CLAW-SWE-BENCH: A BENCHMARK FOR EVALUATING OPENCLAW-STYLE AGENT HARNESSES ON CODING TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

The software engineering capabilities of general-purpose agent harnesses remain underexplored, and existing benchmarks offer limited support for comparing these harnesses under consistent conditions. To address this gap, we introduce Claw-SWE-Bench, a unified benchmark that enables researchers to systematically assess the capabilities and efficiency of general-purpose harnesses on software engineering tasks. The benchmark contains 350 real-world GitHub issue-resolution instances across eight programming languages and 43 repositories and provides a shared adapter protocol to align task inputs, outputs, and execution environments across harnesses. Experiments show that general-purpose harnesses can effectively resolve real-world software issues and that their success rates and resource consumption vary substantially even when the underlying model is held fixed. To lower evaluation costs and support faster debugging and iteration, we also provide Claw-SWE-Bench Lite, an 80-instance subset designed to preserve the key evaluation properties of the full benchmark. We hope this benchmark will help researchers better evaluate and understand the performance of general-purpose harnesses on software engineering tasks and guide the development of more capable and efficient harnesses. Code and dataset links are omitted for anonymous review.

1 INTRODUCTION

Software development and maintenance are important application areas for agents. Automating tasks such as code generation and issue resolution could reduce the workload for developers and improve development and maintenance efficiency (Hou et al., 2024). However, real-world software engineering tasks require more than code generation. Agents must understand problem descriptions, locate relevant code, and verify the correctness of changes through execution and testing (Jimenez et al., 2024; Yang et al., 2024). To address these challenges, prior work has used methods such as supervised fine-tuning (Yang et al., 2025) and reinforcement learning (Luo et al., 2025) to improve the software engineering capabilities of the underlying models.

Task completion depends not only on the capabilities of the underlying model but also on the harness (Yang et al., 2024; Lee et al., 2026). This execution framework coordinates interactions between the model and the external environment. In recent years, agent harnesses, particularly general-purpose harnesses such as OpenClaw (OpenClaw, 2026), have continued to evolve and gain widespread adoption. They organize information about the environment, provide tool interfaces, and manage interaction context and execution workflows. These functions support models in tasks involving productivity tools, browser automation, computer use, and scientific assistance (Ding et al., 2026; Liang et al., 2026). Existing benchmarks and practical applications have demonstrated the strong performance of general-purpose harnesses on everyday tasks (Li et al., 2026).

Despite their strong performance in these settings, the ability of general-purpose harnesses to handle software engineering tasks has received little systematic study. More importantly, it remains unclear whether general-purpose harnesses with different designs perform differently on coding tasks.

In this work, we introduce Claw-SWE-Bench, a benchmark for studying the performance of general-purpose harnesses on coding tasks. It contains 350 real-world GitHub issue-resolution instances from the SWE-bench family (Yang et al., 2025; Hobbhahn, 2025), covering eight programming

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.

languages and 43 repositories. These tasks allow us to compare and evaluate how harnesses affect coding performance. Many existing evaluations of agent coding capabilities focus primarily on models (Deng et al., 2026; Huang et al., 2026). A platform for fair comparisons of different general-purpose harnesses under consistent conditions is still lacking. Consistent with resource-aware agent evaluation (Kapoor et al., 2026; Fan et al., 2025), Claw-SWE-Bench reports task resolved rate , API cost, and average runtime to characterize both the task-solving capabilities and execution efficiency of general-purpose harnesses.

Claw-SWE-Bench provides a shared adapter protocol to align task inputs, outputs, and execution environments. The protocol integrates different harnesses into a unified evaluation pipeline. It standardizes task environments, outer runtime budgets, patch extraction, and test-based evaluation. We identify a source of information leakage in some task environments: Git commits after the specified base commit remain accessible. We remove these commits before evaluation to prevent agents from inspecting subsequent fixes. Using Claw-SWE-Bench, we systematically evaluate seven general-purpose harnesses with three underlying models. We find that general-purpose harnesses can perform software development tasks effectively. We also find substantial differences in task success rates across general-purpose harnesses, even when the underlying model is held fixed. These results show the importance of harness design for agent performance. In terms of efficiency, token usage also varies substantially across harnesses. This indicates that harness design affects both task performance and resource consumption during execution.

Finally, we construct Claw-SWE-Bench Lite, a compact evaluation subset containing 80 instances, to simplify evaluation and iteration for agent developers and reduce debugging costs. Its construction accounts for language coverage and the task difficulty distribution. Lite balances agreement with the full benchmark in task success rates, relative performance across harness–model pairs, and cost characteristics. This preserves the key evaluation properties of the full benchmark at lower cost.

Through Claw-SWE-Bench, we establish a unified benchmark for evaluating the software engineering capabilities of harnesses. We also show that general-purpose harnesses have strong coding capabilities. We hope that the benchmark and its Lite subset will help researchers and developers evaluate and refine agents, supporting progress in agent research and applications.

2 Related Work and Background

Many existing benchmarks for general-purpose agents focus on assistant tasks (Mialon et al., 2024; Yoran et al., 2024; Drouin et al., 2024). Many evaluations of software engineering agents emphasize model comparisons, with less systematic attention to the effects of harness design (Deng et al., 2026; Huang et al., 2026). Taken together, these settings provide limited evidence on performance differences among general-purpose harnesses on real-world software engineering tasks (Kapoor et al., 2025; Yehudai et al., 2026). For example, ClawsBench (Li et al., 2026) evaluates agent capabilities and safety in simulated productivity workflows, including email management and scheduling. WildClawBench (Ding et al., 2026) compares harnesses across a range of assistant tasks. Its coding scenarios, such as writing inference scripts for SAM3 and reproducing benchmark results, are manually authored rather than drawn from real GitHub issues and their corresponding fixes. DeepSWE (Huang et al., 2026) primarily compares model capabilities under a fixed harness rather than systematically evaluating the role of harness design.

2.1 Harnesses

General-Purpose Agent Harnesses. General-purpose harnesses (Wu et al., 2024a; Gao et al., 2024; Wang et al., 2024) connect language models to external environments and support tool use across applications. They construct model context from environmental observations and interaction history, execute tool calls, and maintain session state. OpenClaw (OpenClaw, 2026) adopts a gateway architecture that connects messaging channels to session-based agent execution. Its runtime assembles workspace instructions and reusable skills into the prompt, coordinates tool calls, and persists interaction histories. Hermes Agent (Nous Research, 2026) emphasizes reuse across sessions through persistent memory and a skill library. It retains environment facts and user preferences, retrieves past conversations, and stores task procedures as skills that can be loaded and revised when needed. nanobot (HKUDS, 2026) takes a lightweight approach, organizing a compact Python

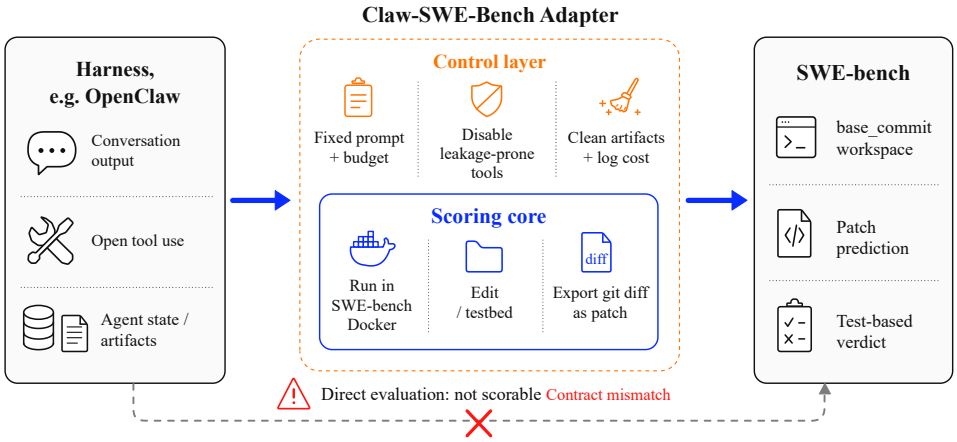


Figure 1: Contract mismatch between OpenClaw-style harnesses and SWE-bench. The adapter converts a general agent interaction into a SWE-bench-scored patch prediction, while outer controls ensure fairness, comparability, and traceable cost.

agent loop around model calls, tool execution, and message handling. More generally, memory and reusable skills can be supplied as context, while file operations, shell commands, and web tools support tasks across applications (Wang et al., 2025b; Wu et al., 2024b). ZeroClaw (ZeroClaw Labs, 2026) emphasizes modularity through a Rust runtime with trait-based interfaces for model providers, tools, communication channels, and memory backends. These interfaces allow components to be replaced independently, while the runtime manages the agent loop and enforces execution policies.

Coding Agent Harnesses. Coding agent harnesses (Yang et al., 2024; Wang et al., 2025a; Zhang et al., 2024) support repository-level software development by connecting code exploration and editing with test-based verification. Claude Code combines source-file search and editing tools in an iterative workflow that diagnoses test failures, modifies relevant code, and reruns tests to check fixes. Project-specific instructions provide coding conventions and build and test procedures to guide work within an existing codebase. Codex CLI (OpenAI, 2026) similarly uses repository-specific guidance to support local code editing, test execution, and lint checks. It also provides a dedicated code review mode that examines code changes and reports potential issues without modifying the working tree.

3 CLAW-SWE-BENCH

We introduce Claw-SWE-Bench to study the performance of general-purpose agent harnesses, referred to here as *claws*, on real-world software engineering tasks. Following the SWE-bench task setting (Jimenez et al., 2024), agents modify existing repositories to resolve real GitHub issues, and the resulting patches are evaluated through repository-level tests. The benchmark contains 350 task instances from SWE-bench-Multilingual (Yang et al., 2025) and SWE-bench-Verified-Mini (Hobbhahn, 2025), spanning 43 repositories and eight language groups: Java, Go, Rust, JavaScript/TypeScript, C/C++, Ruby, PHP, and Python. This coverage across languages and repositories broadens the range of software engineering settings represented in the evaluation.

Evaluating these claws requires both reliable integration and consistent conditions for comparison. We first provide an adapter protocol that connects native harness execution to the repository editing and patch submission required by SWE-bench. This allows general-purpose harnesses to participate in the evaluation while preserving their internal agent loops. Building on this protocol, we define a standardized execution pipeline that controls task inputs, initial repository states, runtime conditions, patch extraction, and test-based scoring across harnesses. Different harnesses can then be compared with the underlying model and outer execution budget held fixed. Their internal tool interfaces and execution strategies remain part of the harness design under evaluation.

3.1 THE ADAPTER PROTOCOL

The core artifact of CLAW-SWE-BENCH is a harness-agnostic adapter protocol. Every supported harness implements the same abstract interface: `create_agent`, `send_task`, `backup_session`, `delete_agent`, `get_docker_args`. The shared orchestrator calls these methods without knowing which harness is underneath. This design separates the benchmark lifecycle from the agent implementation: container management, prompt rendering, patch collection, prediction writing, metadata logging, resume support, and evaluation are implemented once, while each harness adapter only supplies the code needed to invoke its own agent loop inside the container.

At run time, the shared orchestrator owns the common lifecycle shown in Figure 1: container startup, repository reset, prompt rendering, patch collection, prediction writing, metadata logging, and evaluation. The adapter supplies only the harness-specific hooks needed to create or configure an agent, dispatch the rendered task, preserve any run artifacts, and clean up harness state. This boundary is deliberate: the benchmark controls the task-facing environment and evaluator-facing patch format, while leaving the internal agent loop to the harness under study.

Crucially, the candidate patch is collected by the benchmark runner rather than parsed from the agent's final message. Agents communicate a solution only by editing files in the repository. This makes the output contract independent of whether a harness emits JSON, plain text, a final answer, or no structured response at all.

All harnesses are launched through a single command-line entry point, `run_infer.py`. The user specifies the harness name, dataset configuration, model identifier, run identifier, timeout, worker count, and optional instance filters. Dataset metadata are loaded from the configured SWE-bench source, and each instance is represented by the fields needed by the protocol: `instance_id`, `repo`, `base_commit`, and `problem_statement`. A harness registry maps a string identifier to the corresponding adapter class; adding a new claw therefore requires only implementing the adapter interface and registering it in the harness map—the dataset loader, Docker workspace manager, prompt builder, patch collector, prediction writer, and evaluator remain untouched.

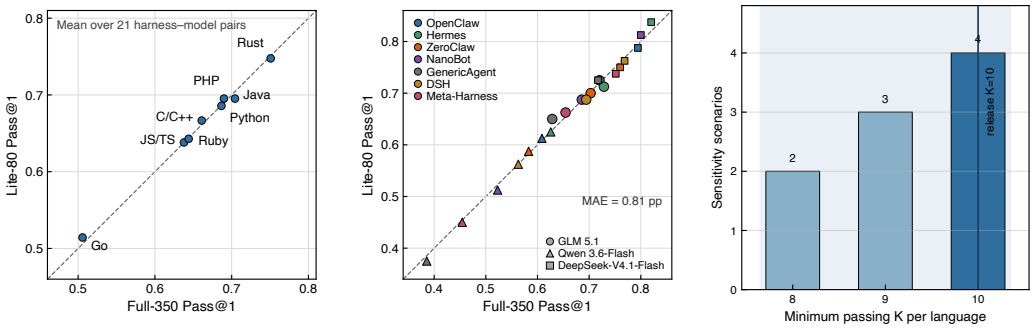
3.2 STANDARDIZED EXECUTION PIPELINE

Around the adapter boundary, CLAW-SWE-BENCH standardizes the parts of the evaluation stack that would otherwise confound harness comparisons.

Runtime and workspace. Each task runs inside its corresponding SWE-bench evaluation Docker image (Jimenez et al., 2024), with the repository reset to the instance's `base_commit` under `/testbed`. To prevent agents from retrieving existing solutions online, we block external network access during task execution, except for connections required for model inference. The same outer budget is used for every harness: `max_turns`=300 where supported, a 3600-second wall-clock timeout, and fixed worker concurrency. Harness-specific dependencies are supplied through Docker arguments or bind mounts, but the repository state, evaluation image, and outer budget seen by the agent are fixed.

Prompt rendering. Every instance is rendered with the same task prompt. The prompt contains the problem statement, identifies `/testbed` as the code repository, and prohibits network access and modifications to test files. This standardizes the task-facing message across harnesses. We do not attempt to equalize each harness's hidden system prompt, tool schema, parser hints, memory policy, or stopping rule; these remain part of the harness design and are therefore part of the experimental variable.

Patch and evaluation contract. Candidate solutions are collected from repository state rather than from an agent's final message. After a harness terminates, times out, or returns an error, the runner extracts the diff against the base commit, removes known non-solution artifacts, and writes a SWE-bench-compatible prediction. This centralized patch contract makes heterogeneous harnesses comparable even when their native outputs differ: JSON, plain text, natural-language summaries, or no structured response all reduce to the same evaluator-facing patch format. Then, evaluation is performed with the official SWE-bench harness (Jimenez et al., 2024).



(a) Per-language parity (b) Cross-claw parity (c) K-sweep sensitivity envelope

Figure 2: Lite-80 parity with full-350. **(a)** Per-language Pass@1 averaged uniformly over 21 pairs. **(b)** Overall Pass@1 for 7 claws $\times$ 3 models. **(c)** K-sweep sensitivity envelope on the 21-column calibration pool; the minimum acceptable K falls in $[8, 10]$ across scenarios.

4 Claw-SWE-Bench Lite

The full 350-instance benchmark (full-350) is used for standard evaluation in this paper, but rerunning all tasks at every development iteration is costly. A full run involves substantial token consumption, API costs, wall-clock time, and log inspection effort. During adapter debugging, prompt editing, model replacement, or regression testing, repeated full runs can make evaluation a major development bottleneck. We therefore construct Claw-SWE-Bench Lite as a low-cost companion subset. Its 80 instances are selected to approximate the overall and per-language Pass@1, relative harness performance, and cost characteristics of full-350. Researchers can use Lite to quickly assess system changes and return to full-350 at key stages for final reporting.

4.1 Lite Subset Definition

Lite-80 selects 10 instances from each of the eight language groups in full-350. It contains 70 non-Python instances from SWE-bench-Multilingual and 10 Python instances from SWE-bench-Verified-Mini, matching the sources used in the full benchmark. In addition to balancing languages, Lite enforces fixed difficulty-quartile quotas within each language group. It selects 2, 3, 3, and 2 instances from Q_1, Q_2, Q_3, and Q_4, respectively. This prevents the subset from concentrating on unusually easy or difficult tasks. The final subset covers 38 of the 43 repositories in the full benchmark, or approximately 88%, retaining broad repository coverage. We use the full-350 evaluation results of seven harnesses paired with three models as calibration data for Lite subset selection.

4.2 Subset Selection Method

Instance selection. TinyBenchmarks (Maia Polo et al., 2024) estimates full-benchmark performance using a small set of representative examples. Following this idea, we formulate Lite construction as a constrained subset selection problem. For Lite-80, we select 10 instances per language group under the 2/3/3/2 difficulty-quartile quotas. We compute the mean resolved rate of each instance across the 21 calibration columns and use these values to define quartiles within each language group. This ensures that the selected instances cover different levels of relative difficulty.

The objective combines resolved-rate parity and pairwise ranking, with cost parity enforced during candidate screening. The resolved-rate parity term minimizes the L_1 difference between Lite and full-350 over the 21×8 grid of calibration columns and language groups. However, small errors in individual scores do not necessarily preserve the relative ordering of systems. We therefore include a pairwise ranking hinge term (Joachims, 2002). Within each language group, this term considers pairs of calibration columns whose full-set resolved rates differ by more than 0.05. It applies a penalty if Lite reverses the ordering or produces a gap below 0.025 in the direction of the full-set ordering. The objective thus accounts for both absolute system performance and relative rankings.

Cost parity is enforced by filtering candidate subsets using the log-space difference between Lite and full-350 mean per-instance costs. We require the mean absolute log error to be at most 0.10 on both

Table 1: Comparison between the bare adapter and the full adapter. Both use the same GLM 5.1 backbone. All metrics are averaged over three runs per configuration. **Apply Failed** is the fraction of instances whose submitted patch cannot be applied to the repository by the SWE-bench evaluator.

Configuration	Resolved	Pass@1(%)	Apply Failed
Bare adapter	67.7/350	19.3	69.1%
Full adapter	249.3/350	71.2	$< 1.5\%$

evaluable columns and columns with complete cost records. This controls relative cost deviations while reducing the number of evaluated instances. We optimize the objective independently for each language group using local search with multiple random restarts (Martí et al., 2013). Each step swaps one selected instance with one unselected instance from the same difficulty quartile. This preserves the instance count and difficulty quotas throughout the search.

Subset size selection. We determine the subset size through a K-sweep on the 21-column calibration pool. We vary K, the number of instances per language group, under different ranking margins, restart counts, random seeds, and mirror-parity settings. The minimum acceptable size ranges from 8 to 10 instances per language group: two scenarios pass at $K = 8$, three require $K = 9$, and four require $K = 10$. We select the largest of these scenario-specific minima, $K = 10$, giving 80 instances across the eight language groups.

4.3 Validation Results

Across the 21 calibration columns, the mean Pass@1 is 0.6610 on full-350 and 0.6607 on Lite-80, a difference of approximately 0.02 percentage points. Per-language deviations are all within 1 percentage point. We also compare Lite and full-350 across seven harnesses paired with three shared models. The mean absolute Pass@1 difference across these 21 pairs is 0.81 percentage points. These results show that Lite-80 closely matches full-benchmark success rates on harness–model pairs used for calibration, covering both model and harness variation.

We also assess cost parity. Across the 11 harness–model pairs with complete per-instance cost estimates, a Lite run costs on average 25.3% of a full-350 run. The cost reduction therefore comes mainly from evaluating fewer instances rather than selecting unusually cheap tasks. Lite-80 can thus provide feedback for debugging and regression testing at roughly one quarter of the evaluation cost.

5 Experimental Setup

We use Claw-SWE-Bench to compare a representative set of recent agent harnesses with different designs. We focus on the capabilities and efficiency of general-purpose harnesses on software engineering tasks. First, we compare the bare and full adapters to assess the role of the adapter protocol in reliable task execution and patch submission. We then conduct the main comparison of seven harnesses with three underlying models on full-350. We further validate Lite-80 on six harness and model combinations excluded from subset construction and calibration. Using the main experimental results, we examine the trade-off between task performance and running cost. Finally, we assess the impact of future-commit leakage in some SWE-bench-Multilingual evaluation environments by comparing results before and after cleanup.

Claws. The main experiment includes seven harnesses: OpenClaw (OpenClaw, 2026), Hermes Agent (Nous Research, 2026), nanobot (HKUDS, 2026), ZeroClaw (ZeroClaw Labs, 2026), GenericAgent (Liang et al., 2026), DeepSeek Harness (DeepSeek-AI, 2026), and Meta-Harness (Lee et al., 2026), a method for automatically evolving harnesses. For Meta-Harness, we use GenericAgent as the initial harness because its Python implementation allows unrestricted code modification and is comparable in code size to the original seed harnesses. We use DeepSeek-V4.1-Flash as the proposer model. We evaluate the evolved harness separately from the original GenericAgent. We also include OpenSquilla (TokenRhythm Technologies, 2026) in the Lite held-out validation.

Models. The main experiment uses GLM 5.1 (Zeng et al., 2026), Qwen 3.6-flash (Qwen Team, 2026), and DeepSeek-V4.1-Flash (Xu et al., 2026). Each model is paired with all seven harnesses on

Table 2: Performance across seven harnesses with three models on Claw-SWE-Bench. Resolved is the number of solved instances. Cost, In, and Out are totals over 350 instances; In excludes cache-read tokens. Dur is mean per-instance duration, and Cache is the token-weighted mean cache hit rate. Bold indicates the highest Resolved and Pass@1 within each model group. † Based on GenericAgent.

Harness	Resolved	Pass@1 (%)	Cost (USD)	Dur (s)	In (M)	Out (M)	Cache (%)
Model: GLM 5.1							
OpenClaw	249.3	71.2	338	511	68.7	8.1	92.0
Hermes Agent	**256.0**	**73.1**	389	800	64.4	6.9	94.1
nanobot	240.3	68.7	525	956	150.4	7.8	87.7
ZeroClaw	244.0	69.7	391	548	93.3	4.3	90.9
GenericAgent	220.3	62.9	86	576	33.1	5.1	66.8
DeepSeek Harness	241.7	69.1	513	916	93.2	5.3	93.7
Meta-Harness†	230.0	65.7	190	379	79.7	5.7	72.2
Model: Qwen 3.6-flash							
OpenClaw	213.0	60.9	87	639	34.6	9.5	97.4
Hermes Agent	**219.7**	**62.8**	104	639	44.5	7.3	97.4
nanobot	183.7	52.5	175	636	466.1	9.9	65.1
ZeroClaw	204.0	58.3	69	429	32.7	6.1	96.9
GenericAgent	134.7	38.5	39	321	73.8	7.5	71.5
DeepSeek Harness	198.7	56.8	91	614	38.4	9.7	97.2
Meta-Harness†	158.0	45.1	49	301	105.8	7.6	68.3
Model: DeepSeek-V4.1-Flash							
OpenClaw	279.3	79.8	50	566	38.5	25.4	97.3
Hermes Agent	**286.0**	**81.7**	36	492	41.8	13.3	96.9
nanobot	278.7	79.6	78	533	184.9	13.6	84.2
ZeroClaw	266.3	76.1	99	1432	57.6	62.2	95.3
GenericAgent	250.3	71.5	12	839	11.7	6.8	88.1
DeepSeek Harness	267.7	76.5	41	1577	56.4	12.9	96.3
Meta-Harness†	262.3	74.9	13	1005	14.0	7.2	86.6

full-350, giving 21 harness–model pairs. We hold the underlying model fixed within each comparison to examine differences in harness performance.

Evaluation metrics. Our primary metric is Pass@1, defined as the fraction of task instances marked as Resolved by the SWE-bench evaluator (Jimenez et al., 2024). We also report total API cost, mean wall-clock duration, token usage, and cache hit rate. API costs are calculated from the per-call token usage recorded by each harness and the fixed unit prices listed in Appendix A.4. Wall-clock duration includes remote API latency. The adapter diagnostic additionally reports the patch application failure rate. For Lite validation, we compare the full-350 and Lite-80 Pass@1 scores of six held-out harness–model pairs. The diagonal represents equal scores on the two task sets.

Runtime configuration. Each instance runs in its corresponding SWE-bench Docker image. The repository is located at `/testbed` and reset to the specified base commit. Standard evaluations apply future-commit cleanup to the Multilingual instances. Unless explicitly varied in the adapter or cleanup diagnostics, all runs share the task-prompt template, patch collector, evaluator, and result aggregation code. We set a per-instance wall-clock timeout of 3,600 seconds and fix worker concurrency at three. For both the adapter comparison and the main comparison of 21 harness–model pairs, we evaluate each configuration on full-350 in three separate runs and report the mean results. Other experiments use a single run per configuration on each task set. All model inference is performed through remote APIs. For the six held-out harness–model pairs, we compare existing full-350 results with separate runs on the fixed Lite-80 subset.

6 Results

6.1 The Adapter Makes a General Agent Scorable

We first test whether the adapter is merely an engineering wrapper or a necessary condition for OpenClaw to be reliably scored by SWE-bench. Table 1 compares the same GLM 5.1 backbone under the bare adapter and the full adapter. The bare adapter places OpenClaw in the SWE-bench

Docker environment and sends the task. The full adapter instead lets the model edit repository files through tools and has the runner export the patch from Git state.

The results show that minimal access is insufficient to create a reliable SWE-bench evaluation target. The bare adapter reaches only 19.3% resolved rate. The main bottleneck is not that the model cannot edit code at all, but the fragility of directly generating unified-diff text: line numbers, context, hunk headers, or trailing newlines can make the patch fail to apply. The full adapter shifts the output responsibility from "the model writes patch text" to "the model edits repository files and the runner exports the patch," reducing apply failures below 1.5% and raising resolved rate to 71.2%. The following experiments therefore measure model and claw differences under a unified scoring contract, rather than testing whether a native agent can hand-write a SWE-bench-compatible diff.

6.2 Performance across Harnesses

To isolate the effect of harness choice, we hold the underlying model fixed and compare seven harnesses under the shared Claw-SWE-Bench protocol. We conduct this comparison with GLM 5.1, Qwen 3.6-flash, and DeepSeek-V4.1-Flash, yielding 21 harness–model pairs. This design allows us to examine whether relative harness performance remains consistent across models or depends on the underlying model. Each pair is evaluated on all 350 instances in three separate runs, giving 22,050 task executions in total. The main table reports resolved instance counts, Pass@1, total API cost, mean per-instance wall-clock duration, total uncached input tokens, total output tokens, and cache hit rate. All metrics are averaged over the three runs for each pair.

Table 2 shows substantial differences in Pass@1 across harnesses even when the underlying model is fixed. The largest gap reaches 24.3 percentage points(pp) on Qwen 3.6-flash. Specifically, Hermes Agent achieves the highest mean Pass@1 with all three models, while some other harnesses show more variation in their relative performance. For example, nanobot trails the best harness by 10.3 pp with Qwen 3.6-flash, but by only 2.1 pp with DeepSeek-V4.1-Flash. Evolving GenericAgent through Meta-Harness also improves Pass@1 across all models. Following the corresponding experimental setup in the original Meta-Harness paper, we use the same instances for evolution and final evaluation. Meta-Harness uses evaluation feedback from these tasks during evolution, so comparisons with other harnesses are not conducted under equal conditions. Its performance gains should therefore not be interpreted as evidence of superiority in a fair comparison. These results show that harness design affects how effectively model capabilities translate into successful issue resolution. They motivate further study of harness optimization and its interaction with the underlying model.

6.3 Efficiency and Cost Analysis

We further analyze the trade-off between task success and API cost using the main results in Table 2. Figure 3 plots part of the results with total API cost on the horizontal axis and Pass@1 on the vertical axis. Each point represents one harness and model combination evaluated on full-350, with both metrics averaged over three runs. We highlight the Pareto frontier across all 21 harness–model pairs.

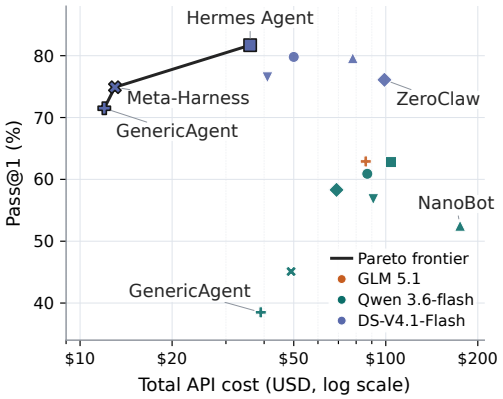


Figure 3: Resolve-rate–cost Pareto frontier for part of the results. The black line connects non-dominated points.

As shown in Figure 3, DeepSeek-V4.1-Flash, the newest of the three evaluated models, generally achieves higher Pass@1 than the other two models. Although it typically generates more output tokens, its low cached-input token price (Appendix A.4) helps keep total API costs lower for most harnesses. The Pareto frontier consists of three harness–model pairs, all using DeepSeek-V4.1-Flash: GenericAgent, Meta-Harness, and Hermes Agent. GenericAgent provides the lowest-cost point, while Hermes Agent reaches the highest Pass@1 of 81.7% at $36. These pairs offer different performance and cost trade-offs under the same model, showing that harness choice affects not only task success but also the cost of agent execution.

6.4 Evaluation of Claw-SWE-Bench Lite

We further assess whether Lite-80 can approximate full-benchmark performance on systems excluded from calibration. We evaluate the fixed subset on six new harness–model pairs: OpenSquilla with GLM 5.1, OpenClaw and Hermes Agent with MiniMax M2.7 (MiniMax, 2026), and OpenClaw with Seed 2.0-mini (Seed, 2026), MiMo-v2.6-flash and Opus 4.7 (Anthropic, 2026). These pairs include four new models that are absent from the calibration pool, while OpenSquilla introduces a new harness. None of these pairs is used for subset construction or calibration.

Following Figure 2(b), Figure 4 compares full-350 Pass@1 on the horizontal axis with Lite-80 Pass@1 on the vertical axis, with one point per pair. The diagonal represents equal scores on the two task sets; points closer to this line indicate closer agreement. Across these six pairs, the maximum absolute difference is 1.70 pp, and their ranking is unchanged between full-350 and Lite-80.

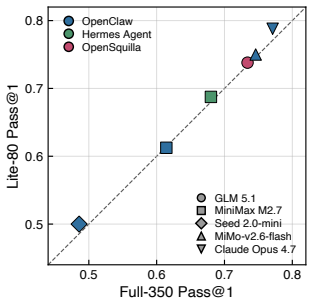


Figure 4: Pass@1 on full-350 and Lite-80 for six harness–model pairs. The dashed line indicates equal scores.

6.5 Effect of Future-Commit Cleanup

During manual inspection of the task environments, we found that SWE-bench-Multilingual containers retained Git history after the specified base commit, including the reference fixes. The image-building code used in these runs cleaned future history for Python repositories but not for non-Python repositories. For the latter, removing the remote origin left the local history intact. Agents could therefore use pull request identifiers from task descriptions to locate repair commits and retrieve reference patches.

We address this issue during workspace preparation by removing accessible Git history beyond the base commit before agent execution. To measure the impact of cleanup, we fix OpenClaw and compare eight models before and after this change on the 300 Multilingual instances. As shown in Figure 5, Pass@1 after cleanup is no higher than before cleanup for any of the eight models. The magnitude of the change varies across models. Opus 4.7 shows the largest decrease, at 8.0 pp. Kimi 2.6 drops by 5.0 pp, and Qwen 3.6-flash by 2.0 pp. We therefore apply future-commit cleanup in standard evaluations to prevent direct access to reference fixes and support fair comparisons.

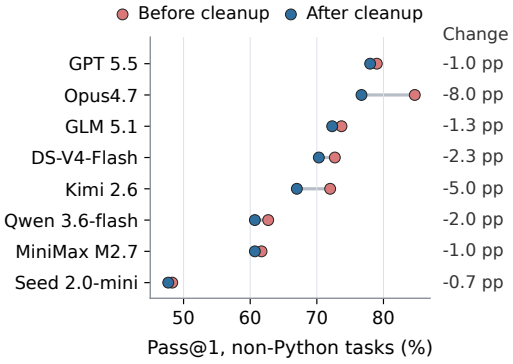


Figure 5: Effect of future-commit cleanup.

7 Conclusion

We introduced Claw-SWE-Bench and its 80-instance companion subset, Claw-SWE-Bench Lite. The shared adapter protocol connects general-purpose agents to the SWE-bench execution environment, patch submission process, and scoring system. This enables consistent, reproducible evaluation of their software engineering capabilities. We evaluated seven harnesses with three models using the same task prompts, outer runtime budgets, and task set. The results show that general-purpose harnesses can effectively complete real-world software engineering tasks. Even with the underlying model fixed, task success rates and running costs vary substantially across harnesses. SWE-style coding agent evaluation should therefore treat the harness as a key controlled variable rather than an implementation detail hidden behind model scores. We hope the full benchmark and Lite subset will provide stable reference points for future research. They support comparisons of new and evolved harnesses under the same tasks, budgets, and cost-accounting conventions. They also make low-cost debugging and replication easier to incorporate into routine research workflows.

AI Use Statement

In this work, we used generative AI tools to assist in reviewing dataset compliance and correctness. We have not used generative AI tools to develop theoretical models or conceptual frameworks, design or provide feedback on research methodology or experiments, implement methods, assist with translation, support qualitative and thematic data analysis, or interpret results. Generating synthetic datasets, proposing or refining research hypotheses, formulating mathematical claims, providing critical ingredients for mathematical proofs, writing proofs, and cleaning or reformatting datasets are not applicable to this work. Additionally, we used generative AI tools to improve the language and identify and correct typographical errors in English text written by the authors. We have reviewed all AI-assisted work. Specifically, the authors checked the language edits to ensure that they preserved the intended meaning and verified AI-assisted findings on dataset compliance and correctness against the original data sources and applicable licenses. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

Ethics Statement

Claw-SWE-Bench evaluates the software engineering capabilities of general-purpose agent harnesses on publicly available, real-world issue-resolution tasks. Relevant upstream license terms are described in Appendix E. Stronger software engineering capabilities could support routine software maintenance and help researchers debug code and develop research software. However, these capabilities could also be misused to exploit vulnerabilities without authorization. We release the benchmark evaluation protocol and its implementation, instance lists, and associated metadata to support systematic evaluation and reproducibility. We do not provide attack methods or tools specifically designed for vulnerability exploitation.

Reproducibility Statement

We include the Claw-SWE-Bench code, benchmark instance lists, and associated metadata in the supplementary material. The Experimental Setup section and Appendices A and B describe the experimental settings, hardware, and runtime environment in detail. They also provide commands for task execution and evaluation to support reproduction of the results.

References

Anthropic. Introducing Claude Opus 4.7. Official release announcement, 2026. URL https://www.anthropic.com/news/claude-opus-4-7. April 16, 2026.

DeepSeek-AI. Deepseek harness: Everything is a plugin. https://github.com/deepseek-ai/deepseek-harness, 2026.

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.

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Yang JingYi, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, Yubo Ma, Haodong Duan, Jing Shao, Jiaqi Wang, Dahua Lin, Kai Chen, and Yuhang Zang. Wildclawbench: A benchmark for real-world, long-horizon agent evaluation, 2026. URL https://arxiv.org/abs/2605.10912.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/drouin24a.html.

Zhiyu Fan, Kirill Vasilevski, Dayi Lin, Boyuan Chen, Yihao Chen, Zhiqing Zhong, Jie M. Zhang, Pinjia He, and Ahmed E. Hassan. SWE-Effi: Re-evaluating software AI agent system effectiveness under resource constraints. *arXiv preprint arXiv:2509.09853*, 2025. URL https://arxiv.org/abs/2509.09853v2.

Dawei Gao, Zitao Li, Xuchen Pan, Weirui Kuang, Zhijian Ma, Bingchen Qian, Fei Wei, Wenhao Zhang, Yuexiang Xie, Daoyuan Chen, Liuyi Yao, Hongyi Peng, Zeyu Zhang, Lin Zhu, Chen Cheng, Hongzhu Shi, Yaliang Li, Bolin Ding, and Jingren Zhou. AgentScope: A flexible yet robust multi-agent platform. *arXiv preprint arXiv:2402.14034*, 2024. URL https://arxiv.org/abs/2402.14034v2.

HKUDS. nanobot. Software and documentation, 2026. URL https://github.com/HKUDS/nanobot. Accessed September 24, 2026.

Marius Hobbhahn. SWEBench-verified-mini. Software and dataset, 2025. URL https://github.com/mariushobbhahn/SWEBench-verified-mini. Accessed September 24, 2026.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):220:1–220:79, 2024. doi: 10.1145/3695988. URL https://doi.org/10.1145/3695988.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks. *arXiv preprint arXiv:2607.07946*, 2026. URL https://arxiv.org/abs/2607.07946v1.

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.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 133–142. Association for Computing Machinery, 2002. doi: 10.1145/775047.775067. URL https://doi.org/10.1145/775047.775067.

Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=Zy4uFzMviZ.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Ndzomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, Juyong Lee, Tengjun Jin, Yifan Mai, Yifei Zhou, Yuxuan Zhu, Rishi Bommasani, Daniel Kang, Dawn Song, Peter Henderson, Yu Su, Percy Liang, and Arvind Narayanan. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=vUaY1t64ZZ.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026. URL https://arxiv.org/abs/2603.28052v1.

Xiangyi Li, Kyoung Whan Choe, Yimin Liu, Xiaokun Chen, Chujun Tao, Bingran You, Wenbo Chen, Zonglin Di, Jiankai Sun, Shenghan Zheng, Jiajun Bao, Yuanli Wang, Weixiang Yan, Yiyuan Li, and Han-chung Lee. ClawsBench: Evaluating capability and safety of LLM productivity agents in simulated workspaces. *arXiv preprint arXiv:2604.05172*, 2026. URL https://arxiv.org/abs/2604.05172v2.

Jiaqing Liang, Jinyi Han, Weijia Li, Xinyi Wang, Zhoujia Zhang, Zishang Jiang, Ying Liao, Tingyun Li, Ying Huang, Hao Shen, Hanyu Wu, Fang Guo, Keyi Wang, Zhonghua Hong, Zhiyu Lu, Lipeng

Ma, Sihang Jiang, and Yanghua Xiao. GenericAgent: A token-efficient self-evolving LLM agent via contextual information density maximization (V1.0). *arXiv preprint arXiv:2604.17091*, 2026. URL https://arxiv.org/abs/2604.17091v1.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. DeepSWE: Training a state-of-the-art coding agent from scratch by scaling RL. Technical blog and model card, 2025. URL https://huggingface.co/agentica-org/DeepSWE-Preview. Accessed September 24, 2026.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinyBenchmarks: evaluating LLMs with fewer examples. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 34303–34326. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/maia-polo24a.html.

Rafael Martí, Mauricio G. C. Resende, and Celso C. Ribeiro. Multi-start methods for combinatorial optimization. *European Journal of Operational Research*, 226(1):1–8, 2013. doi: 10.1016/j.ejor.2012.10.012. URL https://doi.org/10.1016/j.ejor.2012.10.012.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=fibxvahvs3.

MiniMax. MiniMax M2.7. Official model release, 2026. URL https://www.minimax.io/models/text/m27. Accessed September 24, 2026.

Nous Research. Hermes Agent. Software and documentation, 2026. URL https://github.com/NousResearch/hermes-agent. Accessed September 24, 2026.

OpenAI. Codex CLI. Official documentation, 2026. URL https://learn.chatgpt.com/docs/codex/cli. Accessed September 24, 2026.

OpenClaw. OpenClaw. Software and documentation, 2026. URL https://github.com/openclaw/openclaw. Accessed September 24, 2026.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Bytedance Seed. Seed2.0 model card: Towards intelligence frontier for real-world complexity, 2026. URL https://arxiv.org/abs/2607.00248.

TokenRhythm Technologies. OpenSquilla: Token-efficient agent = models + routing harness. aiXiv preprint, August 2026. URL https://aixiv.science/abs/aixiv.260822.000001. Version 1.0.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024. doi: 10.1007/s11704-024-40231-1. URL https://doi.org/10.1007/s11704-024-40231-1.

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, Daniel 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*, 2025a. URL https://openreview.net/forum?id=OJd3ayDDoF.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 13–19 Jul 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024a. URL https://openreview.net/forum?id=BAakY1hNKS.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. OS-Copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024b. URL https://arxiv.org/abs/2402.07456v2.

Anyi Xu, B Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, CC Yu, et al. Deepseek-v4. 1-flash: Pushing the limits of kv cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

John Yang, Kilian Lieret, Carlos Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-3239. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/8b86cf5ace600c48fd188efbb8dedec8-Paper-Datasets_and_Benchmarks_Track.pdf.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. A survey on evaluation of LLM-based agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26690–26714. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1330. URL https://aclanthology.org/2026.findings-acl.1330/.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. AssistantBench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8938–8968. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.505. URL https://aclanthology.org/2024.emnlp-main.505/.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

ZeroClaw Labs. ZeroClaw. Software and documentation, 2026. URL https://github.com/zeroclaw-labs/zeroclaw. Accessed September 24, 2026.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1592–1604. Association for Computing Machinery, 2024. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

A Reproducibility Statement

This appendix consolidates the artefacts and protocol required to reproduce every cell of §6.2.

A.1 Code release

Harness adapters. Adapters for seven claws—OpenClaw, Hermes Agent, ZeroClaw, nanobot, GenericAgent, DeepSeek Harness, and Meta-Harness—are released. The release includes `run_infer.py` and `run_eval.py` together with the per-harness orchestrator, agent adapter, and workspace modules.

A.2 Data release

Claw-SWE-Bench (350 instance IDs plus metadata) and **Claw-SWE-Bench-Lite** (80 instance IDs with cost-aware selection metadata) are released as JSON files. The underlying issues and repositories are hosted on the upstream SWE-bench-Multilingual and SWE-bench-Verified-Mini sources.

A.3 Reproduction protocol

Per-instance run.

```
python3 run_infer.py \
    --claw <claw_name> \
    --dataset multilingual \
    --instance_file config/multilingual_300_instances.txt \
    --model <provider/model_name> \
    --run_id <run_label> \
    --timeout 3600 \
    --workers 3
```

Evaluation.

```
python3 run_eval.py \
    --predictions artifacts/<run_id>/predictions.jsonl \
    --dataset_name SWE-bench/SWE-bench_Multilingual \
    --run_id <run_id> \
    --max_workers 8
```

The commands above cover the 300 Multilingual instances. For the 50 Verified-Mini instances, use `-dataset verified` and `-instance_file config/verified_mini_50.txt` for inference, and `-dataset_name princeton-nlp/SWE-bench_Verified` for evaluation, with a separate run ID.

All seven claws share these inference and evaluation entry points. Claw-specific model settings are documented in the released README; Meta-Harness uses `-claw generic -candidate <candidate_dir>`.

A.4 Model Pricing and API Cost Accounting

We calculate API costs using fixed unit prices and the per-call token usage recorded by each harness. For a given model, the same prices apply across all harnesses. We account separately for uncached input, cached input, and output tokens. Table A.1 lists the prices used for the three models in the main experiment. All prices are in USD per million tokens.

Qwen pricing tier and cache pricing. We use the Qwen 3.6-flash pricing tier for requests with at most 256K input tokens. The official model pricing table does not list a separate price for cached input. The Context Cache documentation specifies that implicit cache hits are charged at 20% of the standard input price. We therefore use a cached-input price of $0.25 \times 20\% =$ US$0.05 per million tokens.

Table A.1: API prices for the three models used in the main experiment (USD per million tokens).

Model	Uncached input	Cached input	Output	Source
GLM-5.1	1.40	0.26	4.40	Z.ai official pricing page
Qwen 3.6-flash	0.25	0.05	1.50	Alibaba Cloud Model Studio international pricing page (Singapore) and Context Cache documentation
DeepSeek-V4.1-Flash	0.30	0.006	1.20	DeepSeek official peak-hour pricing

DeepSeek peak and off-peak pricing. Peak hours for DeepSeek-V4.1-Flash are 01:00 to 04:00 and 06:00 to 10:00 UTC on weekdays. Prices are halved at all other times. The off-peak prices for uncached input, cached input, and output are US$0.15, US$0.003, and US$0.60 per million tokens, respectively. We use the peak-hour prices in Table A.1 for all cost calculations, without off-peak discounts. This makes cost accounting independent of when API calls are made.

Cost calculation. The cost of each API call is calculated as

$$C = \frac{N_{\text{fresh}} p_{\text{in}} + N_{\text{cache}} p_{\text{cache}} + N_{\text{out}} p_{\text{out}}}{10^6}.$$

Here, N_{fresh}, N_{cache}, and N_{out} denote the numbers of uncached input, cached input, and output tokens, respectively. The corresponding unit prices are p_{in}, p_{cache}, and p_{out}, as listed in Table A.1. Cached input tokens are excluded from N_{fresh} to avoid double counting. The cost of a full evaluation run is the sum of all API call costs across all task instances. Costs in the main table are first summed within each full run and then averaged over the three runs.

B Compute and Environment Details

This appendix documents the hardware, run-time parameters, and aggregate wall-clock cost of the experiments reported in Section 5. All seven claws (OPENCLAW, HERMES-AGENT, ZEROCLAW, NANOBOT, GENERIC, DEEPSEEK HARNESS, and META-HARNESS) were executed on a single host with identical run-time parameters; the claw implementation is the only experimental variable.

B.1 Hardware

Item	Value
Architecture	x86_64
OS	Linux 6.8.0-106-generic (Ubuntu)
CPU cores	16
RAM	61 GiB total
GPU	None (model inference is API-only)

Table B.1: Host hardware. Model inference runs on remote provider APIs, so no local GPU is required; the host is used only for harness orchestration, Docker containers, and patch evaluation.

B.2 Run-time parameters (held equal across all seven claws)

Parameter	Value
Per-instance timeout	3600 s (60 min)
Repeats per instance	3
Concurrency	3 threads
Shell tool timeout (each harness)	180 s

Table B.2: Run-time parameters for the main harness comparison. These are overridden via CLI flags so that all seven claws see the same per-instance budget; the prompt template is also identical across claws.

B.3 API provider

All model inference is routed through OpenRouter. **API keys are NOT included in any released artifact**.

B.4 Compute cost (aggregate)

The 21 full-350 runs used for Lite calibration cover seven claws paired with three models, totaling 7,350 instance executions. Their summed per-instance durations amount to approximately **1,430 hours**. This total excludes the additional OpenClaw model sweep, the adapter diagnostic (§6.1), the pre-cleanup runs (§6.5), and Lite-80 validation runs.

C Meta-Harness Implementation Details

We adapt the original outer-loop implementation of Meta-Harness to Claw-SWE-Bench and perform a single evolution round starting from GenericAgent. The resulting harness is reported as Meta-Harness in Table 2. The benchmark adapter and evaluation protocol remain fixed throughout this process.

Seed harness and search space. We initialize evolution from GenericAgent, an open-source Python harness with approximately 3,700 lines of code. The candidate contains a complete copy of the harness source tree. The proposer can modify the agent loop, tools, system prompts, and context management. The evaluation adapter, model configuration, and instance list are excluded from modification.

Proposer and feedback. We use DeepSeek-V4.1-Flash as the proposer model, with reasoning effort set to `xhigh`. The proposer operates in an isolated working directory and can inspect harness source code, per-instance scores, and full execution traces. These artifacts include `agent_stdout.log`, `git.patch`, and evaluation outputs such as `report.json`. It produces one candidate and a falsifiable hypothesis about the proposed change. We adapt the original SKILL instructions to the SWE-bench setting. The instructions require the proposer to identify failure patterns in the traces before editing and to target a single mechanism. They prohibit hard-coded instance identifiers, repository names, and other task-specific content, as well as Git-history inspection and network retrieval.

Search data and auditing. Following the TerminalBench-2 setting of Meta-Harness, we use all 350 Claw-SWE-Bench instances for both evolution and final evaluation, without a held-out task split. To detect task-specific modifications, we apply regular-expression checks to added code for instance identifiers and repository names. We also manually inspect the complete candidate diff. These checks target explicit task-specific code; they do not establish generalization to unseen tasks.

Final evaluation and cross-model transfer. During evolution, both the proposer and the task-execution model use DeepSeek-V4.1-Flash. After evolution, we freeze the resulting harness. We evaluate it with DeepSeek-V4.1-Flash and transfer it to GLM 5.1 and Qwen 3.6-flash by changing only the execution model, without further evolution. The proposer does not participate in these final

task executions. This experiment assesses cross-model transfer on the same benchmark tasks. The main-table results follow the standard evaluation configuration, with three separate runs per model.

Evolution cost. Proposal generation takes approximately eight minutes and costs less than US$0.10 in API fees. The full-350 evaluation of the candidate costs approximately US$13 and takes about 13 hours with eight concurrent workers.

D LITE-80 CONSTRUCTION DETAIL

This appendix records the released Lite-80 construction used in Section 4. The current release is built using full-350 results from 21 harness–model pairs spanning seven claws and three models.

D.1 CALIBRATION POOL AND CONSTRAINTS

Lite selection is calibrated on 21 evaluation columns: seven claws (OpenClaw, Hermes, ZeroClaw, NanoBot, GenericAgent, DSH, and Meta-Harness) paired with GLM 5.1, Qwen 3.6-flash, and DeepSeek-V4.1-Flash. Within-model claw comparisons use this 7×3 grid.

Based on the K-sweep analysis described in Section 4.2, we choose 10 instances per language, giving an 80-instance Lite subset. Within a language, the 10 selected instances must follow the fixed $2/3/3/2$ allocation over difficulty quartiles $Q_1/Q_2/Q_3/Q_4$. Quartiles are computed from the mean resolved rate across the calibration pool, so the strata reflect multi-model and multi-claw difficulty rather than a single harness's behavior.

D.2 OBJECTIVE AND SOLVER

Let $x_i \in \{0, 1\}$ indicate whether instance i is selected. The selection loss combines two terms. First, a resolve-rate L1 term matches Lite-implied and full-350 rates over the 21×8 grid of calibration column by language. Second, a pairwise ranking hinge penalizes column inversions: for column pairs whose full-set rates differ by more than $\texttt{RANK_EPS} = 0.05$, the loss is active when the Lite-predicted ordering is wrong or within a margin of 0.025.

D.3 K-SWEEP DECISION

Table D.1 summarizes the K-sweep on the 21-column calibration pool. The scan evaluates resolve-rate, cost, and operational criteria under different sensitivity scenarios. The minimum acceptable sizes range from 8 to 10 instances per language ($K^* \in [8, 10]$). We select the largest of these minima, $K = 10$, yielding $8 \times 10 = 80$ instances.

Table D.1: Subset-size sensitivity analysis on the 21-column calibration pool. Each scenario row reports the smallest K meeting the evaluation criteria.

Scenario	Driver class	K^*_{scenario}
main	–	9
MARGIN_0	margin	10
MARGIN_0.5overK	margin	10
restart_50	restart	10
restart_500	restart	8
seed_offset_1000	seed	8
seed_offset_2000	seed	10
mirror_B	mirror parity	9
mirror_A	mirror parity	9
release	$K^*_{\max}$	**10**

D.4 COVERAGE AND COST

The released Lite-80 contains 38 unique repositories, covering approximately $38/43 = 88\%$ of the repositories in the full benchmark. The mean Lite/full run-cost ratio is 25.3%. This supports

the intended use of Lite as an approximately four-times cheaper evaluation surface for debugging, regression testing, and preliminary model or claw comparisons.

E LICENSE AND ETHICS

License of the released artifact. This benchmark is derived from two upstream sources, both released under MIT. **(i) SWE-bench Multilingual** is hosted at huggingface.co/datasets/SWE-bench/SWE-bench_Multilingual, released as part of the official SWE-bench project. **(ii) SWE-bench Verified-Mini** is an MIT-licensed subset of SWE-bench Verified (the OpenAI-validated 500-instance Python subset), hosted at github.com/mariushobbhahn/SWEBench-verified-mini.

We retain both upstream LICENSE files and citations. The underlying source code in each task instance retains the license of its original GitHub repository. Users redistributing patches or derivative work must comply with the per-repository license. `REPO_LICENSES.md` in the released artifact lists every underlying repository and its current upstream license.

F VARIATION ALONG THE LLM AXIS

F.1 AGGREGATE RESULTS

We examine variation along the LLM axis using OpenClaw as the reference claw with eleven models on the full 350-instance set. The models are GPT 5.5, Claude Opus 4.7, GLM 5.1, DeepSeek-V4 Pro and Flash, Kimi K2.6, MiniMax M2.7, Qwen 3.6-flash, Seed 2.0-mini, MiMo-v2.6-flash, and LongCat-2.0. Table F.1 reports aggregate results. The highest resolved rate is achieved by GPT 5.5, at 78.0% (273/350), followed by Claude Opus 4.7 at 77.1% (270/350). The lowest cell is Seed 2.0-mini, at 48.6% (170/350). The observed Pass@1 spread across these OpenClaw runs is 29.4 pp.

Accuracy ranking, however, is not cost ranking. GPT 5.5 has the highest Pass@1, but its full 350-instance run costs $1399.1; Claude Opus 4.7 is only 0.9 pp lower, with cost $1082.0. By contrast, DeepSeek-V4 Pro reaches 71.7% Pass@1 at total cost $81.3, while DeepSeek-V4 Flash reaches 70.3% at only $8.2. Qwen 3.6-flash reaches 60.9% Pass@1 at $77.5; GLM 5.1 reaches 72.0% at $290.6.

Table F.1: LLM-axis variation: OpenClaw × 11 models on full-350, with one run per model. Cost is total API cost (USD); Dur is mean duration per instance (seconds); In/Out are total non-cached input/output tokens (millions); Turns is mean turns; Cache is the token-weighted input cache hit rate (%). Rows are grouped by model type and sorted by Pass@1 within each group. The best Pass@1 and lowest Cost are in **bold**.

Model	Type	Resolved	Pass@1	Cost	Dur	In(M)	Out(M)	Turns	Cache
GPT 5.5	Flagship	**273**	**78.0**	1399.1	603.7	40.3	15.7	67.0	97.3
Claude Opus 4.7	Flagship	270	77.1	1082.0	424.6	35.6	6.2	61.6	97.0
GLM 5.1	Flagship	252	72.0	290.6	511.2	60.0	7.0	57.2	91.9
DeepSeek-V4 Pro	Flagship	251	71.7	81.3	662.3	19.3	11.0	47.1	97.4
LongCat-2.0	Flagship	235	67.1	478.0	1091.4	566.8	21.7	65.3	58.2
Kimi K2.6	Flagship	234	66.9	633.7	1235.3	75.6	12.1	78.7	92.1
MiniMax M2.7	Flagship	215	61.4	196.7	1165.6	25.0	9.0	94.8	96.2
MiMo-v2.6-flash	Flash	261	74.6	22.3	1791.0	60.8	36.4	57.7	95.5
DeepSeek-V4 Flash	Flash	246	70.3	**8.2**	430.0	13.8	12.9	51.2	98.5
Qwen 3.6-flash	Flash	213	60.9	77.5	638.9	31.7	8.8	68.2	97.3
Seed 2.0-mini	Flash	170	48.6	19.4	1153.0	89.8	21.6	44.4	79.4

F.2 PER-LANGUAGE RESOLVED RATE

Table F.2 expands the aggregate results in Table F.1 by language for the same eleven OpenClaw–model pairs. Models are listed in descending order of overall resolved rate; each Total matches the corresponding Pass@1 in Table F.1.

Table F.2: Per-language resolved rate (%) under OpenClaw for the 11 models in Table F.1, using the same runs and accounting. Best language per model in **bold**, worst underlined.

Model	Java	Go	Rust	JS/TS	C/C++	Ruby	PHP	Python	Total
GPT 5.5	86.0	61.9	**93.0**	79.1	81.0	70.5	74.4	78.0	78.0
Claude Opus 4.7	86.0	61.9	**88.4**	81.4	71.4	68.2	76.7	82.0	77.1
MiMo-v2.6-flash	**86.0**	64.3	81.4	72.1	78.6	70.5	74.4	70.0	74.6
GLM 5.1	76.7	59.5	**81.4**	69.8	76.2	68.2	74.4	70.0	72.0
DeepSeek-V4 Pro	**79.1**	50.0	**79.1**	74.4	73.8	72.7	76.7	68.0	71.7
DeepSeek-V4 Flash	**81.4**	50.0	79.1	74.4	73.8	63.6	69.8	70.0	70.3
LongCat-2.0	74.4	54.8	**81.4**	67.4	61.9	63.6	67.4	66.0	67.1
Kimi K2.6	69.8	50.0	**74.4**	65.1	71.4	68.2	69.8	66.0	66.9
MiniMax M2.7	65.1	47.6	**76.7**	55.8	61.9	61.4	55.8	66.0	61.4
Qwen 3.6-flash	60.5	47.6	**69.8**	48.8	66.7	68.2	62.8	62.0	60.9
Seed 2.0-mini	**60.5**	33.3	55.8	44.2	54.8	36.4	48.8	54.0	48.6

Commentary. Java and Rust dominate the per-model maxima: Rust is the best language for 7 of 11 models, Java for 3, and the two tie for DeepSeek-V4 Pro. Go is the worst language for all 11 models without exception (range 33.3–64.3%), sitting 10–22 pp below each model's overall mean.