VERIFYING THE VERIFIERS: FAILURE ATTRIBUTION FOR AGENTIC BENCHMARK DIAGNOSTICS AND TRAINING DATA CURATION

Anonymous Authors

ABSTRACT

When coding agents fail benchmark tasks, the failure is opaque: benchmarks report only that the agent failed, not *why*. This matters for two critical use cases. For **evaluation**, practitioners need to know whether failures reflect agent limitations or task defects—ambiguous specifications, flaky tests, broken environments—to diagnose and improve their systems. For **self-evolving agents**, failures serve as reward signal for RL training, but training on task defects as if they were agent errors introduces noise that corrupts learned policies and prevents productive adaptation. We formalize this problem with a failure attribution taxonomy and validate it through a human annotation study on a software engineering benchmark, establishing high inter-annotator agreement ($\kappa = 0.929$). We then present AutoTriage, a system that automates failure attribution by deploying an agentic judge with sandboxed environment access to investigate trajectories—executing code, running tests, and analyzing error logs. We evaluate nine configurations across three models and three access modes (text-only, read-only agent, full sandbox). The best configuration—sandbox execution with GPT-5.2 Codex—achieves near-human agreement ($\kappa = 0.833$), though the benefit of execution access is model-dependent (§5). Error analysis reveals that weaker triage models exhibit a systematic directional bias: they over-attribute agent failures to task defects, constructing plausible defenses of the agent rather than identifying root causes—a failure mode with direct implications for any agent that uses its own critic to drive lifelong self-improvement.

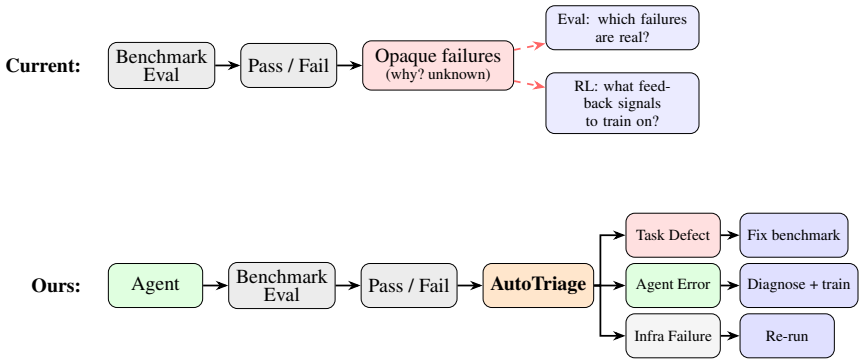
1 INTRODUCTION

LLM agent benchmarks like SWE-bench (Jimenez et al., 2024), HumanEval (Chen et al., 2021), and MLE-bench (Chan et al., 2025) drive progress by providing standardized evaluation. These benchmarks serve two increasingly important roles: as **diagnostic tools** for understanding agent capabilities, and as **training signal** for self-evolving agents that improve through reinforcement learning, where pass/fail outcomes become rewards (Le et al., 2022; Shojaee et al., 2023).

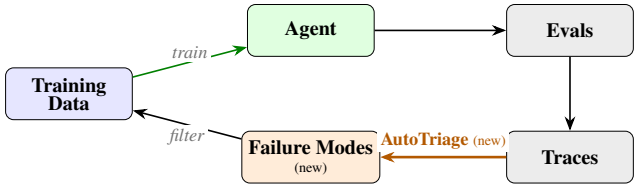
Both roles require understanding *why* an agent failed, not just *that* it failed. When an agent fails a task, the failure could indicate:

- **Agent error**: The agent made a genuine mistake—a bug, misunderstanding, or capability gap that training should correct and that practitioners should diagnose.
- **Task defect**: The task specification was ambiguous, the tests were flaky, or the environment was broken—issues no agent could overcome.
- **Infrastructure failure**: The sandbox crashed, the network timed out, or the evaluation harness malfunctioned—issues orthogonal to both agent and task quality that should trigger a re-run.

This three-way distinction requires jointly reasoning about the task specification, the agent's trajectory, and the execution environment. A test failure might reflect a genuine bug, an overly strict assertion, or a transient environment issue—and the evidence for each can look similar in raw logs. Without this distinction, both use cases break down. For evaluation, practitioners cannot tell whether



(a) AutoTriage replaces opaque pass/fail with actionable failure categories.



(b) The self-evolving agent cycle. AutoTriage closes the loop by automating the previously missing step: attributing failures before traces become training signal.

Figure 1: **(a)** Current benchmarks report only pass/fail; AutoTriage attributes each failure to a root cause—task defects are fixed, agent errors become training signal, infrastructure failures are re-run. **(b)** In the self-evolving agent loop, failure attribution is the critical step between traces and training data. Without it, the agent trains on corrupted signal (§5.4).

to fix the agent or fix the task. For RL training, task defects become negative reward signal, penalizing agents for "failures" that reflect benchmark bugs.

The problem is especially acute for self-evolving agents (Gao et al., 2025)—systems that improve through iterative cycles of evaluation, diagnosis, and self-modification (Schmidhuber, 2003; Wang et al., 2025a). A lifelong agent that continuously adapts must distinguish "I made a mistake" from "the task was broken." Without reliable failure attribution, the self-improvement loop trains on corrupted signal—the agent optimizes against phantom failures rather than genuine capability gaps, discarding valid training data while retaining noise. The quality of the critic is thus a bottleneck for agent evolution, and our error analysis (§5.4) reveals that this concern is not hypothetical: weaker models systematically misattribute agent failures to task defects, exactly the bias that would prevent productive self-modification.

Contributions. We address this gap with four contributions:

1. A **taxonomy** of failure modes distinguishing task defects, agent errors, and infrastructure failures, validated on software engineering benchmarks (§3)

2. A **human validation study** on 118 trajectories establishing that failure attribution is reliable ($\kappa = 0.929$) (§4)

3. **AutoTriage**, an agentic judge for automated failure attribution. We evaluate nine configurations across three frontier models and three access modes, finding that sandbox execution with GPT-5.2 Codex achieves $\kappa = 0.833$, though the benefit of execution access is model-dependent (§5)

4. An **error analysis** revealing that weaker triage models systematically over-attribute agent failures to task defects—a directional bias with implications for self-evolving agents (§5.4)

2 Related Work

Agent Failure Analysis. MAST (Cemri et al., 2025) proposes a taxonomy for multi-agent system failures, focusing on coordination breakdowns. Who&When (Zhang et al., 2025b) formalizes failure attribution in multi-agent systems, achieving 53.5% accuracy in identifying responsible agents but only 14.2% in pinpointing failure steps. DoVer (Ma et al., 2025) advances beyond log-based attribution using intervention-driven debugging through replay and repair. Both operate on multi-agent coordination failures; our work addresses a complementary setting—single-agent SWE task failures—where ground-truth patches enable oracle-grounded validation. Our taxonomy is specifically designed for training data curation, where the task-vs-agent distinction determines signal quality.

Coding Agent Evaluation. SWE-bench Verified (Chowdhury et al., 2024) manually filtered 500 high-quality instances from SWE-bench, demonstrating that benchmark curation matters but not providing a systematic framework for ongoing quality assessment. SWE-Bench+ (Aleithan et al., 2024) revealed that 32.7% of successful patches involved solution leakage and 31% passed due to weak tests—empirically confirming our taxonomy's task defect categories. Wang et al. (2025b) further showed that 7.8% of "solved" SWE-bench Verified patches are actually incorrect. Zhu et al. (2025) analyzed where LLM agents fail and proposed methods for learning from failures, complementing our focus on *attributing* failures to their root causes. Our work complements these efforts by providing automated, ongoing quality assessment rather than one-time manual filtering.

Data Quality and Reward Signal Integrity. Our work connects to two broader threads. First, the data-centric AI paradigm (Zha et al., 2023) emphasizes that data quality—not just model architecture—drives performance; failure attribution implements this principle for agent training data by curating reward signals rather than blindly consuming them. Second, research on reward hacking (Skalse et al., 2022) shows that agents exploit misspecified reward signals to inflate performance without genuine capability gains. Our error analysis reveals a complementary failure mode: when the *critic* rather than the *agent* corrupts the reward signal, misattributing genuine failures to task defects. This critic-side reward corruption is especially pernicious because it is invisible to standard evaluation metrics—the agent's benchmark score appears stable while training signal quality degrades.

LLM-as-Judge and Self-Evolving Agents. Using LLMs as evaluators is standard practice (Zheng et al., 2023). The Agent-as-a-Judge framework (Zhuge et al., 2024) extends this with agentic capabilities but focuses on response quality rather than failure attribution. More broadly, self-evolving agents (Gao et al., 2025) that improve through evolutionary search (Novikov et al., 2025), hill-climbing over self-modifications (Wang et al., 2025a), or open-ended skill accumulation (Wang et al., 2023) depend on the quality of their evaluation signal. The theoretical framework of Gödel machines (Schmidhuber, 2003) establishes that self-improving agents require provably useful self-modifications; in practice, this means the critic must reliably distinguish genuine failures from benchmark artifacts. AutoTriage provides this diagnostic capability, and our error analysis (§5.4) reveals that critic quality is not a given—weaker models systematically misattribute failures in ways that would corrupt self-improvement across the agent's lifespan.

3 Failure Attribution Taxonomy

We propose a three-category taxonomy for failure attribution, developed through iterative analysis of 200+ agent trajectories across software engineering benchmarks.

Task's Fault. The benchmark itself is broken: ambiguous specifications, flaky tests, environment misconfigurations, or solution leakage. These corrupt the reward signal—penalizing agents for unsolvable tasks (false negatives) or rewarding non-solutions that pass weak tests (false positives).

Agent's Fault. The agent made a genuine mistake: logical errors, tool misuse, knowledge gaps, or premature termination. These represent valid negative signal that should be retained for training and diagnosed for improvement.

Infrastructure Failure. The evaluation harness itself malfunctioned: sandbox crashes, network timeouts, resource exhaustion, or harness bugs. These are orthogonal to the task-vs-agent distinction and should trigger a re-run.

Table 1 maps these categories to their impact on reward signal quality. We frame the taxonomy as a signal corruption problem: benchmark outcomes serve as rewards ($r = +1$ for pass, $r = -1$ for fail), and each failure category implies a different intervention.

Table 1: Taxonomy of reward signal corruption. Shaded cells indicate corrupted signal that should be fixed or filtered before use in training. Human annotation (§4) and AutoTriage (§5) classify at the **top level** (Task's Fault vs. Agent's Fault vs. Infrastructure); subcategories provide finer-grained diagnostics. †False positive modes are documented by prior work (Aleithan et al., 2024; Wang et al., 2025b); our annotation focuses on the left column (observed failures).

	Observed Failure ($r = -1$)	**Observed Success** ($r = +1$)
Task's Fault	**False Negative Reward** Agent penalized for task defect. *Ambiguous spec*: overspecified or misleading instructions *Environment fail*: broken deps, missing tools *Brittle tests*: overly strict or flaky assertions **Action:** Fix task or filter from training	**False Positive Reward**† Agent rewarded for non-solution. *Weak tests*: underspecified tests miss errors *Solution leakage*: agent accesses gold answer or test content **Action:** Fix tests or filter from training
Agent's Fault	**True Negative** (keep for training) *Logical failure*: wrong approach, implementation bugs *Tool call error*: incorrect commands or tool misuse *Knowledge gap*: domain misunderstanding *Premature stop*: agent quit early *Timeout*: exceeded time budget **Action:** Improve agent; train on signal	**True Positive** (no attribution needed) Agent solved the task correctly. **Action:** Use as positive signal
Infra Failure	**Corrupted Signal** Failure unrelated to task or agent. *Sandbox crash*: container OOM, disk full *Network issue*: dependency download failure *Harness bug*: evaluation script error **Action:** Re-run evaluation	**Unreliable Signal** Success may be artifact of harness state. **Action:** Re-run to confirm

AutoTriage focuses on the left column (observed failures), where the task-vs-agent distinction determines whether the $r = -1$ signal is valid. The taxonomy also covers the right column: the false positive reward problem identified by prior work showing that 7.8% of "solved" SWE-bench patches are actually incorrect (Wang et al., 2025b) and 31% pass due to weak tests (Aleithan et al., 2024).

4 Human Annotation Study

We conducted a human annotation study to validate that failure attribution is a well-defined task humans can perform reliably.

4.1 Setup

Three expert annotators with experience in LLM agent evaluation independently labeled 118 trajectories from Terminal-Bench, a command-line software engineering benchmark requiring agents to complete terminal-based tasks involving package building, system configuration, and scripting challenges. Terminal-Bench tasks are long-horizon: typical tasks require 20–50 tool calls, spanning dependency resolution, compilation, configuration editing, and test execution. Each task includes a gold solution, a test suite, and a detailed specification. Unlike benchmarks that test isolated code generation, Terminal-Bench evaluates the full agent loop—planning, execution, error recovery, and verification—making it representative of the challenges self-evolving agents face during continuous deployment.

Annotators first jointly labeled a small calibration set to align on taxonomy definitions, then proceeded independently. Each annotator had access to the full trajectory (agent actions and outputs),

task specification, test files, gold solution (when available), and error logs. Annotators assigned a top-level label (task defect, agent error, infrastructure, or valid pass) and subcategory corresponding to the left column of Table 1.

4.2 Inter-Annotator Agreement

Table 2 shows inter-annotator agreement measured by Fleiss' κ (Fleiss, 1971) (chance-corrected multi-rater agreement) and pairwise Cohen's κ (Cohen, 1960).

Table 2: Inter-annotator agreement on Terminal-Bench ($N = 118$). Fleiss' κ establishes "almost perfect" multi-rater agreement. All pairwise Cohen's κ values exceed 0.89.

Metric	κ	**Raw Agreement**
Fleiss' κ (3-way)	0.929	97.7%
A1 vs. A2	0.973	99.2%
A1 vs. A3	0.893	96.6%
A2 vs. A3	0.922	97.5%

Fleiss' $\kappa = 0.929$ demonstrates that failure attribution is not subjective—trained annotators converge on the same labels. This provides both a ceiling for automated systems and validates the taxonomy's discriminative power.

4.3 Disagreement Analysis

The small number of disagreements ($< 3\%$) cluster at the boundary between Agent Error and Task Defect, where an agent made errors that a clearer specification might have prevented. For example, annotators disagreed on cases where a task required a specific library version that was not explicitly stated in the specification: one annotator labeled the agent's use of the wrong version as an agent error (the agent should have checked), while another labeled it as a task defect (the specification should have been explicit). Infrastructure failures are never confused with other categories, confirming that this is a clear-cut failure mode. The asymmetry of disagreements is notable: annotators who disagree tend toward labeling as Task Defect rather than Agent Error, suggesting that the boundary favors giving the agent "benefit of the doubt"—the same directional pattern we observe (at much larger magnitude) in automated systems (§5.4).

5 AutoTriage: Automated Failure Attribution

Human annotation establishes that failure attribution is reliable, but it does not scale: each trajectory requires 15–30 minutes of expert review involving code comprehension, test analysis, and environment reasoning. For benchmarks with hundreds or thousands of tasks evaluated across multiple agents and model versions, manual triage is impractical. We introduce AutoTriage, an automated system that performs failure attribution by deploying an LLM judge with varying levels of environment access.

5.1 Method

AutoTriage operates in three access modes:

LLM-only. The model receives the agent trajectory, task specification, and test logs as text and produces a classification. This serves as a baseline—equivalent to prior LLM-as-judge approaches (Zheng et al., 2023).

Agent (read-only). The model is deployed as an agent with bash tools (`grep`, `diff`, `cat`, `find`) and read-only filesystem access. It can navigate directories, search through logs, diff outputs against gold solutions, and inspect test implementations—but cannot modify any files.

Agent-sandbox (full access). The model operates in a fully executable sandboxed environment—a replica of the original evaluation environment. Beyond reading files, the judge can execute code,

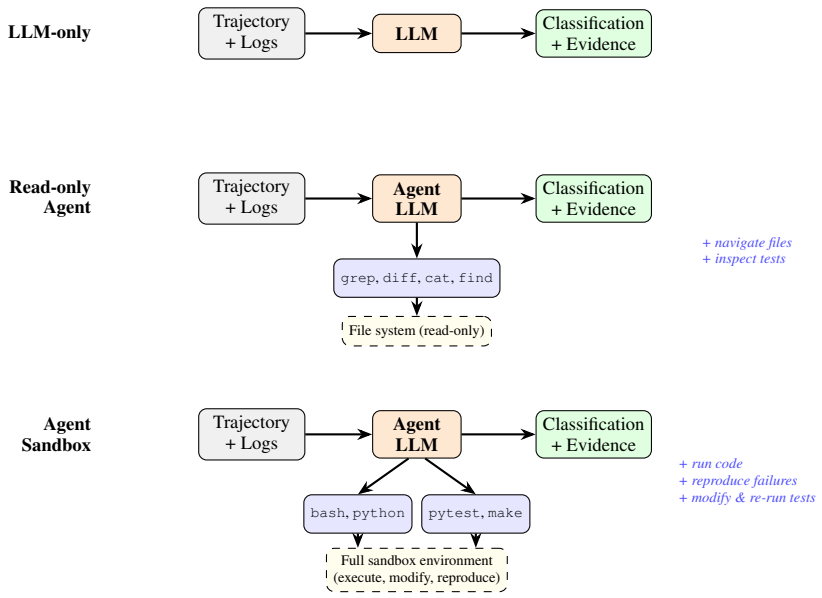


Figure 2: AutoTriage access modes. **LLM-only**: text-based classification (baseline). **Read-only agent**: navigates the file system to inspect task specifications, test implementations, and gold solutions. **Agent-sandbox**: executes code, runs tests, and reproduces failures in a replica of the original evaluation environment. Each mode adds capability (blue annotations); Table 3 evaluates the impact.

run and modify tests, reproduce the agent's steps, and inspect runtime behavior. This is the key differentiator: the judge can verify whether a test failure reflects a genuine correctness issue or a brittle assertion by actually running the test suite.

In all modes, AutoTriage receives the agent's complete trajectory, task specification, gold solution (when available), and test files. The judge is prompted with a structured system prompt defining the taxonomy categories, classification criteria, and requiring chain-of-thought analysis before outputting a classification. In agent modes, the prompt additionally instructs the model to actively investigate—diffing outputs against gold solutions, checking test assertions, and (in sandbox mode) reproducing failures—before reaching a verdict. All configurations use zero-shot prompting with taxonomy definitions. We evaluate each mode across three frontier models: GPT-5.2 Codex, Claude Sonnet 4.5, and Claude Haiku 4.5.

5.2 Evaluation Protocol

We evaluate AutoTriage configurations against human consensus on the annotated Terminal-Bench dataset, computing Cohen's κ between each automated analyzer and the human majority-vote label. We compare nine configurations varying along two axes: **model** (GPT-5.2 Codex, Claude Sonnet 4.5, Claude Haiku 4.5) and **access mode** (LLM-only, read-only agent, agent-sandbox with full execution access). The human-human baseline ($\kappa = 0.929$) provides the ceiling for automated performance. Each configuration was evaluated on the same set of trajectories to ensure comparability; sample sizes in the error analysis (Table 4) differ slightly because some models failed to produce a classification on certain trajectories (e.g., due to context length or timeout). For agent modes, the judge was given up to 30 minutes of wall-clock time per trajectory, matching the upper bound of human annotation time.

5.3 Results

Table 3 shows AutoTriage performance across configurations. The agent-sandbox configuration with GPT-5.2 Codex achieves $\kappa = 0.833$ (88.9% agreement). The consistent ordering across access modes for this model—and the large κ gap between modes—suggests the pattern is robust.

Table 3: AutoTriage evaluation on Terminal-Bench: Cohen's κ and agreement against human consensus. Agent-sandbox mode with GPT-5.2 Codex achieves the highest κ. The benefit of execution access is model-dependent: agent modes improve Codex but hurt Claude models. Higher is better for both metrics (↑).

Mode	Model	$\kappa \uparrow$	Agr. ↑
Agent-sandbox	GPT-5.2 Codex	**0.833**	**88.9%**
Agent (read-only)	GPT-5.2 Codex	0.640	74.2%
LLM-only	GPT-5.2 Codex	0.592	71.0%
Agent-sandbox	Sonnet 4.5	0.433	54.8%
Agent (read-only)	Sonnet 4.5	0.423	54.8%
LLM-only	Sonnet 4.5	0.639	74.2%
Agent-sandbox	Haiku 4.5	0.332	45.2%
Agent (read-only)	Haiku 4.5	0.290	38.7%
LLM-only	Haiku 4.5	0.534	64.5%
Human-Human Baseline		*0.929*	*97.7%*

Key findings:

Executable environment access substantially improves the best model. For GPT-5.2 Codex, agent-sandbox ($\kappa = 0.833$) outperforms both read-only agent ($\kappa = 0.640$) and LLM-only ($\kappa = 0.592$), with the $+0.24$ κ gain suggesting that executing code and reproducing failures provides critical context for reliable attribution.

The benefit of execution is model-dependent. Claude Sonnet 4.5 in LLM-only mode ($\kappa = 0.639$) matches GPT-5.2 Codex in read-only agent mode ($\kappa = 0.640$), but agent modes *hurt* Claude models. Effective tool use for log analysis appears to require model-specific optimization.

5.4 Error Analysis: Where AutoTriage Fails

To understand *how* AutoTriage fails—not just how often—we examine confusion matrices comparing automated predictions to human consensus in read-only agent mode. Table 4 shows results for all three triage models on matched trajectories.

Table 4: AutoTriage confusion matrices (read-only agent mode, Terminal-Bench). Rows are human consensus labels; columns are AutoTriage predictions. Only failure rows are shown (success predictions omitted). Sample sizes differ across models because some models failed to produce a classification on certain trajectories. The dominant error for weaker models is Agent→Task misattribution: Sonnet misclassifies 50% of agent failures as task defects; Haiku misclassifies 44%.

		Predicted					
Model	**Human↓**	**Succ.**	**Agent**	**Task**	**Harn.**	**N**	**Recall**
Codex	Agent Err.	0	**8**	1	0	9	89%
	Harness	0	1	0	**3**	4	75%
Sonnet	Agent Err.	0	**8**	8	0	16	50%
	Harness	0	1	2	**2**	5	40%
Haiku	Agent Err.	0	**7**	7	2	16	44%
	Harness	0	1	4	**0**	5	0%

The dominant error pattern is **systematic over-attribution to task defects**: when the agent made a genuine logical error, weaker triage models blame the benchmark instead. Codex correctly identifies 89% of agent failures; Sonnet and Haiku misattribute roughly half to task defects. This is not random noise—it is a directional bias. Examining AutoTriage reasoning reveals the mechanism: weaker models construct plausible defenses of the agent ("the specification is underspecified," "all agents failed this task") rather than identifying the root-cause error.

Qualitative example. On the `build-pov-ray` task, all three human annotators label the failure as a *logical failure*: the agent chose ZIP archives instead of TAR.Z, producing uppercase filenames that failed hash validation. Codex correctly identifies this as the agent's oversight. Sonnet instead argues: "*The task instruction says 'Find and download the source archives' without specifying which archive format... the agent's functionally correct POV-Ray 2.2 build fails only on undocumented file naming expectations.*" Haiku reaches the same conclusion. Both construct a reasonable-sounding defense—but humans unanimously disagree.

This bias has direct implications for lifelong self-evolving agents: a weaker critic would systematically *under-count* agent errors and *over-flag* valid tasks for removal, corrupting the reward signal in exactly the direction that prevents productive adaptation over the agent's lifespan.

6 Discussion

From Scoreboards to Diagnostics. Current benchmarks produce a single number. Failure attribution decomposes this into actionable categories: tool call errors suggest scaffold improvements, knowledge gaps suggest better retrieval or fine-tuning targets, and task defects flag benchmark issues requiring curation. Each failure category implies a different intervention, transforming evaluation from a measurement activity into a diagnostic one.

Practical Deployment for Lifelong Agents. In a production setting, a lifelong coding agent encounters a continuous stream of tasks with varying quality. Without failure attribution, the agent must treat every failure as a learning signal, leading to two failure modes: training on unsolvable tasks degrades the policy (false negative rewards), while not training on genuine errors wastes opportunities for improvement. AutoTriage acts as a filter in this pipeline, separating actionable signal from noise before it enters the training loop. The key design decision—giving the judge sandbox access to reproduce failures—is motivated by the observation that static log analysis is insufficient for many real-world debugging scenarios. A judge that can re-run tests with modified assertions, check whether failures are deterministic, and verify environment state provides evidence that purely text-based analysis cannot.

Implications for Lifelong Self-Evolving Agents. For agents that improve through iterative evaluation-and-modification cycles—whether via RL (Le et al., 2022), evolutionary code search (Novikov et al., 2025), open-ended skill discovery (Wang et al., 2023), or self-referential optimization (Schmidhuber, 2003; Wang et al., 2025a)—critic quality is a first-order concern. Our error analysis reveals that weaker critics systematically over-attribute agent failures to task defects, constructing plausible defenses rather than identifying root causes. In a closed loop, this bias compounds across iterations: the agent discards valid training signal while retaining noise, inflating perceived performance while degrading actual capabilities. For lifelong agents that must sustain improvement over extended deployment (Gao et al., 2025), the critic must be at least as capable as the agent being improved.

Closing the Loop on Hillclimbing. Recent self-evolving coding agents—Darwin Gödel Machine (Zhang et al., 2025a), SICA (Robeyns et al., 2025), and HGM (Wang et al., 2025a)—improve by hillclimbing over benchmark scores: an agent modifies its own scaffold, evaluates the modification on a benchmark, and keeps it only if the score improves. This paradigm implicitly assumes that benchmark scores are a faithful proxy for capability. Our results challenge this assumption. When a benchmark contains task defects, the score conflates agent capability with benchmark quality: an agent that correctly solves a defective task receives no credit, while one that happens to work around the defect may score higher despite worse general capability. Without failure attribution, hillclimbing optimizes for benchmark-specific performance rather than genuine improvement—a form of overfitting to evaluation artifacts rather than to the underlying task distribution. AutoTriage provides the missing diagnostic layer: by filtering task defects before they enter the evaluation loop, it ensures that the signal hillclimbing optimizes over reflects actual agent capability. This is especially critical for lifelong agents, where hundreds of hillclimbing iterations amplify any systematic bias in the evaluation signal.

Sandboxed Execution and Critic Quality. For GPT-5.2 Codex, full sandbox access ($\kappa = 0.833$) substantially outperforms read-only access ($\kappa = 0.640$), suggesting that executing code and reproducing failures provides critical diagnostic signal. However, this benefit is model-dependent: Claude

models perform *worse* in agent modes, suggesting that effective agentic investigation requires strong tool-use capabilities.

Failure Attribution as Infrastructure for Lifelong Agents. Self-evolving agents that operate over extended deployment face a compound problem: not only must they learn from individual failures, but they must do so *reliably across time*. A lifelong coding agent that encounters thousands of tasks over its operational lifespan will accumulate a training corpus shaped by every attribution decision its critic makes. Our finding that weaker critics systematically over-attribute agent errors to task defects suggests that without calibrated failure attribution, long-running agents will experience a form of reward drift—gradually losing signal quality as corrupted attributions compound. AutoTriage provides a mechanism to maintain signal integrity across an agent's lifespan, complementing existing work on memory management and continual learning for lifelong agents (Gao et al., 2025).

Limitations. Our annotation study covers one benchmark domain (software engineering); broader validation across domains such as web navigation, scientific experimentation, and embodied tasks would strengthen the taxonomy's generality for lifelong agents that operate across diverse environments. AutoTriage evaluation uses a limited number of overlapping trajectories between human and automated annotation. Larger-scale evaluation with representative sampling and additional domains is ongoing. Additionally, our analysis focuses on failure attribution at a single point in time; longitudinal studies tracking how attribution quality affects agent improvement across multiple training iterations would provide direct evidence for the reward drift hypothesis discussed above.

7 Conclusion

We have formalized failure attribution—the task of distinguishing task defects from agent errors—and shown it to be a well-defined, reliable task ($\kappa = 0.929$, $N = 118$) that current benchmarks do not provide. Our three-category taxonomy (task defect, agent error, infrastructure failure) maps directly to the signal corruption problem in agent training: each category implies a different intervention, from fixing the benchmark to retraining the agent to simply re-running the evaluation.

AutoTriage, our agentic judge system, achieves $\kappa = 0.833$ in its best configuration (agent-sandbox with GPT-5.2 Codex), with the $+0.24$ κ gain from text-only to sandbox demonstrating that judges benefit from the ability to *run code*—though this advantage is model-dependent. The finding that Claude models perform worse with tool access than without suggests that effective agentic investigation requires model-specific optimization, and that simply giving a judge more capabilities does not guarantee better attribution.

Error analysis reveals the critical failure mode for self-evolving agents: weaker models systematically over-attribute agent failures to task defects—a directional bias that compounds across iterations. This is not random noise but a consistent pattern of constructing plausible defenses of the agent rather than identifying root causes, with direct consequences for any system that hillclimbs over benchmark scores or uses evaluation outcomes as RL reward signal. Together, the taxonomy, annotation protocol, and automated system provide a missing diagnostic layer, enabling the reliable reward signal that lifelong agent evolution requires.

Future work. Key open questions include: (1) What is the downstream impact of filtering task defects on RL training outcomes—does AutoTriage-curated data produce agents that generalize better than those trained on uncurated signal? (2) How does critic quality interact with the number of self-improvement iterations across an agent's lifespan, and does the reward drift we hypothesize manifest in practice over extended training? (3) Can the agentic judge paradigm extend to other forms of automated quality assurance, such as web navigation or scientific experimentation benchmarks? (4) Does domain-specific prompting close the gap for non-SWE domains? (5) How should failure attribution interact with continual learning—can an agent update its failure attribution model as it encounters new task distributions, or does the critic itself require periodic recalibration?

LLM Usage Disclosure

Large language models (GPT-5.2 Codex, Claude Sonnet 4.5, Claude Haiku 4.5) were used as experimental subjects in this work (§5). Additionally, LLMs were used to assist with manuscript drafting, editing, and analysis of experimental results. All content was reviewed and verified by the authors, who take full responsibility for the paper's claims and conclusions.

REFERENCES

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.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6s5uXNWGIh.

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.

Neil Chowdhury, James Aung, Jun Shern Chan, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified. https://openai.com/index/introducing-swe-bench-verified/, 2024.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian, Zhenghailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven CH Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328, 2022.

Ming Ma, Shaokun Zhang, Qingyun Wu, and Chi Wang. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. *arXiv preprint arXiv:2512.06749*, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025.

Jürgen Schmidhuber. Gödel machines: Self-referential universal problem solvers making provably optimal self-improvements. Technical Report IDSIA-19-03, IDSIA, 2003. arXiv:cs.LO/0309048.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K Reddy. Execution-based code generation using deep reinforcement learning. *arXiv preprint arXiv:2301.13816*, 2023.

Joar Skalse, Nikolaus HR Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

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. *arXiv preprint arXiv:2305.16291*, 2023.

Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-gödel machine: Human-level coding agent development by an approximation of the optimal self-improving machine. *arXiv preprint arXiv:2510.21614*, 2025a.

You Wang, Siegfried Groot, Max Schröder, and Michael Pradel. Are "solved issues" in SWE-bench really solved correctly? An empirical study. *arXiv preprint arXiv:2503.15223*, 2025b.

Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. Data-centric artificial intelligence: A survey. *arXiv preprint arXiv:2303.10158*, 2023.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025a.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025b.

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, 2023.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, et al. Where LLM agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024.