ACTCI-BENCH: EVALUATING SELECTIVE DECISIONS IN CI MAINTENANCE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

A software agent responding to a failed continuous integration (CI) run must decide whether to modify the repository, take a non-patch action, or defer to another owner. Repair benchmarks that start from known patch tasks do not evaluate this decision. We present ActCI-Bench, an evaluation of selective CI maintenance decisions over 1,543 real failure episodes. Each episode links failure evidence to three targets: actionability for the current author, root cause, and appropriate response. The benchmark combines logs and repository context with available CI history and developer handling, and evaluates both classification quality and false repairs on non-actionable episodes. In our evaluation, 509 episodes (33.0%) are non-actionable for the current author. Repository-aware agents achieve higher macro-F1 than specialized triage baselines, yet still propose patches on 41-48% of non-actionable cases. Richer context and an explicit diagnosis-before-response workflow improve response selection in the reported retrospective setting. These findings identify response selection and restraint as distinct dimensions of coding-agent capability. Because some evaluated context includes later developer handling, full-context scores should be read as retrospective evidence-rich results rather than estimates of performance at the moment a CI run fails.

1 INTRODUCTION

A failed continuous integration (CI) run presents an agent with a decision under incomplete evidence. It might reflect a code regression introduced by the current change, a flaky test, dependency drift, a workflow error, or an external failure. These causes call for different actions. Producing a patch whenever a run turns red can add unnecessary changes, hide infrastructure problems, or weaken a test.

Repository-level benchmarks such as SWE-bench Jimenez et al. (2024) and CI repair benchmarks Muna et al. (2026) assess agents after a repair task has been established. Figure 2 illustrates three decisions starting from the same red CI signal. ActCI-Bench instead evaluates the preceding decision: should the current author intervene, what caused the failure, and what response fits the evidence? This formulation connects CI maintenance to selective decision making by agents: a good system must act effectively when a patch is appropriate and refrain from patching when rerun, isolation, workflow maintenance, or escalation is more appropriate.

We assemble 1,543 real CI failure episodes from public sources, retaining failed-run logs, repository and workflow context, dependency evidence, CI history, and developer handling where available. Each episode has actionability, root-cause, and response labels. Our evaluation reports macro-F1 on these three tasks, along with false repair rate on non-actionable cases and patch pass rate on repairable cases. We compare triage rules, specialized flakiness systems, a repair baseline, and repository-aware coding agents; we also analyze context availability and a diagnosis-before-response workflow. The benchmark exposes a practical gap: strong patch performance need not imply good decisions about whether to patch.

Scope of the evidence. We distinguish evidence available when a failed run appears from subsequent reruns, developer discussions, follow-up patches, and later outcomes. The latter are useful for constructing and auditing labels. Where the reported full-context comparison includes such later evidence, it measures retrospective diagnosis with richer information, not a deployable online

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

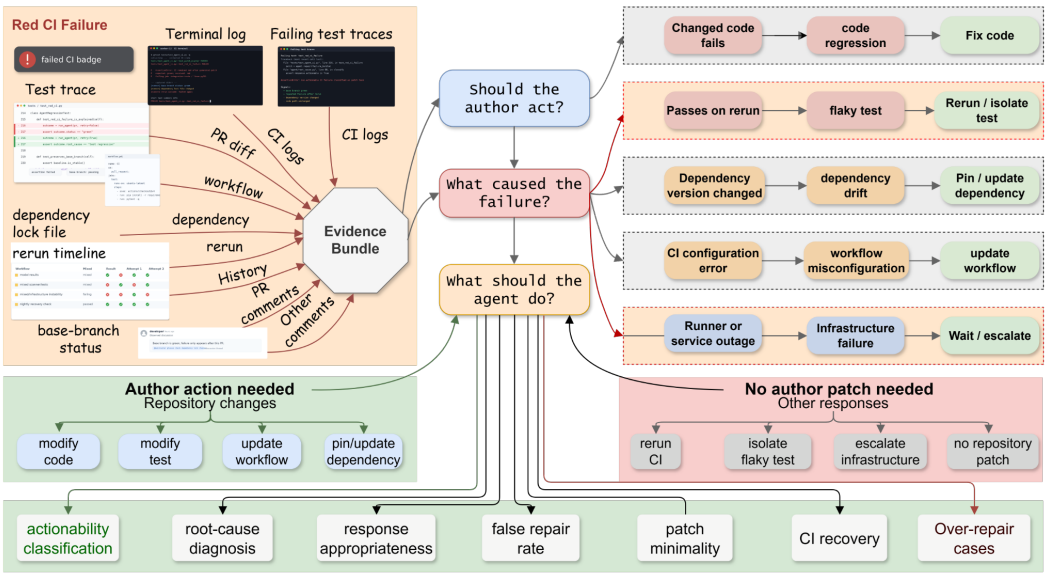


Figure 1: ActCI-Bench overview: evidence supports actionability assessment, root-cause diagnosis, and response selection; responses may involve a repository change or a non-patch action.

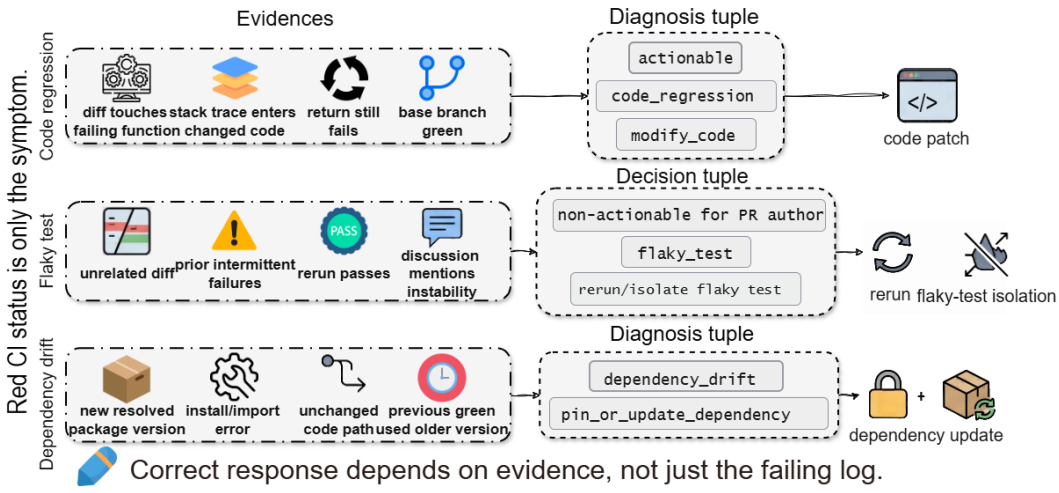


Figure 2: Evidence-grounded CI failure decisions.

agent at failure time. Our claims about context gains are limited accordingly. This distinction matters when interpreting apparent benefits of full context and motivates explicit reporting of evidence provenance.

Our contributions are: (i) a benchmark that frames CI maintenance as joint actionability, diagnosis, and response selection; (ii) a repository-disjoint evaluation with an explicit false-repair metric; and (iii) an empirical analysis of existing baselines, context settings, and diagnosis-first responses. Appendix A.1 gives motivating cases and broader limitations.

2 ActCI-Bench Benchmark

ActCI-Bench evaluates whether software engineering agents can diagnose CI failures and choose responses that match real developer handling. It starts from failed CI episodes and preserves the evidence needed to decide whether a failure is actionable, why it occurred, and how it should be handled.

Table 1: Source composition and evidence coverage.

Source	Episodes	Main evidence types
GHALogs	904	Logs, PR files, rerun evidence, workflow files, dependency files, job metadata, issue comments, review comments
LCA-CI-Builds-Repair	144	Failing logs, repair patches, workflow files, changed files, failed and passed build metadata
IDoFT controlled subset	495	Flaky-test metadata and source provenance
Total	**1,543**	–

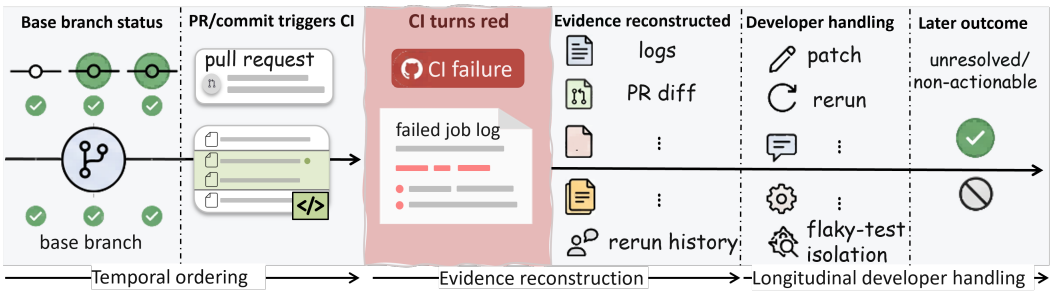


Figure 3: CI failure episode timeline.

2.1 Benchmark Overview

The benchmark contains 1,543 CI failure episodes from three sources: GHALogs Moriconi et al. (2025), LCA-CI-Builds-Repair Bogomolov et al. (2024), and a controlled subset of IDoFT, which builds on the iDFlakies line of flaky-test datasets Lam et al. (2019). Each episode is normalized into a common schema with repository metadata, failed-run metadata, pull-request or commit context when available, labels, and supporting evidence. Table 1 summarizes the source composition and major evidence coverage.

We define a CI failure episode as a temporally grounded event centered on a failed run. It may include the triggering pull request or commit, failed job logs, workflow and dependency files, rerun records, nearby CI runs, developer comments, follow-up commits, and subsequent green runs. Thus, the evaluation unit is the maintenance decision made around a CI failure. Figure 3 illustrates this unit: a failed run is interpreted together with pre-failure context, reconstructed evidence, developer handling, and later outcome.

Each episode has three labels: *actionability*, which states whether the current pull-request author should handle the failure; *root cause*, which identifies the most likely failure type; and *response*, which records the appropriate handling action. These labels define the three tasks used throughout the paper: actionability classification, root-cause diagnosis, and response selection.

2.2 Episode Schema

All sources are converted into a shared, evidence-oriented ActCI schema. The main fields are:

- `episode_id`: a stable identifier for joining task files and metadata;
- `source`: the original data source from which the episode was derived;
- `repo`: repository name, default branch, primary language, star count when available, and archival status;
- `failed_run`: run identifier, attempt number, event type, head commit, workflow path, and failed job information;
- `pr`: pull-request or commit context, including base and head references, changed files, size information, and discussion when available;
- `labels`: normalized actionability, root-cause, and response labels; and
- `evidence`: artifacts supporting the decision, such as logs, workflow files, dependency files, rerun histories, nearby runs, follow-up commits, comments, repair patches, or flaky-test metadata.

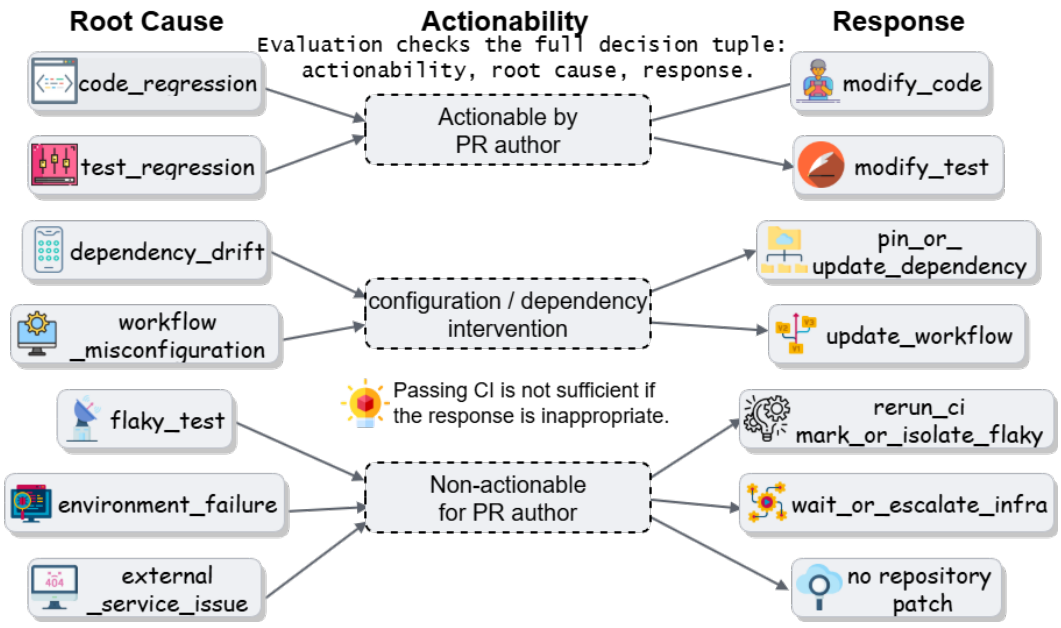


Figure 4: Root-cause, actionability, and response mapping.

Table 2: Label distributions in ActCI-Bench.

Task	Distribution	
	Label	Episodes
Actionability	`actionable_by_pr_author`	1,034
	`non_actionable_for_pr_author`	**509**
	Total	**1,543**
Root cause	**`dependency_drift`**	**500**
	`flaky_test`	**500**
	`code_regression`	260
	`test_regression`	169
	`workflow_misconfiguration`	105
	`environment_failure`	8
	`external_service_issue`	1
	Total	**1,543**
Response	**`pin_or_update_dependency`**	**500**
	`mark_or_isolate_flaky`	**500**
	`modify_code`	260
	`modify_test`	169
	`update_workflow`	104
	`rerun_ci`	8
	`wait_or_escalate_infra`	2
	Total	**1,543**

Because real CI histories are incomplete and heterogeneous, episodes vary in evidence coverage. Each episode records the evidence that can be reconstructed for that failure, enabling log-only, diff-aware, workflow-aware, history-aware, and full-context experimental settings.

2.3 Label Taxonomy

The actionability label captures whether the current developer should intervene after a failed CI run. A failure is actionable when the available evidence indicates that the current change introduced or exposed a problem the pull-request author should handle. It is non-actionable when the failure is unrelated to the current change or better handled through rerun, infrastructure response, maintainer intervention, or another process outside the pull-request author's responsibility. Dependency and workflow updates can be actionable when the current author should make them. We reserve the uncertain label for insufficient or conflicting evidence. In the current public benchmark, 1,034 episodes are labeled `actionable_by_pr_author` and 509 are labeled `non_actionable_for_pr_author`; Table 2 shows the resulting distribution.

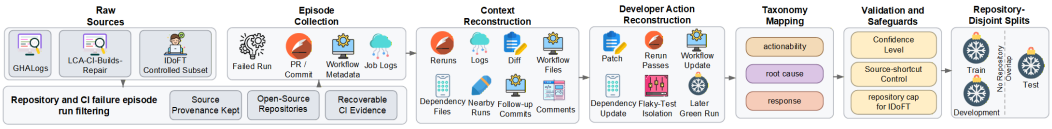


Figure 5: ActCI-Bench construction pipeline.

The root-cause taxonomy covers CI failures that require different diagnostic evidence and different responses. It spans code regressions, test regressions, flaky tests, dependency drift, environment failures, workflow misconfigurations, external-service issues, cache or state issues, base-branch changes, and unknown causes. The current public split contains a subset of these categories, and the taxonomy is defined broadly to support future extension and consistent annotation of CI maintenance episodes. In the current benchmark, the largest root-cause categories are dependency drift and flaky tests, followed by code regressions, test regressions, and workflow misconfigurations. The small number of environment and external-service episodes reflects the difficulty of reconstructing such failures from public project histories and should be considered when interpreting per-class results. The response taxonomy records what an agent should recommend or perform after diagnosis, including code or test modification, dependency pinning or update, workflow update, CI rerun, rebase or synchronization, flaky-test isolation, infrastructure escalation, and no repository action. The response block of Table 2 gives the public benchmark distribution. Figure 4 summarizes how root causes constrain both actionability and appropriate responses; pass/fail recovery alone is therefore an insufficient objective.

2.4 Benchmark Construction Pipeline

A pipeline normalizes heterogeneous CI failure records into a common episode format and maps each episode to the three-label taxonomy. Figure 5 summarizes the steps: source filtering, context reconstruction, taxonomy mapping, validation safeguards, and repository-disjoint splitting.

We start from the open-source repositories in the source datasets, favoring those for which CI evidence can be reconstructed around a failure: logs, workflow configuration, pull-request or commit metadata, and relevant historical signals. Source provenance is retained so that later analysis can separate benchmark behavior by origin. For GHALogs Moriconi et al. (2025) and LCA-CI-Builds-Repair Bogomolov et al. (2024), we identify failed CI runs and link them to the triggering repository state, pull request or commit, and workflow metadata. LCA-CI-Builds-Repair contributes repair-oriented episodes with failing logs, passing-build information, workflow and changed files, and repair patches, while GHALogs contributes a broader set of GitHub Actions episodes with logs, pull-request files, rerun information, workflow and dependency files, job metadata, and developer discussions.

IDoFT is treated differently. As a flaky-test-only source Lam et al. (2019; 2020), using it in full would create an easy source-label shortcut, letting a model equate IDoFT with `flaky_test` without reading CI evidence. We therefore include a deterministic, source-stratified, repository-capped subset of 495 IDoFT episodes, which together with five flaky-test episodes from other sources yields 500 examples, matching the largest non-flaky category, `dependency_drift`. The remaining IDoFT-derived flaky episodes are kept as an optional augmentation set and excluded from headline results.

When possible, we also connect a failure to its subsequent handling: passing reruns without code changes, code or test fixes, dependency or workflow updates, flaky-test isolation, discussion threads, and later green runs. These signals separate superficially similar failures, since a failing test followed by a code patch and a green run points to a code regression, whereas the same test passing on rerun points to a flaky or environmental cause.

Labels come from mapping source-specific evidence, repair artifacts, and developer handling into the ActCI taxonomy. Patch type and build outcome classify repair responses such as `modify_code`, `modify_test`, and `update_workflow`; IDoFT metadata and rerun behavior support `flaky_test` and `mark_or_isolate_flaky`; and dependency files, installation logs, version-resolution evidence, and compatible follow-up changes support `dependency_drift` and `pin_or_update_dependency`. Each episode also carries a confidence level: 1,316 labels are

Table 3: Repository-disjoint benchmark splits.

	Train	Development	Test	Total
Episodes	1,057	221	265	**1,543**

Table 4: Context settings for ActCI-Bench evaluation.

Setting	Evidence provided
Log-only	Failed run metadata and failed job logs
Diff-aware	Log-only context plus PR metadata and changed files
Workflow-aware	Diff-aware context plus workflow configuration
Dependency-aware	Workflow-aware context plus dependency files and version-resolution evidence
History-aware	Dependency-aware context plus reruns, nearby runs, base-branch status, and subsequent green runs
Full context	History-aware context plus developer comments, follow-up commits, links, repair patches, and flaky-test metadata

high confidence, 222 medium, and 5 low. This records evidence limitations; the reported aggregate results do not establish sensitivity to label confidence.

2.5 Tasks and Splits

ActCI-Bench provides three task views over the same 1,543 episodes. The views share `episode_id` values so that predictions can be analyzed jointly across the diagnosis chain.

- **Task 1: Actionability Classification.** Given the episode context, predict whether the failure is actionable, non-actionable for the current author, or uncertain.
- **Task 2: Root-cause Diagnosis.** Given the same context, predict the root cause, such as a code regression, flaky test, dependency drift, or workflow misconfiguration.
- **Task 3: Response Selection.** Given the same context, select the appropriate response, such as modifying code, updating a test, changing dependency constraints, updating workflow configuration, rerunning CI, or isolating a flaky test.

The public benchmark uses repository-disjoint train, development, and test splits, shown in Table 3. Repository-disjoint splitting reduces the risk that an agent or model succeeds by memorizing repository-specific workflow names, dependency conventions, test names, or repeated project contexts, a concern shared by repository-level benchmarks such as BugSwarm Tomassi et al. (2019), BugsInPy Widyasari et al. (2020), and SWE-bench Jimenez et al. (2024). There is no repository overlap between train and development, train and test, or development and test.

Decision structure. The shared episode identifier connects actionability, root-cause, and response predictions, so a system can be evaluated on the consistency of its decisions. An agent may identify a flaky failure correctly while still proposing an unnecessary test edit; this behavior is visible only when diagnosis and response are scored together.

3 Experimental Protocol

We evaluate simple lexical CI triage, flaky-test detectors, a repair baseline, and repository-aware agents. Flaky detectors map predicted flakiness to an isolate-flaky response and otherwise use triage. Repository agents map patches or non-patch recommendations to response labels. Each agent predicts actionability, root cause, and response from the same episode; prompts and parsers are fixed using the development split. The held-out test split uses disjoint repositories. Evidence is presented chronologically, with failed-run metadata and log text followed by pull-request files, workflow and dependency configuration, rerun history, and developer handling when available. Macro-F1 is reported separately for actionability, root cause, and response because major and rare categories coexist. False repair rate is the fraction of non-actionable episodes assigned a patch; patch pass rate is measured only where repair is appropriate. Response appropriateness checks the selected action against the episode label. For context ablation, evidence blocks are added in the fixed order

Table 5: Main results.

Baseline	Act. F1	Root F1	Resp. F1	False repair	Patch pass
Rule-based CI Triage	0.49	0.34	0.31	42.0%	19.0%
FlakeDetector	0.58	0.42	0.36	**24.0%**	–
DeFlaker	0.55	0.39	0.33	28.0%	–
iDFlakies	0.53	0.37	0.31	31.0%	–
FlakeFlagger	0.56	0.40	0.34	27.0%	–
Flakify	0.57	0.41	0.35	26.0%	–
LCA CI-Fixing Baseline	0.52	0.46	0.44	**57.0%**	61.0%
SWE-agent	0.65	0.50	0.48	48.0%	64.0%
Agentless	0.67	0.53	0.50	41.0%	62.0%
AutoCodeRover	**0.69**	**0.55**	**0.53**	44.0%	**67.0%**

above. Diagnosis-first variants explicitly predict actionability and cause before response. The context comparison is descriptive: richer settings may contain evidence recorded after the failed run, so these scores do not isolate a prospective information gain. The diagnosis-first comparison reports an association under the stated adapters rather than a causal guarantee for other agents.

Episode inputs and information timing. A failed run is linked to its triggering repository revision and to the evidence preserved by the source dataset. Log-only input approximates the immediate failure signal; changed files add information about the proposed change; workflow and dependency files expose configuration and version resolution. Historical runs can help distinguish a failure repeatedly observed on the base branch from a regression introduced by the pull request. The full-context condition also incorporates discussion, follow-up commits, and repair artifacts. These latter fields can reveal how the failure was resolved; accordingly, the experiment is interpreted as a retrospective diagnostic comparison. We do not equate its full-context score with an agent's ability to act before developers respond.

Response mapping. Flakiness detectors map positive predictions to isolation and otherwise fall back to CI triage; patch-capable agents map proposed edits to the corresponding response type. These adapters support comparison on the same three prediction tasks, while preserving the distinction between specialist detectors and general-purpose agents.

Scoring and interpretation. Macro-F1 gives equal weight to classes, including rare causes. False repair rate counts patch proposals on non-actionable episodes; patch pass rate applies only to repairable episodes. They describe different populations and should be interpreted together rather than subtracted.

4 RESULTS

RQ1: Composition. Non-actionable failures represent 509/1,543 episodes (33.0%); dependency drift and flaky tests are the two largest root-cause groups (500 each). A red run therefore does not imply a code patch.

RQ2: Baselines. Table 5 reports macro-F1 and the false repair rate. AutoCodeRover has the highest actionability, root-cause, and response F1 (0.69, 0.55, 0.53) but a 44.0% false repair rate. FlakeDetector reduces false repairs to 24.0% while handling a narrower range of causes. The repository agents are SWE-agent Yang et al. (2024), Agentless Xia et al. (2024), and AutoCodeRover Zhang et al. (2024).

Where the baselines differ. Figure 6 makes the safety dimension of the main table explicit. The LCA CI-Fixing baseline has the largest false repair rate, 57.0%, whereas the flakiness tools range from 24.0% to 31.0%. The three repository agents fall between these groups, at 41.0% to 48.0%. This pattern is consistent with their task design: a flakiness signal can prevent some unnecessary patches, while repair-oriented systems are more likely to propose a modification. It does not establish that the specialized detectors are better all-purpose CI agents; their actionability, cause, and

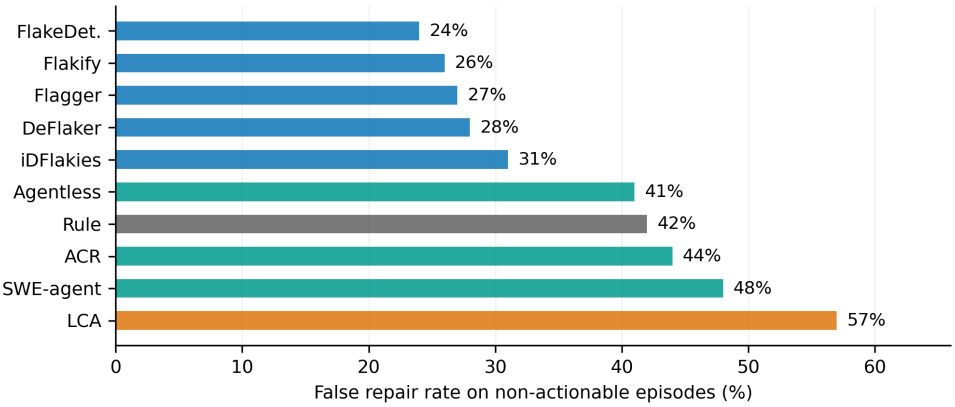


Figure 6: False repair rate among non-actionable episodes for the evaluated baselines. The plot concerns patch proposals, not the success of a patch on repairable cases.

Table 6: Context ablation and diagnosis-first gains.

Context	Act. F1	Root F1	Resp. F1
Log-only	0.56	0.31	0.29
Diff-aware	0.61	0.36	0.34
Workflow-aware	0.63	0.42	0.40
Dependency-aware	0.66	0.49	0.46
History-aware	0.70	0.53	0.50
Full context	**0.71**	**0.55**	**0.53**

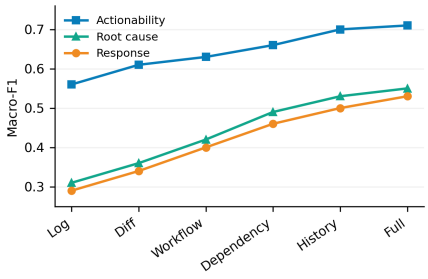


Figure 7: Context ablation.

Diagnosis-first	Direct F1	Diagnosis F1	Direct FR	Diagnosis FR
SWE-agent	0.48	**0.55**	48.0%	**34.0%**
Agentless	0.50	**0.58**	41.0%	**30.0%**
AutoCodeRover	0.53	**0.60**	44.0%	**31.0%**

response macro-F1 scores remain lower than those of the repository agents. The joint view is more informative than ranking systems on patch success alone.

RQ3–RQ4: Context and diagnosis first. In the retrospective context comparison, adding evidence raises average actionability/root-cause/response macro-F1 of the three repository agents from 0.56/0.31/0.29 (logs only) to 0.71/0.55/0.53 (full context). Dependency evidence raises root-cause F1 from 0.42 to 0.49; history raises it to 0.53. In the evaluated workflows, explicit diagnosis before response is associated with higher response F1 for SWE-agent from 0.48 to 0.55, Agentless from 0.50 to 0.58, and AutoCodeRover from 0.53 to 0.60, while lowering their false repair rates from 48/41/44% to 34/30/31%.

Reading the context ablation. The log-only setting gives an agent the failed job and its immediate diagnostic text, but similar stack traces may arise from distinct causes. Changed files and workflow configuration add evidence about what the pull request modified and how CI was invoked. Dependency records help identify installation and resolution failures; reruns and neighboring builds reveal whether symptoms recur without a new repository change. These evidence sources plausibly explain why root-cause macro-F1 rises from 0.31 with logs alone to 0.49 after adding dependency context and 0.53 after adding history. The last step, from history-aware to full context, is smaller in the reported averages, but its interpretation is different: the full bundle can contain information produced after the decision point. Thus the plotted progression is a comparison of available information in the retrospective dataset, not a controlled account of what an online agent could know at every stage. The benchmark is most useful when each score is reported with its precise evidence bound-

ary. Likewise, the diagnosis-first comparison changes the order in which predictions are elicited from the agents. Its lower false repair rates support a practical hypothesis that explicit diagnosis can encourage restraint, but the present comparison alone does not show that the same gains hold for every model, repository, or operational setting.

RQ5: Patch-only evaluation. On repairable episodes, patch pass rates for LCA CI-Fixing, SWE-agent, Agentless, and AutoCodeRover are 61/64/62/67%, respectively; full response F1 scores are 0.44/0.48/0.50/0.53. The metrics have different denominators and should not be interpreted as a numerical performance gap. Their joint reporting distinguishes patch success on repairable cases from decision quality across all cases; the LCA repair baseline falsely patches 57.0% of non-actionable episodes.

5 Related Work and Limitations

Specialized flaky-test detectors Lam et al. (2019); Bell et al. (2018) target one source of CI noise; repository repair benchmarks Jimenez et al. (2024); Tomassi et al. (2019) assess known patch tasks. ActCI-Bench tests whether, why, and how to act before editing. Public CI histories can have missing evidence and noisy inferred labels. GitHub Actions dominates the sources; environmental and external-service categories are small, and reruns or escalation cannot always be replayed offline. Appendix A.1 gives additional protocol details and threats. The case-specific evidence retained by the benchmark also makes it possible to inspect disagreement between an agent's actionability, cause, and response predictions. Such disagreement matters even when a patch passes CI, since passing tests alone cannot establish that the agent selected the appropriate maintenance action for the original failure.

Actionability, diagnosis, and response scores reveal different agent behaviors. A specialist can avoid unnecessary patches on flaky failures yet miss other causes; a repository agent can repair more cases while over-patching non-actionable episodes. CI recovery evaluates only executable repair cases and cannot establish that a patch was necessary. The metrics should therefore be read together, with the stated evidence boundary of each context condition. In particular, the retrospective full-context setting may include later developer handling, so its result is an evidence-rich diagnostic score rather than an estimate of immediate autonomous response quality.

6 Conclusion

ActCI-Bench evaluates the complete CI maintenance decision. In the reported evaluation, repository agents achieve the highest structured-task scores while frequently proposing patches on non-actionable failures. Retrospective evidence and diagnosis-first prompting are associated with stronger response selection; prospective performance remains outside the scope of these measurements.

Reproducibility Statement

The main text and appendix specify the episode schema, labels, splits, baseline adapters, and evaluation metrics. An anonymous replication package is linked for review.

AI Use Statement

Generative AI tools were used for language polishing and for adapting the manuscript's LaTeX formatting to the ICLR 2027 submission template. The authors are responsible for the final content of the paper.

REFERENCES

Abdulrahman Alshammari, Christopher Morris, Michael Hilton, and Jonathan Bell. Flakeflagger: Predicting flakiness without rerunning tests. In *43rd IEEE/ACM International Conference on Software Engineering: Companion Proceedings, ICSE Companion 2021, Madrid, Spain, May 25-28, 2021*, pp. 187. IEEE, 2021. doi: 10.1109/ICSE-COMPANION52605.2021.00081. URL https://doi.org/10.1109/ICSE-Companion52605.2021.00081.

Jonathan Bell, Owolabi Legunsen, Michael Hilton, Lamyaa Eloussi, Tifany Yung, and Darko Marinov. Deflaker: automatically detecting flaky tests. In Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (eds.), *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pp. 433–444. ACM, 2018. doi: 10.1145/3180155.3180164. URL https://doi.org/10.1145/3180155.3180164.

Moritz Beller, Georgios Gousios, and Andy Zaidman. Oops, my tests broke the build: an explorative analysis of travis CI with github. In Jesús M. González-Barahona, Abram Hindle, and Lin Tan (eds.), *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, pp. 356–367. IEEE Computer Society, 2017. doi: 10.1109/MSR.2017.62. URL https://doi.org/10.1109/MSR.2017.62.

Berkay Berabi, Jingxuan He, Veselin Raychev, and Martin T. Vechev. Tfix: Learning to fix coding errors with a text-to-text transformer. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 780–791. PMLR, 2021. URL http://proceedings.mlr.press/v139/berabi21a.html.

Egor Bogomolov, Aleksandra Eliseeva, Timur Galimzyanov, Evgeniy Glukhov, Anton Shapkin, Maria Tigina, Yaroslav Golubev, Alexander Kovrigin, Arie van Deursen, Maliheh Izadi, and Timofey Bryksin. Long code arena: a set of benchmarks for long-context code models. *CoRR*, abs/2406.11612, 2024. doi: 10.48550/ARXIV.2406.11612. URL https://doi.org/10.48550/arXiv.2406.11612.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Sakina Fatima, Taher Ahmed Ghaleb, and Lionel C. Briand. Flakify: A black-box, language model-based predictor for flaky tests. *IEEE Trans. Software Eng.*, 49(4):1912–1927, 2023. doi: 10.1109/TSE.2022.3201209. URL https://doi.org/10.1109/TSE.2022.3201209.

Yonghui Andie Huang, Daniel Alencar da Costa, Grant Dick, Mariam El Mezouar, and Liwen Xiao. Is this build failure related to my patch? an empirical study of unrelated build failures in continuous integration. *Empir. Softw. Eng.*, 31(6):148, 2026. doi: 10.1007/S10664-026-10874-8. URL https://doi.org/10.1007/s10664-026-10874-8.

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. *CoRR*, abs/2403.07974, 2024. doi: 10.48550/ARXIV.2403.07974. URL https://doi.org/10.48550/arXiv.2403.07974.

Nan Jiang, Thibaud Lutellier, and Lin Tan. CURE: code-aware neural machine translation for automatic program repair. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pp. 1161–1173. IEEE, 2021. doi: 10.

1109/ICSE43902.2021.00107. URL https://doi.org/10.1109/ICSE43902.2021.00107.

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, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

René Just, Darioush Jalali, and Michael D. Ernst. Defects4j: a database of existing faults to enable controlled testing studies for java programs. In Corina S. Pasareanu and Darko Marinov (eds.), *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*, pp. 437–440. ACM, 2014. doi: 10.1145/2610384.2628055. URL https://doi.org/10.1145/2610384.2628055.

Wing Lam, Reed Oei, August Shi, Darko Marinov, and Tao Xie. idflakies: A framework for detecting and partially classifying flaky tests. In *12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019*, pp. 312–322. IEEE, 2019. doi: 10.1109/ICST.2019.00038. URL https://doi.org/10.1109/ICST.2019.00038.

Wing Lam, Kivanç Muslu, Hitesh Sajnani, and Suresh Thummalapenta. A study on the lifecycle of flaky tests. In Gregg Rothermel and Doo-Hwan Bae (eds.), *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pp. 1471–1482. ACM, 2020. doi: 10.1145/3377811.3381749. URL https://doi.org/10.1145/3377811.3381749.

Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. Genprog: A generic method for automatic software repair. *IEEE Trans. Software Eng.*, 38(1):54–72, 2012. doi: 10.1109/TSE.2011.104. URL https://doi.org/10.1109/TSE.2011.104.

Claire Le Goues, Neal J. Holtschulte, Edward K. Smith, Yuriy Brun, Premkumar T. Devanbu, Stephanie Forrest, and Westley Weimer. The manybugs and introclass benchmarks for automated repair of C programs. *IEEE Trans. Software Eng.*, 41(12):1236–1256, 2015. doi: 10.1109/TSE.2015.2454513. URL https://doi.org/10.1109/TSE.2015.2454513.

Kui Liu, Anil Koyuncu, Dongsun Kim, and Tegawendé F. Bissyandé. Tbar: revisiting template-based automated program repair. In Dongmei Zhang and Anders Møller (eds.), *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*, pp. 31–42. ACM, 2019. doi: 10.1145/3293882.3330577. URL https://doi.org/10.1145/3293882.3330577.

Tianyang Liu, Canwen Xu, and Julian J. McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *CoRR*, abs/2306.03091, 2023. doi: 10.48550/ARXIV.2306.03091. URL https://doi.org/10.48550/arXiv.2306.03091.

Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. An empirical analysis of flaky tests. In Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey (eds.), *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pp. 643–653. ACM, 2014. doi: 10.1145/2635868.2635920. URL https://doi.org/10.1145/2635868.2635920.

Pooya Rostami Mazrae, Alexandre Decan, Tom Mens, and Mairieli Wessel. An empirical study of the evolution of github actions workflows. *J. Syst. Softw.*, 236:112824, 2026. doi: 10.1016/J.JSS.2026.112824. URL https://doi.org/10.1016/j.jss.2026.112824.

Florent Moriconi, Thomas Durieux, Jean-Rémy Falleri, Raphaël Troncy, and Aurélien Francillon. Ghalogs: Large-scale dataset of github actions runs. In *22nd IEEE/ACM International Conference on Mining Software Repositories, MSR@ICSE 2025, Ottawa, ON, Canada, April 28-29, 2025*, pp. 669–673. IEEE, 2025. doi: 10.1109/MSR66628.2025.00104. URL https://doi.org/10.1109/MSR66628.2025.00104.

Rabeya Khatun Muna, Md Nakhla Rafi, and Tse-Hsun Chen. Ci-repair-bench: A repository-aware benchmark for automated patch validation via CI workflows. *CoRR*, abs/2604.27148, 2026. doi: 10.48550/ARXIV.2604.27148. URL https://doi.org/10.48550/arXiv.2604.27148.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *CoRR*, abs/2412.21139, 2024. doi: 10.48550/ARXIV.2412.21139. URL https://doi.org/10.48550/arXiv.2412.21139.

Owain Parry, Gregory M. Kapfhammer, Michael Hilton, and Phil McMinn. A survey of flaky tests. *ACM Trans. Softw. Eng. Methodol.*, 31(1):17:1–17:74, 2022. doi: 10.1145/3476105. URL https://doi.org/10.1145/3476105.

Shanto Rahman, Bala Naren Chanumolu, Suzzana Rafi, August Shi, and Wing Lam. Ranking relevant tests for order-dependent flaky tests. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*, pp. 1999–2011. IEEE, 2025. doi: 10.1109/ICSE55347.2025.00178. URL https://doi.org/10.1109/ICSE55347.2025.00178.

Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. In Jesús M. González-Barahona, Abram Hindle, and Lin Tan (eds.), *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, pp. 345–355. IEEE Computer Society, 2017. doi: 10.1109/MSR.2017.54. URL https://doi.org/10.1109/MSR.2017.54.

Nuno Saavedra, André Silva, and Martin Monperrus. Gitbug-actions: Building reproducible bug-fix benchmarks with github actions. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, ICSE Companion 2024, Lisbon, Portugal, April 14-20, 2024*, pp. 1–5. ACM, 2024. doi: 10.1145/3639478.3640023. URL https://doi.org/10.1145/3639478.3640023.

August Shi, Wing Lam, Reed Oei, Tao Xie, and Darko Marinov. ifixflakies: a framework for automatically fixing order-dependent flaky tests. In Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (eds.), *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pp. 545–555. ACM, 2019. doi: 10.1145/3338906.3338925. URL https://doi.org/10.1145/3338906.3338925.

David A. Tomassi, Naji Dmeiri, Yichen Wang, Antara Bhowmick, Yen-Chuan Liu, Premkumar T. Devanbu, Bogdan Vasilescu, and Cindy Rubio-González. Bugswarm: mining and continuously growing a dataset of reproducible failures and fixes. In Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (eds.), *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pp. 339–349. IEEE / ACM, 2019. doi: 10.1109/ICSE.2019.00048. URL https://doi.org/10.1109/ICSE.2019.00048.

Ratnadira Widyasari, Sheng Qin Sim, Camellia Lok, Haodi Qi, Jack Phan, Qijin Tay, Constance Tan, Fiona Wee, Jodie Ethelda Tan, Yuheng Yieh, Brian Goh, Ferdian Thung, Hong Jin Kang, Thong Hoang, David Lo, and Eng Lieh Ouh. Bugsinpy: a database of existing bugs in python programs to enable controlled testing and debugging studies. In Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (eds.), *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pp. 1556–1560. ACM, 2020. doi: 10.1145/3368089.3417943. URL https://doi.org/10.1145/3368089.3417943.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. *CoRR*, abs/2407.01489, 2024. doi: 10.48550/ARXIV.2407.01489. URL https://doi.org/10.48550/arXiv.2407.01489.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated

software engineering. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida Wang, and Ofir Press. Swe-bench multimodal: Do AI systems generalize to visual software domains? In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=riTiq3i21b.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diago, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/8b86cf5ace600c48fd188efbb8dedec8-Abstract-Datasets_and_Benchmarks_Track.html.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In Maria Christakis and Michael Pradel (eds.), *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, pp. 1592–1604. ACM, 2024. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

Lianyu Zheng, Shuang Li, Xi Huang, Jiangnan Huang, Bin Lin, Jinfu Chen, and Jifeng Xuan. Why do github actions workflows fail? an empirical study. *ACM Trans. Softw. Eng. Methodol.*, 35(5):139:1–139:29, 2026. doi: 10.1145/3749371. URL https://doi.org/10.1145/3749371.

Hao-Nan Zhu, Kevin Z. Guan, Robert M. Furth, and Cindy Rubio-González. Actionsremaker: Reproducing GITHUB actions. In *45th IEEE/ACM International Conference on Software Engineering: ICSE 2023 Companion Proceedings, Melbourne, Australia, May 14-20, 2023*, pp. 11–15. IEEE, 2023. doi: 10.1109/ICSE-COMPANION58688.2023.00015. URL https://doi.org/10.1109/ICSE-Companion58688.2023.00015.

Celal Ziftci and Jim Reardon. Who broke the build? automatically identifying changes that induce test failures in continuous integration at google scale. In *39th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017, Buenos Aires, Argentina, May 20-28, 2017*, pp. 113–122. IEEE Computer Society, 2017. doi: 10.1109/ICSE-SEIP.2017.13. URL https://doi.org/10.1109/ICSE-SEIP.2017.13.

A Additional Details

A.1 Benchmark Characteristics

ActCI-Bench has four characteristics that distinguish it from patch-only CI repair datasets. First, it separates the decision to act from the act of producing a repair, exposing false-actionable errors where an agent tries to fix a failure that should have been rerun, isolated, escalated, or otherwise treated as non-actionable for the current developer. Second, it is built from real CI histories and preserves concrete artifacts such as logs, workflow files, dependency metadata, historical runs, rerun outcomes, discussion, and follow-up actions. Third, it deliberately includes both repairable failures and failures for which editing code is inappropriate, enabling measurement of over-repair on flaky,

dependency-related, environmental, workflow-related, or external-service failures. Finally, because each episode includes actionability, root-cause, and response labels, the benchmark supports both stage-wise and end-to-end evaluation. An agent may identify that a failure is actionable and choose the wrong response, or diagnose flakiness correctly and still modify a test unnecessarily; this multi-level structure supports the experimental analyses in Section 3.

B Motivating Examples

We motivate CI failure handling as a decision task beyond patch generation using three episodes that share one surface symptom, a red CI run, yet require different judgments about actionability, root cause, and response. Although simplified, they reflect the failure modes and evidence types captured by ActCI-Bench.

B.1 Example 1: Code Regression

Consider a pull request that refactors input validation in a library module. CI fails in a unit test on the modified path, and the log shows an invalid input now being accepted. The failing test is in the changed package, the stack trace enters a modified function, a rerun reproduces the failure, and recent base-branch runs are green.

This resembles the repairable bugs targeted by APR benchmarks such as Defects4J Just et al. (2014) and ManyBugs Le Goues et al. (2015). The red CI signal is actionable by the pull-request author, and the most plausible root cause is a `code_regression`: the change altered behavior and violated an existing test. The right response is to fix the implementation while leaving the workflow and test suite untouched. A patch-only benchmark handles this case well, since the expected action is a repository-side code fix.

The example also shows why evidence matters: actionability follows from combining the log, stack trace, diff, rerun behavior, and base-branch history. Without that connection, the same assertion failure could pass for a stale test expectation, a dependency-induced behavior change, or a flaky test.

B.2 Example 2: Flaky Test

Now consider a pull request that touches only documentation and a small formatting script, while CI fails in an asynchronous integration test that times out waiting for a service. The diff is unrelated to the failing path, nearby pull requests show the same test failing intermittently, a rerun passes without any code change, and developer comments note recent instability.

This is characteristic of flaky tests, a well-studied source of CI noise Luo et al. (2014); Parry et al. (2022); Lam et al. (2020). Unlike the previous case, the failure is a non-actionable `flaky_test` episode for the current pull request, and an appropriate response is to rerun CI, isolate the flaky test, or route it to the test owners. Modifying application code would be speculative and could hide the real problem.

Such episodes expose a weakness of repair-only evaluation. Rewarded only for turning the workflow green, an agent may weaken the timeout, delete an assertion, or otherwise edit the test until it passes, improving the immediate CI status while degrading the project's test signal. A realistic benchmark must therefore separate successful mitigation from unnecessary or harmful repair.

B.3 Example 3: Dependency Drift

As a third example, consider a repository whose dependency file allows a broad version range for a third-party package. A pull request changes an unrelated module, but CI resolves a newer dependency version than the last green runs, and the job fails during test collection with an import error from a removed API. The diff is unrelated, while the dependency metadata and job log show that the resolved version changed between the last green run and the failure.

This is neither a code regression nor a flaky test. The root cause is `dependency_drift`: the constraints no longer select a compatible environment. Depending on project policy, the response

may pin the dependency, update a lock file, adapt a compatibility layer, or coordinate an upgrade. In each case, it targets dependency management, not the application logic the pull request changed.

Dependency and workflow failures are common in GitHub Actions pipelines Zheng et al. (2026); Mazrae et al. (2026). Dependency drift is especially hard when an agent sees only the final error message, since an import or type error looks like an ordinary bug without installation logs, dependency files, version constraints, and historical runs. This motivates supplying dependency evidence and testing whether agents select `pin_or_update_dependency`.

Figure 2 summarizes the evidence chains behind the three examples, showing how one red CI signal can demand different diagnosis-response tuples.

B.4 Implications for CI Failure Evaluation

These examples imply three requirements for evaluating CI maintenance agents.

First, a red CI run is an incomplete task specification. The same status can mean a code regression, flaky test, dependency drift, workflow misconfiguration, environment failure, or external-service problem Rausch et al. (2017); Beller et al. (2017); Zheng et al. (2026). Treating every failure as a patchable bug collapses these into one repair task and skips the developer's first decision: whether the current change should be touched at all.

Second, actionability depends on contextual evidence. CI logs are necessary but rarely sufficient. Diffs, workflow files, dependency metadata, rerun outcomes, nearby runs, base-branch status, discussions, and follow-up commits can all change the diagnosis. A timeout signals flakiness once paired with rerun success and prior intermittent failures; an import error signals dependency drift only when version-resolution evidence shows the environment changed.

Third, the desired output is a response, not always a patch. A CI assistant should choose among modifying code, updating tests, pinning or updating dependencies, editing workflow configuration, rerunning CI, rebasing, isolating flaky tests, escalating infrastructure issues, or taking no action. Scoring only whether generated patches pass, as in repair-only benchmarks Tomassi et al. (2019); Jimenez et al. (2024); Muna et al. (2026), neither penalizes over-repair on non-actionable failures nor rewards correct non-patch mitigations.

ActCI-Bench follows these requirements. For each episode, an agent receives an evidence bundle E with the failed run, logs, pull-request or commit context, workflow and dependency artifacts, and any historical or developer signals. It predicts three labels: y_a for actionability, y_c for root cause, and y_r for response. A correct solution is a decision tuple (y_a, y_c, y_r) matching the project-grounded handling of the episode, and patch generation is required only when y_r calls for a code or test change. The general objective is the project-grounded decision.

C Threats to Validity

ActCI-Bench faces several validity threats. For construct validity, actionability is inherently context-dependent: whether a CI failure should be handled by the current pull-request author may vary across project policies, and root-cause or response labels necessarily simplify mixed failures and project-specific handling choices. We mitigate this by grounding labels in developer actions, reruns, follow-up commits, repair patches, later CI outcomes, and preserved evidence, while treating response selection as a decision task alongside full repository-specific validation. For internal validity, labels inferred from weak supervision, source metadata, repair patches, rerun behavior, or developer follow-up may be noisy, as in prior CI and bug-fix datasets Tomassi et al. (2019); Saavedra et al. (2024); public CI evidence may also be incomplete, with truncated logs, changed workflow files, missing discussions, or unrecoverable runs, a challenge also noted by GitHub Actions reconstruction tools. We therefore record evidence availability, confidence levels, and context-ablation settings, and keep baseline adapters fixed, since flaky-test detectors and repair systems were originally designed for different objectives than actionability, root-cause, and response prediction. Some full-context evidence, including developer follow-up and repair patches, occurs after the failure and can leak outcome information into prediction. Consequently, the full-context ablation is a retrospective diagnostic comparison and does not quantify prospective deployment performance. Source-specific artifacts may also allow models to infer labels without substantive diagnosis, particularly

because most IDoFT episodes are flaky; repository-disjoint splitting alone does not eliminate this risk. For external validity, ActCI-Bench is built from GHALogs, LCA-CI-Builds-Repair, and a controlled IDoFT subset, which may overrepresent GitHub Actions, open-source projects, recoverable histories, and certain languages; results may not fully generalize to private CI, monorepos, proprietary infrastructures, or organizations with different policies. Some categories, such as environment and external-service failures, remain rare, and the downsampled IDoFT subset avoids a flaky-source shortcut while covering only part of flaky-test benchmarking. Although the same actionability concepts apply beyond GitHub Actions, platform-specific conventions in Travis CI and GitHub Actions influence observed failures Beller et al. (2017); Zheng et al. (2026); Mazrae et al. (2026), so broader CI-platform coverage remains future work. For conclusion validity, accuracy and macro-F1 do not capture the asymmetric costs of false actionable and false non-actionable decisions; we therefore report false repair rate, response appropriateness, patch quality, per-class metrics, and recommend paired bootstrap intervals, especially for small classes. Some responses, such as reruns, waiting for external-service recovery, or infrastructure escalation, cannot be fully replayed offline, and LLM-agent results may vary with model versions, prompts, decoding, tools, and context budgets, as also observed in code and agent benchmarks Chen et al. (2021); Jain et al. (2024); Pan et al. (2024); Yang et al. (2025b). Finally, benchmark users should guard against shortcut learning and taxonomy overfitting: models may exploit source-specific artifacts, as in broader repository-level benchmark concerns Tomassi et al. (2019); Liu et al. (2023); Jimenez et al. (2024), and non-actionable means avoiding speculative modification of the current change when rerun, flaky-test isolation, dependency maintenance, infrastructure escalation, or external recovery is the more appropriate response.

D Related Work

D.1 CI Failure Diagnosis and Repair

Empirical studies show that CI failures stem from diverse causes, including compilation errors, test failures, dependency issues, configuration problems, and execution-environment instability Rausch et al. (2017); Beller et al. (2017); Zheng et al. (2026). Studies of GitHub Actions workflow evolution and log datasets such as GHALogs Moriconi et al. (2025) further indicate that relevant evidence is distributed across logs, workflow files, and repository history, so a failed CI run provides a broader diagnostic signal than source-code defect evidence alone. Existing techniques support parts of this process: Culprit Finder Ziftci & Reardon (2017) ranks likely responsible changes, Huang *et al.* Huang et al. (2026) study whether build failures are attributable to the triggering patch, BART provides build repair hints, and ActionsRemaker Zhu et al. (2023), GitBug-Actions Saavedra et al. (2024), and CI-Repair-Bench Muna et al. (2026) support reproduction or repair-oriented evaluation of CI failures. ActCI-Bench treats CI maintenance as a decision problem: the correct response may be a code patch, dependency update, workflow change, rerun, infrastructure escalation, or a justified decision not to modify the repository.

D.2 Flaky Test Detection and Test Failure Triage

Flaky tests produce inconsistent outcomes without relevant code changes and are a major source of CI noise, reduced trust, and unnecessary debugging effort Luo et al. (2014); Parry et al. (2022). Prior work detects, reproduces, ranks, or repairs flaky tests using coverage information, repeated or order-dependent executions, language-model features, and test-outcome analysis Bell et al. (2018); Lam et al. (2020; 2019); Shi et al. (2019); Alshammari et al. (2021); Fatima et al. (2023); Rahman et al. (2025); related CI triage systems such as BuildSheriff group failures by likely shared root causes. ActCI-Bench incorporates flakiness as one root-cause category among code regressions, dependency drift, workflow misconfiguration, environment failures, and external-service issues, and evaluates the downstream handling decision. Thus, it tests whether an agent avoids over-repairing noisy failures and chooses the correct response after recognizing flakiness.

D.3 Automated Program Repair

Automated Program Repair (APR) synthesizes patches to restore correctness. GenProg Le Goues et al. (2012), ManyBugs Le Goues et al. (2015), and Defects4J Just et al. (2014) established test-suite-based repair evaluation; template-based and neural approaches include TBar Liu et al. (2019),

CURE Jiang et al. (2021), and TFix Berabi et al. (2021). APR typically assumes a defect requiring a patch, whereas CI may fail because of dependency resolution, runner instability, external services, or flaky tests. ActCI-Bench also measures whether a repository modification is justified.

D.4 Repository-Level Software Engineering Benchmarks

Repository-level benchmarks provide realistic artifacts for testing, debugging, and repair. Defects4J Just et al. (2014), BugsInPy Widyasari et al. (2020), ManyBugs Le Goues et al. (2015), and BugSwarm Tomassi et al. (2019) collect failures, fixes, and test infrastructure. SWE-bench Jimenez et al. (2024), SWE-bench Multimodal Yang et al. (2025a), RepoBench Liu et al. (2023), and Long Code Arena Bogomolov et al. (2024) extend evaluation to issue resolution, visual tasks, repository completion, and long-context code reasoning. They usually begin from a defined repair task. ActCI-Bench begins with a CI failure and includes cases where the appropriate response does not involve patching.