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

Jesse Hu* **Pratyush Shukla*** **Ke Huang**
Abundant AI
{jesse, pratyush, ke}@abundant.ai
*Equal contribution

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 **recursive self-improvement**, 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 self-modification. 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 system that uses its own critic to drive self-improvement.

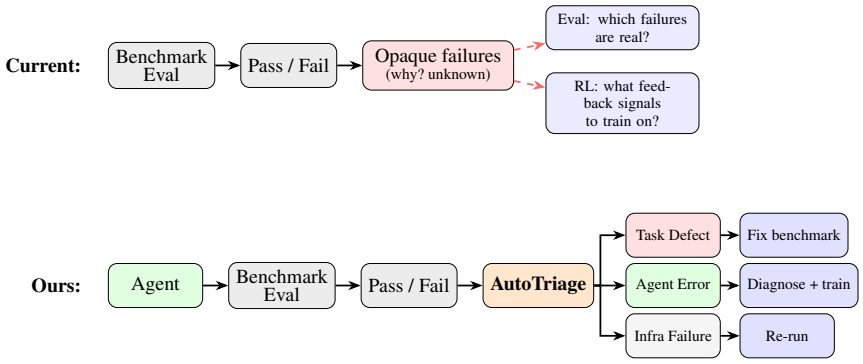
1 INTRODUCTION

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 reinforcement learning, where pass/fail outcomes become rewards (Le et al., 2022; Shojaee et al., 2023).

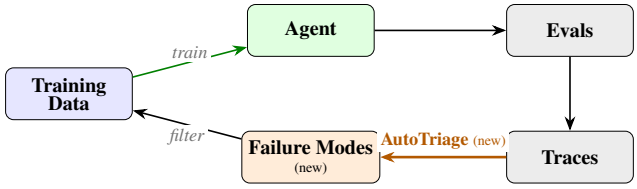
Both roles require understanding *why* an agent failed, not just *that* it failed. When an agent fails a task, the failure could indicate:

- **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



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



(b) The self-improvement 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 recursive self-improvement loop, failure attribution is the critical step between traces and training data. Without it, the agent trains on corrupted signal (§5.4).

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 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 recursive self-improvement (Schmidhuber, 2003; Wang et al., 2025a). An agent that improves through iterative cycles of evaluation, diagnosis, and self-modification 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 self-improvement, 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 recursive self-improvement (§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., 2025) 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. Our work complements these efforts by providing automated, ongoing quality assessment rather than one-time manual filtering.

LLM-as-Judge and Recursive Self-Improvement. 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, coding agents that improve recursively—through evolutionary search (Novikov et al., 2025) or hill-climbing over self-modifications (Wang et al., 2025a)—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.

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.

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).

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

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.

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).

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.

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%

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. 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, 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.

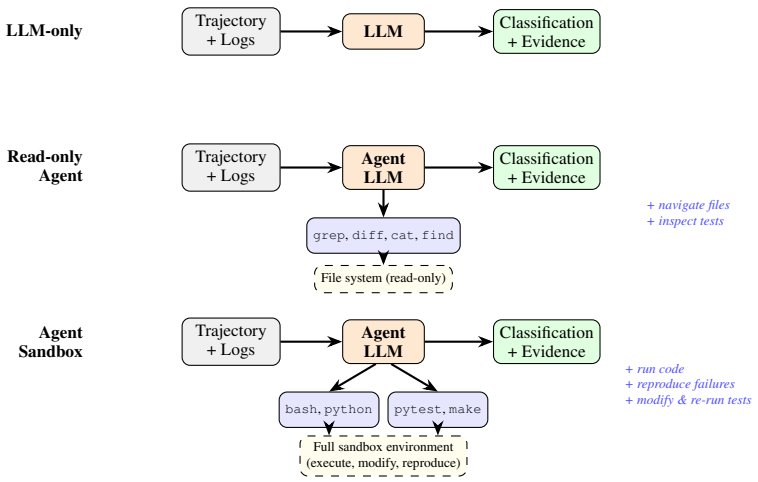


Figure 2: AutoTriage access modes. **LLM-only**: text-based classification (baseline). **Read-only agent**: navigates the file system to inspect task specifications, test implementations, and gold solutions. **Agent-sandbox**: executes code, runs tests, and reproduces failures in a replica of the original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

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.

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	κ ↑	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 due to varying trajectory overlap with the human-annotated set. 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%
	Bucket accuracy: 84.6% on failures ($\kappa_{\text{code}} = 0.69$)						
Sonnet	Agent Err.	0	**8**	8	0	16	50%
	Harness	0	1	2	**2**	5	40%
	Bucket accuracy: 36.4% on failures ($\kappa_{\text{code}} = 0.17$)						
Haiku	Agent Err.	0	**7**	7	2	16	44%
	Harness	0	1	4	**0**	5	0%
	Bucket accuracy: 31.8% on failures ($\kappa_{\text{code}} = 0.08$)						

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 recursive self-improvement: 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 self-modification.

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.

Implications for Recursive Self-Improvement. For agents that improve through iterative evaluation-and-modification cycles—whether via RL (Le et al., 2022), evolutionary code search (Novikov et al., 2025), 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. The critic must be at least as capable as the agent being improved.

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.

Limitations. Our annotation study covers one benchmark domain (software engineering); broader validation across domains would strengthen the taxonomy. AutoTriage evaluation uses a limited number of overlapping trajectories. Larger-scale evaluation with representative sampling and additional domains is ongoing.

7 Conclusion

We have shown that failure attribution—distinguishing task defects from agent errors—is a well-defined task ($\kappa = 0.929$, $N = 118$) that current benchmarks do not provide. 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 suggesting that judges benefit from the ability to *run code*—though this advantage is model-dependent. Error analysis reveals the critical failure mode for self-improvement: weaker models systematically over-attribute agent failures to task defects—a directional bias that compounds across iterations. Together, the taxonomy, annotation protocol, and automated system provide a missing diagnostic layer, enabling the reliable reward signal that recursive self-improvement requires.

Future work. Key open questions include: (1) What is the downstream impact of filtering task defects on RL training outcomes? (2) How does critic quality interact with the number of self-improvement iterations? (3) Can the agentic judge paradigm extend to other forms of automated quality assurance? (4) Does domain-specific prompting close the gap for non-SWE domains?

Acknowledgments

We thank Yiren Lu, Kelly Buchanan, and Derek Chen for detailed feedback on earlier drafts. Maharshi Mandapati and Rohan Jayaprakash contributed to the human annotation study. Meji Abidoye contributed to the production implementation of AutoTriage.

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.

Safety & Ethics Note

AutoTriage could inform recursive self-improvement loops. Our error analysis (§5.4) highlights a concrete risk: biased critics misattribute agent failures to task defects, causing self-improving agents to discard valid training signal. We recommend monitoring critic calibration before deploying automated triage in closed-loop training.

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.

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.

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.

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.

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*, 2025.

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.

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.