SWE-BENCH-XL: EVALUATING CODING AGENTS ON VERY LARGE REPOSITORIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents are evaluated on benchmarks of repository-level tasks mined from real pull requests, but current reference benchmarks mostly draw from small codebases. Real industrial and open-source development happens in much larger repositories, where an agent must navigate the code and manage a limited context window. We close this gap with SWE-Bench-XL, a benchmark of 100 reproducible tasks mined from four of the largest open-source repositories on GitHub, with a mean per-instance source-code size roughly an order of magnitude larger than reference benchmarks. Across five frontier models evaluated under OpenHands, pass@1 ranges from 22.5% to 47.6%. Trajectory analysis shows that agents make more codebase searches and visit more directories than in any other benchmark, indicating that SWE-Bench-XL better exercises the capabilities of agents in navigating codebases. We release SWE-Bench-XL as reproducible Harbor tasks.

1 INTRODUCTION

Coding agents increasingly write or contribute to large-scale software systems. Google reports that 75% of all new code it writes is AI-generated, a rise from 25% in 2024 and 50% in late 2025 (Pichai, 2024; 2026), with many predicting nearly all code to be AI-generated in the near future. Coding agents capabilities are measured by benchmarks of software engineering tasks. SWE-bench (Jimenez et al., 2024) is an influential benchmark with mined issue-resolving tasks from merged pull requests automatically graded by running the test suites. More recently, SWE-Bench-Pro (Deng et al., 2026) and DeepSWE (Huang et al., 2026) extend the scope of software engineering tasks towards harder, longer-horizon problems.

Existing benchmarks, however, do not capture repositories as large as the ones developers work in every day. SWE-Bench-Verified, SWE-Bench-Pro, and DeepSWE include codebases with a median of 3.6M, 1.2M, and 0.29M source-code tokens, respectively. The largest instance in any of the mentioned benchmarks measures 35.6M tokens. Many real-world systems are orders of magnitude larger than this. Yet frontier models now accept a million tokens of context. The median SWE-Bench-Verified codebase is only 3.6 such windows, the median SWE-Bench-Pro codebase only 17% larger than one, and the median DeepSWE codebase is less than one such window. Basically, tasks in current reference benchmarks are not enough to truly exercise the ability of agents to navigate and manage context windows in the context of real-world software development.

We close that gap with SWE-Bench-XL, a benchmark of 100 reproducible tasks mined from extra-large open-source codebases. Figure 1 positions SWE-Bench-XL against existing benchmarks on the codebase size axis. The median SWE-Bench-XL codebase has a size of 44.9M source-code tokens, 13 times the SWE-Bench-Verified median, 38 times the SWE-Bench-Pro median, and 153 times the DeepSWE median. SWE-Bench-XL's smallest instance, with a codebase of 14.8M tokens, is already larger than 94% of all SWE-Bench-Verified and SWE-Bench-Pro instances, and its largest reaches 148.2M. At this scale, agents have to locate relevant code while selecting what to read and retain in context.

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.

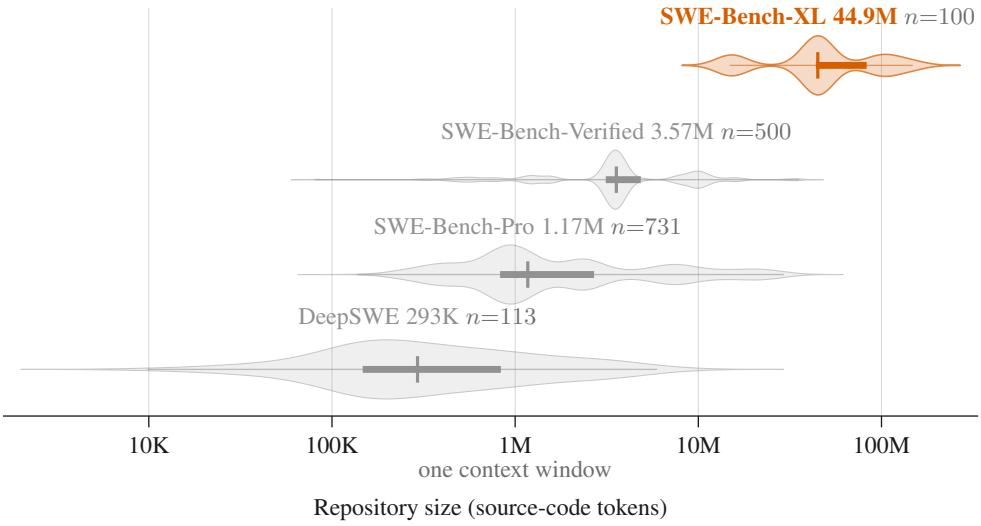


Figure 1: Size distribution of coding benchmarks measured in source-code tokens at each task's base commit, on a log axis; the 1M tick is one frontier context window. Each row is a kernel-density violin over that benchmark's instances, estimated in log size and normalised to a common height, with its interquartile range as a bar and its median as a tick through it. The SWE-Bench-XL row has one lump per repository: `transformers` at 15M, `typescript` and `elasticsearch` together near 45M, and `vscode` spread from 80M to 148M as it grew over the months the pull requests span. The median SWE-Bench-XL task holds 44.9M source-code tokens, 13× the SWE-Bench-Verified median and 38× the SWE-Bench-Pro median. SWE-Bench-XL spans 14.8M to 148.2M tokens across its 100 instances.

We evaluate five frontier models under OpenHands. Overall pass@1 ranges from 22.5% to 47.6%, while the leading model varies by repository (Section 4.1). In a cross-benchmark comparison using the same agent, SWE-Bench-XL and DeepSWE have similar pass@1 (22.52% vs. 25.27%) and operations per attempt (61.9 vs. 58.3). On SWE-Bench-XL, however, the agent makes 1.42 times as many searches, visits 1.69 times as many distinct directories, and uses 2.66 times as many operations per reference-patch source file (Section 4.2). These trajectories are consistent with agents spending more effort finding the relevant code, even when the eventual edit is small.

To summarize, we make three contributions:

- We construct SWE-Bench-XL, a benchmark of 100 software engineering tasks mined from four of the largest open-source repositories, with a median per-instance codebase size of 44.9M source-code tokens. We analyze its reference patches and their direct dependencies to characterize the local repairs within these large repositories.
- We evaluate five models under OpenHands and compare performance and trajectories across repositories and benchmarks, finding strong repository dependence and greater codebase navigation on SWE-Bench-XL than on DeepSWE at similar pass@1 and trajectory length.
- We make SWE-Bench-XL publicly available and ready to use via Harbor (Harbor Framework Team, 2026).

2 SWE-Bench-XL

SWE-Bench-XL is a benchmark of coding tasks mined from merged pull requests in very large open-source repositories on GitHub. It measures how coding agents behave on codebases an order of magnitude larger than the ones in existing benchmarks. At this scale no existing language model's context can hold significant portions of the codebases at once, so solving a task requires agents to navigate and manage context in a way that is closer to real workloads.

Repository	Language, domain	#Tasks	Codebase size (million tokens) Mean	Min	Max
microsoft/vscode	TypeScript, desktop IDE	27	111.7	79.7	148.2
elastic/elasticsearch	Java, search engine	39	45.3	43.9	46.9
microsoft/typescript	TypeScript, compiler	15	44.3	43.4	48.3
huggingface/transformers	Python, ML library	19	15.2	14.8	15.6
Total		**100**	**57.4**	**14.8**	**148.2**

Table 1: The four SWE-Bench-XL repositories, with instance counts and per-instance codebase size in million source-code tokens, measured at each instance's base commit by the procedure of Section 2.5. The smallest repository, huggingface/transformers, has at least 14.8M tokens per instance, larger than 94% of SWE-Bench-Verified and SWE-Bench-Pro instances.

2.1 Benchmark Overview

SWE-Bench-XL contains 100 tasks from four repositories, shown in Table 1. Each task is a self-contained instance with a synthesized natural-language problem description, a deterministic test oracle synthesized from the original pull requests, and a container image containing the codebase at the moment before the pull request was merged.

We select the four repositories based on two criteria. First, all four repositories are larger in size than the median size of SWE-Bench-Verified, SWE-Bench-Pro, and DeepSWE. Second, they are diverse in terms of application domain. elasticsearch is a Java distributed-search engine, transformers a Python machine-learning library, typescript a compiler, and vscode a desktop application built on Electron. Together they span vastly different languages, build systems, and applications.

Table 1 reports the number of source-code tokens per instance, measured at the base commit with the procedure of Section 2.5. The four repositories span 14.8M to 148.2M tokens, and the largest instance, microsoft__vscode-306139, is 4.2 times the size of the largest instance in any of the three reference benchmarks.

2.2 Patch Analysis

Figure 2 characterizes the gold patch of every task in SWE-Bench-XL and in the three benchmarks we compare against, measured with the same procedure on all four. The median SWE-Bench-XL patch edits 2 source files and 51 source lines, which is 0.046% of the source files in its repository. The files it modifies directly import a median of 38 other files of the repository, which together hold 184K tokens of code.

Patch size. SWE-Bench-XL patches are larger than those of SWE-Bench-Verified (a median of 1 file and 7 lines) and smaller than those of SWE-Bench-Pro (3 files, 81 lines). The sharpest contrast is with DeepSWE, the benchmark closest to SWE-Bench-XL in difficulty and trajectory length (Section 4.2). A median DeepSWE patch edits 6 source files and 642 lines, 13 times as many lines as a SWE-Bench-XL patch, and 25% of the source files DeepSWE patches touch are new, against 4% in SWE-Bench-XL. Because DeepSWE's codebases are also 153 times smaller, a DeepSWE patch touches 4.7% of its repository's source files, about 100 times the share a SWE-Bench-XL patch touches. The two benchmarks are hard in opposite ways: DeepSWE asks for large changes to small codebases, while SWE-Bench-XL asks for localized changes within very large ones.

Dependencies. Localized changes are not necessarily self-contained ones. In the median SWE-Bench-XL task, the modified files directly import 38 files holding 184K tokens of code, against 9, 24, and 17 files holding 34K, 47K, and 30K tokens in SWE-Bench-Verified, SWE-Bench-Pro, and DeepSWE, which is 3.9 times the code of the nearest benchmark. In 59% of SWE-Bench-XL tasks these direct imports alone exceed 128K tokens, against 14%, 21%, and 18% in SWE-Bench-Verified, SWE-Bench-Pro, and DeepSWE. This shows that, even when the changes themselves are small, the code they build on is broad. To correctly solve a SWE-Bench-XL task an agent has to

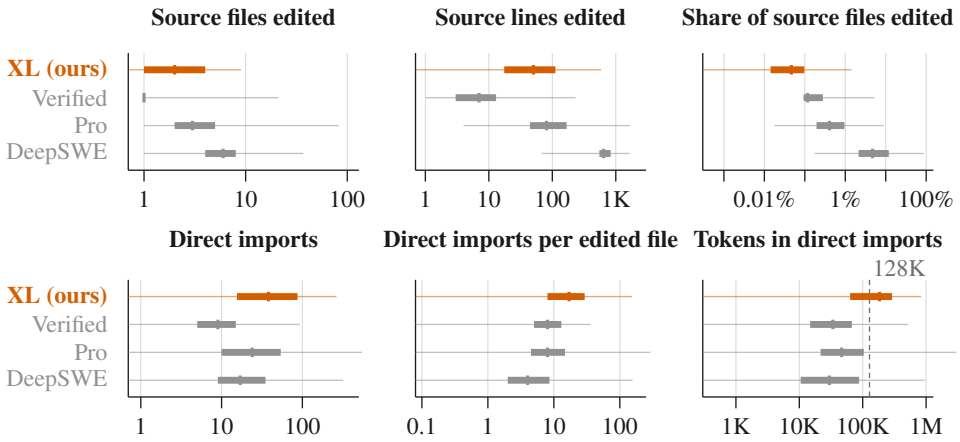


Figure 2: Size of the gold patches (top) and of the code that the edited files directly depend on (bottom), one row per benchmark, on log scales. Each row marks the full range (hairline), the interquartile range (bar), and the median (tick). The share of source files edited counts edited and added source files against the repository's source files after the patch. Direct imports are the repository files that the modified source files import at the base commit (we report the number per task and per edited file).

locate and understand the parts of a surrounding context that already occupies a significant part of frontier model's context windows, inside a codebase that is even larger.

2.3 Task Specification

Each SWE-Bench-XL task gives the agent a natural-language problem statement and a repository checked out at the commit before the fix. The task asks to modify the code so that a held-out test-based oracle passes. The agent receives no issue link, no reference patch, and no test file. The agent is expected to work without internet access.

Listing 1 shows an example of a problem statement from a SWE-Bench-XL instance built from an `elastic/elasticsearch` PR. The original pull request information [1] is transformed and augmented by the synthetic pipeline to ensure that the problem statement contains enough information for it to be possible to autonomously complete the task without leaking the solution. The task's verifier is strengthened with additional test cases synthesized from the original pull request, such as the one shown in Listing 2, while also retaining existing tests from the repository as regression checks.

Problem statement. Each task provides a problem statement defining the issue to resolve. The statement is synthetically generated based on the original pull request.

Environment. Each task ships with a containerized environment which contains the codebase at the base commit and all required dependencies for building and testing the project. We specify every task in the Harbor task format (Harbor Framework Team, 2026), a framework for evaluating agents and models in container environments.

Verifier. The canonical verifier for SWE-Bench-XL consists of automated tests synthetically generated from the original pull request, based on the intention of the original author. Each task includes a list of `FAIL_TO_PASS` tests, which fail at the base commit and verify the task has been resolved, and a set of `PASS_TO_PASS` tests which guard against regressions. Listing 2 shows one generated `FAIL_TO_PASS` test for the Elasticsearch task: it fixes a random seed that exposes the bug and checks that estimates do not decrease as quantiles increase.

[1] `https://github.com/elastic/elasticsearch/pull/144588`

 # Exponential Histogram Percentiles Can Be Non−Monotonic

When computing multiple percentiles from an exponential histogram, the resulting values should be non−decreasing (i.e., a higher percentile must return a value greater than or equal to a lower percentile). This invariant is occasionally violated, causing intermittent test failures.

Target audience

Internal codebase developers and QA engineers working with exponential histogram aggregations.

Current behavior

When querying a sequence of increasing quantiles from an exponential histogram, the returned values are occasionally ∗∗not∗∗ monotonically non−decreasing. That is, for two quantile values `q1 < q2`, the estimated result for `q2` can be strictly less than the result for `q1`. This can be reproduced by running the percentile tests over many random seeds.

Expected behavior

For any sequence of increasing quantile inputs `q1 <= q2 <= ... <= qN` queried against the same exponential histogram, the resulting estimated values must be monotonically non−decreasing: `result(q1) <= result(q2) <= ... <= result(qN)`. This invariant must hold consistently regardless of the histogram's data distribution or the specific quantile values requested.

Task

Fix the described bug. The repository is located at `/app`.

Listing 1: Problem statement for `elastic/elasticsearch-144588`, synthesized from the original PR description and solution.

```
public void testQuantileMonotonicityDeterministicSeed() {
    // Seed 801 with these parameters produces data that exposes the
    // non-monotonicity bug in the old interpolation formula
    // prev*(1-f) + curr*f.
    Random rng = new Random(801L);
    int n = 100 + (int) (rng.nextDouble() * 900);   // yields 734
    double[] values = generateLogNormal(rng, n);
    int buckets = 5 + (int) Math.round(Math.pow(1995.0,
    ↪ rng.nextDouble()));  // yields 8

    ExponentialHistogram histogram =
    ↪ createAutoReleasedHistogram(buckets, values);

    double prevQuantileValue = Double.NEGATIVE_INFINITY;
    for (double q : QUANTILES_TO_TEST) {
        double result =
    ↪ ExponentialHistogramQuantile.getQuantile(histogram, q);
        if (Double.isNaN(result)) {
            continue;
        }
        assertTrue(
            "Quantile values must be monotonically non-decreasing",
            result >= prevQuantileValue
        );
        prevQuantileValue = result;
    }
}
```

Listing 2: Example FAIL_TO_PASS test from `elastic/elasticsearch-144588`, part of the test oracle synthesized from the ground-truth solution available in the original PR.

2.4 Benchmark Construction

We rely on an agent-driven pipeline to collect all 100 instances. The pipeline takes a merged pull request and produces a task in Harbor format, without necessarily requiring a linked issue or pre-existing tests in the patch. Then, it verifies that the task is reproducible and that the verifier (the held out test) correctly distinguishes correct and incorrect solutions. Additionally, the agent builds containerized environments based on each repository's documentation and CI configuration, ensuring every instance is reproducible, self-contained, and resembles the environment a human developer would use to work on the repository.

For the task in Listing 1, the source pull request changed one method and added a monotonicity check in an adjacent test. The fix replaces a weighted sum of two histogram values with interpolation from the previous value using their difference, avoiding a rounding pattern that caused the failure. Our pipeline turned the pull request into the problem statement above and an augmented test oracle, checking that the oracle fails on the pre-fix commit and passes with the reference patch.

For each of the identified repositories, we harvest a pool of candidate pull requests spanning January 1 to May 31 of 2026, with the goal of collecting as recent as possible instances to reduce the risk of evaluating models on tasks they may have seen during training.

2.5 Codebase Size

There exist several ways for calculating the size of a codebase, including lines of code and number of files. We calculate the size of each instance's codebase in tokens with a recursive tree walk of the repository at the base commit. Token count is restricted to source-code files only. Files whose extensions are in a predefined list of binary file types, and directories holding build artifacts, virtual environments, dependency caches, or version control metadata are excluded. We encode the text of every remaining file with the `cl100k_base` tokenizer through the `tiktoken` library (ope, 2026) and attribute the token count to that file. This is a novel and relevant measure for benchmarks, as coding agents have the context window as the key constraint to respect.

3 Experimental Methodology

3.1 Agents and Runs

We evaluate a series of frontier models, all under the same harness, OpenHands (Wang et al., 2025). Every agent is run on all 100 instances for five independent attempts. Agents start inside each instance's container with the problem statement as its only input, and interact with the codebase until they emit a task termination signal or the time limit. The verifier grades the diff produced by the agent according to each instance's test cases.

3.2 Metrics

After excluding error and missing outcomes, we estimate pass@1 (Chen et al., 2021) as the fraction of attempts that resolve each instance, then average those fractions equally over all instances. We run five independent attempts per instance (Bjarnason et al., 2026).

For trajectory metrics, we count normalized file reads, writes, searches, and other recognized file operations. We also measure the number of distinct directories visited, navigation operations per write, operations per gold file, and the position of the first write. The gold file set G contains the source-code files modified by the reference patch, excluding test, documentation, and specification files. Trajectory values are computed per attempt, averaged within each instance, and then averaged equally over instances with valid attempts. Table 2 defines the reported metrics and their abbreviations.

Metric	Definition
pass@1	Mean per-instance fraction of attempts that resolve the task, reported as a percentage.
Ops	Number of operations in a trajectory, including reads, writes, searches, etc. We normalize tool calls into operations to ensure that, for example, tool calls that read multiple files are interpreted as multiple read operations.
Reads	Number of file-read operations.
Writes	Number of file-write operations.
Srch	Number of file searches using commands like `grep`, `rg`, `ag`, `find`, `ls`, `tree`, `fd`, or `locate`.
Dirs	Number of distinct directories targeted by non-write operations.
Nav/W	(Reads + Srch) / Writes, computed per attempt
Ops/$\|G\|$	Ops / $\|G\|$, computed per attempt, where $\|G\|$ is the number of source-code files in the gold file set G.
FW@	Position of the first write in the full operation sequence

Table 2: Metrics and abbreviations used in the results tables.

Model	Overall	`vscode`	`elasticsearch`	`typescript`	`transformers`
GPT-5.6 Sol	**47.6**	9.6	**68.3**	61.4	**48.2**
Claude Opus 5	35.1	13.5	27.4	**78.7**	47.4
Claude Fable 5.1	27.3	**25.2**	7.4	64.0	42.1
GPT-5.6 Terra	27.1	10.4	22.8	65.3	29.5
Claude Sonnet 5	22.5	6.7	13.0	61.6	33.7

Table 3: Pass@1 (%) overall and by repository under OpenHands. Overall averages across all 100 instances; the repository columns follow the descending mean codebase-size order in Table 1. Bold indicates the highest point estimate in each column.

4 Experimental Results

4.1 Leaderboard and Per-Repository Performance

Table 3 reports overall and per-repository pass@1 for the five models on the 100 instances of SWE-Bench-XL. GPT-5.6 Sol leads overall at 47.6%, followed by Claude Opus 5 at 35.1%. The remaining models range from 22.5% to 27.3%.

The ranking changes across repositories. GPT-5.6 Sol has the highest point estimate on `elasticsearch` and `transformers`, Claude Opus 5 on `typescript`, and Claude Fable 5.1 on `vscode`. The gap between each model's strongest and weakest repository exceeds 54 percentage points, while the gap between the first and fifth models overall is 25.1 points.

4.2 Cross-Benchmark Comparison

We compare SWE-Bench-XL with SWE-Bench-Verified, SWE-Bench-Pro, and DeepSWE using the same OpenHands / Claude Sonnet 5 configuration, with five independent attempts per task. Outcome and overall trajectory metrics are computed consistently across all four benchmarks. We first use the full comparison to position XL. We then contrast it with DeepSWE because it is the benchmark that most closely matches the target difficulty and trajectory length.

Difficulty and Trajectory Size. As Table 4 shows, pass@1 is 77.31% on Verified and 55.75% on Pro but only 25.27% on DeepSWE and 22.52% on XL. This shows that DeepSWE and SWE-Bench-XL clearly are more difficult. XL averages 61.9 operations and 28.4 reads per attempt, against 58.3 and 26.6 on DeepSWE. The difference between SWE-Bench-XL and DeepSWE is not statistically significant.

Trajectory Behavior. Despite nearly equal difficulty and total trajectory size, SWE-Bench-XL and DeepSWE exercise different behaviors in the coding agent. While DeepSWE exercises the

	Difficulty	Trajectory size			Navigation				Timing
Benchmark	pass@1 (%)	Ops	Reads	Writes	Srch	Dirs	Nav/W	Ops/$\|G\|$	FW@
Verified	77.31	18.4	7.7	3.9	6.4	4.1	4.80	16.1	8.1
Pro	55.75	29.3	12.1	6.6	10.2	5.6	4.32	11.7	11.7
DeepSWE	25.27	58.3	26.6	15.2	16.1	6.1	3.40	10.6	17.3
XL	22.52	61.9	28.4	10.1	22.9	10.3	7.08	28.2	18.7

Table 4: Cross-benchmark outcomes and mean trajectory metrics for the same agent configuration. Nav/W is reads plus searches per write, FW@ is the position of the first write in the operation sequence, and G is the gold file set.

ability of agents to implement large changes in the codebase, SWE-Bench-XL focuses on exercising the capabilities of agents in navigating large scale codebases.

On XL, the agent issues 22.9 searches and visits 10.3 distinct directories per attempt, 1.42 and 1.69 times the corresponding DeepSWE values. Trajectories on SWE-Bench-XL also observe a first write operation later (18.7) than in any other benchmark. XL also requires 28.2 operations per gold file, compared with 10.6 on DeepSWE (2.66×), and leads to more navigation operations per write (7.08) than on DeepSWE (3.40).

These patterns are consistent with the codebase-size hypothesis. The median SWE-Bench-XL instance is about 153 times the size of the median DeepSWE instance, and agents spend more effort exploring its codebase. Overall, SWE-Bench-XL is harder and longer than Verified and Pro. Relative to DeepSWE, SWE-Bench-XL elicits broader codebase navigation and exploration.

5 Discussion

5.1 Limitations

SWE-Bench-XL contains tasks from four public repositories. They span several languages and domains, but do not represent the full range of large software projects, including even larger open-source codebases such as the Linux kernel and proprietary repositories. One notable threat is that repository size alone does not establish that a task requires broad codebase knowledge. Some tasks might remain local after the relevant component is found. Our trajectory analysis numbers indicate that SWE-Bench-XL is better at capturing this aspect than existing benchmarks, but does not prove that all tasks require it. Finally, our results rely on a single agent harness. We believe so but only future usage of SWE-Bench-XL will provide sound empirical validation of generalization beyond OpenHands.

5.2 Future Work

We aim to identify and validate tasks whose solutions require understanding interactions across components, rather than a single isolated component. For long-lived repositories, future benchmarks should assess whether agent changes remain understandable and adaptable through subsequent tasks (i.e., code maintenance), in addition to passing functional tests. It is also on the roadmap to develop better tools for analyzing trajectories which could distinguish and measure which capabilities (e.g., navigation, implementation, testing) each benchmark exercises, and clarify how those contribute to overall performance.

6 Related Work

Coding benchmarks are the foundation for measuring progress in AI coding. HumanEval and MBPP test models on generating single Python functions from natural language descriptions (Chen et al., 2021; Austin et al., 2021). LiveCodeBench (Jain et al., 2024) increased the difficulty by drawing fresh competitive programming problems from online platforms. Frontier models, however, have saturated these benchmarks and the field has moved to measuring progress and capabilities with more complex and realistic software engineering tasks.

Benchmarks like SWE-bench (Jimenez et al., 2024) mine tasks from merged pull requests in open-source repos (Saavedra et al., 2024), ensuring every problem corresponds to a task a developer actually completed. Multi-SWE-bench and SWE-bench Multilingual (Zan et al., 2025; kha, 2025) focus on increasing programming language coverage, SWE-bench Multimodal (Yang et al., 2025) on including vision problems, while SWE-bench-Live and SWE-rebench (Zhang et al., 2025; Badertdinov et al., 2025) continuously mine fresh instances to resist data contamination.

More recently, there has been a focus on increasing task horizon. SWE-Bench-Pro (Deng et al., 2026) mines tasks whose reference patch spans several files and over a hundred lines with the goal of increasing task difficulty and horizon. DeepSWE (Huang et al., 2026) focuses on the same goal but creates from scratch a new set of tasks. ProgramBench (Yang et al., 2026) and MirrorCode (Adamczewski et al., 2026) propose a new kind of task: asking agents to rebuild programs from scratch using documentation and the original binary as behavior specification. SWE-Bench-XL focuses on problems which require navigating and reasoning over very large codebases, and is unique in doing so.

LoCoBench and LoCoBench-Agent (Qiu et al., 2025a;b) synthesize large codebases and tasks with the goal of testing the ability of agents to navigate and reason over them. On the contrary, SWE-Bench-XL is not synthetic, it mines tasks from very large open-source repositories available on GitHub, ensuring that the tasks are realistic and representative of real-world software engineering problems. LinuxFLBench (Zhou et al., 2026) collects tasks from the Linux kernel, but focuses on fault localization alone. SWE-Bench-XL contains end to end tasks related to issue fixing.

7 Conclusion

We have introduced SWE-Bench-XL, a benchmark for evaluating coding agents on very large codebases. We evaluate five frontier models under one harness on it. The best model resolves 47.6% of instances at pass@1. On XL, agents navigate more broadly and use more operations per gold file than on DeepSWE. SWE-Bench-XL is publicly released, with every instance packaged in a containerized, reproducible format.

AI Use Statement

We have used generative AI tools in the following ways. First, we have used AI to partly synthesize problem statements and test oracles that make up the instances in SWE-Bench-XL, as described in Section 2.4. Second, we have used coding agents for parts of the execution of our project. Third, we have used AI to adversarially reason about our experimental methodology and results interpretation. Fourth, we have used AI to polish writing of the paper, specifically to guarantee consistency between writing and data, in improving figures and tables, and in correcting grammatical mistakes.

Ethics Statement

SWE-Bench-XL is constructed entirely from pull requests merged into public, actively maintained open-source repositories, and the released dataset redistributes only content those repositories' licenses permit.

References

SWE-bench Multilingual · Kabir Khandpur, May 2025. URL https://kabirk.com/multilingual.html.

openai/tiktoken, June 2026. URL https://github.com/openai/tiktoken. original-date: 2022-12-01T23:22:11Z.

Tom Adamczewski, David Owen, David Rein, Florian Brand, Giles Edkins, Allen Hart, and Daniel O'Connell. Mirrorcode: Ai can rebuild entire programs from behavior alone. *arXiv preprint arXiv:2606.30182*, 2026.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program Synthesis with Large Language Models, August 2021. URL http://arxiv.org/abs/2108.07732. arXiv:2108.07732 [cs.PL].

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: An Automated Pipeline for Task Collection and Decontaminated Evaluation of Software Engineering Agents. October 2025. URL https://openreview.net/forum?id=nMpJoVmRy1.

Bjarni Haukur Bjarnason, André Silva, and Martin Monperrus. On Randomness in Agentic Evals, March 2026. URL http://arxiv.org/abs/2602.07150. arXiv:2602.07150 [cs.LG].

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde 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, Josh 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, July 2021. URL http://arxiv.org/abs/2107.03374. arXiv:2107.03374 [cs].

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa R Kundurthy, Sean M. Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-bench pro: Can AI agents solve long-horizon software engineering tasks? In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=uEVTdoAbnK.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, September 2026. URL https://github.com/harbor-framework/harbor.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. Deepswe: Measuring frontier coding agents on original, long-horizon engineering tasks, 2026. URL https://github.com/datacurve-ai/deep-swe.

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, June 2024. URL http://arxiv.org/abs/2403.07974. arXiv:2403.07974 [cs.SE].

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can Language Models Resolve Real-world Github Issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Sundar Pichai. Alphabet Q3 earnings call: CEO Sundar Pichai's remarks, October 2024. URL https://blog.google/company-news/inside-google/message-ceo/alphabet-earnings-q3-2024/. "More than a quarter of all new code at Google is generated by AI, then reviewed and accepted by engineers.".

Sundar Pichai. Cloud Next '26: Momentum and innovation at Google scale, April 2026. URL https://blog.google/innovation-and-ai/infrastructure-and-cloud/google-cloud/cloud-next-2026-sundar-pichai/. "Today, 75% of all new code at Google is now AI-generated and approved by engineers, up from 50% last fall.".

Jielin Qiu, Zuxin Liu, Zhiwei Liu, Rithesh Murthy, Jianguo Zhang, Haolin Chen, Shiyu Wang, Ming Zhu, Liangwei Yang, Juntao Tan, Zhepeng Cen, Cheng Qian, Shelby Heinecke, Weiran Yao, Silvio Savarese, Caiming Xiong, and Huan Wang. LoCoBench: A Benchmark for Long-Context Large Language Models in Complex Software Engineering, September 2025a. URL http://arxiv.org/abs/2509.09614. arXiv:2509.09614 [cs.SE].

Jielin Qiu, Zuxin Liu, Zhiwei Liu, Rithesh Murthy, Jianguo Zhang, Haolin Chen, Shiyu Wang, Ming Zhu, Liangwei Yang, Juntao Tan, Roshan Ram, Akshara Prabhakar, Tulika Awalgaonkar, Zixiang Chen, Zhepeng Cen, Cheng Qian, Shelby Heinecke, Weiran Yao, Silvio Savarese, Caiming Xiong, and Huan Wang. LoCoBench-Agent: An Interactive Benchmark for LLM Agents in Long-Context Software Engineering, November 2025b. URL http://arxiv.org/abs/2511.13998. arXiv:2511.13998 [cs.SE].

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*, pp. 1–5, 2024.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL http://arxiv.org/abs/2407.16741.

John Yang, Carlos E. Jimenez, Alex Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik Narasimhan, Diyi Yang, Sida Wang, and Ofir Press. SWE-bench Multimodal: Do AI Systems Generalize to Visual Software Domains? In *International Conference on Learning Representations*, May 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/07d6332ae36730707fddddba736d7b6c-Abstract-Conference.html.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, et al. Programbench: Can language models rebuild programs from scratch? *arXiv preprint arXiv:2605.03546*, 2026.

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. Multi-SWE-bench: A Multilingual Benchmark for Issue Resolving, April 2025. URL http://arxiv.org/abs/2504.02605. arXiv:2504.02605 [cs].

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. SWE-bench Goes Live!, June 2025. URL http://arxiv.org/abs/2505.23419. arXiv:2505.23419 [cs.SE].

Zhenhao Zhou, Zhuochen Huang, Yike He, Chong Wang, Jiajun Wang, Yijian Wu, Xin Peng, and Yiling Lou. Taming System Complexity: Demystifying Software Engineering Agents in Diagnosing Linux Kernel Faults, June 2026. URL http://arxiv.org/abs/2505.19489. Annual Meeting of the Association for Computational Linguistics (ACL 2026). arXiv:2505.19489 [cs.SE].