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

Anonymous authors
Paper under double-blind review

ABSTRACT

When coding agents fail benchmark tasks, the failure is opaque: benchmarks report only that the agent failed, not *why*. This matters for two increasingly 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 **RL training**, failures serve as negative reward signal, but training on task defects as if they were agent errors introduces noise that corrupts learned policies. We formalize this problem with a failure attribution taxonomy and validate it through human annotation ($\kappa > 0.84$, $N = 158$) across two benchmarks, revealing significant variation in task defect rates. 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, navigating file systems, and analyzing error logs. We evaluate nine configurations across three models and three access modes (text-only, read-only agent, full sandbox). On a software engineering benchmark, AutoTriage achieves $\kappa = 0.83$, reaching 90% of human inter-annotator agreement. To our knowledge, this is the first use of an agentic judge with full execution access for failure triage. Our framework provides a missing diagnostic layer in the agent development pipeline, transforming benchmarks from pass/fail scoreboards into tools for both targeted improvement and clean training data curation.

1 INTRODUCTION

LLM agent benchmarks like SWE-bench (Jimenez et al., 2024), HumanEval (Chen et al., 2021), MLE-bench (Chan et al., 2024), and domain-specific evaluations such as FinanceAgentBench 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.

Without this distinction, both use cases break down. For evaluation, practitioners cannot determine whether low scores reflect agent limitations or benchmark noise—they 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 rather than agent limitations. Yet current benchmarks provide only a binary pass/fail, leaving the quality of both diagnostic signal and training data unknown.

Failure attribution addresses both problems simultaneously. By classifying each failure, practitioners gain a decomposition of evaluation results into actionable categories—tool call errors suggest

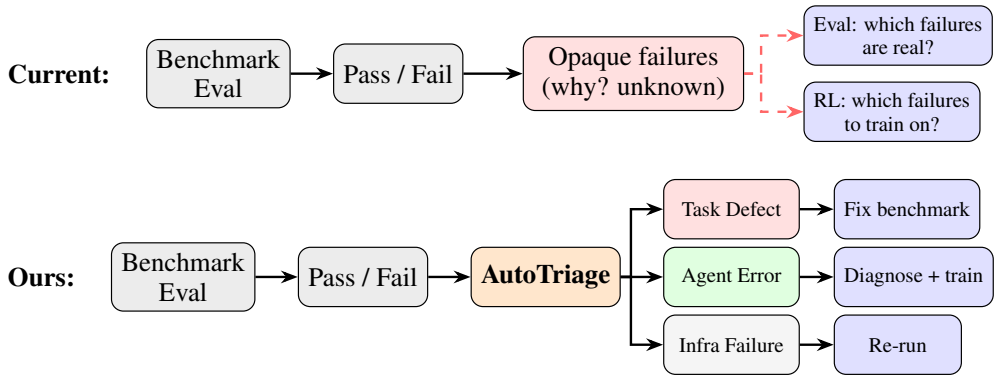

Figure 1: Current benchmarks report only pass/fail, leaving both evaluation and training signal opaque. AutoTriage attributes each failure to a root cause, enabling downstream action: **task defects** trigger benchmark fixes, **agent errors** provide actionable diagnostic and training signal, and **infrastructure failures** are re-run.

scaffold improvements, knowledge gaps suggest better retrieval, and task defects flag benchmark issues. The same attribution filters noisy failures from RL reward computation. This closes the loop between evaluation, diagnosis, and improvement.

Contributions. We address this gap with three contributions:

1. A **taxonomy** of failure modes distinguishing task defects from agent errors, applicable across LLM agent domains (§3)
2. A **human annotation study** on 158 trajectories establishing that attribution is reliable ($\kappa > 0.84$) and revealing significant variation in task defect rates across benchmarks (§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 access achieves $\kappa = 0.83$ (90% of human agreement) on a software engineering benchmark (§5)

2 Related Work

Benchmark Curation. 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 the task defect categories in our taxonomy. LiveCodeBench (Jain et al., 2024) addresses contamination through temporal splits but does not examine task quality. Our work complements these efforts by providing automated, ongoing quality assessment that can scale with benchmark growth.

Agent Failure Analysis. MAST (Cemri et al., 2025) proposes a taxonomy for multi-agent system failures, focusing on coordination and communication breakdowns. AgentErrorBench (Zhu et al., 2025) benchmarks error detection but does not distinguish task-attributable from agent-attributable failures. Our taxonomy is specifically designed for the RL training data curation use case, where this distinction determines signal quality.

LLM-as-Judge. Using language models to evaluate outputs has become standard practice (Zheng et al., 2023). The Agent-as-a-Judge framework (Zhuge et al., 2024) extends this by equipping evaluators with agentic capabilities such as tool use and code execution. However, both paradigms focus on response quality (helpfulness, correctness) rather than failure attribution. AutoTriage applies the agentic judge paradigm specifically to failure diagnosis, requiring deeper analysis of agent trajectories and task specifications to distinguish task defects from agent errors.

Reward Signal Quality. Work on reward hacking (Skalse et al., 2022) and RLHF (Ouyang et al., 2022; Bai et al., 2022) assumes access to clean reward signals. More broadly, the data-centric AI movement (Zha et al., 2023) has emphasized that data quality is often more impactful than model architecture—yet this insight has not been systematically applied to agent benchmark data. Our work addresses the upstream problem: ensuring that the binary pass/fail signals from benchmarks actually reflect agent performance rather than task defects.

3 Failure Attribution Taxonomy

We develop a two-level taxonomy (Table 1) based on analysis of 200+ failed trajectories across multiple benchmarks spanning software engineering, financial analysis, and general tool use.

Table 1: Failure attribution taxonomy. Task defects indicate unreliable signal that should be filtered from training; agent errors indicate valid signal.

Category	Subcategory	Description
Task Defect (filter)	Ambiguous Spec	Requirements unclear or admit multiple valid interpretations
	Environment Fail	Dependencies broken, build failures, missing resources
	Brittle Tests	Tests check implementation details rather than correctness
	Solution Leakage	Gold solution or key hints visible to agent
Agent Error (valid signal)	Logical Failure	Incorrect algorithm, wrong approach, bugs in implementation
	Tool Call Error	Malformed commands, wrong tool selection, syntax errors
	Knowledge Gap	Missing domain knowledge or API familiarity
	Premature Stop	Gave up early, incomplete solution attempt
	Timeout	Ran out of time or compute budget
Infrastructure	Infra Failure	Sandbox crash, network issues, harness bugs

The key distinction: **task defects** indicate the benchmark failed the agent (unreliable signal), while **agent errors** indicate the agent failed the task (valid training signal). Infrastructure failures are neither—they should be re-run rather than used for training.

Cross-Domain Applicability. While specific failure modes vary by domain, the top-level taxonomy applies broadly:

- **Software engineering**: Ambiguous specs often involve unclear requirements; brittle tests check exact string matches rather than semantic correctness
- **Financial analysis**: Ambiguous specs involve underspecified analysis criteria; environment failures include missing data sources or API issues
- **Tool use**: Task defects include impossible tool sequences or missing permissions

4 Human Annotation Study

We conducted a human annotation study to (1) validate that failure attribution is a well-defined task humans can perform reliably, and (2) measure task defect rates across benchmarks.

4.1 Setup

Three expert annotators with experience in LLM agent evaluation independently labeled 158 failed trajectories from two benchmarks:

- **Terminal-Bench** (118 trajectories): A command-line software engineering benchmark requiring agents to complete terminal-based tasks
- **FinanceAgentBench** (40 trajectories): A financial analysis benchmark requiring agents to analyze documents, extract data, and produce reports

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 labels from the taxonomy in Table 1.

4.2 Inter-Annotator Agreement

Table 2 shows inter-annotator agreement measured by Fleiss' κ (Fleiss, 1971) (chance-corrected multi-rater agreement). We also report pairwise Cohen's κ (Cohen, 1960) in Table 3 to characterize individual annotator consistency.

Table 2: Multi-rater inter-annotator agreement (Fleiss' κ). Both benchmarks achieve "almost perfect" agreement ($\kappa > 0.8$), establishing failure attribution as a reliable task.

Benchmark	N	Fleiss' κ	Raw Agreement
Terminal-Bench	118	0.929	97.7%
FinanceAgentBench	40	0.848	94.2%

Table 3: Pairwise inter-annotator agreement (Cohen's κ). All pairs achieve substantial or almost perfect agreement on both benchmarks.

	Terminal-Bench		FinanceAgentBench	
Annotator Pair	κ	Agree.	κ	Agree.
A1 vs. A2	0.973	99.2%	0.929	97.5%
A1 vs. A3	0.893	96.6%	0.815	92.5%
A2 vs. A3	0.922	97.5%	0.809	92.5%

Fleiss' $\kappa > 0.84$ on both benchmarks demonstrates that failure attribution is not subjective—trained annotators converge on the same labels even across domains. This provides both a ceiling for automated systems and validates the taxonomy's discriminative power.

4.3 Disagreement Analysis

To characterize the nature of disagreements, we examine a detailed breakdown on a calibration subset (61 trajectories: 22 from Terminal-Bench, 39 from FinanceAgentBench). Table 4 shows the top-level confusion matrix aggregating all pairwise comparisons across three annotator pairs ($N = 181$).

Table 4: Human-human confusion matrix on calibration subset (top-level categories). The primary source of disagreement is Agent Error vs. Task Defect (10 cases, 5.5%).

	Task Defect	Agent Error	Infra	Total
Task Defect	93	2	0	95
Agent Error	8	63	0	71
Infra	0	0	15	15
Total	101	65	15	181

Infrastructure is never confused with other categories (clear failure mode). The 8 cases where one annotator labeled Agent Error and another labeled Task Defect typically involve situations where an agent made errors that a clearer specification might have prevented—a genuinely ambiguous boundary.

4.4 Failure Attribution Reveals Benchmark Quality Variation

To illustrate the practical impact of failure attribution, Table 5 shows the distribution of majority-vote labels on a pilot subset of 61 trajectories. These trajectories were selected for annotator calibration

rather than sampled representatively, so exact rates should be treated as illustrative rather than population estimates. Updated rates from systematic sampling will accompany the camera-ready version.

Table 5: Failure mode distribution on pilot subset (majority vote, $N = 61$). Selected for annotator calibration, not representative sampling. The qualitative contrast—Terminal-Bench dominated by agent errors, FinanceAgentBench dominated by task defects—is consistent across subsets we have examined.

Category	Terminal-Bench	FinanceAgentBench	Total
Task Defects			
Ambiguous Spec	1	31	32
Environment Fail	0	0	0
Agent Errors			
Logical Failure	14	6	20
Tool Call Error	2	0	2
Knowledge Gap	0	1	1
Infrastructure			
Infra Failures	5	0	5
Total	22	38	60*
Task Defect Rate	**4.5%**	**81.6%**	53.3%

*One trajectory received no majority label and is excluded.

Even on this pilot subset, the qualitative contrast is stark. Terminal-Bench failures are overwhelmingly agent errors—genuine mistakes that are informative for both diagnosis and training. FinanceAgentBench failures are dominated by ambiguous specifications—these mislead diagnostic evaluation (suggesting agent weakness where none exists) and corrupt RL training signal. Crucially, this variation is invisible to standard benchmark harnesses, which report only pass/fail rates. Failure attribution makes benchmark quality measurable, and even order-of-magnitude differences in task defect rates have major practical implications for both evaluation trust and training data curation.

5 AutoTriage: Automated Failure Attribution

Human annotation does not scale. We introduce AutoTriage, an automated system for failure attribution.

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. It outputs a classification from the taxonomy with supporting evidence. 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 datasets, 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 ($\kappa = 0.929$ for Terminal-Bench, $\kappa = 0.848$ for FinanceAgentBench) provides the ceiling for automated performance.

5.3 Results

Table 6 shows AutoTriage performance across configurations. On Terminal-Bench, the agent-sandbox configuration with GPT-5.2 Codex achieves $\kappa = 0.833$ (88.9% agreement), reaching **90% of human inter-annotator agreement**. Performance varies substantially across both models and access modes.

Table 6: AutoTriage evaluation: Cohen's κ against human consensus. Agent-sandbox mode (full execution access) with GPT-5.2 Codex reaches 90% of human agreement on Terminal-Bench. FinanceAgentBench remains challenging for all configurations.

		Terminal-Bench		**FinanceAgentBench**	
Mode	**Model**	κ	**Agr.**	κ	**Agr.**
Agent-sandbox	GPT-5.2 Codex	**0.833**	**88.9%**	0.242	75.0%
Agent (read-only)	GPT-5.2 Codex	0.640	74.2%	—	—
LLM-only	GPT-5.2 Codex	0.592	71.0%	**0.328**	**80.0%**
Agent-sandbox	Sonnet 4.5	0.433	54.8%	0.042	60.0%
Agent (read-only)	Sonnet 4.5	0.423	54.8%	—	—
LLM-only	Sonnet 4.5	0.639	74.2%	0.188	45.0%
Agent-sandbox	Haiku 4.5	0.332	45.2%	0.073	65.0%
Agent (read-only)	Haiku 4.5	0.290	38.7%	—	—
LLM-only	Haiku 4.5	0.534	64.5%	0.222	65.0%
Human-Human Baseline		*0.929*	*97.7%*	*0.848*	*94.2%*

Key findings:

Executable environment access substantially improves the best model. For GPT-5.2 Codex on Terminal-Bench, agent-sandbox (full execution, $\kappa = 0.833$) outperforms both the read-only agent ($\kappa = 0.640$) and LLM-only ($\kappa = 0.592$). The $+0.24$ κ gain from text-only to full sandbox access shows that the ability to execute code, run tests, and reproduce failures provides critical context for reliable attribution.

Model capability is necessary but not sufficient. GPT-5.2 Codex outperforms Claude models in agent modes on Terminal-Bench, but Claude Sonnet 4.5 in LLM-only mode ($\kappa = 0.639$) matches GPT-5.2 Codex in read-only agent mode ($\kappa = 0.640$). The interaction between model and mode is complex: agent modes help GPT-5.2 Codex but hurt Claude models, suggesting that effective tool use for log analysis requires model-specific optimization.

Domain difficulty varies dramatically. All configurations perform substantially worse on FinanceAgentBench (best $\kappa = 0.328$) than on Terminal-Bench (best $\kappa = 0.833$). This parallels the finding that humans also find FinanceAgentBench slightly harder ($\kappa = 0.848$ vs. 0.929), but the gap is far larger for automated systems—suggesting that financial domain failures involve more nuanced judgment that current models struggle to replicate.

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, enabling targeted improvement rather than undirected iteration. This transforms evaluation from a measurement activity into a diagnostic one.

Implications for RL Training. The same attribution that enables diagnosis also enables training data curation. Without failure attribution, the quality of RL training data derived from benchmarks is unknown and unknowable. Our annotation study reveals that this quality varies dramatically across benchmarks. Benchmark selection is therefore a first-order decision for training pipelines, yet currently an invisible one. Systematic failure attribution provides the missing quality metric, filtering task defects before they become reward signal.

Sandboxed Execution for Agentic Judges. Our strongest finding is that full sandbox access ($\kappa = 0.833$) substantially outperforms read-only agent access ($\kappa = 0.640$) for the same model. The $+0.19$ κ gain from read-only to sandbox mode shows that the ability to execute code, run tests, and reproduce failures—not just navigate files—provides critical diagnostic signal. The full $+0.24$ κ improvement from LLM-only to sandbox mode (Table 6) confirms that structured investigation via tools provides context far beyond what text summaries offer. To our knowledge, this is the first use of a coding agent with full execution access as a judge for failure analysis.

Domain-Dependent Difficulty. The gap between Terminal-Bench ($\kappa = 0.833$) and FinanceAgentBench ($\kappa = 0.328$) automated performance suggests that attribution is substantially easier for domains with clear correctness criteria (code that passes or fails tests) than for domains requiring subjective judgment (financial analysis quality). Software engineering benchmarks may be ready for fully automated triage, while other domains still require human oversight.

Limitations. Our annotation study covers two benchmarks; broader validation would strengthen the taxonomy. AutoTriage evaluation on Terminal-Bench uses 18 overlapping trajectories for the best configuration and FinanceAgentBench uses 20, limiting statistical power. The interaction between model and access mode (agent modes help GPT-5.2 Codex but hurt Claude models) warrants further investigation. Larger-scale evaluation is ongoing.

7 Conclusion

We have shown that failure attribution—distinguishing task defects from agent errors—is a well-defined task ($\kappa > 0.84$, $N = 158$) that reveals significant, previously invisible variation in benchmark quality. We introduce AutoTriage, an agentic judge system for automated failure triage, and evaluate nine configurations across three frontier models and three access modes. Our best configuration—agent-sandbox with full execution access using GPT-5.2 Codex—achieves $\kappa = 0.833$ on a software engineering benchmark, reaching 90% of human agreement. The $+0.24$ κ gain from text-only to sandbox access demonstrates that judges need to *run code*, not just read logs. Together, the taxonomy, annotation protocol, and automated system provide a missing diagnostic layer in the agent development pipeline, serving two critical needs: transforming evaluations into actionable diagnostics for agent improvement, and curating clean reward signal for RL training.

Future work. Key open questions include: (1) What is the downstream impact of filtering task defects on RL training outcomes? (2) Can domain-specific prompting or fine-tuning close the gap on FinanceAgentBench? (3) Why do agent modes help GPT-5.2 Codex but hurt Claude models? (4) Can we extend the agentic judge paradigm to other forms of automated quality assurance in ML pipelines?

Acknowledgments

Removed for anonymous review.

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.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

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? *arXiv preprint arXiv:2503.13657*, 2025.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Makatura, Gaetan Hadjeres, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

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.

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. *arXiv preprint arXiv:2403.07974*, 2024.

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? *arXiv preprint arXiv:2310.06770*, 2024.

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.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

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. *Advances in Neural Information Processing Systems*, 35, 2022.

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.

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. *Advances in Neural Information Processing Systems*, 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.