WHAT DOES A HARNESS BUY? TOKENS, MOSTLY.

Anonymous authors
Paper under double-blind review

ABSTRACT

A coding agent is a language model wrapped in a harness: the system prompt, the tool set, and the context management that turn a chat model into something that can work inside a repository. Much of the day-to-day engineering of these agents happens in the harness. Production harnesses ship releases daily, vendors advertise pass-rate gains from harness changes, and leaderboards mix harnesses freely. What is rarely measured is how much the harness itself moves the score when the model is held fixed. This paper measures it. We run five models through three production harnesses, Claude Code, mini-SWE-agent, and OpenCode, on SWE-bench Verified, and rerun the same configurations to calibrate how much a score moves when nothing changes but the run. The answer to the title question is: not pass rate. On 447 tasks and the two models we ran there, Claude Code and mini-SWE-agent, the heaviest and the lightest harness, are equivalent within five points. On a 45-task hard subset and five models, swapping the harness flips as many tasks as rerunning the same harness, 13% in both cases, and the tasks a harness wins in one run are not the tasks it wins in the next. The one harness effect that clears the noise is a loss, not a gain: OpenCode trails by up to 9 points on the large pool, and on one model half of that gap sits in runs its output cap cut short. What the harness does decide is the bill. With the same model, the same tasks, and one price list, cost per task differs by up to 3$\times$ across harnesses. The gap is set at the first call, by the preamble of system prompt and tool schemas each harness sends with every step, and scaled by the number of steps; per-step growth and per-call tool output differ far less. The provider's price for cached input scales the bill and does not reorder it. The rerun data also give the resolution a harness comparison needs: at the discordance we observe, 45 tasks catch a 13-point gap only half the time and no gap with 80% power, and 447 tasks resolve 5 points, still coarser than the gains many harness changes claim.

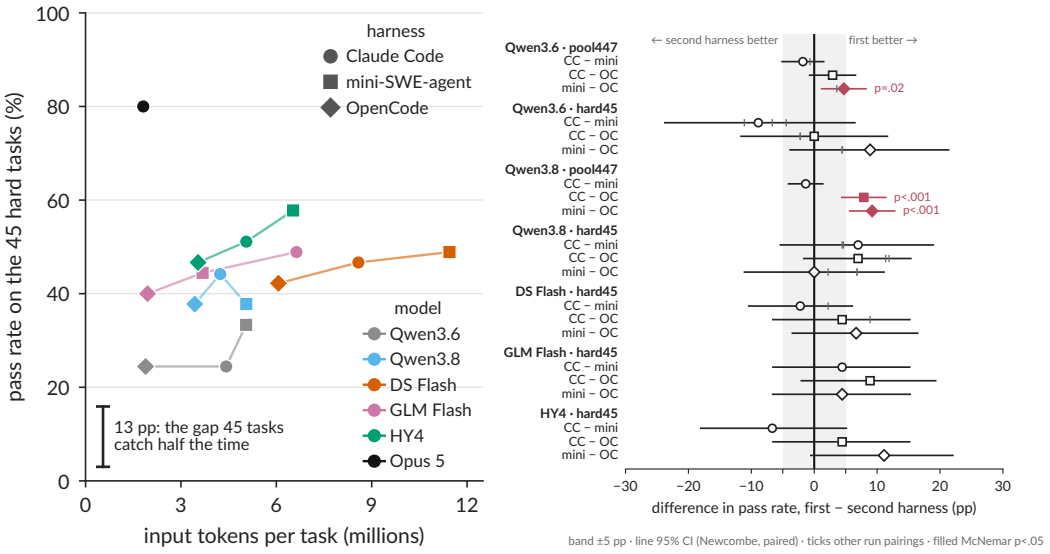
1 INTRODUCTION

Coding agents have turned into a product category, and the product is a model plus a harness. The harness, which part of the literature calls a scaffold, is everything around the model: the system prompt, the tools it can call, the way tool output is fed back, and the way the conversation is trimmed when it grows too long. It is also the part that changes most: popular harnesses ship more than two releases a day (Ben Sghaier et al., 2026), most of it prompts, tool surfaces, and context management rather than the model.

The field has come to believe that this engineering moves the score. The belief has evidence behind it. On a public terminal benchmark, the same model is listed at 52.1% under one harness and 57.8% under another, with the lower-scoring arm consuming 65$\times$ the input tokens (Vats & Golev, 2026). Six harnesses run over the same model pool span 23.8 points (Yao et al., 2026), and five harnesses on one fixed model span 27.4 points (Zheng et al., 2026). Vendors sell harnesses on this basis: one advertises a 2.4-point gain over Claude Code with its own model held fixed (Zhipu AI, 2026). Leaderboards, meanwhile, mix harnesses freely, so a ranking of models is also a ranking of whatever harness each submitter chose (Zhang et al., 2026).

These numbers are hard to read, for two reasons. First, almost all of them come from one run per cell, while ten reruns of one configuration on SWE-bench Verified (Jimenez et al., 2024; OpenAI,

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.



(a) (b)

Figure 1: **Same model, different harness.** (a) Input tokens per task against pass rate on the 45 hard tasks, one run of every model under every harness. Within a model the harnesses spread two- to three-fold in tokens and stay inside the gap that 45 tasks catch only half the time. (b) Paired pass-rate difference for each pair of harnesses on each model and task set, with 95% intervals for the paired difference; vertical ticks mark the rerun range where a configuration was run more than once. Filled points are the pairs that clear the noise, and all of them are OpenCode trailing on the 447-task pool.

2024) already spread by 2.2 to 6.0 points (Bjarnason et al., 2026). Second, the harnesses being compared are often not on the same footing. In the same study, one harness scores 19.1% with a minimal adapter to the benchmark and 73.4% with the full one on the same backbone (Zheng et al., 2026), so part of what such a comparison measures is how well each harness was wired to the benchmark. The study closest to ours shows how much this leaves open. It runs three production harnesses on two models over 50 tasks, one run per cell, and finds within-model differences of 0 to 8 points with intervals that cover zero except the largest, beside a 40× spread in tokens per solved task (Vats & Golev, 2026). Whether those 8 points come from the harness or from the run cannot be told from that design, since ten reruns of a single configuration already spread by up to 6 points. So the question a harness claim rests on, whether swapping the harness moves a score more than running the same harness again, has not been measured on the same tasks and models.

We measure it. We take three production harnesses as they ship, Claude Code, mini-SWE-agent, and OpenCode (Anthropic, 2026; Lieret et al., 2026; Anomaly Innovations, 2026), pin their versions, and run the same models through all three on SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024). The models are Qwen3.6-35B-A3B, Qwen3.8-27B, DeepSeek-V4-Flash, GLM-5.3-Flash, and HY4-Preview, with Claude Opus 5 added inside Claude Code as a strong anchor. Of the 492 Verified tasks that run in our offline setup, two sets play different roles. The 45 tasks in the two hardest annotated difficulty bands, all of them, are where we expect a harness to matter most. The remaining 447 give the power to bound a small effect. Ten rerun pairs of the same configuration let the harness gap be read next to the rerun gap, all runs are offline, and every arm is billed under one price list from its own token counts.

Held to that standard, the harness barely moves pass rate (Figure 1). On the hard tasks, the spread across the three harnesses is two to five tasks out of 45 for every model, and rerunning one harness moves it by up to three. The two flip the same share of tasks, 13% at the median, and the tasks a harness wins in one run are not the tasks it wins in the next. On the 447 tasks, Claude Code and mini-SWE-agent are equivalent within five points on both models we ran there. We also looked where a harness effect should hide if it existed: a vendor's own harness on its own model scored inside the range of the other three.

What does clear the noise is a loss. OpenCode trails by up to 9 points on the large pool, and half of that gap sits in runs its output cap cut short before a patch was written. Every other harness failure we can name is of the same kind: a hard stop at the context window, a crash at the step cap, a search tool that could not run offline. Every harness effect we can name is a way to lose a task, and none we measured is a way to win one.

The harness does decide the bill, and it decides it in a way the pass rate hides. With the same model on the same tasks, cost per task differs by up to 3× across the three harnesses. Little of that comes from what the tools return: per call, the three harnesses hand the model about the same amount of text. It comes from what each harness carries into every step, a fixed preamble of system prompt and tool schemas that is 16k tokens in Claude Code, 7k in OpenCode, and under a thousand in mini-SWE-agent, plus the model's own reasoning kept in the transcript, which our reconstruction says is re-sent on every turn. The third term is the number of steps the model takes. Step by step, the three contexts then grow at rates within a factor of one and a half of one another, so the bill is set at the first call and multiplied by the step count. The price of cached input scales that bill and does not reorder it: under either vendor's price list, the three harnesses rank by their tokens.

The rerun data also settle what a harness comparison can resolve. At the discordance we observe, 45 tasks catch a 13-point gap half the time and no gap at 80% power, and 447 tasks resolve about 5 points. Many of the published harness gains sit below the first number, and some sit below the second as well.

2 Related Work

Harness comparisons. Several recent studies swap whole harnesses under a fixed model and report large spreads. Six configurable harnesses over the same model pool span 23.8 points on a graded task score (Yao et al., 2026), five harnesses on one model span 27.4 points of Pass@1 (Zheng et al., 2026), and one fixed model costs 139× more tokens under its worst scaffold than under its best on a single git task (Alier Forment et al., 2026). Two findings in this line point the way we go. Three production harnesses paired on the same terminal tasks differ by 0 to 8 points with intervals that cover zero except the largest, while tokens per solved task differ 40× (Vats & Golev, 2026), and 35 consecutive releases of one harness bring no significant score gain on a fixed model while nearly doubling token use (Ben Sghaier et al., 2026). The scaffold papers on SWE-bench before them, SWE-agent, OpenHands, and the agentless pipeline, each compared their own design against the others under a fixed model (Yang et al., 2024; Wang et al., 2025; Xia et al., 2025). Every study in this line runs each cell once or twice. Rankings also move with the harness: which model is most robust to meaning-preserving rewrites of a task depends on whether it runs under mini-SWE-agent or OpenCode (Mahmud et al., 2026), and a position paper asks that comparisons disclose the harness for this reason (Zhang et al., 2026). We keep the whole-harness unit and the production artifacts, and add the two things this line lacks: reruns of the same cell, and a bill computed under one price list. On pass rate our result agrees with theirs, with the power to bound it.

What is inside a harness. A source-code taxonomy of coding agents lists twelve design dimensions and asks for exactly the control we use, a fixed model with the scaffold varied (Rombaut, 2026). The three harnesses we run sit at different points of that taxonomy. mini-SWE-agent gives the model a single `bash` tool and no context management, and reaches SWE-agent's score in about a hundred lines of Python (Yang et al., 2024; Lieret et al., 2026). Claude Code and OpenCode add typed file tools, long system prompts, and automatic compaction (Anthropic, 2026; Anomaly Innovations, 2026). Component studies show that individual pieces matter under pressure: a context-management package nearly doubles a small model's score in a 20k-token window (Lewis, 2026), and restricting the tool surface changes cost by task regime and agent while pass rates stay tied (Yang et al., 2026). A component study on SWE-bench Verified varies context management, action space, and planning inside one lightweight harness over 176 matched settings, and finds that context management pays as the window tightens and that capable models do as well with a bash-only tool set at lower cost (Fan et al., 2026). We ask the complementary question: what the whole production harness, as users run it, changes against a rerun of itself. Models fine-tuned on one scaffold's trajectories lose as much as 29 points when moved to another, while base models vary far less (Thangarajah et al., 2026). Our check of a vendor's own harness on its own model is the same question asked of an API model.

Noise and power in agent evaluation. The case for error bars on evaluations has been made for language models in general (Miller, 2024) and for agents in particular, where a score reported without its cost or its variance rewards the wrong things (Kapoor et al., 2025). Ten reruns of one model–scaffold pair on SWE-bench Verified spread by 2.2 to 6.0 points, with standard deviations above 1.5 points even at temperature zero, so reported gains of 2 to 3 points can be noise (Bjarnason et al., 2026). A generalizability-theory decomposition finds that the agent main effect explains under 3% of outcome variance while agent-by-task interaction explains 7 to 23% (Srinivasan, 2026), and a pre-registered comparison on GAIA fits the scaffold-by-model interaction directly and finds it significant (Starace, 2026). We take the rerun spread as the yardstick, and report every harness gap next to it with the sample size it would take to resolve.

3 SETUP

Harnesses. We run three production harnesses as they ship, with the version pinned per arm: Claude Code 2.1 (releases 2.1.259 to 2.1.273 in the scored arms; 2.1.263 on GLM and Opus), mini-SWE-agent 2.4.6, and OpenCode 1.18 (1.18.30 and 1.18.31 in the scored arms; 1.18.30 on GLM) (Anthropic, 2026; Lieret et al., 2026; Anomaly Innovations, 2026). The three span the design space of Section 2: one minimal harness and two full products, one closed and one open. They differ in what they give the model. mini-SWE-agent exposes one `bash` tool, trims only very long observations, and has no context management beyond a hard stop at the model's window. Claude Code and OpenCode expose typed file, search, and edit tools, prepend long system prompts, and compact the transcript automatically when it grows. None of them sends sampling parameters, so decoding is whatever the provider does by default. We hold one budget fixed across all three: a step is one model call, and every trial gets 300 of them. Wall-clock limits are a safety net, and on the locally served Qwen arms they are not equal: Claude Code, whose steps against vLLM run about twice as slow, could run for 36,000 s where OpenCode's own soft timeout stopped it at 5,760 s (Appendix A). On the GLM model, where the API accepts a reasoning-effort setting, all three harnesses pin it to `high`. Everything else is left as the harness authors set it, and Table A1 in the appendix lists what that is. We also run DeepSeek's own command-line harness, dsh in the tables, on DeepSeek's model as a home-field check.

Models. The ladder has five models that run in all three harnesses: Qwen3.6-35B-A3B and Qwen3.8-27B, served locally with vLLM 0.27.1 in bfloat16 precision, and DeepSeek-V4-Flash (release 0731), GLM-5.3-Flash, and HY4-Preview, called through their vendor APIs (DS Flash, GLM Flash, and HY4 for short in figures and wide tables). The three API models are among the most used on OpenRouter: in its trailing-30-day ranking by tokens processed through 2026-09-20, DeepSeek-V4-Flash-0731 ranks second, HY4-Preview third, and GLM-5.3-Flash fourth (OpenRouter, 2026). Claude Opus 5 runs inside Claude Code only, through the harness's own channel, and anchors the top of the ladder. Harbor 0.22.0 (Harbor Framework Team, 2026) orchestrates the runs, one container per trial, and runs the benchmark's own tests to score each one.

Tasks. SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024) has 500 tasks, and 8 of them need a public web service to verify and are dropped, leaving 492. From these we take two disjoint sets. hard45 is every task in the two hardest annotated difficulty bands, 1 to 4 hours and over 4 hours, 42 and 3 tasks, taken whole so that there is no sampling to tune. pool447 is the remaining 447 tasks. Every arm on hard45 runs the same 45 tasks. On pool447 we run the two Qwen models, which we serve ourselves. The agent phase has no network beyond the model endpoint. An audit of our early web-enabled runs found that 71% of DeepSeek and Qwen trials had fetched the upstream fix from GitHub, so those runs are discarded and the condition is enforced at the network layer (Appendix D). Offline, OpenCode's search tools fail because they download a binary at first use. We place it ahead of time and report that arm, marked rg in the tables.

Runs and scoring. The quantity we estimate is the difference in pass rate between two harnesses on the same model over the tasks of a set, for one run of each configuration. The reported score of a cell is its first scheduled run of every task, a rule fixed before the results were read. A trial is usable if it was scored and raised no infrastructure exception. Trials that overflowed the window or hit a step, time, or output limit are usable and are scored by the verifier like any other trial, which nearly all of them fail. Where the first run lost a task to infrastructure, the slot is refilled

Table 1: **Paired comparison on pool447.** For each pair $A - B$, b is the number of tasks passed by A only and c by B only. Δ is the paired difference with a Newcombe 95% interval, p is McNemar's exact test, unadjusted. The last column reads *equivalent* when the two one-sided tests at δ = 5 points pass on the 90% interval, *inferior* when McNemar's test rejects, and *inconclusive* otherwise.

Model	Pair $A - B$	n	A	B	b	c	Δ (pp)	McNemar p	Verdict
Qwen3.6-35B-A3B	CC−mini	445	310	318	26	34	−1.8 [−5.2, +1.6]	.37	equivalent
	CC−OC	445	310	297	43	30	+2.9 [−0.9, +6.7]	.16	inconclusive
	mini−OC	445	318	297	45	24	+4.7 [+1.1, +8.4]	.02	inferior B
Qwen3.8-27B	CC−mini	444	347	353	17	23	−1.4 [−4.2, +1.5]	.43	equivalent
	CC−OC	444	347	312	52	17	+7.9 [+4.3, +11.5]	<.001	inferior B
	mini−OC	445	353	312	57	16	+9.2 [+5.5, +12.9]	<.001	inferior B

Table 2: **Pass rate on hard45.** Each cell is the first run, as passed/45 with the Wilson 95% interval; the superscript is how many times that cell was run. Claude Opus 5 runs inside Claude Code only. Qwen3.8-27B in Claude Code has 43 scored tasks; two trials were lost to infrastructure. The pass count of every rerun is in Table A3.

Model	Claude Code		mini-SWE-agent		OpenCode (rg)	
Qwen3.6-35B-A3B	11/45 $^{\times 2}$	24% [14, 39]	15/45 $^{\times 2}$	33% [21, 48]	11/45 $^{\times 2}$	24% [14, 39]
Qwen3.8-27B	19/43 $^{\times 2}$	44% [30, 59]	17/45 $^{\times 2}$	38% [25, 52]	17/45 $^{\times 2}$	38% [25, 52]
DeepSeek-V4-Flash	21/45 $^{\times 2}$	47% [33, 61]	22/45 $^{\times 1}$	49% [35, 63]	19/45 $^{\times 1}$	42% [29, 57]
GLM-5.3-Flash	22/45 $^{\times 1}$	49% [35, 63]	20/45 $^{\times 1}$	44% [31, 59]	18/45 $^{\times 1}$	40% [27, 55]
HY4-Preview	23/45 $^{\times 1}$	51% [37, 65]	26/45 $^{\times 1}$	58% [43, 71]	21/45 $^{\times 1}$	47% [33, 61]
Claude Opus 5	36/45 $^{\times 3}$	80% [66, 89]		—		—

from a rerun of the identical configuration on that task, which never displaces a usable first-run trial (Appendix C). Tasks with no usable trial are removed from that cell's denominator, and the count is shown wherever it is below the size of the set. Reruns of the same cell are kept apart and used only to measure how much a score moves on its own. A trial that hits the step cap, the wall-clock limit, the context window, or the output cap is scored as a failure, and we record which of these ended it. Task-level outcomes are paired across harnesses on the same model, so a harness gap is tested with McNemar's exact test and reported with a Newcombe interval, neither adjusted for the number of pairs. On pool447 we also test equivalence at δ = 5 points with two one-sided tests, which pass when the 90% interval of the difference sits inside $\pm\delta$. For each task set we report the smallest gap the exact test detects at 80% power, computed from the discordance we observe, and where no gap reaches that power we say so. Cost is computed from each trial's own token counts, split into cache-hit input, cache-miss input, and output, under the vendor's list price for the model. The two vendor-API models in the cost analysis, DeepSeek-V4-Flash and GLM-5.3-Flash, therefore carry a bill, the locally served models carry token counts only, and Opus runs on a subscription with no per-token price.

4 Results

4.1 The harness barely moves pass rate

The 447-task pool has the power to bound the harness effect, and the bound is tight (Table 1). Claude Code and mini-SWE-agent, the heaviest harness and the lightest, differ by -1.8 points on Qwen3.6-35B-A3B and -1.4 on Qwen3.8-27B. On both models the 90% interval of the difference sits inside $\pm$5, so the two are equivalent at that margin and Claude Code is at most about one point ahead. The tasks on which they disagree split almost evenly, the b and c columns of the table, as they do for the one rerun pair on the pool. The pairs that do clear the noise all have OpenCode on the losing side, by +4.7 points on Qwen3.6-35B-A3B and +7.9 to +9.2 on Qwen3.8-27B. Under Holm's correction over the six comparisons on the pool, the two Qwen3.8-27B gaps survive and the Qwen3.6-35B-A3B gap does not. Section 4.3 traces those gaps to how OpenCode ends a trial.

The 45 hardest tasks are where we expect a harness to matter most, and there the harness gap is the size of the rerun gap (Table 2). The spread across the three harnesses is two to five tasks for every model, and rerunning one cell moves it by up to three. The ordering is not stable either: mini-SWE-agent leads on three of the five models and Claude Code on the other two, and every paired difference on this set has an interval that covers zero (Figure 1b). HY4-Preview, the strongest model in all three harnesses, does not break the pattern: Claude Code trails mini-SWE-agent by -6.7 points and mini-SWE-agent leads OpenCode by +11.1, both below what 45 tasks can resolve (p = .45 and .12). On this set, a ranking of harnesses cannot be told from a ranking of runs.

4.2 The noise floor and what it can resolve

Swapping the harness flips as many tasks as rerunning the same harness (Figure 2). The flip rate of a pair of runs, the share of tasks whose outcome differs between them, puts a rerun, a harness swap, and a model swap on one scale. Reruns on hard45 flip 13% of tasks at the median, and harness swaps flip the same 13%. Swapping the model flips more, 22% at the median, and the pool shows the same picture at a larger n (Appendix F). The score moves less than the flip rate because flips go both ways, but the task-level churn from changing the harness is the churn of a rerun.

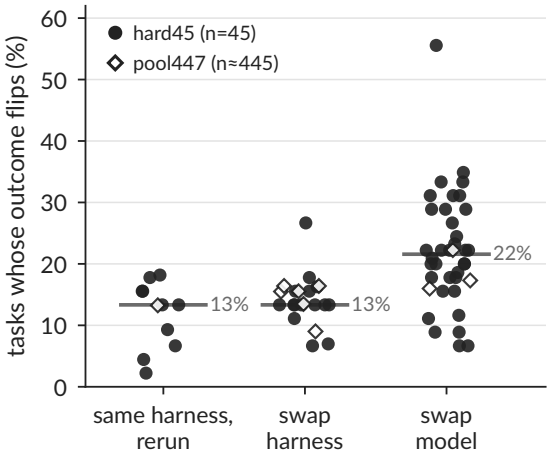


Figure 2: **Swapping the harness flips as many tasks as rerunning the same harness.** Each point is one pair of runs on the same tasks, and the value is the share of tasks whose outcome differs between the two. Rerun pairs share model, harness, and settings. Harness pairs share the model. Model pairs share the harness. Filled points are hard45, open points pool447; bars are medians.

The harness's flips do not replicate at the task level either. If a harness won a fixed set of tasks, a split in run 1 would repeat in run 2. On the six harness pairs on hard45 run twice on both sides, three on each Qwen model, the split repeats no more often than it reverses: 5 tasks split the same way and 7 the opposite way, the rest tie, and a coin gives that count under either test in Appendix F. A real effect leaves a different trace: for the model swap under the same harness, 10 of 15 such tasks split the same way both times, even though Qwen3.8-27B leads Qwen3.6-35B-A3B by only two to ten tasks on this set. On hard45 the harness gap behaves like a fresh draw of the same noise, with no set of tasks that one harness wins.

The flip rates set what a comparison of this kind can resolve. With 14% of tasks discordant between two arms, 45 tasks catch a 12.9-point gap half the time and no gap at 80% power, and 447 tasks detect 5.2 points at that power (Figure A2 in the appendix). So the two-to-five-task gaps in Table 2 cannot be told from zero. By the same scale, a reported gain of a few points needs a paired pool larger than ours, and one read off a single run of a few dozen tasks is inside the noise.

4.3 A harness can lose tasks

The one harness effect that clears the noise is OpenCode losing tasks, and the trajectories say how (Figure 3). The first way is an output cap with no recovery: when the model's reasoning runs past the provider's cap in the middle of a step, OpenCode does not re-prompt, and a trial cut on its final step ends with whatever is on disk, which is usually nothing. On pool447 with Qwen3.8-27B, 33 OpenCode trials fail this way, where Claude Code, which hits the same cap on the same model and re-prompts the model to continue, loses one. Scoring only the tasks where OpenCode was not cut off roughly halves its gap to the other two harnesses (Table A7), and transplanting Claude Code's recovery prompt into OpenCode recovers tasks at a rate the affected tasks cannot tell from zero (Appendix H). The rest of OpenCode's gap goes with a second habit: under OpenCode the model

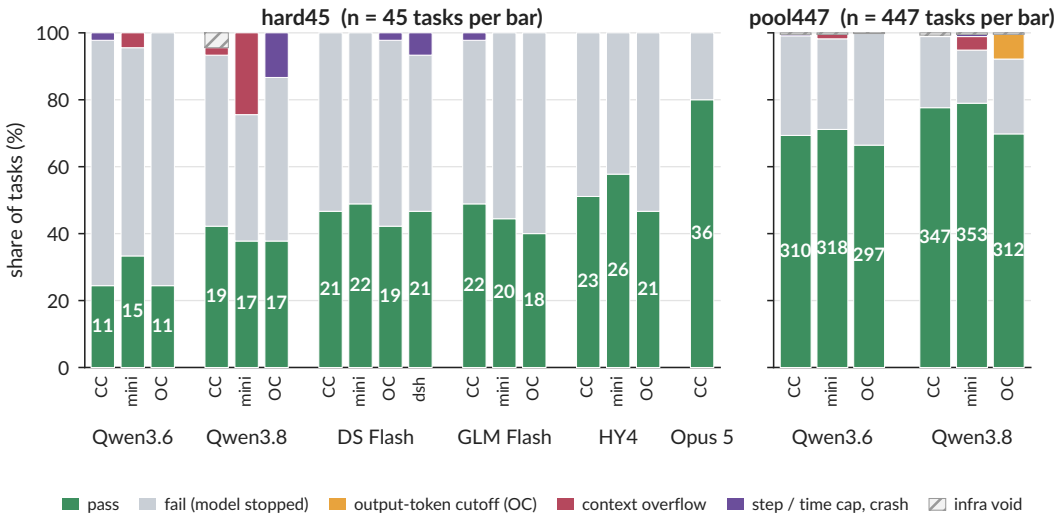


Figure 3: **How trials end.** One bar per model and harness, split into pass, fail with the model stopping on its own, output-token cutoff, context overflow, step or time cap or crash, and trials lost to infrastructure, which are not scored. dsh is DeepSeek's own harness.

stops sooner. On Qwen3.6-35B-A3B over the pool, where no trial is cut off, OpenCode ends after a median of 28 model calls against 71 under mini-SWE-agent and 55 under Claude Code. Its gap to mini-SWE-agent is still +4.7 points. OpenCode also stops soonest on GLM-5.3-Flash (Table A4). The data do not say what in OpenCode makes the model stop sooner, only that shorter runs and lost tasks travel together.

The other failure modes are also losses, and each traces to one design choice. mini-SWE-agent has no context management, so a long trial runs into the model's window, which on Qwen3.8-27B happens on 11 of the 45 hard tasks and 18 of the pool. Most of those overflows are symptoms: the other two harnesses mostly failed those tasks too, so a compaction step that recovered the ones another harness passed would move mini-SWE-agent by 6.7 points on the hard tasks and less on the pool, and on rerun most overflow again (Appendix K). An overflow mostly marks a task the model was already losing. The two full harnesses fail at their own limits instead: Claude Code and OpenCode compact the transcript and lose at most one trial to the window, and Claude Code loses at most two trials per arm at the step cap. OpenCode hits the wall clock on six of the 45 hard tasks on Qwen3.8-27B, where it had a sixth of Claude Code's time, and dropping those tasks does not shrink its gap there (Table A7). OpenCode as shipped also cannot run its search tools offline, and the shipped arm sits lower than the reported arm on one Qwen model and higher on the other (same table). None of these is a way to win a task. The pairs that clear the noise, the filled points in Figure 1b, all point the same way: toward a harness losing.

4.4 The harness sets the bill

The harness sets the token bill, by up to 3× on the same model (Figure A3 in Appendix G). A bill is mostly per-step input times steps, and the harness moves both terms. On GLM-5.3-Flash Claude Code and mini-SWE-agent take about as many steps, so their gap is per-step weight: a trial costs 1.81 CNY under Claude Code and 0.97 under mini-SWE-agent, because Claude Code feeds the model most input per step (Table A4). OpenCode carries less per step and also stops soonest, so it is cheapest on both counts, at 0.56 CNY. On DeepSeek-V4-Flash the step counts diverge instead: mini-SWE-agent takes 141 steps to Claude Code's 104, its transcript grows with every one of them, and the lightest harness ends up moving the most tokens. Per-step weight and step count trade against each other, and which one wins depends on the model.

The per-step weight is not what the tools return (Table A4 in the appendix). Per call, the three harnesses hand the model about the same amount of text, at the mean and at the 90th percentile. What differs is the preamble each harness carries into every call, its system prompt, tool schemas,

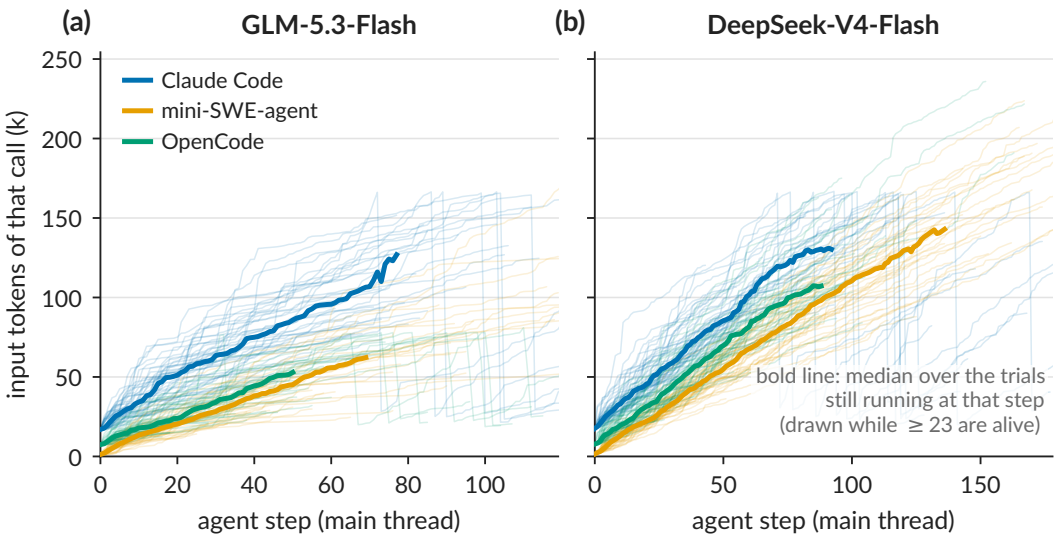


Figure 4: **What the model reads at each step.** Input tokens of every call against the step index, one faint line per trial and a bold median over the trials still running at that step, drawn while at least half of them are. The harnesses differ first in where the line starts, which is the preamble, and less in how fast it climbs. The cliffs are compactions.

and message wrapper: 16,581 tokens in Claude Code, 7,025 in OpenCode, and 829 in mini-SWE-agent, paid again on every step. Claude Code also keeps the model's own reasoning in the transcript, so a run accumulates many times the model text of a mini-SWE-agent run, and it takes the most steps. Adding these terms to the tool returns and call arguments approximately reconstructs the measured input per step in all three arms (Appendix G gives the gaps). What separates the harnesses' per-step weight is their own text, not the repository's.

Read step by step, the same accounting shows the harnesses separating at the first call (Figure 4). The input the model reads rises along a near-straight line in every arm, so a harness is described by where its line starts and how fast it climbs. The harnesses differ first in the start: on GLM-5.3-Flash the first call carries 17.1k tokens under Claude Code, 7.6k under OpenCode, and 1.4k under mini-SWE-agent, and the same three sizes on DeepSeek-V4-Flash. The slopes sit within a factor of one and a half of one another on both models, and the reasoning Claude Code keeps in the transcript accounts for part of its steeper climb (Appendix G). On DeepSeek-V4-Flash the three lines run nearly parallel, so Claude Code's per-step input ends up tied with mini-SWE-agent's and the bill is decided by the step count. Claude Code compacts in a fifth to a third of its trials, at the cliffs in the figure, and mini-SWE-agent never compacts and overflows instead. The first call, the growth per step, and the step count are the three numbers a harness's bill turns on.

The price of a cache hit scales the bill and does not reorder it. Each step re-sends the transcript of the last one, so 96 to 99% of input tokens in every arm on both vendor models are cache hits, and the hit price decides how much of the bill the transcript is. Zhipu charges $0.287\times$ the miss price for a hit, so cached input is most of the GLM bill and Claude Code's weight shows up as nearly twice mini-SWE-agent's cost. DeepSeek charges $0.033\times$, so the cached transcript is under half of the bill and output is as large a term: mini-SWE-agent re-reads more and Claude Code writes more, and the two come out the same. Repricing each model's tokens under the other vendor's list keeps the order of the three harnesses in both cases (Appendix G). What flips Claude Code from cheaper than mini-SWE-agent to twice its cost is the model's step profile, not the price list.

4.5 What we looked for and did not find

A vendor's own harness on its own model buys no score advantage. DeepSeek's harness on DeepSeek-V4-Flash passes 21 of 45, the same as Claude Code and one fewer than mini-SWE-agent,

at 1.04 CNY per trial against Claude Code's 0.98 (Appendix B). Home field is inside the range on score and at the top of it on bill.

Claude Code makes the transcript longer without making the model reason harder. On Qwen3.8-27B over the pool, reasoning is 82.5% of the model's output under Claude Code and 82.9% under OpenCode, the same share, while total output per trial is 1.23× under Claude Code. Its reasoning grows with the transcript, not faster.

The weakest model is not the most harness-sensitive. On Qwen3.6-35B-A3B the widest gap that does not involve OpenCode is Claude Code minus mini-SWE-agent, -8.9 points on hard45 ($p = .39$). The next widest is on Qwen3.8-27B, +7.0 points the other way. Both are inside the hard45 noise, the Qwen3.6 gap shrinks on rerun, and on the pool the two harnesses are equivalent within five points on both models.

Inside one harness, neither two tool ablations nor a release from a year earlier moves the score. Taking away Claude Code's file-edit tools, so that Claude Opus 5 edits through `bash`, and taking away the shell instead both pass within one task of the three control runs (Table A5). A release from a year earlier, 1.0.100, also passes within that band, at 0.44× the list-price cost, because it did not enable extended thinking by default. The year-old release moves the bill and not the score.

The one interaction we did find is with thinking. Turning it off on Qwen3.6-35B-A3B moves Claude Code and OpenCode by +2.2 and +3.4 points over two runs each, with about as many tasks going up as down, and costs mini-SWE-agent 11 points ($p = .04$; Table A6). This suggests that the dependence on thinking belongs to the model and harness pair, not to the model alone.

5 Discussion and Conclusion

What a harness buys, on this evidence, is fewer ways to lose a task, and a bill, not pass rate. Every harness effect we could measure is a way to lose a task the model would otherwise solve: an output cap with no recovery, a context window with no compaction, a search tool that does not run offline. We did not find a harness winning tasks, which puts the engineering where the losses are. The one change that might move a score in our grid is a recovery step for OpenCode's output cutoff, which Claude Code already has. Transplanted, it recovered tasks at a rate that 41 tasks cannot tell from zero, because the runaway reasoning behind the cut recurs on only about 15% of reruns of the same task. The three harnesses differ most in the parts that get the most attention, the prompt, the tool surface, the compaction policy, and none of those differences buys a score gain.

The thinking interaction is the one lead we would chase. Switching thinking off on Qwen3.6-35B-A3B costs mini-SWE-agent 11 points and moves the other two by less than the noise. On HY4-Preview, however, the same switch inside Claude Code costs 11 points, as much as the widest harness gap on that model, and cuts output tokens by more than half. The two results suggest that dependence on explicit reasoning is specific to the model and harness pair, but our grid is too small to identify the mechanism.

Two limits on scope. Our results come from one benchmark. The one-model check on Terminal-Bench 2.1 (Appendix L) is a transfer check, and general-purpose agents may be a different regime. And the closed anchor model runs in Claude Code only, so the three-harness ladder stops at HY4-Preview and cannot tell whether a closed frontier model shows the same gaps.

Swapping the harness under a fixed model moves pass rate on SWE-bench Verified by as much as rerunning the same harness on the 45 hardest tasks, across five models. On the 447-task pool, the pairs that clear the noise are all OpenCode trailing, by 8 to 9 points on Qwen3.8-27B, tied to how it ends a trial. What the harness does decide is the bill, by up to 3× on the same model, through what it carries into each step and how many steps the model takes, and the cache price scales that bill without reordering it. The numbers to carry between vendors are the first call, the growth per step, and the step count. At the discordance we observe, a harness comparison on 45 tasks catches a 12.9-point gap half the time and nothing at 80% power, and on 447 tasks it resolves 5.2 points. The cheapest control in such a comparison is a second run of the same cell, the one most comparisons skip, and we recommend reporting it next to any harness gap.

ETHICS STATEMENT

All tasks come from the public SWE-bench Verified benchmark, and no human subjects or personal data are involved. The upstream fixes that networked agents retrieved are public GitHub content; we report retrieval rates by model to inform evaluation practice and make no claim that any vendor trained on or intended to use the test data. API keys and credentials were removed from all released files.

REPRODUCIBILITY STATEMENT

Every number in the paper is generated from one table of trials by the scripts that draw the figures and tables, and Appendix A lists the harness versions, the settings each arm ran with, and how the offline condition was enforced. The trial table, the analysis scripts, and the Harbor job files, with credentials removed, are provided as supplementary material. The full trajectories of the mini-SWE-agent and OpenCode runs, and the tool-call sequences and results of the Claude Code runs, will be released with the paper.

AI USE STATEMENT

In this work, we used generative AI tools to implement standard methods, to assist with data analysis, and to draft and edit parts of the paper. The agent trajectories this paper measures are model output by design and are the object of study itself. We have reviewed all AI-assisted work: the analysis scripts were rerun by the authors against the trial table behind every table and figure, and all drafted and edited text was read and approved by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Marc Alier Forment, María José Casañ Guerrero, Francisco José García-Peñalvo, and Juanan Pereira. The scaffolding matters more than the interface: A controlled comparison of MCP and CLI tool use across seven agent scaffoldings, five language models, and one software task. *arXiv preprint arXiv:2608.08654*, 2026.

Anomaly Innovations. OpenCode. `https://github.com/anomalyco/opencode`, 2026. Version 1.18.30.

Anthropic. Claude code. `https://github.com/anthropics/claude-code`, 2026. Version 2.1.263.

Oussama Ben Sghaier, Hao Li, Bram Adams, and Ahmed E. Hassan. Don't blame the large language model: How agent harness evolution shapes coding agent quality. *arXiv preprint arXiv:2607.03691*, 2026.

Bjarni Haukur Bjarnason, André Silva, and Martin Monperrus. On randomness in agentic evals. *arXiv preprint arXiv:2602.07150*, 2026. ICLR 2026 Workshop on Agents in the Wild.

Run-Ze Fan, Zihao Zhang, Simin Ma, Yebowen Hu, Shouju Wang, Kaiqiang Song, Fei Liu, Hamed Zamani, and Xiaoyang Wang. An empirical study of harness design for coding agents. *arXiv preprint arXiv:2609.20804*, 2026.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. `https://doi.org/10.5281/zenodo.20953922`, 2026. Version 0.22.0; source at `https://github.com/laude-institute/harbor`.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Transactions on Machine Learning Research*, 2025.

Sydney Lewis. Same model, different harness: Different coding-agent results. *arXiv preprint arXiv:2608.26218*, 2026.

Kilian Lieret, John Yang, and SWE-agent contributors. mini-SWE-agent. `https://github.com/SWE-agent/mini-swe-agent`, 2026. Version 2.4.6.

Hasan Najib Mahmud, Shreya Gupta, Isha Chaudhary, Nathaniel Enis, et al. A jagged frontier: Evaluating robustness of code agents to semantics-preserving transformations. *arXiv preprint arXiv:2608.18389*, 2026.

Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations. *arXiv preprint arXiv:2411.00640*, 2024.

OpenAI. Introducing SWE-bench Verified. `https://openai.com/index/introducing-swe-bench-verified/`, 2024.

OpenRouter. AI model rankings. `https://openrouter.ai/rankings`, 2026. Trailing 30 days by tokens processed, usage data through 2026-09-20; archived 2026-09-21 at `https://web.archive.org/web/20260921030005/https://openrouter.ai/rankings`.

Benjamin Rombaut. Inside the scaffold: A source-code taxonomy of coding agent architectures. *arXiv preprint arXiv:2604.03515*, 2026.

Vasundra Srinivasan. Deployment decision reliability: A generalizability-theory framework for sizing long-horizon agent evaluations. *arXiv preprint arXiv:2608.11323*, 2026.

Jason Starace. Scaffold effects on GAIA: A controlled comparison. *arXiv preprint arXiv:2606.08529*, 2026.

Kishanthan Thangarajah, Boyuan Chen, and Ahmed E. Hassan. DCAS: Decoupling CLI agent scaffolding to internalize planning across scaffolds. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2026.

Naman Vats and Oleg Golev. The scaffold effect in coding agents: Harness choice as a hidden variable in coding-agent evaluation. *arXiv preprint arXiv:2607.22585*, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. In *ACM International Conference on the Foundations of Software Engineering (FSE)*, 2025.

Hong Yang, Qi Yu, and Travis Desell. When does restricting a coding agent to `execute_code` help? a regime $\times$ agent-design ablation. *arXiv preprint arXiv:2607.10569*, 2026.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, et al. Harness-bench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026.

Yunbei Zhang, Janet Wang, Jihun Hamm, Yingqiang Ge, Weijie Xu, and Chandan K. Reddy. Stop comparing LLM agents without disclosing the harness. *arXiv preprint arXiv:2605.23950*, 2026.

Mengyu Zheng, Kai Han, et al. Claw-SWE-bench: A benchmark for evaluating OpenClaw-style agent harnesses on coding tasks. *arXiv preprint arXiv:2606.12344*, 2026.

Zhipu AI. ZCode: the official harness for GLM-5.3. `https://zhuanlan.zhihu.com/p/2070504528416337941`, 2026. Vendor post, 11 Aug 2026.

APPENDIX

A WHAT WAS HELD FIXED

Table A1 lists the configuration of each harness as run. Three settings were aligned across the harnesses on purpose; the rest were left as shipped.

Table A1: **What was held fixed.** Verified from the harness configurations, the Harbor job files, and the proxy logs.

	Claude Code	mini-SWE-agent	OpenCode
Version (pinned per arm)[a]	2.1.259–2.1.273 (2.1.263 on GLM and Opus)	2.4.6	1.18.30–1.18.31 (1.18.30 on GLM)
Sampling	none sent (no flag)	none sent	none sent
Thinking / effort	GLM: effort `high` (CLI flag); Qwen: effort `xhigh` via env (vLLM compat., a no-op)[b]; DS: provider default	GLM: `reasoning_effort` = `high` (request body); Qwen/DS: default	GLM: `reasoningEffort` = `high` (model config); Qwen/DS: default
Tools	built-in set (Bash, Read, Edit, Write, Grep…); `WebSearch` disallowed	`bash` only	bash, read, edit, write, glob, grep, webfetch, task; ripgrep pre-placed (rg arms)
Context management	auto-compaction: 110k window (env override) on the Qwen arms, client default elsewhere[c]	none; per-observation head/tail trim (10k chars); hard stop at 131k	prune (40k tool-output tail) + auto-compaction at `limit.input` minus a 20k reserve (78k); input limit 98k, output limit 32k
Termination	`--max-turns 300`; soft timeout (exit 124)	`step_limit=300`; no soft timeout	`build.steps=300`; soft timeout (exit 124)
Wall clock (safety net)[d]	Qwen arms: 12× the task's agent timeout; vendor-API arms: the batch's single soft timeout	Qwen arms: 6× the task's agent timeout; vendor-API arms: the same soft timeout	Qwen arms: 6× the task's agent timeout; vendor-API arms: the same soft timeout
Network (agent phase)	allowlist: model API host + 172.17.0.1	same	same
Runner	Harbor 0.22.0, one container per trial, verifier after the agent exits	same	same

[a] the versions of the arms scored in Tables 2 and 1, read off every scored trial; OpenCode 1.18.25–1.18.29 appear only in the as-shipped OpenCode arm of Table A7 and in the ablations. [b] a compatibility setting, not a treatment: `xhigh` is the Qwen chat template's own default, and vLLM's Anthropic-compatible `/v1/messages` endpoint ignores `effort` altogether (verified by a single-prompt probe against the same serving stack); Claude Code's reasoning per step is within 3–6% of OpenCode's on the same model, and OpenCode sends no effort at all, so all three harnesses see the same thinking budget on the Qwen arms. [c] `CLAUDE_CODE_AUTO_COMPACT_WINDOW`=110000 is set on the Qwen arms only, where it fires at a measured median of 78k tokens and so matches OpenCode by construction; on the vendor-API arms Claude Code keeps its own default window and compacts much later (on GLM-5.3-Flash at a median of 168k prompt tokens, Table A4). [d] on the locally served Qwen arms the limit is a multiple of the task's agent timeout, six times for mini-SWE-agent and OpenCode and twelve for Claude Code; on the vendor-API arms (DeepSeek-V4-Flash, GLM-5.3-Flash, Opus 5) one soft timeout applies to all three harnesses, 2,760 s on DeepSeek and Opus and 5,760 s on GLM.

Budget. Every trial has a cap of 300 steps, set through each harness's own knob: `--max-turns` in Claude Code, `step_limit` in mini-SWE-agent, `build.steps` in OpenCode. Wall-clock limits are kept only as a safety net. On the locally served Qwen arms Harbor's multiplier on the task's 3,000 s agent timeout is six for mini-SWE-agent and OpenCode and twelve for Claude Code, whose prefix-cache pattern against vLLM makes each step about twice as slow; OpenCode's run command also carries its own soft timeout of 5,760 s, and that is the limit that binds. On the vendor-API arms one soft timeout applies to every harness, 2,760 s on DeepSeek and Opus and 5,760 s on GLM. The asymmetry costs OpenCode six trials on Qwen3.8-27B × hard45: nine of its trials hit the soft timeout at 5,760 s, where Claude Code could run to 36,000 s, and six of them failed. All six

were still making tool calls with a patch on disk, none was near the step cap, and we report this as an uncontrolled factor rather than a harness effect. A trial that reaches either limit is scored as a failure.

Context window. The two Qwen models are served with a 131k-token window. OpenCode is configured with an input limit of 98k tokens and an output limit of 32k, so the two together fit the window; it compacts when the conversation exceeds the input limit minus a 20k reserve, at about 78k tokens. On the Qwen arms Claude Code's auto-compaction window is set to 110k, which fires at a measured median of 78k tokens, matching OpenCode by construction; on the vendor-API arms Claude Code runs with its default window and compacts much later, on GLM-5.3-Flash at a median of 168k prompt tokens. mini-SWE-agent has no compaction and runs into the model's window; that outcome is reported as its own column.

Network. The agent phase of every reported run has no network beyond the model endpoint and the host's package mirror. On the servers this is Harbor's allowlist mode, which puts the trial container behind an egress sidecar. The Opus runs, which must run on a local machine with the subscription channel, use an internal Docker network and a phase-aware proxy that admits only the model API during the agent phase and any host during verification; every denial is logged, and a content-level audit found no scored trial in any offline arm that retrieved GitHub content. Two harness-specific consequences of the offline condition are handled explicitly. OpenCode downloads ripgrep at first use, so its `grep` and `glob` tools fail offline; the reported arm pre-places the binary, and the shipped arm is kept for Table A7. mini-SWE-agent calls the model without streaming, and the egress sidecar drops plain-HTTP connections that do not answer within 15 seconds, so its local-model arms go through a TLS forwarder on the host; this changes transport, not sampling.

Other fixed conditions. OpenCode selects its system prompt by model-name pattern, and every model in this study falls into its default prompt. Claude Code runs with a fresh configuration directory per trial and with web search disallowed; against non-Anthropic endpoints its `Grep` tool is unavailable, and those arms search through `Bash`. Thinking is on by default in every reported arm, including the three HY4-Preview arms, which are called through the vendor's API. The Qwen3.6-35B-A3B thinking-off arms of Appendix J switch it off at the serving layer, so the switch is the same for all three harnesses; the HY4-Preview thinking-off arm switches it off through the API's thinking field, which the endpoint treats as on when it is missing, and it runs in Claude Code only. For the HY4-Preview Claude Code arms the reverse proxy folds Claude Code's mid-conversation system messages into the preceding user message and sets the thinking field explicitly, which aligns the vendor endpoint with the defaults of the Anthropic API; the content and the order of the conversation are unchanged. Claude Code sends a reasoning-effort setting of `xhigh` to every endpoint; vLLM's Anthropic-compatible endpoint ignores it, and on the Qwen arms the reasoning per step under Claude Code is within a few percent of OpenCode's, which sends none.

B DeepSeek's own harness

The home-field arm runs DeepSeek's command-line harness, release 0.1.2-rc.1, on DeepSeek-V4-Flash over hard45, driven by Harbor like the other three and under the same soft timeout. All 45 trials were scored, none was lost to infrastructure, and 21 passed, the same count as Claude Code on the same model. Six trials ended with a non-zero exit from the harness, three of them after a passing patch. The harness takes 111.2 model calls per trial, between OpenCode's 88.2 and mini-SWE-agent's 141.1 on this model, and costs 1.04 CNY per trial at DeepSeek's list price, two fen above mini-SWE-agent, the most expensive of the three general harnesses on this model. A vendor's own harness lands inside the range of the three general ones on score and at the top of it on bill.

C Task pools and coverage

Table A2 traces the denominators from the 500 tasks of SWE-bench Verified to the two task sets, and counts, for the canonical run of every cell, the trials that were lost to infrastructure and the trials that ended at a limit. Table A3 lists every cell of the grid with its runs, its usable count, and the jobs it came from. The eight dropped tasks are the `psf/requests` instances, whose verifier calls a public web service. One further task, `django-11133`, served as the smoke test for every arm and

is filed under the hard45 label in the job tree while belonging to pool447; it has no scored hard45 row, so the hard45 universe is 45 tasks and disjoint from pool447. Two scikit-learn tasks time out in the verifier in most arms, which is why pool447 cells have 444 to 446 scored tasks. Eight tasks in three cells have two usable trials for the same run; the latest-finishing one is taken, and taking the earliest instead changes no count.

Table A2: **Denominator ledger.** (a) From SWE-bench Verified to the two task sets. (b) For the canonical run of every cell, the trials lost to infrastructure (void, removed from the denominator) and the trials that ended at a limit (kept and scored as failures).

(a) Denominator ledger

SWE-bench Verified: **500** tasks

− 8 `psf/requests` instances (verifier depends on `httpbin.org`, hangs offline) → **pool492** = 492

→ **hard45** = 45 (whole difficulty bands 1–4 h and >4 h: 42 + 3; zero sampling degrees of freedom) + **pool447** = 447 (the complementary remainder, disjoint)

`django__django-11133` is filed under the hard45 label as the universal smoke task but belongs to pool447; it has no usable hard45 row, so the scored hard45 universe is 45.

(b) What the denominator loses, canonical run-1 trials

Subset	Model	Arm	n	ok	void	overfl.	t/out	step	out-cut	other
hard45	Qwen3.6	CC	45	45	0	0	0	1	0	0
		mini	45	45	0	2	0	0	0	0
		OC-rg	45	45	0	0	0	0	0	0
	Qwen3.8	CC	45	43	2	1	0	0	0	0
		mini	45	45	0	11	0	0	0	0
		OC-rg	45	45	0	0	6	0	0	0
	DS Flash	CC	45	45	0	0	0	0	0	0
		mini	45	45	0	0	0	0	0	0
		OC-rg	45	45	0	0	0	0	0	1
	GLM Flash	CC	45	45	0	0	1	0	0	0
		mini	45	45	0	0	0	0	0	0
		OC-rg	45	45	0	0	0	0	0	0
	HY4	CC	45	45	0	0	0	0	0	0
		mini	45	45	0	0	0	0	0	0
		OC-rg	45	45	0	0	0	0	0	0
	Opus 5	CC	45	45	0	0	0	0	0	0
pool447	Qwen3.6	CC	447	445	2	0	0	2	0	0
		mini	447	445	2	6	0	0	0	0
		OC-rg	447	446	1	0	0	0	0	0
	Qwen3.8	CC	447	444	3	1	0	0	1	0
		mini	447	445	2	18	1	0	0	2
		OC-rg	447	445	2	0	0	0	33	0

Classes follow `analysis/paper_numbers.py:outcome()` on the canonical run-1 trial of every task. *void* = infra exception (not scored as an outcome, removed from the denominator); *overflow* = `ContextWindowExceeded` or the extractor's context-overflow flag; *timeout* = `SoftTimeout` / `AgentTimeoutError`; *step-cap* = `StepCapReached`; *out-cut* = the run ended on the 32 k output-token ceiling (OpenCode: a terminal `step_finish` with `reason=length`, from `analysis/cutoff_audit.txt`; Claude Code: a fatal `OutputTokenExceededError`). Overflow, timeout, step-cap and out-cut trials are *kept* and scored as failures in every main table; column (b) says how many they are. *other* = `NonZeroAgentExitCodeError`, the remaining member of the cap set `paper_numbers.CAP_EXC`. The wall-clock limit is not symmetric on the locally served Qwen arms: the six OpenCode timeouts on Qwen3.8-27B × hard45 are a 5,760 s limit against Claude Code's 36,000 s on the same model, and all six were still making tool calls, with a patch already on disk, when they were stopped (none near the 300-step cap).

Table A3: **Coverage matrix.** Every cell of the grid, with the runs, the scored task count, the passes of each run, and the Harbor jobs it is assembled from.

Model	Subset	Arm	Run	n	ok	pass	void	Status	Jobs
Qwen3.6	hard45	CC	1	45	45	11	0	complete	`cc-qwen36-hard45-nw`
		CC	2	45	45	10	0	complete	`cc-qwen36-hard45-nw-k2`
		mini	1	45	45	15	0	complete	`mini-qwen36-hard45-nw` (+ `-r`, `-r2`)
		mini	2	45	45	13	0	complete	`mini-qwen36-hard45-nw-k2`
		OC-rg	1	45	45	11	0	complete	`oc-qwen36-hard45-nw-rg` (+ `-r`)
		OC-rg	2	45	45	11	0	complete	`oc-qwen36-hard45-nw-rg-k2`
	pool447	CC	1	447	445	310	2	partial (445/447)	`cc-qwen36-pool447-nw` (+ `-r`, `-r2`)
		mini	1	447	445	318	2	partial (445/447)	`mini-qwen36-pool447-nw-c` (+ `mini-qwen36-pool447-nw`, `mini-qwen36-pool447-nw-r`, `mini-qwen36-pool447-nw-r2`)
		mini	2	447	445	313	2	partial (445/447)	`mini-qwen36-pool447-nw-k2-c2` (+ `mini-qwen36-pool447-nw-k2`, `mini-qwen36-pool447-nw-k2-c`)
		OC-rg	1	447	446	297	1	partial (446/447)	`oc-qwen36-pool447-nw-rg` (+ `-r`, `-r2`, `-r3`)
Qwen3.8	hard45	CC	1	45	43	19	2	partial (43/45)	`cc-qwen-hard45-nw` (+ `-r`)
		CC	2	45	45	20	0	complete	`cc-qwen-hard45-nw` (+ `-r`)
		mini	1	45	45	17	0	complete	`mini-qwen-hard45-nw` (+ `-r`)
		mini	2	45	45	18	0	complete	`mini-qwen-hard45-nw`
		OC-rg	1	45	45	17	0	complete	`oc-qwen-hard45-nw-rg`
		OC-rg	2	45	44	14	1	partial (44/45)	`oc-qwen-hard45-nw-rg`
	pool447	CC	1	447	444	347	3	partial (444/447)	`cc-qwen-pool447-nw`
		mini	1	447	445	353	2	partial (445/447)	`mini-qwen-pool447-nw` (+ `-r`, `-r2`, `-r3`)
		OC-rg	1	447	445	312	2	partial (445/447)	`oc-qwen-pool447-nw-rg`
DS Flash	hard45	CC	1	45	45	21	0	complete	`cc-ds-hard45-nw` (+ `-r`)
		CC	2	45	45	23	0	complete	`cc-ds-hard45-nw-k2` (+ `-r`)
		mini	1	45	45	22	0	complete	`mini-ds-hard45-nw` (+ `-r`)
		OC-rg	1	45	45	19	0	complete	`oc-ds-hard45-nw-rg` (+ `-r`, `-r2`)
GLM Flash	hard45	CC	1	45	45	22	0	complete	`cc-glm53f-hard45-nw`
		mini	1	45	45	20	0	complete	`mini-glm53f-hard45b-nw`
		OC-rg	1	45	45	18	0	complete	`oc-glm53f-hard45-nw-rg` (+ `oc-glm53f-r1-nw-rg`)
HY4	hard45	CC	1	45	45	23	0	complete	`cc-hy4-hard45-nw-r`
		mini	1	45	45	26	0	complete	`mini-hy4-hard45-nw`
		OC-rg	1	45	45	21	0	complete	`oc-hy4-hard45-nw-rg` (+ `-r`)
Opus 5	hard45	CC	1	45	45	36	0	complete	`cc-opus-hard45-nw` (+ `-r`, `-r2`, `-r3`)
		CC	2	45	45	35	0	complete	`cc-opus-hard45-nw-k2` (+ `-r`, `-r2`)
		CC	3	45	45	35	0	complete	`cc-opus-hard45-nw-k3` (+ `-r`)

Arms: *CC* = Claude Code, *mini* = mini-SWE-agent, *OC* = OpenCode as shipped, *OC-rg* = OpenCode with ripgrep pre-placed (the canonical OpenCode arm wherever it exists). Not-excluded jobs only; smoke and crashed batches carry `exclude=1` in `analysis/extract_trials.py` and never enter a cell. *Run*: the job-level repeat, except in the Qwen3.8 hard45 and Qwen3.6/3.8 k=2 jobs, where the within-job attempt index is the run index, so one job appears under both runs. *void* = scored trial lost to one of the 14 infra exception types (`main_table.INFRA`); the task is then absent from the paired analyses. Arms outside the main grid — `nothink`, `midsys`, `c2-*`, `cc-v1.0.100`, the `dsh` harness and every web-enabled arm — are excluded from this matrix and reported in their own appendix sections.

D LEAKAGE AUDIT OF THE WEB-ENABLED RUNS

Before the offline condition was enforced, the DeepSeek and Qwen arms ran with network access. An audit of those runs scanned tool-call arguments for retrieval of the upstream fix: a GitHub search hit, a commit patch, a pull-request diff, or the post-fix file from the raw content host. 312 of 439 trials had fetched the fix, 152 of them citing the task's own issue or pull-request number, and 310 of the 312 went on to edit a file after the lookup. The Claude models did this in 4 of 445 trials. Pass rate followed: 96% of the scored DeepSeek and Qwen trials that retrieved the fix passed, against 71% of those that never touched the network. All web-enabled pass-rate results were discarded, and the condition was moved from the prompt to the network layer.

E PER-TASK OUTCOMES

Figure A1 shows every hard45 task under every model and harness. Rows are sorted by how many arms solved the task.

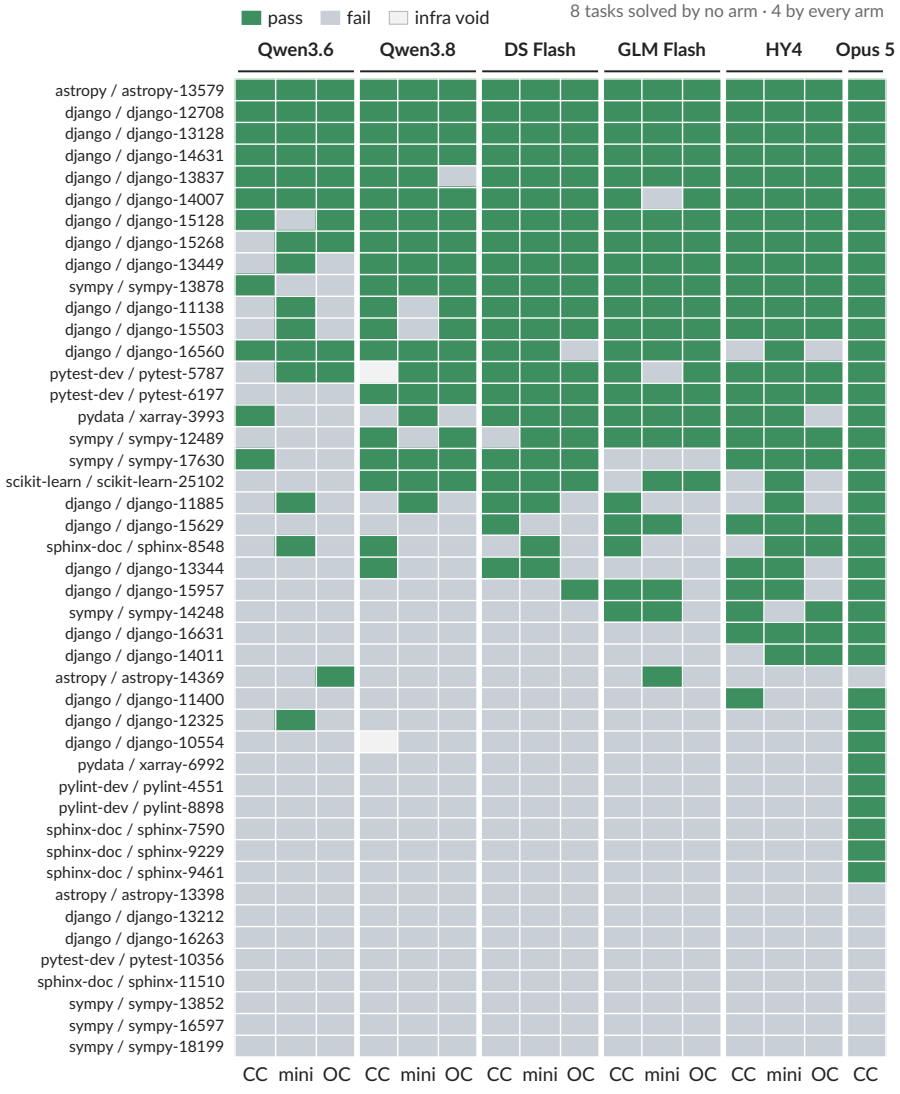


Figure A1: **hard45, task by task.** Canonical run of every cell. Eight tasks are solved by no arm and four by every arm.

F RESOLUTION OF A PAIRED COMPARISON

The flip rate of a pair of runs is the share of tasks whose outcome differs between them, $(b+c)/n$ in the notation of Table 1. Across the ten rerun pairs on hard45 the median flip rate is 13%, and 16% over the 7 pairs on the five ladder models, since Claude Opus 5's three reruns are the quietest; across the 15 harness pairs it is 13%, and across the model pairs 22% at the median and up to 56%. On pool447 the one rerun pair flips 13% of tasks and the six harness pairs flip 9 to 16%. The replication test of Section 4.2 starts from the 41 run-1 splits of the six twice-run harness pairs, on 16 distinct tasks; among the 12 tasks split in both runs, 5 same and 7 opposite give a binomial p = .77, and p = .821 under a permutation that flips the sign of a whole task at once. The smallest paired difference the exact McNemar test detects at 80% power depends on n and on the discordance rate; Figure A2 plots it at the 14% we observe pooled over the harness pairs, with a band for the range across pairs. Power is unconditional: the number of discordant tasks is binomial in n and the discordance rate, and given that number the two-sided exact test at α = .05 rejects or not. Fewer than six discordant tasks split one way cannot reject at all, so below 55 tasks no difference reaches 80% power at this discordance; at 45 tasks the ceiling is 62%, reached when every discordant task points the same way. The asymptotic formula that inverts the usual sample-size expression is drawn dashed for reference and is optimistic by half a point to a point everywhere. The equivalence verdicts on pool447 rest on the 90% interval of the Claude Code minus mini-SWE-agent difference, [-4.7, +1.1] on Qwen3.6-35B-A3B and [-3.7, +1.0] on Qwen3.8-27B, both inside $\pm$5.

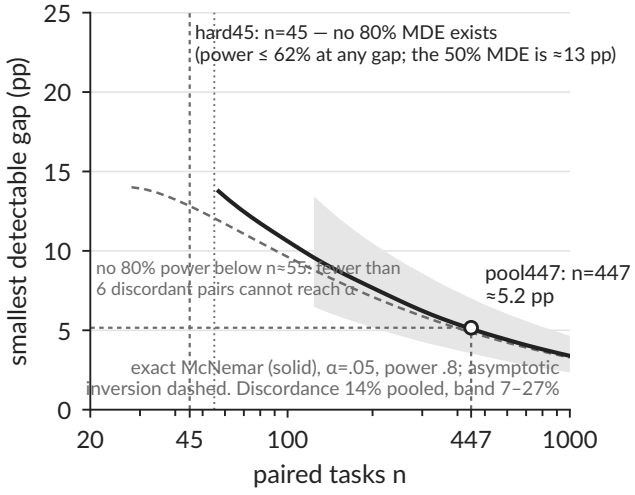


Figure A2: **Smallest detectable paired difference against sample size.** Exact McNemar test at α = .05 and 80% power, at the pooled discordance rate (solid line) and across the range observed between harness pairs (band); the asymptotic formula is dashed. At 45 tasks no difference reaches 80% power.

G WHERE THE PER-STEP INPUT GOES

Table A4 decomposes the input each harness sends per step, on the one model where all three harnesses ran the same 45 tasks with the same settings and complete trajectories, GLM-5.3-Flash. Three notes on how it was measured. The fixed cost per call is the prompt size of the first call, because the system prompts and tool schemas of Claude Code and OpenCode are not in the trajectories; the harness preamble is that size minus the issue statement, which is the same text in every harness, with mini-SWE-agent's instance template counted as preamble. Claude Code's tool returns are taken from its raw stream log rather than from the converted trajectory, because the converter appends the SDK's tool-result metadata to every observation and about doubles its size. The reconstruction uses each harness's own empirical characters-per-token ratio, and for Claude Code it reproduces the measured input only if the model's reasoning blocks are counted as re-sent history; that term is inferred from the fit, not observed, since the provider's usage does not separate reasoning from text. The reconstructed input per step lands within +6%, $-$13%, and $-$6% of the measured input in the three

arms. The per-step growth behind Figure 4 is the median over trials of the per-trial slope of input tokens against step index: on GLM-5.3-Flash 937 tokens per step under Claude Code against 648 under mini-SWE-agent and 681 under OpenCode, and on DeepSeek-V4-Flash 1,308 against 1,037 and 1,116; the reasoning Claude Code keeps in the transcript is 82 and 187 tokens of that per step on the two models. Repricing each model's tokens under the other vendor's list keeps the order of the harnesses: on the GLM tokens Claude Code costs 1.86× mini-SWE-agent at Zhipu's prices and 2.09× at DeepSeek's, and on the DeepSeek tokens 0.95× and 0.79×; the DeepSeek-V4-Flash trial costs 0.98 CNY under Claude Code and 1.02 under mini-SWE-agent at DeepSeek's list price.

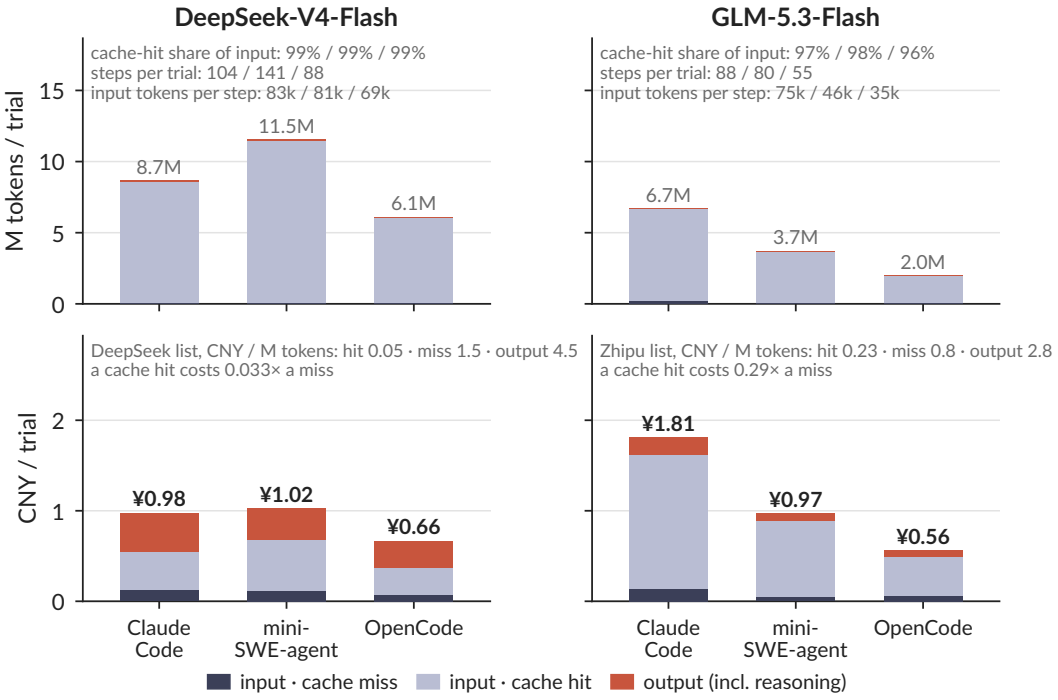


Figure A3: **Where the bill comes from.** Top: tokens per trial on DeepSeek-V4-Flash and GLM-5.3-Flash, split into cache-miss input, cache-hit input, and output including reasoning. Bottom: the same tokens priced at the vendor's list price. Annotations give the cache-hit share, steps per trial, and input tokens per step for the three harnesses.

Table A4: **Where the per-step input goes.** GLM-5.3-Flash on hard45, 45 runs per harness. (a) Size of a single tool return. (b) The per-step input budget, reconstructed from the trajectories and compared with the provider's measured usage.

(a) Tool-return size per call (chars)

Harness	calls	mean	median	p90	Largest sources
Claude Code	4,403	1,237	392	3,095	`Bash` 68%, `Read` 30%
mini-SWE-agent	3,787	1,186	461	3,121	`bash` 100%
OpenCode	2,633	1,103	311	3,049	`read` 49%, `bash` 39%, `grep` 9%

(b) Per-step input budget (tokens, hard45 means over 45 runs)

	Claude Code	mini-SWE-agent	OpenCode
Fixed cost per call[a]	17,134	1,382	7,579
of which the issue statement	554	554	554
of which harness preamble	16,581	829	7,025
Model text + reasoning per run[b]	58,359	1,715	21,900
Tool returns (observations) per run[b,c]	31,557	44,871	16,781
Tool-call arguments per run[b]	11,719	14,554	5,130
chars per token (empirical)[b]	3.84	2.22	3.85
Steps per run	88.1	80.2	55.1
Runs with an auto-compaction (of 45)[d]	8	0	0
Reconstructed input per step	80,060	39,933	33,068
Measured input per step[e]	**75,286**	**45,897**	**35,039**

GLM-5.3-Flash on hard45, 45 runs per harness (`cc-glm53f-hard45-nw`, `mini-glm53f-hard45b-nw`, `oc-glm53f-hard45-nw-rg`); the model, its settings and the task set are identical across the three columns, so every difference here belongs to the harness. [a] the first call's prompt tokens, i.e. what every later call also pays before any transcript: the harness's system prompt and tool schemas, its own user-message wrapper, and the issue statement. The split subtracts the issue statement at 3.7 chars/token, measured on the first user turn, which OpenCode and Claude Code receive verbatim and identically task by task (2,049 chars = 554 tokens on average); mini-SWE-agent's user turn additionally carries its instance template (a constant 2,860 chars), which is counted as harness preamble. The harnesses' own system prompts are in neither the trajectory nor the agent log, so the preamble is a residual. [b] characters appended to the conversation over one run, converted at the arm's own empirical rate (fitted on step-to-step Δprompt_tokens vs. Δchars): Claude Code and OpenCode carry prose at $\approx$3.8 chars/token, mini-SWE-agent's raw shell output is token-dense at 2.2. Everything a step appends is re-sent with every later request; the fit only closes for Claude Code if its reasoning blocks are re-sent too, which we infer from the fit rather than observe directly. [c] Claude Code's observation sizes are read from the raw stream log (`agent/claude-code.txt`), because the trajectory converter appends the SDK's `tool_use_result` metadata to every observation and roughly doubles it; that blob was never sent to the model. mini-SWE-agent and OpenCode trajectories match their raw logs and are used as stored. [d] auto-compaction boundaries recorded by the Claude Code SDK; mini-SWE-agent and OpenCode write no such record and never reset in these runs, so Claude Code's reconstruction is an upper bound and the other two are exact. On this arm Claude Code runs with its own default auto-compaction window — the 110k override of Table A1 is set on the Qwen arms only — and compacts at a median of 168k prompt tokens, where OpenCode compacts at `limit.input` minus its 20k reserve, i.e. at 78k, which is why the compaction counts differ. [e] input tokens reported by the provider, averaged over all steps of all 45 runs.

H THE OUTPUT CUTOFF AND A RECOVERY TRANSPLANT

OpenCode's provider configuration caps a single response at 32k output tokens, and the trajectories show the cut landing at exactly that size in the middle of a reasoning block. On pool447 with Qwen3.8-27B, 38 of 447 OpenCode trials contain such a cut and in 34 of them it is the final step, all but one of which fail. Those trials pass at 7.9% against 75.9% for the rest, and 32 of the 34 end with the repository untouched, after a median of 11 steps. Claude Code on the same model and pool is cut in 52 of 447 trials, 66 events in all; 64 of them are followed by the harness's own resume prompt and the session continues, and one trial ends on the cap. Its trials with a cut pass at 69% against 79% without. mini-SWE-agent never reaches the cap; its largest single step is 13k output tokens.

To test whether the recovery prompt is the difference, we patched Harbor's OpenCode wrapper to re-invoke the session with Claude Code's resume string, verbatim, whenever the last step was cut or the repository was untouched, and reran the 41 pool447 tasks whose parent trial had failed that way under both the stock and the patched wrapper, two attempts each, interleaved on one host. Mean per-task reward is 0.585 stock and 0.634 with recovery; six tasks improve, two worsen, 33 tie (p = .29, exact sign test). The prompt fired on 15% of the patched trials, against the 33 of 41 parent trials that had been cut, so the failure belongs to the run rather than to the task. Recovery cost 1.23× the steps and 1.30× the input tokens.

I CLAUDE CODE SETTINGS ON CLAUDE OPUS 5

Claude Code with Claude Opus 5 was run on hard45 in four settings (Table A5): three control runs of the pinned release; the model restricted to `bash`, twice; the model with the shell removed; and release 1.0.100, a year older than the pinned one. Every setting passes between 34 and 37 of 45, within one task of the control runs' 35 to 36. The task-level flips against the three control runs are 16/135 and 7/135 for `bash` only, 4/135 without a shell, and 12/135 for the old release, against 6/135 between the control runs themselves. The old release moves the bill and nothing else: it does not enable extended thinking by default, its median output per trial is 0.24× that of the controls, and its cost per trial 0.44×.

Table A5: **Claude Code on Claude Opus 5, four settings, hard45.** Flips are counted against the three control runs pooled. Cost is what Claude Code itself reports, at Anthropic's list price.

Setting	CC version	Solved / 45 k	95% CI (%)	flips /135	McNemar p vs. s1	output tok.	reas. tok.	steps	USD/trial (mean)
Control, run 1	2.1.263	36	[66.2, 89.1]	—	—	15,984	4,916	34	1.83
Control, run 2	2.1.263	35	[63.7, 87.5]	—	1.00	12,777	5,927	32	1.51
Control, run 3	2.1.263	35	[63.7, 87.5]	—	1.00	14,930	5,345	39	1.60
Bash-only, run 1	2.1.263	36	[66.2, 89.1]	16/135	1.00	13,539	5,378	31	1.42
Bash-only, run 2	2.1.263	37	[68.7, 90.7]	7/135	1.00	12,250	5,853	28	1.63
No shell	2.1.263	34	[61.3, 85.8]	4/135	0.50	16,734	10,031	26	1.86
Claude Code 1.0.100	1.0.100	34	[61.3, 85.8]	12/135	0.62	3,351	0	26	0.73

Claude Opus 5 through Claude Code on hard45, 45 tasks per row, no web access; every row is one run, and all 45 tasks are usable in every row. *Control* is the arm of the main tables (three independent runs). *Bash-only* removes the file-editing tools with `--disallowedTools(Edit,Write,MultiEdit,NotebookEdit, WebSearch)`, so the model can only edit through the shell; it was run twice. *No shell* removes execution instead (`--disallowedTools WebSearch,Bash`), leaving the editing tools. *Claude Code 1.0.100* is the npm release of 2025-09-01, one year older than the 2.1.263 the other six rows run, with the job YAML otherwise identical. *Solved* is tasks with reward 1 out of the usable trials, with a Wilson 95% interval. *flips* counts tasks whose outcome differs from a control run, pooled over all three runs (3×45); the within-control floor — the same count between the three control runs themselves — is 6/135 = 4.4%, so no setting here separates itself from rerun noise by more than about a factor of two. *McNemar* is the two-sided exact binomial on the discordant pairs against control run 1. *output tok.* and *reas. tok.* are per-trial medians of the provider's output-token and reasoning-token counts; 1.0.100 predates extended thinking being on by default, which is where its 4× smaller output and its zero reasoning come from. *USD/trial* is the mean of the `cost_usd` Claude Code reports for itself, i.e. Anthropic list price for the tokens used; these runs were served by a Claude Code subscription and were not billed per token, so the column is a token-weighted effort index, not spend.

J THINKING ON AND OFF

Thinking was switched off for Qwen3.6-35B-A3B at the serving layer, with a chat template that closes the thinking block at once and a ban on the opening token, so the switch is the same for all three harnesses. Each harness ran hard45 twice with thinking on and twice with it off; Table A6 pairs the per-task mean pass over the two runs. Claude Code and OpenCode move by +2.2 and +3.4 points, with as many tasks going up as down. mini-SWE-agent moves by −11.1 points, with 10 tasks going down and 2 up. Without thinking, Claude Code and OpenCode take more steps, 103.5→126 and 56→78.5 at the median, and mini-SWE-agent fewer.

On HY4-Preview the same switch was thrown inside Claude Code alone, through the API's thinking field, with one run per side. The model passes 23 of 45 with thinking on and 18 with it off, 51→40%, a drop of 11 points; 9 tasks are lost and 4 gained (p = .27). Output falls by more than half, 43.9k to 17.5k tokens per trial, and the median step count rises, 75→90. The bill follows the tokens, 2.86 to 2.14 CNY per trial at the vendor's list price.

Table A6: **Thinking on and off, hard45.** Top block: Qwen3.6-35B-A3B, three harnesses, two runs per side, thinking switched at the serving layer; the per-task mean is paired over tasks both sides scored, and p is a two-sided sign test on the tasks whose mean changed. The OpenCode arms here run without ripgrep in place. Bottom block: HY4-Preview, Claude Code only, one run per side, thinking switched through the API's thinking field; with one run per side the same sign test is McNemar's exact test on the 9 lost and 4 gained tasks.

Harness	solved / attempts on	off	per-task mean on → off	Δ (pp)	up / down	sign p	med. steps on → off	med. output on → off
Claude Code	21/90	22/87	0.233 → 0.256	+2.2	7 / 5	0.77	103.5 → 126	27,761 → 24,550
mini-SWE-agent	28/90	18/90	0.311 → 0.200	−11.1	2 / 10	0.04	106.5 → 98	25,198 → 20,582
OpenCode	15/88	16/86	0.148 → 0.182	+3.4	7 / 5	0.77	56 → 78.5	14,018 → 15,949.5
HY4-Preview, Claude Code only, one run per side, thinking switched at the API								
Claude Code	23/45	18/45	0.511 → 0.400	−11.1	4 / 9	0.27	75 → 90	43,189 → 15,001

Qwen3.6-35B-A3B on hard45, no web access, two independent runs per side and per harness (90 attempts), thinking switched off at the server, not in the harness: a chat template that closes the think block before generation, no reasoning parser, and the `<think>` token banned by a logits processor, so all three harnesses see the same model. *solved* counts attempts with reward 1 over the attempts that are usable (scored, no infra exception); 7 thinking-off and 2 thinking-on attempts were lost to verifier faults (`verifier_incomplete`, one `VerifierTimeoutError`) and were not rerun, which is why some denominators are below 90 and why the OpenCode pairing is over 44 tasks: `django-13449` voided in both thinking-off attempts. *per-task mean* is the mean outcome of a task over the usable attempts of that side, and the comparison is paired over the tasks both sides scored (n = 45, 45 and 44); Δ is off − on in percentage points. *up / down* counts the tasks whose per-task mean rose / fell; the sign test is the two-sided exact binomial on those tasks, the remaining tasks being ties. This is one k=2 observation on 45 tasks. Three caveats. Both OpenCode arms here run *without* the pre-placed ripgrep, because the thinking-off arm had none either; they are therefore not the OC-rg arm of the main tables. The Claude Code arms carry reasoning effort `xhigh`, its setting throughout this paper. And mini-SWE-agent's `reasoning_chars` are visible narration duplicated from the message text, not hidden thinking, so its reasoning length is not comparable with the other two and is omitted here. The second block is a different experiment: HY4-Preview through the vendor's API under Claude Code, one run per side on the same 45 tasks, with thinking switched off by setting the API's thinking field to disabled — the endpoint treats a missing field as on — instead of at the serving layer. With one run per side a per-task mean is 0 or 1, so *up / down* are the discordant tasks and the two-sided sign test on them is McNemar's exact test. At Tencent's list price the trial costs 2.86 CNY with thinking on and 2.14 with it off, from mean output of 43.9k and 17.5k tokens per trial.

K SENSITIVITY OF THE PAIRED DIFFERENCES

Table A7 recomputes every paired difference under four changes: dropping the trials that overflowed the context window, scoring per protocol (dropping tasks where either arm ended at a limit or was lost), using OpenCode as shipped instead of the arm with ripgrep in place, and substituting the second run for the first wherever one exists. No conclusion of the paper changes under any of them. On the pool, the only entries that move by more than 3 points are the two OpenCode gaps

on Qwen3.8-27B under per-protocol scoring, which is the cutoff effect of Appendix H. On hard45, eleven entries move by more than 3 points, all inside that set's resolution: the largest is the shipped OpenCode arm on Qwen3.6-35B-A3B, 11 points worse than the arm with ripgrep in place, and one sign flips, Claude Code minus mini-SWE-agent on DeepSeek-V4-Flash under the second run, by one task each way. mini-SWE-agent's overflows on Qwen3.8-27B mostly fall on tasks the other harnesses also fail: of the 11 hard tasks it lost that way, 3 were passed by either other harness, and of the 18 lost on the pool, 6, so recovering every one of them would move it by 6.7 and 1.3 points, and on rerun 7 of the 11 overflow again.

Table A7: **Sensitivity of every paired difference.** Baseline as in Tables 2 and 1; the other columns recompute the difference after dropping overflowed trials, scoring per protocol, using OpenCode as shipped, and swapping in the second run. = means the variant removes no task and the value is unchanged.

Model	Pair	baseline Δ (pp) [95% CI] (n)	no ovfl.	per-prot.	OC ship	run 2
hard45						
Qwen3.6	CC−mini	-8.9 [-23.9, +6.6] (45)	-9.3^{43}	-6.8^{44}	—	-6.7
	CC−OC	+0.0 [-11.8, +11.8] (45)	=	+0.0^{44}	+11.1	-2.2
	mini−OC	+8.9 [-4.0, +21.5] (45)	+9.3^{43}	=	+20.0	+4.4
Qwen3.8	CC−mini	+7.0 [-5.5, +19.1] (43)	+0.0^{32}	+7.0	—	+4.4^{45}
	CC−OC	+7.0 [-1.8, +15.5] (43)	+7.1^{42}	+8.1^{37}	+4.7	+11.4^{44}
	mini−OC	+0.0 [-11.2, +11.2] (45)	+5.9^{34}	+0.0^{39}	-2.2	+6.8^{44}
DS Flash	CC−mini	-2.2 [-10.5, +6.2] (45)	=	=	—	+2.2*
	CC−OC	+4.4 [-6.7, +15.3] (45)	=	+4.5^{44}	+0.0	+8.9*
	mini−OC	+6.7 [-3.6, +16.6] (45)	=	+6.8^{44}	+2.2	—
GLM Flash	CC−mini	+4.4 [-6.7, +15.3] (45)	=	+6.8^{44}	—	—
	CC−OC	+8.9 [-2.2, +19.5] (45)	=	+11.4^{44}	—	—
	mini−OC	+4.4 [-6.7, +15.4] (45)	=	=	—	—
HY4	CC−mini	-6.7 [-18.2, +5.2] (45)	=	=	—	—
	CC−OC	+4.4 [-6.7, +15.3] (45)	=	=	—	—
	mini−OC	+11.1 [-0.7, +22.2] (45)	=	=	—	—
pool447						
Qwen3.6	CC−mini	-1.8 [-5.2, +1.6] (445)	-1.8^{439}	-1.8^{443}	—	-0.7*
	CC−OC	+2.9 [-0.9, +6.7] (445)	=	+3.2^{443}	+5.0^{421}	—
	mini−OC	+4.7 [+1.1, +8.4] (445)	+5.0^{439}	+4.7	+6.7^{421}	+3.6*
Qwen3.8	CC−mini	-1.4 [-4.2, +1.5] (444)	-2.4^{425}	-1.4^{440}	—	—
	CC−OC	+7.9 [+4.3, +11.5] (444)	+7.9^{443}	+3.4^{410}	+5.9	—
	mini−OC	+9.2 [+5.5, +12.9] (445)	+10.5^{427}	+4.6^{409}	+7.0	—

Every cell is the paired difference (first harness − second) over the tasks both arms scored, computed by `paper_numbers.paired` on the canonical trials; the baseline column is the number in Fig. 1b / Table 2 (intention-to-treat: overflow, cap and cut trials all count as failures). *no ovfl.*: tasks where either arm hit the context window are dropped. *per-prot.* (per protocol): tasks where either arm ended on a step cap, a timeout, an output cut or an infra void are dropped. = marks a variant that is defined but drops no task, i.e. reproduces the baseline exactly. *OC ship*: the as-shipped OpenCode arm (no pre-placed ripgrep) replaces OC-rg. *run 2*: the second independent run replaces run 1. — means the variant does not exist for that cell (no such trial, no as-shipped arm, no second run). * only one of the two arms has a second run; the other stays at run 1. A superscript is the variant's n where it differs from the baseline's.

L A TRANSFER CHECK ON TERMINAL-BENCH 2.1

We ran the three harnesses once with Qwen3.6-35B-A3B on the 53 Terminal-Bench 2.1 tasks whose reference solutions need no network during the agent phase. On the 50 tasks all three arms scored, mini-SWE-agent passes 19, OpenCode 20, and Claude Code 20, with every pairwise McNemar p = 1.0; the flip rates between arms are 10 to 26%. At this size no ordering is detectable, the same as on hard45.