BeyondSWE: Can Current Code Agent Survive Beyond Single-Repo Bug Fixing?

Guoxin Chen[*], **Fanzhe Meng**[*], **Jiale Zhao**[*], **Minghao Li, Daixuan Cheng, Huatong Song, Jie Chen, Yuzhi Lin, Hui Chen, Xin Zhao**[†], **Ruihua Song**[†], **Chang Liu, Cheng Chen, Kai Jia**[†] **and Ji-Rong Wen**
[1]Gaoling School of Artificial Intelligence, Renmin University of China, [2]Independent Researcher, [3]AweAI Team

Current code-agent benchmarks primarily evaluate localized issue resolution within a single target repository, leaving under-tested many software engineering tasks that require external knowledge or broader repository-level changes. We introduce BeyondSWE, a 500-instance benchmark drawn from 246 real-world GitHub repositories to evaluate code agents beyond single-repository bug fixing. BeyondSWE covers four representative settings: cross-repository issue resolution, domain-specific issue resolution, dependency-driven migration, and document-to-repository generation, spanning both broader knowledge scope and broader resolution scope. Our evaluation shows that BeyondSWE remains far from saturated: the best OpenHands-based agent reaches 46.12 average score, while the strongest Codex harness with GPT-5.4 (xhigh) reaches 56.65 under a search-aware prompt. To study whether external information access closes this gap, we use SearchSWE as a controlled diagnostic baseline for search-augmented coding. Search access improves most models and substantially helps some tasks, but the gains remain limited and uneven, showing that current agents still struggle to convert retrieved information into precise, version-compatible, and locally actionable code changes. These results suggest that deep search for coding remains an open problem: progress requires agents that can reliably combine external evidence with repository-local reasoning and execution-based verification.

🤗 Benchmark Repo Scaffold WebPage

[*]Equal Contributions. [†]Corresponding authors.

Date: Feb. 18, 2026.

1. Introduction

Modern code agents have made rapid progress on software-engineering benchmarks, led by SWE-bench Verified (Jimenez et al., 2024), which grounds evaluation in real GitHub issues. Subsequent work has extended this paradigm through live updates, broader repository coverage, multilingual settings, and more complex issue instances (Deng et al., 2025; Zan et al., 2025a; Zhang et al., 2025).

Yet this evaluation paradigm still leaves important parts of software engineering under-tested. Most existing benchmarks center on localized issue resolution within a single target repository, where the relevant context is expected to be recoverable from the issue and the codebase. In practice, many engineering tasks fail precisely because the necessary information is outside the repository, or because the required change must be coordinated across a broader portion of the system. Developers may need to consult upstream projects, documentation, or domain knowledge. They may also need to migrate APIs across a codebase or build a coherent repository from a specification. These settings stress capabilities that are only partially captured by single-repository bug-fixing benchmarks.

This gap motivates a simple question: *how well do current code agents perform beyond single-repository bug fixing?* To study this question, we introduce **BeyondSWE**, a benchmark designed around two practical dimensions. The first, *knowledge scope*, asks whether solving a task requires only

Emails: {gx.chen.chn, mengfanzhe16, marshmallowzjl, batmanfly, jiakai0419}@gmail.com, songruihua_bloon@outlook.com

Benchmark	Scope		Statistics		
	Resol.	Knowledge	#Repo	#Files	#Lines
SWE-bench-Verified	Local Func	Within Repo	12	1.3	11.6
SWE-bench-Live	Local Func	Within Repo	223	2.7	65.1
SWE-bench Pro	Local Func	Within Repo	41	4.1	107.4
CrossRepo	Local Func	Cross Repo	67	4.1	190.7
DomainFix	Local Func	Domain	12	4.2	157.6
DepMigrate	Global Repo	Official Docs	120	8.4	281.6
Doc2Repo	Global Repo	Human Spec	50	26.8	3528.4
BeyondSWE	**Mix**	**Mix**	**246**	**10.9**	**1039.6**

Table 1. Comparison with existing SWE benchmarks

repository-local information or also external software, domain, documentation, or specification knowledge. The second, *resolution scope*, asks whether the required solution is a localized fix, a repository-wide transformation, or full repository construction. Together, these dimensions allow us to stress-test capabilities that remain underrepresented in existing benchmarks while retaining executable, test-based evaluation.

Following this lens, we define four representative stress tests. *Cross-repository issue resolution (CrossRepo)* keeps the familiar issue-resolution format but requires agents to use information from related external repositories. *Domain-specific issue resolution (DomainFix)* tests whether agents can combine code reasoning with specialized knowledge from scientific and engineering domains. *Dependency-driven migration (DepMigrate)* evaluates repository-wide coordination under breaking changes in upstream dependencies. *Document-to-repository generation (Doc2Repo)* asks agents to construct a functional repository from a natural-language specification. These tasks comprise 500 instances drawn from 246 real-world GitHub repositories. Their target solutions affect an average of 10.9 files and 1039 lines per instance, substantially exceeding the modification scale of existing SWE-bench-style benchmarks. Our evaluation shows that BeyondSWE remains challenging even for strong coding agents. Under the OpenHands scaffold (Wang et al., 2025b), the best model reaches 46.12 average score, far from saturating the benchmark. The results expose task-specific weaknesses: CrossRepo stresses external software knowledge, DomainFix requires domain-specific reasoning, DepMigrate demands coordinated repository-level edits, and Doc2Repo often yields partial functionality without fully correct repositories.

Because several tasks require information beyond the local repository, we instantiate **SearchSWE** as a controlled diagnostic baseline for search-augmented coding. SearchSWE minimally extends a code-agent workflow with web search and fetch tools, allowing us to compare otherwise similar settings with and without external information access. The results indicate that search access for coding is genuinely useful: most models improve under SearchSWE, and the Codex harness using GPT-5.4 (xhigh) improves from 48.48 to 56.65 when explicitly prompted to integrate search into the coding workflow. Yet these gains remain limited and uneven, and our case studies show that agents often fail to turn retrieved information into precise, version-compatible, and locally actionable code changes. This suggests that deep search for coding remains an open problem: search and coding have each advanced rapidly, but current agents do not yet reliably synthesize them into robust search-augmented coding workflows.

To summarize, our contributions are as follows:

- We introduce BeyondSWE, a 500-instance benchmark for evaluating code agents beyond single-repository bug fixing, spanning four representative settings that vary in knowledge scope and resolution scope.

- We conduct a broad empirical evaluation of frontier and code-specialized agents, showing that current systems remain brittle across cross-repository reasoning, domain-specific repair, dependency migration, and repository generation.
- We use SearchSWE as a controlled diagnostic baseline to study deep search for coding, showing that search access is useful but insufficient: current agents still struggle to convert retrieved information into precise, version-compatible, and locally actionable code changes.

2. Related Work

SWE Benchmark. SWE-bench-Verified (Chowdhury et al., 2024) established executable GitHub-issue resolution as the dominant evaluation setting for code agents. Follow-up benchmarks improve this setting through live updates, multilingual coverage, decontamination, and harder issue instances (Amin et al., 2026; Badertdinov et al., 2025; Ding et al., 2025; Liu et al., 2025b,c; Rashid et al., 2025; Tian et al., 2024; Yang et al., 2025c; Zan et al., 2025b). Most preserve the core assumption that the target repository contains the main problem-solving context. BeyondSWE keeps executable test-based evaluation, but shifts the task distribution toward external knowledge requirements and broader resolution scopes.

Code agents and search. Recent frontier coding harnesses, including Codex and Claude Code (Anthropic, 2025; OpenAI, 2025a), increasingly expose web search or fetch capabilities as part of coding workflows. However, these production systems couple search with proprietary prompts, tool policies, model choices, and scaffolding, making it difficult to isolate how external information access affects coding performance. We therefore instantiate SearchSWE as a controlled search-augmented baseline, allowing us to study when search helps, when it remains insufficient, and why on BeyondSWE.

3. BeyondSWE

BeyondSWE is a benchmark for evaluating code agents on software engineering tasks that require broader knowledge or broader resolution scope than single-repository bug fixing. As shown in Figure 1, we use two practical dimensions to guide task selection. *Knowledge scope* distinguishes tasks whose relevant information is recoverable from the target repository from tasks that require external software artifacts, scientific domain knowledge, API documentation, or a specification. *Resolution scope* distinguishes localized fixes from repository-wide transformations and full repository construction. These dimensions serve as a design lens for stress-testing underrepresented capabilities while preserving executable, test-based evaluation. Overall, BeyondSWE contains 500 instances drawn from 246 GitHub repositories. The target solutions affect an average of 10.9 files and 1039.6 lines per instance, substantially exceeding the modification scale of existing SWE-bench-style benchmarks.

3.1. Task Formulation

Each instance contains three components. First, the *problem statement* specifies the task to be solved. For CrossRepo, DomainFix, and DepMigrate, it follows the GitHub issue-resolution format. For Doc2Repo, it is a natural-language specification of the target repository's API and behavior. Second, the *Docker environment* provides a reproducible runtime with the target repository, dependencies, and test commands. Third, the *test suite* defines executable success criteria. For CrossRepo, DomainFix, and DepMigrate, we follow the SWE-bench convention of pass-to-pass (P2P) tests that should remain passing and fail-to-pass (F2P) tests that should pass only after the correct fix is applied. For

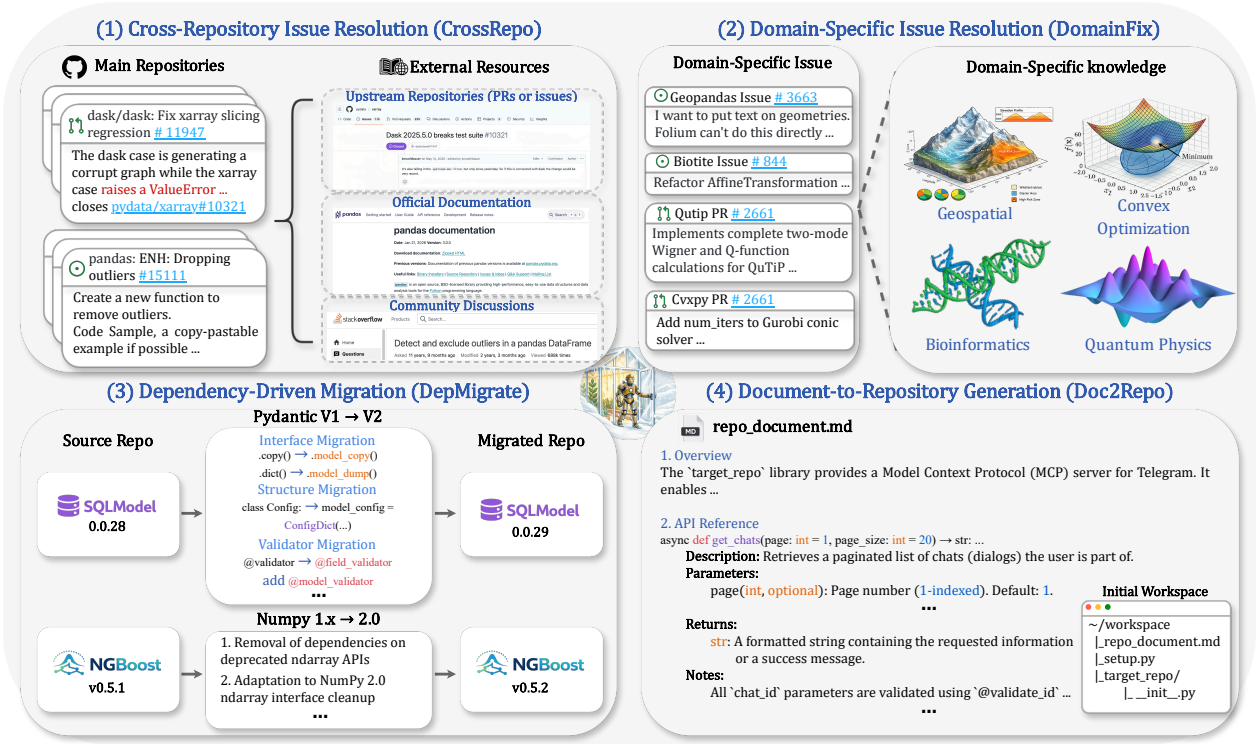


Figure 1. Overview of BeyondSWE. Our benchmark extends evaluation along two practical dimensions—*knowledge scope* and *resolution scope*: CrossRepo and DomainFix expand knowledge scope by requiring external software resources and domain expertise respectively; DepMigrate and Doc2Repo expand resolution scope from localized patches to codebase-wide transformations.

Doc2Repo, where the agent starts from an empty workspace, success is measured by the complete test suite for the generated repository.

3.2. Benchmark Tasks

Cross-repository issue resolution. CrossRepo keeps the familiar issue-resolution setting but requires agents to use information from related external repositories or linked artifacts. This setting reflects cases where a bug report, API behavior, or implementation pattern cannot be understood from the target repository alone. To construct the task, we scan Python-dominant GitHub repositories for merged pull requests containing external links and collect about 3,000 candidates. After environment construction and stability filtering, about 800 candidates remain. We manually verify that the external links are relevant to the issue and that the rewritten problem statement preserves the task context without revealing solution-specific details. This process yields 200 issues across 67 repositories, with an average of 1.3 external links per issue.

Domain-specific issue resolution. DomainFix evaluates whether agents can combine code reasoning with specialized knowledge from scientific and engineering domains. The target projects come from 11 research fields, including quantum physics, molecular dynamics, geospatial analysis, bioinformatics, and materials science. We collaborate with domain experts to identify 21 high-quality repositories and collect about 800 candidate pull requests. After executable environment construction, around 200 candidates pass stability inspection. Each remaining instance is independently reviewed by three domain experts for environmental correctness, genuine domain complexity, and solution non-triviality. Only unanimously accepted instances are retained. The final task contains

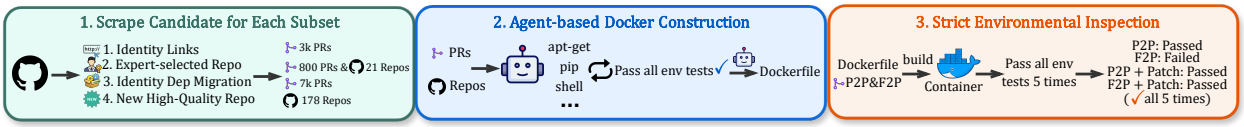


Figure 2. Environment Construction Pipeline for BeyondSWE. We collect task-specific candidates, build executable Docker environments through an agent-assisted setup process, and retain only instances whose tests exhibit stable fail-to-pass and pass-to-pass behavior across repeated runs.

72 issues across 12 repositories.

Dependency-driven migration. DepMigrate tests whether agents can coordinate repository-level edits under breaking changes in upstream dependencies. Unlike localized bug fixing, migration requires agents to understand an upstream API change, find affected call sites, and update the codebase consistently. We identify 23 widely used packages with significant version upgrades and collect pull requests whose descriptions or commit messages mention these packages and relevant version numbers. LLM-based filtering removes candidates that are not genuine migration efforts, producing about 7,000 candidates before environment construction. After stability inspection, about 1,000 candidates remain. Four software engineering experts then verify migration validity, producing 178 issues across 120 repositories. For each instance, the environment installs the upgraded dependency while the codebase is checked out before the migration patch, so agents must adapt the repository to the new dependency behavior.

Document-to-repository generation. Doc2Repo evaluates repository construction from a specification rather than repair of an existing codebase. Agents receive a natural-language document describing the intended API and behavior, and must create a complete repository from an empty workspace. To reduce contamination risk, we collect Python repositories created between January and November 2025, require continued activity after August 2025, and retain projects with at least three contributors and more than 20 stars. For each repository, Gemini 3 Pro (Google, 2025b) explores the codebase and generates a specification covering purpose, usage examples, public classes and functions, parameters, return types, and behavior, while removing implementation details and directory structure. We mask repository names with `target_repo` and require agents to infer the structure from contextual cues such as import paths. We adapt tests from the original repositories with LLM assistance and human review. After environment construction, 60 candidates remain, from which we select 50 high-quality instances.

3.3. Environment Construction

Reliable evaluation requires runnable historical environments, but many real repositories suffer from dependency decay, unavailable packages, deprecated APIs, and missing system libraries. As shown in Figure 2, we use an agent-assisted process to construct each Docker environment. The agent starts from a base Ubuntu container, clones the repository, checks out the pre-PR commit, and iteratively resolves setup failures until the existing tests can be executed. The agent can use shell commands to install missing packages, compilers, and libraries, which are often absent from repository-level dependency files. We then distill the successful command history into a reproducible `Dockerfile`.

Each generated environment undergoes strict stability inspection. We build the Docker image and execute the relevant tests five times. For CrossRepo, DomainFix, and DepMigrate, we require P2P tests to pass and F2P tests to fail before applying the reference patch, and require both sets to pass after applying the patch. Instances with flaky behavior, incomplete setup, or invalid fail-to-pass

transitions are discarded. For Doc2Repo, the environment is validated by running the adapted test suite against the reference implementation and by manually auditing tests whose expected behavior depends on the generated specification.

3.4. Evaluation Protocol

We separate agent execution from final verification so that scores reflect the submitted code changes rather than artifacts of the agent's workspace. During evaluation, an agent works in its own Docker environment and produces a patch or a generated repository. We then extract the resulting changes and apply them to a fresh container that is independent of the agent's workspace. This prevents environmental side effects, cached artifacts, or modified local configuration from affecting the final score.

We also apply integrity safeguards against solution leakage and test manipulation. Following concerns raised in prior benchmark analyses (Xiao et al., 2026), we remove git commits, logs, and metadata after the target commit while retaining earlier history that a developer could realistically inspect. After applying an agent patch, we restore all test files to their original state before running evaluation. Thus, success must come from changing the target implementation rather than editing tests or exploiting future repository history.

For CrossRepo, DomainFix, and DepMigrate, we report **Resolved Rate**, the percentage of instances where all P2P and F2P tests pass in the fresh container. For Doc2Repo, we report **Pass Rate**, the average percentage of tests passed, and **(Almost) Correct Count**, the number of repositories that pass all tests or at least 90% of tests.

3.5. Quality Control and Human Verification

BeyondSWE combines automated filtering with task-specific human verification. Automated filtering removes candidates with unstable environments, invalid test transitions, flaky behavior, or solution-revealing problem statements. Human review checks whether retained tasks represent genuine SWE challenges rather than artifacts of environment setup or underspecified tests.

The verification process uses task-appropriate expertise. DomainFix uses domain experts to confirm that each retained issue requires specialized scientific or engineering knowledge beyond general programming. DepMigrate uses software engineering experts to verify that each instance corresponds to a real dependency migration rather than an incidental version bump. Across all tasks, five senior software engineers inspect environment construction and data cleaning, and five senior PhD researchers in software engineering and LLMs audit final task quality. For Doc2Repo, tests are additionally reviewed to ensure that the specification and expected behavior are aligned; when a test reflects stricter behavior than the original repository, the requirement is documented in the instance metadata. Detailed reviewer backgrounds, agreement criteria, rejection reasons, and compensation information are reported in the appendix.

4. Search-Augmented Diagnostic Baseline

Several BeyondSWE tasks require information that may not be present in the target repository. CrossRepo may depend on related implementations or upstream discussions, DomainFix may require specialized scientific knowledge, and DepMigrate may require migration guides or dependency documentation. This motivates a controlled diagnostic question: *when a code agent has access to external information, does it reliably turn that information into correct code changes?*

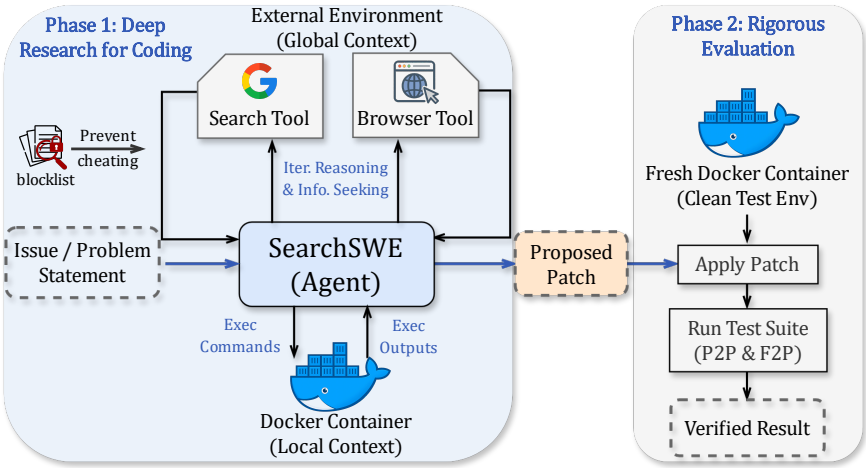


Figure 3. Overview of SearchSWE as a controlled search-augmented coding baseline. The agent retains the local coding workflow, while adding web search and fetch tools under a blocklist that prevents direct access to solution-revealing target-repository artifacts.

We instantiate SearchSWE to study this question. SearchSWE minimally extends a standard code-agent workflow (Wang et al., 2025b) with two external-information tools: a *search tool* that queries the web for potentially relevant resources, and a *browser tool* that fetches and summarizes the content of a specified webpage. This design lets us compare otherwise similar settings with and without external information access, isolating the effect of search augmentation on coding performance.

Controlled access. To keep the comparison focused, SearchSWE exposes web search and fetch as general tools rather than task-specific retrieval modules. The agent decides when to search, what query to issue, which result to inspect, and how to integrate retrieved evidence into local implementation and verification. This setup is deliberately simple: performance improvements are informative, but the primary goal is to diagnose whether current agents can combine search with coding under realistic repository constraints.

Cheating prevention. Because BeyondSWE instances are derived from real repositories, unrestricted web access could allow an agent to retrieve the original issue or pull request. We therefore implement a blocklist over both search results and shell commands. The blocklist filters URLs and operations matching the target repository across GitHub, GitLab, raw-content endpoints, API endpoints, and direct git operations. It also blocks solution-revealing artifacts associated with the target instance. These safeguards force agents to use indirect external evidence, such as documentation, related projects, or general technical resources, rather than directly copying the gold solution. With this setup, the experiments allow us to attribute performance changes to the availability and use of external information.

5. Experiments

5.1. Experimental Setup

We evaluate current code agents on BeyondSWE under four settings. OpenHands (Wang et al., 2025b) provides a standard open-source code-agent scaffold. SearchSWE is used as a controlled diagnostic baseline that keeps the same local coding workflow while adding web search and fetch tools. We also evaluate the Codex (OpenAI, 2025a) harness (`v0.118.0`) in two prompt settings: its default coding prompt and a SearchSWE-style prompt that explicitly encourages search-aware

Model	CrossRepo	DomainFix	DepMigrate	Doc2Repo		AVG
	%Resolved	%Resolved	%Resolved	Pass Rate	#(Alm.) Corr.	
Codex						
GPT-5.4 (xhigh) w/ SearchSWE Prompt	**55.17** +8.9	**61.11** +19.4	**48.59** +4.2	**61.74** +0.1	**(7) / 2**	**56.65** +8.2
GPT-5.4 (xhigh) w/ Default Prompt	46.23	41.67	44.38	61.64	**(7) / 2**	48.48
OpenHands						
DeepSeek-V4-Pro(Max) (DeepSeek-AI, 2026)	44.00	**38.89**	**44.38**	**57.20**	(5) / 1	**46.12**
GLM-5 (Zeng et al., 2026)	**44.67**	33.33	42.13	56.76	(7) / 3	44.22
Qwen3.5-Plus (Qwen Team, 2026)	41.50	**38.89**	41.01	52.41	(3) / 1	43.45
Gemini 3 Pro (Google, 2025b)	41.50	31.94	41.81	52.03	**(8) / 2**	41.82
GPT-5.4 (OpenAI, 2026)	43.00	27.78	37.50	56.30	(5) / 3	41.15
Kimi-K2.5 (Team et al., 2026)	40.50	34.72	39.89	51.36	(3) / 1	41.62
Seed-Coder-2.0(Seed et al., 2025)	39.50	24.29	33.15	55.54	(5) / 2	38.12
MiniMax-M2.5 (MiniMax, 2026)	40.00	25.00	37.64	46.57	(5) / 1	37.30
SearchSWE						
DeepSeek-V4-Pro(Max) (DeepSeek-AI, 2026)	**48.50** +4.5	**43.06** +4.2	**47.19** +2.8	56.16 -1.0	(5) / 2	**48.73** +2.6
GLM-5 (Zeng et al., 2026)	43.88 -0.8	35.94 +2.6	47.13 +5.0	**60.05** +3.3	(7) / 3	46.75 +2.5
Qwen3.5-Plus (Qwen Team, 2026)	41.50	34.72 -4.2	39.89 -1.1	54.90 +2.5	(3) / 1	42.75 -0.7
Gemini 3 Pro (Google, 2025b)	41.12 -0.4	39.44 +7.5	44.07 +2.3	50.73 -1.3	(4) / 2	43.84 +2.0
GPT-5.4 (OpenAI, 2026)	45.00 +2.0	31.94 +4.2	38.76 +1.3	58.35 +2.1	**(7) / 4**	43.51 +2.4
Kimi-K2.5 (Team et al., 2026)	43.00 +2.5	34.72	37.64 -2.3	52.22 +0.9	(5) / 2	41.90 +0.3
Seed-Coder-2.0 (Seed et al., 2025)	43.15 +3.7	30.43 +6.1	35.80 +2.7	49.67 -5.9	(3) / 1	39.76 +1.6
MiniMax-M2.5 (MiniMax, 2026)	39.00 -1.0	31.94 +6.9	37.08 -0.6	50.66 +4.1	(3) / 0	39.67 +2.4

Table 2. Main Results on BeyondSWE. We organize the results by three evaluation settings: Codex, OpenHands, and SearchSWE. For Codex, we report both the default prompt setting and the SearchSWE-style prompt setting to highlight the sensitivity of frontier search-code agents to prompt design. Red/green values indicate gains/drops relative to the matched no-search setting: the default prompt for Codex and OpenHands for SearchSWE. **Bold** and underlined denote the best and second-best results across each evaluation settings.

coding. The average score (AVG) is the mean of the four task-level scores, using Doc2Repo pass rate as its task score. Table 2 summarizes the main results.

5.2. Overall Performance on BeyondSWE

How close are current agents to solving BeyondSWE? The main result is that BeyondSWE remains far from saturated even for strong contemporary coding agents. Under the OpenHands scaffold, the best model, DeepSeek-V4-Pro (Max), reaches only 46.12 average score. The Codex harness with GPT-5.4 (xhigh) performs better, reaching 48.48 with the default prompt, but this still leaves substantial headroom across all four task families. When prompted with the SearchSWE-style workflow, the same Codex configuration reaches the strongest overall result, 56.65, yet the benchmark is still not close to being solved.

The task-level pattern shows that BeyondSWE is not difficult for a single reason. CrossRepo stresses external software knowledge, DomainFix requires domain-specific reasoning, and DepMigrate demands coordinated repository-level edits. Doc2Repo exposes a different limitation: its pass rate can overstate complete success, since even the best configuration produces only 2 fully correct repositories out of 50. Overall, current agents often make meaningful partial progress, but still fail to deliver reliable end-to-end solutions across broader knowledge and resolution scopes.

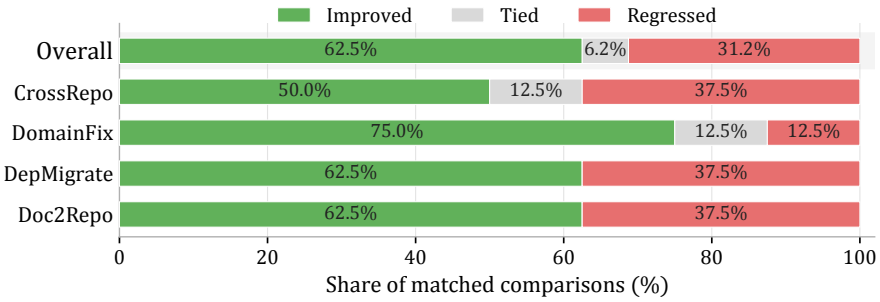


Figure 4. Task-wise decomposition of SearchSWE outcomes relative to OpenHands. Each task row aggregates the eight matched model pairs. The Overall row aggregates all 32 task-model comparisons.

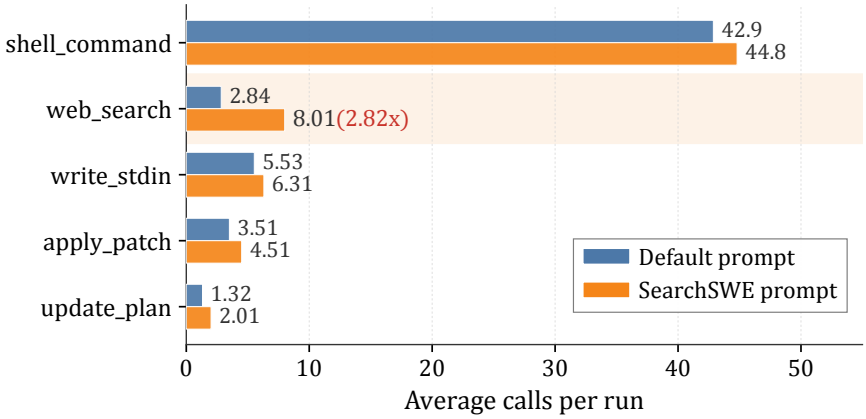


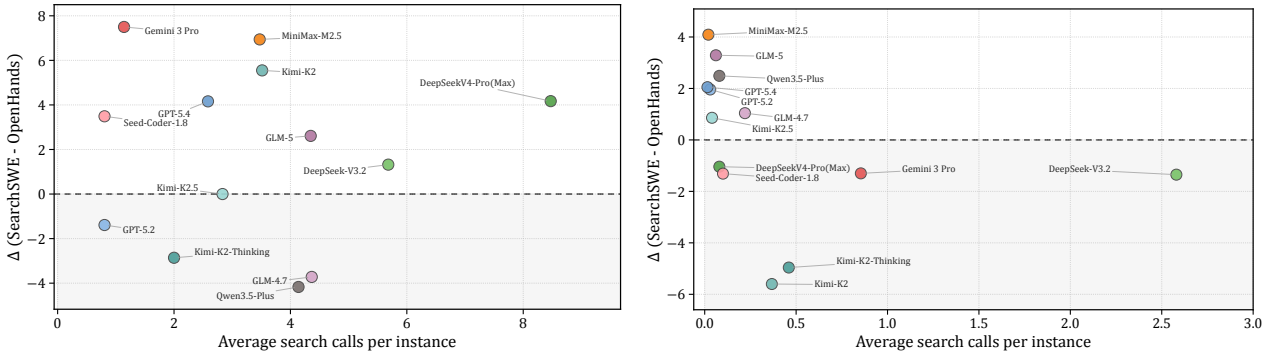
Figure 5. Average tool calls per instance for Codex with GPT-5.4 (xhigh) under the default prompt and the SearchSWE-style prompt. The search-oriented prompt substantially increases web-search use while keeping local coding-tool usage comparable.

5.3. Search Helps, but Integration Matters

Does external search access improve coding performance? The answer is broadly yes, but the gains are limited and harness-dependent. Compared with OpenHands, SearchSWE improves seven of eight evaluated models, with the strongest SearchSWE configuration reaching 48.73. At the task level, 20 of 32 paired comparisons improve, while 31.2% regress, showing that search access is useful but not uniformly reliable. Figure 4 decomposes these outcomes by task. Overall, DomainFix benefits most, consistent with its reliance on externally retrievable domain knowledge, whereas CrossRepo, DepMigrate, and Doc2Repo retain substantial regressions.

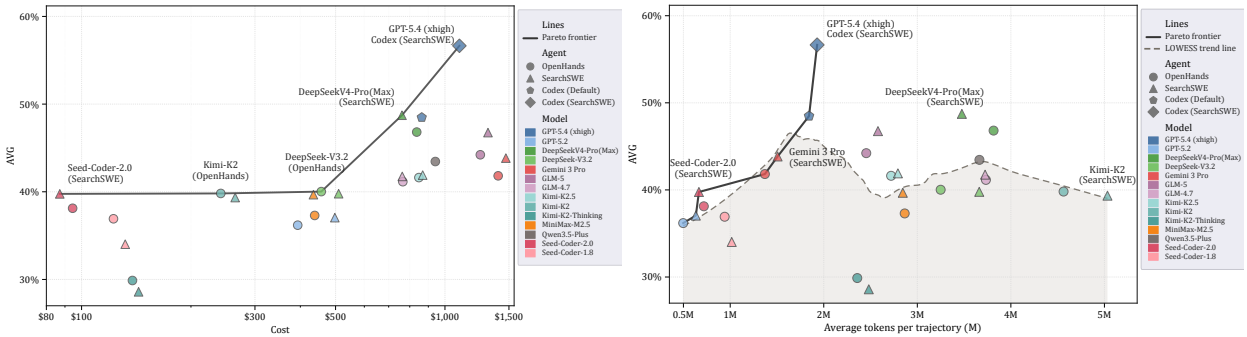
The Codex comparison complements this aggregate view. Using the same GPT-5.4 (xhigh) model and harness, the SearchSWE-style prompt improves AVG from 48.48 to 56.65, with gains concentrated on DomainFix (+19.4) and CrossRepo (+8.9). Figure 5 shows that the prompt increases average web-search calls by about 2.8× while preserving a similar local coding-tool profile. This suggests that search access is useful but not self-activating: frontier harnesses may still need explicit guidance to turn external information into local, version-compatible code changes.

Is more search necessarily better? Figure 6 compares search-related calls with SearchSWE gains on DomainFix and Doc2Repo, the tasks with the strongest and weakest average search gains. DomainFix contains many positive-gain points, while Doc2Repo remains clustered around small or negative gains. Within each task, more search calls do not monotonically imply larger gains.



(a) DomainFix: largest average gain. (b) Doc2Repo: weakest average gain.

Figure 6. Search effort versus SearchSWE gain on two contrasting tasks. Each point is an evaluated model with matched OpenHands and SearchSWE runs. The y-axis shows task-score of SearchSWE delta over OpenHands. Within each task, more search calls do not monotonically imply larger gains.



(a) All Cost ($) vs. Performance. (b) Average tokens (M) vs. Performance.

Figure 7. Cost and token-budget efficiency on BeyondSWE. Solid black lines denote the Pareto frontier among evaluated configurations. In the token plot, the dashed LOWESS curve summarizes the local performance trend: points below it are less token-productive than configurations with similar token budgets, and the high-token flattening shows that longer trajectories alone do not reliably improve performance.

This suggests that the bottleneck is not how often agents search, but whether they retrieve relevant evidence and ground it in the local task context.

Why search remains insufficient. The qualitative cases in Appendix B identify three failure sites in the search-to-code pipeline: evidence retrieved at the wrong granularity, external knowledge not grounded in the local dependency version, and keyword-matched but semantically unrelated results contaminating the coding context. Thus, the limitation is not access to search alone, but the ability to decide which external evidence is trustworthy and locally actionable. Robust search-augmented coding still requires source discrimination, version grounding, and local verification before retrieved evidence becomes a code change.

5.4. Cost and Token Budget Are Not Enough

Are stronger results simply a consequence of spending more money or tokens? Figure 7 suggests not. Higher API cost does not reliably correspond to higher performance: some lower-cost configurations, such as Seed-Coder-2.0 (Seed et al., 2025) under SearchSWE, remain competitive,

while several high-cost configurations do not lie on the performance frontier. This indicates that benchmark performance is not merely purchased through larger inference budgets.

The token-performance view shows a similar pattern from trajectory length. The Pareto frontier highlights the token-efficient configurations: many higher-token runs fall below this frontier without corresponding performance gains. The LOWESS trend rises in the low-token region but flattens or declines at higher token budgets, indicating that longer trajectories do not reliably translate into better task resolution. Many high-token runs appear to spend budget on repeated exploration, noisy search, or ineffective repair loops rather than successful fixes. Taken together, these results point to token productivity rather than token volume as the key bottleneck. More budget can help when paired with a strong model, scaffold, and search-aware workflow, but current agents do not automatically convert additional tokens into grounded search, correct local reasoning, and targeted edits.

6. Conclusion

We introduced BeyondSWE, a 500-instance benchmark for evaluating code agents beyond single-repository bug fixing, spanning cross-repository reasoning, domain-specific repair, dependency migration, and repository construction from specifications. Our evaluation shows that stronger models and harnesses improve performance, but current agents remain far from saturating these settings. Using SearchSWE as a controlled diagnostic baseline, we find that external search is genuinely useful, especially when missing knowledge is externally retrievable, but its gains are uneven and depend on whether agents can ground retrieved evidence in the local repository, dependency version, and task specification. These results suggest that deep search for coding remains an open problem: progress requires agents that not only retrieve information, but also decide when to trust it, when to ignore it, and how to convert it into precise, executable code changes.

References

N. Amin, Z. Fei, X. Li, J. Petke, and H. Ye. Jmigbench: A benchmark for evaluating llms on source code migration (java 8 to java 11). *arXiv preprint arXiv:2602.09930*, 2026.

Anthropic. Introducing web search on the anthropic api. https://www.anthropic.com/news/web-search-api, 2025.

I. Badertdinov, A. Golubev, M. Nekrashevich, A. Shevtsov, S. Karasik, A. Andriushchenko, M. Trofimova, D. Litvintseva, and B. Yangel. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=nMpJoVmRy1.

G. Chen, M. Liao, P. Yu, D. Wang, Z. Qiao, C. Yang, X. Zhao, and K. Fan. C-3PO: Compact plug-and-play proxy optimization to achieve human-like retrieval-augmented generation. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=hlpwAmQ4wr.

G. Chen, Z. Qiao, X. Chen, D. Yu, H. Xu, X. Zhao, R. Song, W. Yin, H. Yin, L. Zhang, K. Li, M. Liao, Y. Jiang, P. Xie, F. Huang, and J. Zhou. Iterresearch: Rethinking long-horizon agents with interaction scaling. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=qQ5MZ5Mx7p.

N. Chowdhury, J. Aung, C. J. Shern, O. Jaffe, D. Sherburn, G. Starace, E. Mays, R. Dias, M. Aljubeh, M. Glaese, et al. Introducing swe-bench verified. *arXiv preprint arXiv:2407.01489*, 2024.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro.

X. Deng, J. Da, E. Pan, Y. Y. He, C. Ide, K. Garg, N. Lauffer, A. Park, N. Pasari, C. Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

J. Ding, S. Long, C. Pu, H. Zhou, H. Gao, X. Gao, C. He, Y. Hou, F. Hu, Z. Li, et al. Nl2repo-bench: Towards long-horizon repository generation evaluation of coding agents. *arXiv preprint arXiv:2512.12730*, 2025.

Google. Deep research is now available on gemini 2.5 pro experimental, 2025a. URL https://blog.google/products/gemini/deep-research-gemini-2-5-pro-experimental/.

Google. Gemini 3 pro, 2025b. URL https://deepmind.google/models/gemini/pro/.

L. Guo, Y. Wang, C. Li, P. Yang, J. Chen, W. Tao, Y. Zou, D. Tang, and Z. Zheng. Swe-factory: Your automated factory for issue resolution training data and evaluation benchmarks. *arXiv preprint arXiv:2506.10954*, 2025. URL https://arxiv.org/abs/2506.10954.

Z. He, Q. Yang, W. Sheng, X. Zhong, K. Zhang, C. An, W. Shi, T. Cai, D. He, J. Chen, and J. Xu. Swe-swiss: A multi-task fine-tuning and rl recipe for high-performance issue resolution, 2025. URL https://www.notion.so/SWE-Swiss-A-Multi-Task-Fine-Tuning-and-RL-Recipe-for-High-Performance-Issue-Resolution-21e174dedd4880ea829ed4c861c44f88. Notion Blog.

N. Jain, J. Singh, M. Shetty, L. Zheng, K. Sen, and I. Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*, 2025.

C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. 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.

A. Liu, A. Mei, B. Lin, B. Xue, B. Wang, B. Xu, B. Wu, B. Zhang, C. Lin, C. Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025a.

J. Liu, C. Huang, Z. Guan, W. Lei, and Y. Deng. E2edev: Benchmarking large language models in end-to-end software development task. *arXiv preprint arXiv:2510.14509*, 2025b.

L. Liu, X. Liu, Q. Zhou, L. Chen, Y. Liu, H. Nguyen, B. Omidvar-Tehrani, X. Shen, J. Huan, O. Tripp, et al. Migrationbench: Repository-level code migration benchmark from java 8. *arXiv preprint arXiv:2505.09569*, 2025c.

J. J. Ma, M. Hashemi, A. Yazdanbakhsh, K. Swersky, O. Press, E. Li, V. J. Reddi, and P. Ranganathan. Swe-fficiency: Can language models optimize real-world repositories on real workloads? *arXiv preprint arXiv:2511.06090*, 2025.

MiniMax. MiniMax M2.5: Built for real-world productivity, Feb. 2026. URL https://www.minimax.io/news/minimax-m25.

OpenAI. Deep research system card, 2025. URL https://cdn.openai.com/deep-research-system-card.pdf.

OpenAI. Introducing upgrades to codex. https://openai.com/index/introducing-upgrades-to-codex/, 2025a.

OpenAI. Introducing GPT-5.2, Dec. 2025b. URL https://openai.com/index/introducing-gpt-5-2/.

OpenAI. Introducing GPT-5.4, Mar. 2026. URL https://openai.com/index/introducing-gpt-5-4/.

A. Orwall. Moatless tools: A framework for repository-level code understanding and editing. https://github.com/aorwall/moatless-tools, 2024.

Perplexity. Introducing perplexity deep research, 2025. URL https://www.perplexity.ai/hub/blog/introducing-perplexity-deep-research.

Z. Qiao, G. Chen, X. Chen, D. Yu, W. Yin, X. Wang, Z. Zhang, B. Li, H. Yin, K. Li, et al. Webresearcher: Unleashing unbounded reasoning capability in long-horizon agents. *arXiv preprint arXiv:2509.13309*, 2025.

Qwen Team. Qwen3-coder: Agentic coding in the world. https://qwenlm.github.io/blog/qwen3-coder/, Jul 22 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, Feb. 2026. URL https://qwen.ai/blog?id=qwen3.5.

M. S. Rashid, C. Bock, Y. Zhuang, A. Buchholz, T. Esler, S. Valentin, L. Franceschi, M. Wistuba, P. T. Sivaprasad, W. J. Kim, et al. Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents. *arXiv preprint arXiv:2504.08703*, 2025.

B. Seed, Y. Zhang, J. Su, Y. Sun, C. Xi, X. Xiao, S. Zheng, A. Zhang, K. Liu, D. Zan, et al. Seed-coder: Let the code model curate data for itself. *arXiv preprint arXiv:2506.03524*, 2025.

C. Tao, J. Chen, Y. Jiang, K. Kou, S. Wang, R. Wang, X. Li, S. Yang, Y. Du, J. Dai, et al. Swe-lego: Pushing the limits of supervised fine-tuning for software issue resolving. *arXiv preprint arXiv:2601.01426*, 2026.

K. Team, Y. Bai, Y. Bao, G. Chen, J. Chen, N. Chen, R. Chen, Y. Chen, Y. Chen, Y. Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025a.

K. Team, T. Bai, Y. Bai, Y. Bao, S. Cai, Y. Cao, Y. Charles, H. Che, C. Chen, G. Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

T. D. Team, B. Li, B. Zhang, D. Zhang, F. Huang, G. Li, G. Chen, H. Yin, J. Wu, J. Zhou, et al. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*, 2025b.

M. Tian, L. Gao, S. Zhang, X. Chen, C. Fan, X. Guo, R. Haas, P. Ji, K. Krongchon, Y. Li, et al. Scicode: A research coding benchmark curated by scientists. *Advances in Neural Information Processing Systems*, 37:30624–30650, 2024.

J. Wang, D. Zan, S. Xin, S. Liu, Y. Wu, and K. Shen. Swe-mirror: Scaling issue-resolving datasets by mirroring issues across repositories. *arXiv preprint arXiv:2509.08724*, 2025a.

X. Wang, S. Rosenberg, J. Michelini, C. Smith, H. Tran, E. Nyst, R. Malhotra, X. Zhou, V. Chen, R. Brennan, et al. The openhands software agent sdk: A composable and extensible foundation for production agents. *arXiv preprint arXiv:2511.03690*, 2025b.

Y. Wei, O. Duchenne, J. Copet, Q. Carbonneaux, L. Zhang, D. Fried, G. Synnaeve, R. Singh, and S. I. Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv preprint arXiv:2502.18449*, 2025.

C. S. Xia, Y. Deng, S. Dunn, and L. Zhang. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

B. Xiao, B. Xia, B. Yang, B. Gao, B. Shen, C. Zhang, C. He, C. Lou, F. Luo, G. Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. R. Narasimhan, and O. Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=mXpq6ut8J3.

J. Yang, S. Guo, L. Jing, W. Zhang, A. Liu, C. Hao, Z. Li, W. X. Zhao, X. Liu, W. Lv, et al. Scaling laws for code: Every programming language matters. *arXiv preprint arXiv:2512.13472*, 2025b.

J. Yang, C. E. Jimenez, A. L. Zhang, K. Lieret, J. Yang, X. Wu, O. Press, N. Muennighoff, G. Synnaeve, K. R. Narasimhan, D. Yang, S. Wang, and O. Press. SWE-bench multimodal: Do AI systems generalize to visual software domains? In *The Thirteenth International Conference on Learning Representations*, 2025c. URL https://openreview.net/forum?id=riTiq3i21b.

J. Yang, K. Lieret, C. E. Jimenez, A. Wettig, K. Khandpur, Y. Zhang, B. Hui, O. Press, L. Schmidt, and D. Yang. Swe-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025d.

Z. Yang, S. Wang, K. Fu, W. He, W. Xiong, Y. Liu, Y. Miao, B. Gao, Y. Wang, Y. Ma, et al. Kimi-dev: Agentless training as skill prior for swe-agents. *arXiv preprint arXiv:2509.23045*, 2025e.

Z.ai Team. GLM-4.7: Advancing the coding capability, Dec. 2025. URL https://z.ai/blog/glm-4.7.

D. Zan, Z. Huang, W. Liu, H. Chen, S. Xin, L. Zhang, Q. Liu, A. Li, L. Chen, X. Zhong, S. Liu, Y. Xiao, L. Chen, Y. Zhang, J. Su, T. Liu, R. LONG, M. Ding, and liang xiang. Multi-SWE-bench: A multilingual benchmark for issue resolving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025a. URL https://openreview.net/forum?id=MhBZzkz4h9.

D. Zan, Z. Huang, W. Liu, H. Chen, L. Zhang, S. Xin, L. Chen, Q. Liu, X. Zhong, A. Li, et al. Multi-swe-bench: A multilingual benchmark for issue resolving. *arXiv preprint arXiv:2504.02605*, 2025b.

A. Zeng, X. Lv, Z. Hou, Z. Du, Q. Zheng, B. Chen, D. Yin, C. Ge, C. Huang, C. Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

L. Zhang, S. He, C. Zhang, Y. Kang, B. Li, C. Xie, J. Wang, M. Wang, Y. Huang, S. Fu, E. Nallipogu, Q. Lin, Y. Dang, S. Rajmohan, and D. Zhang. SWE-bench goes live! In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=OGWkr7gXka.

J. Zhao, G. Chen, F. Meng, M. Li, J. Chen, H. Xu, Y. Sun, X. Zhao, R. Song, Y. Zhang, et al. Immersion in the github universe: Scaling coding agents to mastery. *arXiv preprint arXiv:2602.09892*, 2026.

Contents of Appendix

Failure Mode	Instance	Search Behavior	Local Failure	Takeaway
Evidence availability	unidata_siphon_pr234	Search returns high-level documentation instead of source-level backend logic	Agent implements a brittle interpretation of an ambiguous API description	Search must recover evidence at the right technical granularity
Version grounding	behave_behave-django_pr162	Agent searches around an unsupported newer Django assumption instead of local constraints	Agent applies a method pattern incompatible with the repository's legacy test lifecycle	External evidence must be filtered through local dependency versions
Evidence filtering	abravalheri_validate-pyproject_pr105	Keyword-matched results drift into unrelated domains due to overloaded terminology	Agent falls back to a generic plugin-registration pattern that creates test side effects	Agents need semantic source discrimination, not only retrieval

Table 3. Summary of search-augmented coding failure modes analyzed in the case studies.

A. Additional Related Work

Deep research agents. Recent deep research systems extend LLM agents with iterative web search, browsing, source synthesis, and long-horizon information gathering (Chen et al., 2025, 2026; Google, 2025a; OpenAI, 2025; Perplexity, 2025; Qiao et al., 2025; Team et al., 2025b). These systems demonstrate strong progress in open-domain research workflows, but their objectives are usually information synthesis rather than producing executable code changes under repository-local constraints. BeyondSWE uses search-augmented coding to study a different setting: retrieved information must be converted into precise, version-compatible, and test-passing code edits.

Code agents and agent training. A parallel line of work improves code agents through stronger scaffolds, agent-computer interfaces, trajectory data, supervised fine-tuning, and reinforcement learning (Guo et al., 2025; He et al., 2025; Orwall, 2024; Tao et al., 2026; Wang et al., 2025b; Xia et al., 2024; Yang et al., 2024, 2025d; Zhao et al., 2026). Most of this work optimizes agents for repository-local issue resolution, especially SWE-bench-style tasks (Jain et al., 2025; Ma et al., 2025; Wang et al., 2025a; Wei et al., 2025; Yang et al., 2025b,e). Our analysis complements these efforts by evaluating whether agents can combine local coding with external information seeking across tasks that require broader knowledge or broader resolution scope.

B. Qualitative Analysis of Search-Augmented Coding Failures

This section examines why external search does not always translate into better coding performance. The three cases below are not isolated anecdotes; they illustrate recurring breaks in the pipeline from external evidence to local code changes. Table 3 summarizes the failure modes.

At a high level, search-augmented coding fails in three places. First, *the relevant evidence may not be what search engines rank highly*: agents may receive user-facing documentation when they need source-level implementation logic. Second, *retrieved information must be grounded in the local repository state*: a correct pattern for a newer library version can be wrong for the pinned environment. Third, *agents must filter semantically irrelevant results*: keyword overlap can pull in authoritative but unrelated sources that contaminate the coding context. Together, these cases explain why search access is useful but insufficient on BeyondSWE.

B.1. Failure Mode I: Source-Level Evidence Is Hard to Retrieve

Failure mode. Some coding tasks require low-level artifacts such as source files, commit diffs, or backend implementation logic, but web search often prioritizes curated, user-facing documentation. The result may be conceptually relevant but too imprecise to support a robust code change.

Case. In `unidata_siphon_pr234`, the agent needs to extend `IAStateUpperAir` so it can fetch data for all stations at once. The behavior is enabled by an update in the Iowa Environmental Mesonet backend, but the target repository does not contain the backend implementation. To implement the change correctly, the agent needs to understand the exact parameter handling expected by the backend CGI script.

What went wrong. The agent correctly recognizes that local context is insufficient and searches for the backend artifact using a query targeting `akrherz/iem` and `raob.py`. However, as shown in Table 4, search returns a high-level API help page rather than the backend source code. The help page says the service can be queried with "just a timestamp," which is directionally useful but syntactically ambiguous: it does not specify whether to remove the `station` parameter, set it to `None`, pass a wildcard, or follow additional backend-specific error-handling logic.

Component	Agent's Intent / Target	Search Engine Response
Target Artifact	**Backend Source Code** (`akrherz/iem/.../raob.py`)	**User-Facing Documentation** (`.../json/raob.py?help`)
Information Type	**Precise Logic** Explicit conditionals, error handling, and parameter parsing logic.	**Ambiguous Natural Language** High-level description: "approach this service... with **just a timestamp**."
Resulting Action	*(Hypothetical)* Implement robust parameter negotiation mirroring backend logic.	*(Actual)* Implemented a brittle solution based on literal interpretation of "just a timestamp".

Table 4. Analysis of the Information Landscape Mismatch in `unidata_siphon_pr234`. The search engine's bias towards curated content obscures the precise logic required for robust code implementation.

The agent then implements the most literal interpretation of the documentation by omitting the station parameter in selected cases. This solves part of the intended behavior, but it misses edge-case handling encoded in the source-level backend logic. The resulting patch fails comprehensive tests such as `test_no_future_data_with_pressure_iastate`.

Lesson. The failure is not simply that search failed to find anything relevant. Rather, search found evidence at the wrong granularity, and the agent did not recognize that the retrieved documentation was insufficient for implementation. Search-augmented coding therefore requires agents to judge whether retrieved evidence is precise enough for source-level changes.

B.2. Failure Mode II: Retrieved Knowledge Must Be Version-Grounded

Failure mode. External information can conflict with the version constraints of the local repository. For maintenance tasks, the right answer is not necessarily the newest API pattern, but the pattern compatible with the installed dependency versions and the repository's existing architecture.

Case. In `behave_behave-django_pr162`, the agent must fix a fixture-loading issue in a repository pinned to legacy Django versions. The relevant constraints are available locally through project con-

figuration and installed packages. The task therefore requires grounding any external information in the actual environment before editing code.

What went wrong. Instead of first verifying the installed Django version, the agent forms an unsupported assumption about a newer Django setting and searches for evidence around that assumption. As Table 5 shows, this pushes the agent toward a modern class-method pattern for lifecycle hooks. That pattern conflicts with the local repository, where `_pre_setup` participates in a legacy instance-method lifecycle.

Context Source	Code Pattern & Logic	Status
Local Environment (Django 2.2/3.x)	**Instance Method (Legacy)** `def _pre_setup(self): ...` *Constraint: Must maintain state on the test instance.*	**Ignored**
Agent Search (Hallucinated "v5.2")	**Class Method (Modern/Future)** `@classmethod` `def _pre_setup(cls): ...` *Bias: Assumes newer patterns apply universally.*	**Prioritized**
Implementation (The Error)	**Signature Mismatch** Agent applies `@classmethod` to legacy hook. *Result: Breaks MRO and instance state access.*	✗

Table 5. Analysis of Version Conflict in `behave_behave-django_pr162`. The agent prioritizes hallucinated future specifications over explicit local constraints.

The implementation then changes `_pre_setup` into a `@classmethod`. This breaks the expected interaction with instance-level state and the inherited test lifecycle, causing the test suite to fail. The key error is not merely a bad search query; it is the absence of a local-version check before applying retrieved or recalled patterns.

Lesson. Search for coding is not a search for the most recent answer. It is a search for evidence compatible with the local environment. Agents need to treat local dependency versions, installed APIs, and inherited code structure as constraints that filter external knowledge.

B.3. Failure Mode III: Keyword Matches Can Contaminate Context

Failure mode. Technical terms are often overloaded across domains. For niche libraries, a reasonable query can retrieve high-ranking results that match the keywords but not the repository's semantic context. If an agent fails to reject these results, search becomes a source of context contamination.

Case. In abravalheri_validate-pyproject_pr105, the agent must implement support for `repo-review`, where checks are grouped into "families." The agent searches for `"repo-review define checks fixtures families"`, a query that is reasonable given the task wording.

What went wrong. As shown in Table 6, the first result is relevant, but later results drift into unrelated domains such as Autodesk Revit and RelativityOne. The drift is caused by overloaded terms such as "family" and "review," combined with the low web footprint of the target library. Rather than isolating the single relevant source and discarding unrelated entries, the agent fails to recover a precise integration pattern.

Search Result Title	Domain	Snippet Content	Rel.
Families - repo-review 0.12.4.dev15 ...	**Target**	"Families are a set of simple strings that group together similar checks..."	✓
Writing Family Instances with the Revit Writer	Architecture (BIM)	"...highlight some notable points on **Revit family instances** and the generic API..."	✗
RelativityOne - User Guide	Legal Tech	"...launch the Review interface from the Documents and **Family card**..."	✗
Maryland Workforce Innovation...	Gov. Policy	"...conducts an onsite **review** to examine all resource documents..."	✗

Table 6. Analysis of Retrieval Noise in `abravalheri_validate-pyproject_pr105`. The polysemy of the term "Family" causes the search engine to drift into unrelated technical domains.

The agent then falls back to a generic Python plugin-registration prior using `entry_points`. This change is plausible in isolation but incompatible with the target tests, which already register plugins through fixtures. The implementation registers the plugin twice and triggers a side-effect failure:

```
________ TestDisable.test_parse ________
> assert len(params.plugins) == 1
E AssertionError: assert 2 == 1
E  +  where 2 = len([<PluginWrapper...>, <PluginWrapper...>])
```

Lesson. The challenge is not only retrieving information, but discriminating which sources belong to the repository's semantic context. Search-augmented coding requires agents to reject high-ranking but out-of-domain evidence and to verify that an externally suggested pattern is locally actionable before modifying code.

C. More Implementation Details

C.1. Data and Code Availability

We submit anonymized supplementary material containing the benchmark data, evaluation code, and documentation needed to inspect BeyondSWE and reproduce the evaluation protocol. Public release links are omitted during anonymous review and will be provided after the review process.

C.2. Agent and Tool Configuration

Evaluation Details. For both OpenHands and SearchSWE, we set the maximum number of interaction turns to 200. The maximum context length is determined by each model's native limit. OpenHands (Wang et al., 2025b) is equipped with three core tools: `ExecuteBashTool` for command execution, `StrReplaceEditorTool` for file editing, and `FinishTool` for task completion. SearchSWE extends this toolset with two additional capabilities: `SearchTool` for web search and `BrowserTool` for webpage browsing. The search functionality is powered by Google Search via SerpAPI[1], while the browser tool utilizes Jina Reader[2] for content extraction, with DeepSeek-V3.2 serving as the summarization model. For Codex experiments, we use Codex (`v0.118.0`).

[1] https://serpapi.com/
[2] https://jina.ai/

Research Field	Repositories
Scientific Computing	
Astronomy	astroplan
Bioinformatics	biotite, Biopython
Computational chemistry	cclib
Plasma physics	PlasmaPy
Quantum physics	qutip
Seismology	obspy
Engineering	
Convex optimization	cvxpy
Geospatial	geopandas
Materials science	pymatgen
Molecular dynamics	mdanalysis
Photonic IC design	gdsfactory

Table 7. Research fields and repositories included in the DomainFix task.

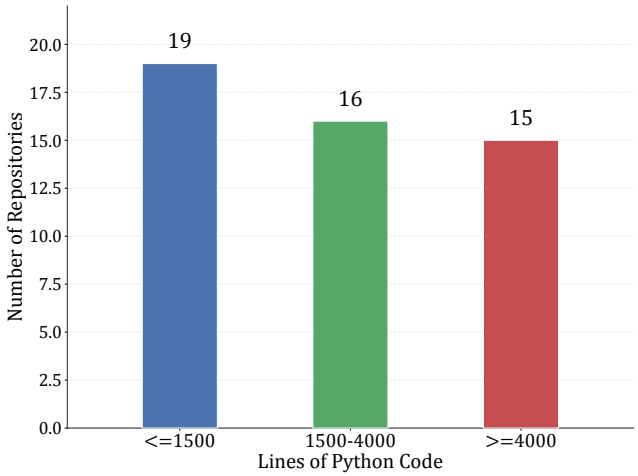


Figure 8. Distribution of lines of code across the 50 Doc2Repo repositories.

Controlled comparison. The comparison between OpenHands and SearchSWE is designed to isolate the effect of external information access. Both settings use the same local execution environment, editing interface, shell access, test commands, and maximum interaction budget. SearchSWE differs only by exposing web search and fetch tools and by using prompts that instruct the agent how to incorporate retrieved evidence into coding and verification.

C.3. BeyondSWE Dataset Details

Repository Details of DomainFix. Table 7 lists the 11 research fields and corresponding repositories included in the DomainFix task. These repositories span diverse scientific domains, from astronomy and quantum physics to bioinformatics and materials science, each requiring specialized domain knowledge to resolve issues.

Doc2Repo Repository Statistics. Figure 8 and Figure 9 present the code size statistics for the 50 repository instances in the Doc2Repo task. The repositories range from approximately 400 to over 13,000 lines of code, with the majority (31 out of 50) exceeding 1,500 lines. This distribution demonstrates that Doc2Repo poses a substantial challenge, requiring agents to generate non-trivial, real-world scale codebases rather than simple toy examples.

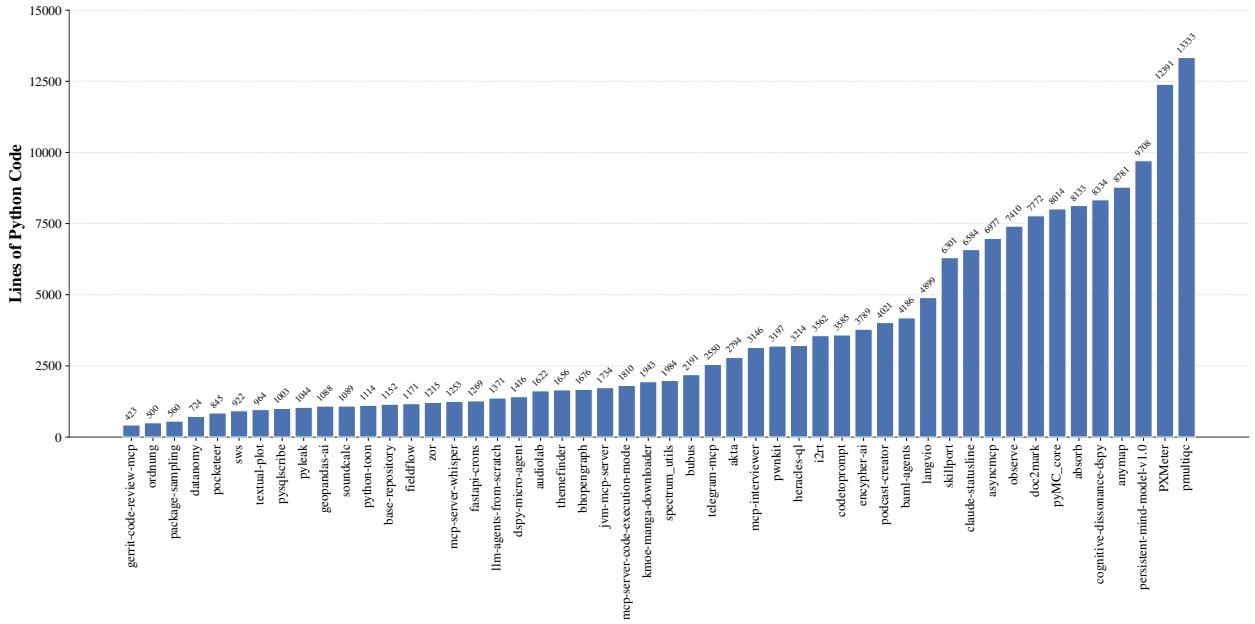


Figure 9. Lines of code for each Doc2Repo repository, sorted by size.

Data Format. Each instance in BeyondSWE is stored as a JSON object containing the following fields (Table 8). For Doc2Repo, the `problem_statement` field contains the generated repository specification, and evaluation uses the complete adapted test suite rather than a P2P/F2P split.

C.4. Additional Evaluation Results

Table 9 reports additional OpenHands and SearchSWE results for models not included in the main table. These runs support the same qualitative pattern as the main experiments: search access often improves performance, but the gains remain task- and model-dependent, with regressions on some task-model pairs.

C.5. BeyondSWE Construction Details

This section expands the construction procedure described in Section 3. Table 10 summarizes the candidate sources, filtering stages, human verification, and final retained instances for the four tasks.

Candidate collection. For CrossRepo, DomainFix, and DepMigrate, candidates are collected from merged pull requests so that each instance is grounded in a real developer change and has an executable reference patch. CrossRepo starts from Python-dominant repositories whose pull requests contain external links; the links serve as evidence that developers used or referenced information outside the target repository. DomainFix starts from repositories selected with domain experts across scientific and engineering fields, then collects pull requests from those repositories. DepMigrate starts from pull requests mentioning major version changes of widely used dependencies. Doc2Repo differs from the repair tasks: it starts from recently created Python repositories and converts each repository into a clean-room specification and test suite.

Filtering and problem statements. The filtering process removes candidates that are not executable, not stable, or not aligned with the intended task category. For issue-resolution tasks, raw pull requests often contain implementation details, commit references, or code-level hints. We therefore

Field	Description
`instance_id`	Unique identifier for the instance, typically in the format `user_repo_prN`.
`dataset_id`	Dataset identifier (e.g., `realswe_bench`).
`task`	Task category: `crossrepo`, `domainfix`, `depmigrate`, or `doc2repo`.
`user`	GitHub organization or user name.
`repo`	Repository name.
`language`	Primary programming language of the repository.
`workdir`	Working directory path inside the Docker container.
`image_url`	Docker image URL for the evaluation environment.
`patch`	Gold patch (ground truth solution) in unified diff format.
`pr_commit`	Commit hash of the pull request that resolved the issue.
`parent_commit`	Commit hash of the codebase before the fix (evaluation starting point).
`problem_statement`	Natural language description of the issue to be resolved.
`github_url`	URL to the original GitHub repository.
`pre_commands`	Shell commands to initialize the environment before evaluation.
`FAIL_TO_PASS`	List of test cases that should change from failing to passing after the fix.
`PASS_TO_PASS`	List of test cases that should remain passing after the fix.

Table 8. Data fields for each instance in BeyondSWE.

rewrite them into issue-style problem statements that preserve the observable task requirements while removing solution-specific details. For CrossRepo, reviewers additionally check that the external links are relevant to the issue rather than incidental references. For Doc2Repo, repository names are masked as `target_repo`; the specification preserves public API and behavioral requirements but removes implementation details and explicit directory structure.

Retention criteria. An instance is retained only if its environment is reproducible and its tests distinguish the buggy state from the fixed state. For CrossRepo, DomainFix, and DepMigrate, this requires P2P tests to pass and F2P tests to fail before applying the reference patch, and all selected tests to pass after applying the reference patch. For Doc2Repo, the generated specification and adapted tests are checked against the reference implementation and manually audited for consistency.

C.6. Human Verification and Quality Control

Table 11 summarizes the human verification protocol.

Human verification serves two purposes. First, it checks whether a retained instance reflects the intended capability rather than an artifact of data collection, environment setup, or test construction. Second, it checks whether the problem statement and test suite are fair: the task should provide enough information to identify the required behavior, but should not reveal the gold solution.

Review protocol. DomainFix uses domain experts because the core question is whether the issue requires specialized scientific or engineering knowledge. DepMigrate uses software engineering experts because the central question is whether the pull request represents a genuine dependency migration rather than an incidental version update. Across all tasks, senior software engineers inspect environment construction and data cleaning, while senior PhD researchers in software engineering and LLMs audit final task validity. Instances are rejected when reviewers identify unstable tests, insufficient task information, solution leakage, mismatch between the problem

Model	CrossRepo	DomainFix	DepMigrate	Doc2Repo		AVG
	%Resolved	%Resolved	%Resolved	Pass Rate	#(Alm.) Corr.	
OpenHands						
GLM-4.7 (Z.ai Team, 2025)	40.20	**36.11**	**39.89**	48.40	(3) / 1	**41.15**
DeepSeek-V3.2 (Liu et al., 2025a)	38.00	30.56	36.52	**54.99**	(3) / 0	40.02
Kimi-K2 (Team et al., 2025a)	37.00	27.78	39.53	54.91	**(6) / 2**	39.81
GPT-5.2 (OpenAI, 2025b)	33.00	23.61	34.27	53.89	**(6) / 2**	36.19
Seed-Coder-1.8 (Seed et al., 2025)	**41.92**	18.57	31.46	42.71	(1) / 0	33.67
Qwen3-Coder-Plus (Qwen Team, 2025)	19.19	5.56	15.43	1.87	(1) / 0	10.51
Qwen3-235BA22B (Yang et al., 2025a)	15.50	5.71	13.56	4.03	(0) / 0	9.70
SearchSWE						
GLM-4.7 (Z.ai Team, 2025)	**45.40** +5.2	32.39 -3.7	**39.77** -0.1	49.44 +1.0	(3) / 1	**41.75** +0.6
DeepSeek-V3.2 (Liu et al., 2025a)	39.49 +1.5	31.88 +1.3	34.09 -2.4	53.64 -1.4	(4) / 0	39.78 -0.2
Kimi-K2 (Team et al., 2025a)	39.90 +2.9	**33.33** +5.6	34.83 -4.7	49.31 -5.6	(2) / 1	39.34 -0.5
GPT-5.2 (OpenAI, 2025b)	36.22 +3.2	22.22 -1.4	33.90 -0.4	**55.85** +2.0	**(7) / 2**	37.05 +0.9
Seed-Coder-1.8 (Seed et al., 2025)	36.92 -5.0	22.06 +3.5	32.57 +1.1	41.40 -1.3	(1) / 0	33.24 -0.4
Qwen3-Coder-Plus (Qwen Team, 2025)	17.50 -1.7	17.14 +11.6	16.28 +0.9	1.38 -0.5	(0) / 0	13.08 +2.6
Qwen3-235BA22B (Yang et al., 2025a)	16.58 +1.1	9.72 +4.0	14.12 +0.6	7.00 +3.0	(1) / 0	11.86 +2.2

Table 9. Evaluation results of other models on BeyondSWE. This table includes BeyondSWE entries that are not reported in Table 2. **Bold** and underlined values indicate the best and second-best results within each OpenHands/SearchSWE block, respectively. For SearchSWE rows, red/green deltas compare against the matched OpenHands result for the same model.

statement and reference patch, or a task-category mismatch.

Compensation policy. Some reviewers and experts were members of the author team and participated as part of the research process. External reviewers and experts were compensated according to the applicable project and institutional policies for their review roles. We report this at the policy level rather than listing payment amounts, since compensation depends on the review arrangement and task batch rather than on a single uniform rate.

D. Representative Task Examples

This section provides compact examples of the four task formats in BeyondSWE. Each example highlights what information is given to the agent, what capability the instance stresses, and how the output is evaluated.

E. Prompt Design and Excerpts

This section summarizes the prompts used for benchmark construction and search-augmented coding. We avoid printing the full prompts in the paper because they are long and mostly operational. Instead, Table 12 summarizes their roles, and the boxes below show the key constraints needed to understand the evaluation design.

E.1. Problem Statement Generation

For CrossRepo, DomainFix, and DepMigrate, raw pull-request descriptions may contain implementation details, commit references, or direct descriptions of the fix. We therefore use a problem-statement generation prompt to rewrite pull requests into issue-style task descriptions. The prompt is designed to preserve observable task context while removing solution-specific information.

Task	Candidate Source	Automatic Filtering	Human Verification	Final Instances
CrossRepo	Merged pull requests in Python-dominant repositories containing external links	Environment construction and stability inspection reduce about 3,000 candidates to about 800 executable candidates	Review external-link relevance and check that rewritten issue statements preserve task context without solution-specific details	200 issues / 67 repos
DomainFix	Pull requests from expert-selected repositories across 11 scientific and engineering fields	Environment construction and stability inspection reduce about 800 candidates to about 200 executable candidates	Three domain experts independently verify environmental correctness, domain complexity, and solution non-triviality	72 issues / 12 repos
DepMigrate	Pull requests mentioning 23 widely used packages and relevant version upgrades	LLM-based migration filtering plus environment construction reduce about 7,000 candidates to about 1,000 executable candidates	Four software engineering experts verify that each instance is a genuine dependency migration	178 issues / 120 repos
Doc2Repo	Recently created Python repositories with continued activity, at least three contributors, and more than 20 stars	Repository masking, specification generation, test adaptation, and executable environment construction produce 60 candidates	Manual review selects high-quality specifications and checks alignment between adapted tests and documented behavior	50 repos

Table 10. Candidate collection, filtering, and verification pipeline for BeyondSWE.

E.2. SearchSWE Prompt Design

SearchSWE uses a shared system prompt plus task-specific user prompts. The shared system prompt defines the local coding workflow, search-use policy, and verification discipline. Task-specific prompts adjust the mission context: CrossRepo encourages external software investigation; DomainFix highlights domain knowledge; DepMigrate emphasizes compatibility with upgraded dependencies; Doc2Repo treats the provided specification as the authoritative source.

E.3. Task-Specific Prompt Roles

The task-specific user prompts differ mainly in the constraints they emphasize. For CrossRepo, the prompt asks the agent to inspect external artifacts when local context is insufficient. For DomainFix, it encourages the agent to verify scientific definitions, formulas, or domain conventions before modifying code. For DepMigrate, it forbids dependency downgrades and requires compatibility with the upgraded package versions installed in the environment. For Doc2Repo, it treats the provided specification as the ground truth and restricts search to missing dependencies or library-version syntax, rather than architectural invention.

These prompts are used to make search behavior explicit and auditable. They are not intended as a new agent architecture; they define a controlled way to test whether agents can combine external evidence with repository-local coding and verification.

Task	Reviewers	Verification Criteria	Agreement / Retention Rule
CrossRepo	Senior software engineers and senior PhD auditors	External links are relevant to the issue; rewritten statements preserve task requirements; no solution-specific details are exposed; tests exhibit valid fail-to-pass behavior	Retain only instances passing manual relevance, leakage, and stability checks
DomainFix	Three domain experts, plus final senior PhD audit	Environment executes as expected; the issue requires domain-specific knowledge beyond general programming; the solution is non-trivial and cannot be inferred from error messages alone	Retain only instances accepted by all three domain experts
DepMigrate	Four software engineering experts, plus final senior PhD audit	The pull request corresponds to a real dependency migration; the upgraded dependency is installed in the evaluation environment; the required fix is not an incidental local patch	Retain only instances verified as genuine migration tasks
Doc2Repo	Senior software engineers and senior PhD auditors	Specification preserves public API and behavior while removing implementation details; adapted tests match the specification; stricter-than-source requirements are documented	Retain only specifications and tests judged aligned and executable
All tasks	Five senior software engineers and five senior PhD researchers	Environment construction, data cleaning, problem-statement quality, test validity, and final task-category fit	Reject instances with instability, solution leakage, underspecified behavior, or task-category mismatch

Table 11. Human verification protocol for BeyondSWE.

CrossRepo Example: `kitware_trame-server_pr8`

Task goal. Fix an issue where the server ignores the explicit `host` argument and the `TRAME_DEFAULT_HOST` environment variable, causing it to bind to `localhost` even when downstream integrations require another interface.

Capability stressed. The issue is linked to downstream PyVista usage, so the agent must reason beyond the target repository and understand how server binding behavior affects related projects.

Input signal. The problem statement provides a reproduction script, expected binding behavior, and an external integration reference, but not the implementation location of the fix.

Evaluation signal. A valid patch must preserve existing behavior while making the host argument and environment-variable override pass the target F2P tests.

Figure 10. Representative CrossRepo instance.

DomainFix Example: `cvxpy_cvxpy_pr2125`

Task goal. Add sparse Cholesky decomposition support for positive definite sparse matrices in `cvxpy.utilities.linalg`.

Capability stressed. The task is not only an API addition: the agent must understand the mathematical contract of sparse Cholesky factorization, including the permutation relation $PAP^T = LL^T$, and implement it in a way compatible with sparse matrix representations.

Input signal. The problem statement gives the desired function behavior and example usage, but the implementation requires domain knowledge about numerical linear algebra and sparse factorization.

Evaluation signal. A valid patch must return the expected sparse triangular factor and permutation behavior while preserving existing CVXPY linear-algebra functionality.

Figure 11. Representative DomainFix instance.

DepMigrate Example: `stanfordnlp_dsp_pr403`

Task goal. Update an LM client to support `openai>=1.0`, including Azure OpenAI configurations where `model`, `engine`, and `deployment_id` handling changed.

Capability stressed. The agent must perform migration reasoning rather than a localized bug fix: it needs to map old SDK assumptions to the new client interface, handle Azure-specific behavior, and avoid breaking existing cache compatibility.

Input signal. The issue states current failure modes under the upgraded dependency and lists compatibility constraints, but the agent must inspect the repository to find all affected client initialization and inference paths.

Evaluation signal. A valid patch must pass tests under the upgraded dependency while preserving behavior for standard OpenAI and Azure-backed configurations.

Figure 12. Representative DepMigrate instance.

Doc2Repo Example: `doc2mark`

Task goal. Construct a repository from a specification for a unified document-processing library that converts PDFs, Office files, images, HTML, and related formats into Markdown.

Capability stressed. The agent starts from an empty workspace and must infer the repository structure, public API, dependencies, and implementation behavior from the specification alone.

Input signal. The specification describes classes such as `UnifiedDocumentLoader`, method signatures, supported formats, OCR configuration, caching behavior, and expected exceptions, while masking the original repository identity as `target_repo`.

Evaluation signal. A valid submission must implement a coherent package whose public API and behavior pass the adapted repository test suite.

Figure 13. Representative Doc2Repo instance.

Prompt Component	Purpose	Key Constraints	Used For
Problem-statement generation	Convert raw pull requests into issue-style task descriptions	Remove solution-specific details while preserving reproduction steps, expected behavior, logs, and necessary context	CrossRepo, DomainFix, DepMigrate
SearchSWE system prompt	Define the shared code-agent workflow with search and local verification	Search only when local context is insufficient; fetch full pages before using results; prioritize local environment and user instructions	All SearchSWE runs
CrossRepo user prompt	Encourage use of external software artifacts when target-repository context is incomplete	Inspect related projects or documentation, but adapt evidence to the local repository and tests	CrossRepo
DomainFix user prompt	Guide agents to combine code reasoning with domain-specific knowledge	Verify scientific definitions, formulas, or domain conventions before applying a fix	DomainFix
DepMigrate user prompt	Frame the task as compatibility with upgraded dependencies	Do not downgrade dependencies; use migration guides or release notes only when compatible with installed versions	DepMigrate
Doc2Repo user prompt	Constrain clean-room repository construction from a specification	Treat the provided specification as authoritative; use search only for dependency syntax or missing imports, not to override the spec	Doc2Repo

Table 12. Summary of prompt components used for benchmark construction and SearchSWE evaluation.

Problem Statement Generation Prompt: Key Constraints

```
You are an expert-level autonomous software engineer and open-source maintainer.

Your sole task is to draft a concise, human-like GitHub issue based on a provided pull request.



Critical rules:

- Do not reveal the solution, diff, internal function names, line numbers, PR author, commit hash, or fix logic.

- Write from the perspective of a user or developer reporting the problem before the fix exists.

- Preserve reproduction steps, observed behavior, expected behavior, error logs, environment details, and
    external context needed to make the task solvable.

- The reproduction should be a natural script or command sequence, not a unit test that asserts the answer.
```

Figure 14. Excerpt of the prompt used to convert pull requests into issue-style problem statements.

SearchSWE System Prompt: Search-Use Policy Excerpt

```
Use search strategically, not exclusively.
Search only when the task requires information that is missing from the local repository.

When to search:
- Unknown syntax or API details of a specific library version.
- Obscure error messages that cannot be diagnosed from local code.
- Missing implementation details not provided in the local context.

Search methodology:
- Use focused queries to identify relevant sources.
- If a result looks promising, fetch and read the actual page content.
- Never write code based only on search snippets.
- Before using an online library pattern, check whether the library is installed locally and verify the local
    version/API.
- User instructions, local files, and provided specifications are the highest-priority constraints.
```

Figure 15. Excerpt of the shared SearchSWE system prompt governing search use and local verification.