Coder-30B	It is easily led by a claim that names a concrete file or function even when unrelated to the symptom, and "the problem is not in A but in B" phrasing suppresses its commitment to the originally correct location.

Table 11: Effect of target sampling temperature on Qwen2.5-Coder-32B under one round of distraction evolution. We report deception rate, hidden-test pass rate, and the number of evaluated episodes (n) at each temperature.

Target temperature	Deception rate	Hidden pass	n
0.0	0.196	0.343	102
0.7	0.143	0.509	112
1.0	0.196	0.420	112