CI-Bench: A Framework for Evaluating Large Language Model Tools on CI Failures

Raian Latif Nabil*
University of California, Davis
United States of America
mnabil@ucdavis.edu

Hao-Nan Zhu*
University of California, Davis
United States of America
hnzhu@ucdavis.edu

Cindy Rubio-González
University of California, Davis
United States of America
crubio@ucdavis.edu

Abstract

Large language models (LLMs) have demonstrated their potential in performing complex software engineering (SE) tasks. Rigorous evaluation of LLMs and LLM-based tools requires massive, up-to-date data derived from real-world SE processes. Existing continuous integration & delivery (CI/CD) datasets, such as BugSwarm, provide evolving data mined from software build processes, including complete context of build failures. To harness the CI/CD data, we propose CI-Bench, a unified benchmarking framework designed to evaluate LLM-based program repair tools on software failures from CI/CD processes. CI-Bench retrieves data from the BugSwarm dataset, parses the build logs, and constructs appropriate prompts before invoking LLM-based program repair tools. Additionally, CI-Bench includes an executor to facilitate dynamic evaluation within an environment identical to the original build process. With CI-Bench, we evaluate three state-of-the-art LLM-based program repair tools, Agentless, SWE-Agent, and AutoCodeRover, on a code repair task involving 100 real-world CI/CD failures using GPT-4o, Claude-3.5-Sonnet, and Deepseek-V3 as foundation models. Our evaluation shows that Agentless, SWE-Agent, and AutoCodeRover achieve success rates of up to 28%, 35%, and 13%, respectively, in generating correct patches. CI-Bench is available on GitHub at https://github.com/bugswarm/ci-bench, with an accompanying tool demo at https://youtu.be/BM0K-P38MOg.

Keywords

Large Language Models; Benchmarking; Program Repair

ACM Reference Format:
Raian Latif Nabil, Hao-Nan Zhu, and Cindy Rubio-González. 2026. CI-Bench: A Framework for Evaluating Large Language Model Tools on CI Failures. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-Companion '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3774748.3787606

1 Introduction

Large Language Models (LLMs) have revolutionized software engineering (SE) by enabling the automation of complex tasks such as fault detection, test generation, and automated code repair. LLMs and LLM-based tools have transformed how researchers and developers approach SE tasks, allowing them to tackle challenges traditional models struggle with. These tools are designed in various fashions, ranging from fixed, pre-defined steps [7] to agentic behaviors involving multi-step reasoning and planning [5].

However, evaluating the performance of LLMs and LLM-based tools on SE tasks remains a significant challenge. Existing benchmarks like SWE-Bench [4] and its extensions [9] primarily focus on GitHub issues, which do not fully capture the complexities of real-world software development processes. Additionally, these benchmarks do not automatically evolve to include new data, leading to potential data leakage when evaluating LLMs with a later cutoff date than the benchmark release. Furthermore, these benchmarks lack sufficient details to fully replicate the original software failures, which limits their utility for reproducibility studies.

We introduce CI-Bench, a unified framework that enables the evaluation of various LLM-based program repair tools to address the limitations of existing benchmarks. To harness the rich context from the software build process, CI-Bench retrieves software build failures from the BugSwarm [6, 11] dataset, comprising more than 3,500 reproducible CI/CD build failures containing complete context, such as the execution environment and logs that capture build failures (e.g., compilation errors, test failures, linting violations, misconfigurations, and environment-related issues), along with error messages, exceptions, and tracebacks. Such data is crucial for evaluating LLM-based program repair tools in real-world development and deployment scenarios. Furthermore, CI-Bench also leverages the evolving nature of BugSwarm to ensure evaluations include the latest data after an LLM's cutoff date. Lastly, CI-Bench reconstructs the original execution environment of each build failure, ensuring that evaluations faithfully mirror the real-world scenarios encountered by developers.

Given a targeted LLM-based program repair tool and a CI/CD build failure from BugSwarm, CI-Bench parses the build logs to extract information related to the failure. This information is then used to construct prompts for the targeted program repair tool. Additionally, using environmental information from BugSwarm, CI-Bench sets up a containerized environment that mirrors the original build process, allowing the tool under evaluation to run in the identical environment as the original build failure. With this setup, CI-Bench replays the CI/CD build process and automatically determines whether the targeted tool successfully fixes the failure. To demonstrate the utility of CI-Bench, we integrate three state-of-the-art LLM-based program repair tools: Agentless [7], SWE-Agent [8], and AutoCodeRover [10], with various underlying foundation models. However, CI-Bench is not limited to these; it can be extended to support any program repair tool that can process build failures.

We evaluate the three integrated LLM-based program repair tools (Agentless, SWE-Agent, and AutoCodeRover) on 100 real-world Java build failures from the BugSwarm dataset (63 projects), measuring performance via plausibility and syntactic equivalence. For each tool, we assess the effectiveness of several foundation models, specifically GPT-4o, Claude-3.5-Sonnet, and Deepseek-V3. Our results demonstrate that Agentless and SWE-Agent significantly outperform AutoCodeRover, achieving average plausibility

*Both authors contributed equally.

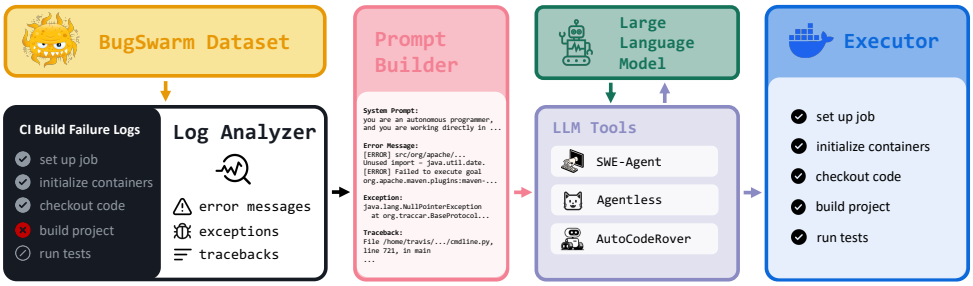

Figure 1: Components of CI-Bench

success rates of 28% and 30% respectively, compared to 11.3% for AutoCodeRover; peak performance for these tools reached 32%, 36%, and 13%. Manual verification against developer-provided fixes finds that Agentless, SWE-Agent, and AutoCodeRover produce on average correct patches in 25.3%, 28.3%, and 11.3% of cases, respectively. Notably, while AutoCodeRover had a lower success rate, all of its generated plausible patches were validated as correct.

With CI-Bench, researchers and developers can more easily evaluate LLM-based program repair tools using any subset of CI/CD build failures from the BugSwarm dataset and obtain performance metrics. CI-Bench's extensible design allows users to integrate new tools, models, and performance metrics seamlessly. The remainder of this paper presents the technical details of the framework and a more detailed discussion of our evaluation results.

2 Overview of CI-Bench

CI-Bench consists of several key components as shown in Figure 1. Our framework leverages BugSwarm, as it remains the largest and most diverse dataset of reproducible CI failures available to date. The *Log Analyzer* parses the historical CI failure logs provided by BugSwarm to identify key diagnostic information, such as error messages. The *Prompt Builder* transforms the extracted information into a generic prompt for the LLM-based program repair tools. These tools then incorporate this prompt into their system prompts and interact with the LLM to generate candidate patches. Finally, the *Executor* validates each patch by applying it to the failing program using the original execution environment, which is facilitated by BugSwarm via Docker containers and the necessary build scripts. We further describe each component along with a running example using the BugSwarm artifact `tananaev-traccar-64783123`.

The BugSwarm Dataset. CI-Bench enables LLM-based program repair tools to be evaluated on existing CI datasets. In this paper, we select BugSwarm, which is an established and maintained dataset built by mining consecutive failed-pass job pairs from two popular CI/CD services, GitHub Actions and Travis-CI. Each BugSwarm artifact consists of a Docker image that includes the source code to reproduce the failure and the corresponding fix, scripts to build the code, and historical build logs from the original CI runs. A BugSwarm artifact also encapsulates critical metadata such as the failed and passed commits and information about the CI system. Therefore, if artifacts from sources other than BugSwarm are to be used, they must provide equivalent metadata so that the framework can seamlessly evaluate tools against them. The BugSwarm dataset supports both Java and Python. Currently, it has 3,598 active artifacts; 2,245 are Java artifacts, and the rest are Python. In the rest of this section we will consider the BugSwarm artifact `tananaev-traccar-64783123`. This artifact is mined from a Java GitHub repository named `traccar`, where a Null Pointer Exception occurs. We use this artifact to showcase our evaluation of the SWE-Agent tool.

Log Analyzer. CI-Bench's first component is the *Log Analyzer*, which parses the failed build log of the artifact to extract error messages, exceptions, and tracebacks. This information is important because it usually includes the location of the failure such as file and line number, class name and method name. For our example, artifact `tananaev-traccar-64783123`, the Log Analyzer processes the failed build log and extracts various error messages, the Null Pointer Exception, and its tracebacks that indicate the location of the failure. This information serves as input to the *Prompt Builder*.

Prompt Builder. The *Prompt Builder* creates a *generic prompt* from the error messages, exceptions, and/or tracebacks provided by the Log Analyzer to guide LLMs in code repair tasks. This prompt is the same for all the considered tools, and it also includes metadata such as the artifact name, its programming language, and its CI build system, which are essential for guiding the execution pipelines of certain tools. By design, the generic prompt remains simple yet effective, functioning as a structured problem description which each tool tries to repair. Each repair tool has its own *system prompt*: a predefined instruction that sets the overall task and its behavior. The generic prompt is inserted within each tool's system prompt. The tools are originally designed to fix bugs using bug reports, not build logs, thus the need for our own *Prompt Builder*.

Integrated LLM-Based Repair Tools. CI-Bench is a unified framework that can plug in specific LLM program repair tools for benchmarking. To use the CI-Bench framework, tools must meet the following requirements: (1) provide support for Java code repair, (2) be capable of processing sequential text input (required to process CI/CD build logs), and (3) generate patches. Currently, CI-Bench has integrated three state-of-the-art LLM-based program repair tools, Agentless [7], SWE-Agent [8], and AutoCodeRover [10]. While these tools were originally designed for Python, we adapted them for Java programs. We also modified the tools to interact with BugSwarm containers instead of relying on GitHub issue threads or source repositories. For example,

```
protected Object decode( /* ... */ ) {
  /* ... */
  } else if (type == MSG_LOGIN) {
-   if (channel == null) {
+   if (channel != null) {
      /* ... */
      response.writeByte(0x0A);
      channel.write(response, remoteAddress);
    /* ... */
    }
  /* ... */
```

(a) Patch written by the developer.

```
protected Object decode( /* ... */ ) {
  /* ... */
  } else if (type == MSG_LOGIN) {
    if (channel == null) {
      /* ... */
      response.writeByte(0x0A);
-     channel.write(response, remoteAddress);
+     if (channel != null) {
+       channel.write(response, remoteAddress);
+     }
    /* ... */
```

(b) Patch generated by SWE-Agent.

Figure 2: Patches for `tananaev-traccar-64783123`.

we adapted the parsing and filtering logic to handle Java files and functions, and modified prompt templates to reflect Java syntax.

Executor. The *Executor* validates the patches generated by the program repair tools. It accepts a patch file as input and applies it within a Docker container specifically configured with the artifact's original dependencies. The *Executor* then replicates the build process by executing the exact commands from the original failed GitHub Actions job—typically involving a full project build and the execution of associated unit tests. A patch is deemed plausible if the repository builds successfully and all commands execute without error. In our running example, the patch generated by SWE-Agent (Figure 2b) results in a successful build that passes all tests, and thus is considered plausible. However, this patch is not syntactically equivalent to the ground-truth fix provided by the developer (Figure 2a). We provide a detailed analysis of tool performance across various metrics in the next section.

3 Experimental Evaluation

This evaluation aims to determine how the LLM-based program repair tools perform on the CI failures. We vary the foundation models within the LLM-based program repair tools and assess the performance of each tool using different correctness metrics. To the best of our knowledge, no prior work evaluates LLM-based repair tools on CI-failure data; thus, we report cross-tool and cross-model comparisons within CI-Bench as reference points.

Artifact and Model Selection. To select BugSwarm artifacts for this evaluation, we adopt the issue selection criteria established by SWE-bench Lite [3]: (1) patches that do not include images, external hyperlinks, or references to specific commit SHAs and references to other pull requests or issues; (2) patches that contain one file edit; (3) patches with three or fewer edit hunks; (4) patches that do not create or remove files; (5) patches that are not for test files. From those artifacts, we selected 100 from 63 projects. This selection prioritized real-world patches that involved the smallest code changes, followed by those with the fewest edit hunks. This choice ensures that the evaluation focuses on patches with minimal edits, enabling a clearer assessment of the tool's ability to repair issues that human developers typically resolve with small, localized modifications to the source code. However, CI-Bench is not constrained by the nature and/or complexity of the required fixes. CI-Bench supports evaluating tools on the full BugSwarm corpus, though we selected only 100 artifacts here for a tractable evaluation. We chose three foundation models that have recently performed better on code generation tasks: GPT-4o, Claude-3.5-Sonnet, and Deepseek-V3.

Correctness Metrics. We use two metrics to measure correctness: plausibility (PL) and syntactic equivalence (SYE). Plausibility is the measure of patches that result in a successful build of a BugSwarm artifact. Syntactic equivalence refers to the token-level equivalence between the generated patch and the developer patch. CI-Bench applies the generated patch and uses ANTLR [2] to tokenize the patched source for comparison with the developer-patched file. Furthermore, we additionally performed a manual inspection comparing the suggested patches against developer-provided patches.

Evaluation Results. As shown in Table 1, we observe that all three tools are able to generate patches with all three foundation models. Agentless and SWE-Agent generate more patches than AutoCodeRover in general, with up to 62 and 67 patches generated, respectively. Our investigation shows that AutoCodeRover generates fewer patches due to the non-deterministic behavior of its patch agent, which sometimes fails to locate the exact code region for applying the patch, even when the patch is successfully generated by the foundational model. We have also investigated the reasons why Agentless and SWE-Agent do not generate patches for the remaining artifacts. For Agentless, the main reason is that it fails to localize the bug location correctly, which is an essential step before generating a patch. And SWE-Agent fails to generate patches due to the cost limit of the foundational model, which prevents it from generating a patch for every artifact.

Not all generated patches are plausible (PL) to make a successful build. SWE-Agent generates up to 36 plausible patches across the three models, while Agentless achieves a comparable number (up to 32), indicating their ability to effectively fix the issues that block the build process. On the other hand, AutoCodeRover generates up to 13 plausible patches, which is lower than the other two tools.

The trend persists when considering syntactic equivalence (SYE), i.e., when patches are plausible and also syntactically equivalent to the ground truth. Agentless and SWE-Agent consistently outperform AutoCodeRover, achieving up to 24 and 28 syntactically equivalent patches respectively across the models. Notably, Claude-3.5-Sonnet leads with 24 SYE patches under Agentless, suggesting stronger alignment with the human-made fixes. On the other hand, Deepseek-V3 achieves 28 SYE patches under SWE-Agent.

Furthermore, we additionally performed a manual evaluation of the generated patches to assess whether they genuinely addressed the underlying causes of build failures. Plausibility was initially

Table 1: Results for 100 Artifacts (#PG: Patches Generated, #PL: Plausible, #SYE: Syntactic Equivalent, #CO: Correct).

	Agentless				SWE-Agent				AutoCodeRover			
Model	#PG	#PL	#SYE	#CO	#PG	#PL	#SYE	#CO	#PG	#PL	#SYE	#CO
GPT-4o	62	32	22	28	67	30	23	28	21	13	11	13
Claude-3.5-Sonnet	52	30	24	27	45	24	20	22	22	10	10	10
Deepseek-V3	44	22	19	21	64	36	28	35	19	11	9	11

measured by checking if a previously failing repository could be built successfully after applying a tool-generated patch. However, in some cases, patches produced successful builds without resolving the actual exception or error in the source code. For example, a patch that simply modified or disabled failing test cases could yield a successful build, since success is defined only by all tests passing. To account for such cases, we conducted manual inspection of each patch to ensure that the observed build success reflected a meaningful fix rather than a superficial change.

Manual inspection is consistent with other metrics. Agentless and SWE-Agent clearly outperform AutoCodeRover, achieving success rates of up to 28% and 35%, respectively, compared to only 13% for AutoCodeRover. Specifically, GPT-4o produced 28 successful patches when combined with Agentless, while Deepseek-V3 achieved 35 successful patches with SWE-Agent.

Among the foundation models, GPT-4o and Claude-3.5-Sonnet exhibit comparable performance across all tools, with GPT-4o maintaining a slight lead in average plausibility. While Deepseek-V3 achieves the highest performance when paired with SWE-Agent, it produces fewer plausible or syntactically equivalent patches when integrated with AutoCodeRover and Agentless.

4 Threats to Validity

We evaluated three foundation LLMs, GPT-4o, Claude-3.5-Sonnet, and Deepseek-V3, on 100 BugSwarm artifacts. Each of these models has its own training cut-off date, and it is possible that they were partially trained on GitHub data, though the specifics are not publicly known. This raises a valid concern for all real-world datasets used in program repair. However, CI-Bench remains applicable regardless of whether a fix was pushed into the associated repository before or after a model's training cut-off date, as it can be applied to any artifact in the dataset. This indicates the critical importance of evolving datasets such as BugSwarm, which continuously grow over time with new artifacts. For the manual inspection in our evaluation, we carefully compared each generated patch against the developer patch; however, some false positives may remain.

5 Related Work

SWE-Bench [4] is a benchmark of 2,294 real-world GitHub issues for evaluating code repair task. SWE-Bench Lite [3] is a subset of SWE-Bench, popular among SE research communities. Other benchmarks, such as Multi-SWE-Bench [9], expand the benchmark to support multiple programming languages beyond Python. Agentic tools like SWE-Agent [8], AutoCodeRover [10], AIDER [1], and the non-agentic Agentless [7] have shown significant performance on these datasets. However, these benchmarks face significant limitations: they often lack dataset diversity, fail to provide robust support for reproducibility, and are primarily restricted to GitHub issues [12, 13]. In contrast, BugSwarm provides reproducible CI failures for both Java and Python, covering a broader range of repositories. CI-Bench is a unified framework that leverages BugSwarm artifacts to enable the systematic evaluation of LLM-based program repair tools on diverse and reproducible real-world CI failures.

6 Conclusion

We introduced CI-Bench, the first benchmark specifically designed to evaluate LLM-based program repair tools on CI failures. While existing benchmarks often lack reproducibility and recency, CI-Bench addresses these limitations by leveraging the BugSwarm dataset. Using our framework, we evaluated three state-of-the-art repair tools—originally developed for resolving bug reports—on 100 real-world CI failures. Our findings reveal that Agentless and SWE-Agent significantly outperform AutoCodeRover in generating correct patches. In future work, we plan to generalize CI-Bench to tasks such as test generation and fault localization.

7 Acknowledgments

This work was supported by the National Science Foundation under awards CNS-2016735, CNS-2346396, and CCF-2119348.

References

[1] 2025. AIDER. https://aider.chat/2024/06/02/main-swe-bench.html.
[2] 2025. antlr. https://www.antlr.org/.
[3] 2025. SWE-Bench Lite. https://www.swebench.com/lite.html.
[4] Carlos Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-Bench: Can Language Models Resolve Real-world Github Issues?. In *ICLR*. OpenReview.net.
[5] Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2024. Cognitive Architectures for Language Agents. *Trans. Mach. Learn. Res.* 2024 (2024).
[6] David A. Tomassi, Naji Dmeiri, Yichen Wang, Antara Bhowmick, Yen-Chuan Liu, Premkumar T. Devanbu, Bogdan Vasilescu, and Cindy Rubio-González. 2019. BugSwarm: mining and continuously growing a dataset of reproducible failures and fixes. In *ICSE*. IEEE / ACM, 339–349.
[7] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. Demystifying LLM-Based Software Engineering Agents. *Proc. ACM Softw. Eng.* 2, FSE (2025), 801–824.
[8] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *NeurIPS*.
[9] Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Kai Shen, and Liang Xiang. 2025. Multi-SWE-bench: A Multilingual Benchmark for Issue Resolving. *CoRR* abs/2504.02605 (2025).
[10] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. AutoCodeRover: Autonomous Program Improvement. In *ISSTA*. ACM, 1592–1604.
[11] Hao-Nan Zhu, Kevin Z. Guan, Robert M. Furth, and Cindy Rubio-González. 2023. Actionsremaker: Reproducing GitHub Actions. In *ICSE Companion*. IEEE, 11–15.
[12] Hao-Nan Zhu and Cindy Rubio-González. 2023. On the Reproducibility of Software Defect Datasets. In *ICSE*. IEEE, 2324–2335.
[13] Hao-Nan Zhu, Robert Furth, Michael Pradel, and Cindy Rubio-González. 2026. From Bugs to Benchmarks: A Comprehensive Survey of Software Defect Datasets. *ACM Comput. Surv.* (Feb. 2026). doi:10.1145/3797033