WHY DO COMPUTER-USING AGENTS FAIL? A TRACE-GROUNDED TAXONOMY AND ITS RELIABILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer-using agents (CUAs) are increasingly evaluated on graphical interfaces and live production websites, yet benchmarks typically report only a single terminal reward. This conflates failure origins across the model, agent scaffold, benchmark evaluator, execution harness, and task specification. We introduce a frozen CUA failure taxonomy—6 families and 18 categories—built by grounded-theory analysis over a $1{,}712$-trace construction corpus, and apply it through a three-stage judging pipeline that decouples execution outcome from task success and attributes failures across system components. We evaluate on a disjoint held-out corpus of $3{,}825$ traces produced by a Harbor-based framework running four unmodified agent CLIs across two generations of OSWorld and ClawBench respectively. Category-level classification reaches Krippendorff's $\alpha = 0.840$ on the construction set and holds between 0.764 and 0.915 across the 26 held-out populations with $n > 11$ and two blind annotators agree with the pipeline at $\kappa = 0.69$–0.71, against $\kappa = 0.77$ with each other over a validation set. The diagnostic infrastructure proves to be part of the measurement: trace normalization shifts attribution by 15.3 percentage points, withholding screenshots raises the agent share by 7.9, and one revision of the judge's prompt moves it by roughly twenty—each exceeding the judge's own sampling noise.

1 INTRODUCTION

Computer-using agents (CUAs) turn natural-language instructions into sequences of clicks, keystrokes, and tool calls on real software. Benchmarks such as OSWorld and OSWorld 2.0 run these agents inside full desktop operating systems (Xie et al., 2024; Yuan et al., 2026), and ClawBench runs them on live production websites (Zhang et al., 2026b). These benchmarks make end-to-end success measurable in realistic environments, and they show that current agents still fail on a large share of realistic tasks.

When an episode fails, however, the benchmark reports a single terminal reward, and that number cannot say *why*. The backbone model may have misread the screen, the scaffold may have stopped too early, the evaluator may have rejected a correct final state, the harness or a live website may have blocked progress, or the task may not have been fully specified. Each calls for a different fix and belongs to a different team, so without separating them effort spent improving an agent can be wasted on failures it never caused.

Answering "why did it fail?" at scale needs three things: a way to decide which component failed, a stable vocabulary for the failures that belong to the agent, and evidence that the vocabulary applies consistently to agents and models it was not built from. Each is harder than it looks. Asking an LLM to induce a taxonomy directly from failure descriptions gave us 93, 16, and 32 categories across three replicate runs on the same input; concurrent work reports the same path dependence (Zhang et al., 2026a). Diagnoses also depend on how the judge is configured: normalization, screenshot availability, and the prompt are treated as implementation details, yet each changes the verdict.

In this paper we build the infrastructure to answer this question end to end and use it to study three research questions: **RQ1:** Where do CUA failures originate, and what do agent-caused failures look like? **RQ2:** Does a taxonomy frozen on one corpus transfer reliably to unseen scaffolds, backbones, and benchmark generations? **RQ3:** Do diagnostic-infrastructure choices—trace normalization, visual evidence, and the judge's prompt—change the answer beyond the judge's own sampling noise?

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

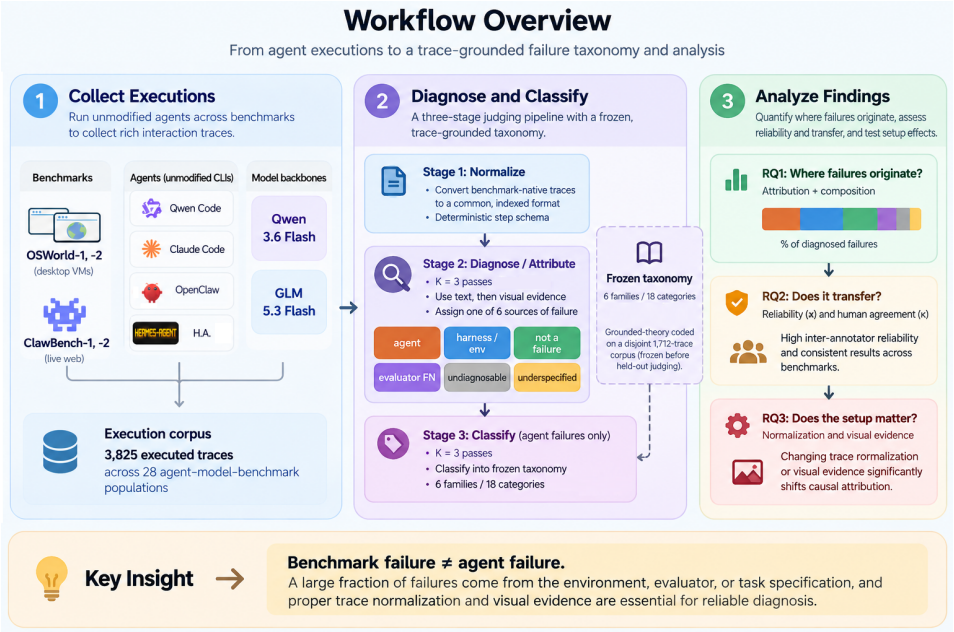

Figure 1: From agent runs to failure analysis. Four unmodified agent CLIs, each paired with two model backbones, produce 3,825 traces across OSWorld and ClawBench. A three-stage pipeline normalizes each trace, attributes every benchmark-reported failure to one of six system components, and classifies agent-attributed failures under a taxonomy frozen on a separate 1,712-trace corpus.

Our infrastructure has three parts (Figure 1): a taxonomy of 6 families and 18 categories frozen before any held-out trace is seen (Section 3); a three-stage judging pipeline that attributes each failure to one of six components and classifies the agent-attributed ones (Section 4); and a Harbor-based framework (Team, 2026) running four unmodified agent CLIs with two backbones across four benchmark generations (Section 5).

Our experiments support four findings. **First**, a large share of diagnosed failures lies outside the agent: under our production judge specification the agent accounts for 47.6%, against 27.7% harness or environment and 14.5% that are not failures at all. The absolute share depends on the judge's prompt (Section 6.4); the ordering of sources does not, and on live websites agent and harness are nearly level. **Second**, agent failures are dominated by *how agents stop*, not by where they click: incomplete termination accounts for 52% of the 3,863 categorized failures. **Third**, the taxonomy transfers, with α between 0.76 and 0.92 across the 26 held-out populations with $n > 11$ and two blind annotators agreeing with the pipeline at $\kappa = 0.69$–0.71, close to the $\kappa = 0.77$ they reach with each other. **Fourth**, the infrastructure itself moves the diagnosis: normalization shifts attribution by 15.3 points beyond resampling noise, screenshots move the agent share down 7.9, and the judge's prompt moves it by roughly twenty.

We contribute a frozen, trace-grounded CUA failure taxonomy of 6 families and 18 categories validated by blind two-annotator agreement (Section 3); a three-stage judging pipeline that attributes each failure to a system component before classifying the agent-caused ones (Section 4); an account of where CUA failures originate across 28 held-out populations (Sections 6.1–6.2); and controlled measurements of how the pipeline's own settings change attribution (Section 6.4).

2 Related Work

Computer-use benchmarks. Interactive benchmarks score an agent by the state it produces rather than by matching a reference action sequence. WebArena and VisualWebArena provide self-hosted websites with execution-based evaluation (Zhou et al., 2024; Koh et al., 2024); OSWorld extends this to desktop operating systems with state checkers over 369 tasks (Xie et al., 2024), and OSWorld 2.0 to 108 longer, multi-application workflows (Yuan et al., 2026). ClawBench moves to 153 write-

heavy tasks on live websites, blocking the final consequential request so an attempted submission is evaluated without real side effects (Zhang et al., 2026b). All provide realistic outcomes, but a terminal score cannot say which component failed.

Failure taxonomies and diagnosis. MAST derives 14 failure modes for multi-agent LLM systems through grounded theory and scales annotation with an LLM judge (Cemri et al., 2026), but its unit is a team of collaborating agents rather than a computer-use trajectory. CUADebug is closest to our setting: it introduces a CUA error taxonomy and a human-annotated set of OSWorld failures, localizes the root-cause step from paired screenshots, and tests whether diagnoses improve re-execution (Zhang et al., 2026a). GUIDE decomposes GUI trajectories into hierarchical subtasks (Zhai et al., 2026), and Shah et al. (2026) characterize faults from issues and pull requests. Where that work builds debuggers and repair loops, we ask whether one frozen vocabulary applies consistently across four scaffolds, two backbones, and four benchmark generations.

Evaluation infrastructure. Harbor provides common abstractions for running heterogeneous agents across tasks, environments, models, and runtime providers (Team, 2026), and recent adapter work emphasizes evaluation parity across execution substrates (Shi et al., 2026). We use it as the orchestration substrate; it neither evaluates tasks nor produces taxonomy labels.

3 Building a Frozen Failure Taxonomy

A vocabulary for agent failures is only useful if two people, or two runs of a judge, assign the same label to the same trace.

Automated induction is not stable enough to anchor a study. Prompting an LLM to induce categories from pooled failure descriptions produced 93, 16, and 32 categories across three replicate runs on identical input. Unsupervised clustering of the same 2,047 descriptions points the same way, yielding hundreds of clusters under both settings tried. A vocabulary of that size, or one that varies several-fold under resampling, cannot support a reliability claim, so we built the taxonomy by hand; the clustering settings are released, and the induction runs are reported as motivation rather than a reproducible comparison.

Construction corpus. Coding was done over a *construction corpus* of 1,712 traces from earlier runs, using different backbones, earlier harness versions, and only two of the four held-out scaffolds. It shares no traces with the held-out corpus, which makes the later results a test of the taxonomy rather than a fit to it.

From open codes to frozen categories. Two researchers coded the corpus following grounded theory (Cemri et al., 2026). We did not read raw trajectories cold: the diagnosis stage (Section 4) writes, for each agent-attributed failure, a description naming the failure, its definition, a summary, and evidence tied to specific steps, and those descriptions are what we open-coded. As codes accumulated we merged those describing the same phenomenon, split those covering two mechanisms, and sampled again, so the scheme was grown rather than designed.

Coding stopped at conceptual saturation, operationalized in advance as **50 consecutive newly sampled instances producing no new category**. We then used an LLM to tighten category names and definitions for consistency of phrasing, after their content was fixed by the human coding. Each category also carries worked examples as annotation aids, released but not analyzed: every number here is at the 18-category level.

Coverage and freeze. To test whether the scheme covers the failure space rather than only the traces we read, we classified the remaining construction-corpus failures against it with an explicit escape hatch: of 2,253 failures, 11 (0.49%) came back `OTHER`. The hierarchy was then frozen (Table 1) before any held-out trace was judged.

4 The Judging Pipeline

The judge runs strictly after execution: it reads frozen traces and never changes a benchmark-native outcome. All model calls use Gemini 3.5 Flash-Lite (Google DeepMind, 2026) under fixed prompt versions and output schemas.

Table 1: The frozen taxonomy (18 categories in 6 families) and each category's share of the 3,863 categorized agent failures in the held-out corpus, per benchmark and overall (%). Families are ordered by overall share; bold marks the largest category per column. OSW: OSWorld; CB: ClawBench.

ID	Family / category	OSW-1	OSW-2	CB-1	CB-2	All
F1	**Incomplete or premature termination**	48.8	57.4	55.3	51.0	52.0
18	Preparatory or exploratory phase mistaken for completion	**23.5**	**35.5**	13.2	17.8	**19.8**
01	Final persistence, commit, or secondary action omitted	7.6	8.5	19.8	13.2	12.9
16	Premature abandonment of a recoverable non-bot obstacle	7.2	3.5	17.6	14.3	11.9
06	Declared complete without verifying the actual result	7.9	4.3	2.1	2.3	4.5
04	Partial completion across repeated or multi-instance targets	2.6	5.7	2.6	3.4	3.0
F6	**Process inefficiency**	9.5	16.0	20.2	24.6	17.0
05	Step-budget exhaustion through process inefficiency	9.5	16.0	**20.2**	**24.6**	17.0
F5	**External blockers**	2.2	3.2	14.5	15.1	9.3
03	Bot-detection or anti-automation challenge	1.8	0.0	14.5	15.1	8.9
11	Environment or dependency setup failure	0.4	3.2	0.0	0.0	0.4
F2	**Grounding and targeting**	12.7	2.8	6.8	5.0	8.2
10	Target or scope misidentification	8.5	0.0	2.8	3.0	4.7
02	Blind or incorrect UI-element targeting	4.0	1.8	3.1	2.0	3.1
17	Imprecise interactive spatial or motor control	0.2	1.1	0.9	0.0	0.4
F4	**Method and output structure**	15.5	14.5	0.4	1.1	7.2
13	Method violates a structural or schema constraint	11.9	8.9	0.3	1.1	5.4
12	Insufficient fidelity for complex generated content	2.1	3.9	0.1	0.0	1.1
14	Output-format or schema non-compliance	1.5	1.8	0.0	0.0	0.7
F3	**Data and value errors**	11.4	6.0	2.9	3.3	6.3
15	Explicitly specified task parameter set incorrectly	5.0	1.8	2.1	2.4	3.2
08	Misread, miscounted, or misjudged source data	3.2	4.3	0.0	0.3	1.6
07	Fabricated or assumed values not grounded in source data	2.5	0.0	0.0	0.1	0.9
09	Simple transcription, typographic, or formatting slip	0.8	0.0	0.7	0.4	0.6
	Categorized failures (n)	1,425	282	1,210	946	3,863

Stage 1: Normalize. Each benchmark-native trajectory is converted deterministically into a common step schema, with step indices assigned in code so that the diagnosis, the image lookup, and the released record all refer to the same step. This stage makes no judgment.

Stage 2: Diagnose. Every benchmark-reported failure receives $k = 3$ independent passes at temperature 0.7. Each reads the text trajectory and evaluator evidence, re-diagnoses with a screenshot if it identifies a suspect step with indexed visual evidence, and assigns one of six sources: `agent`, `harness_or_environment`, `evaluator_false_negative`, `task_underspecified`, `undiagnosable`, and `not_a_failure` (Appendix C). Agent-attributed passes also write the failure description used downstream, and the trace label is the majority vote. One convention is counter-intuitive enough to state: a CAPTCHA or bot-detection block is an *agent* failure here, classified by how the agent responds (category 03); `harness_or_environment` is reserved for cases where nothing could have succeeded as configured.

Stage 3: Classify. Each agent-failure description is mapped to one of the 18 categories, or `OTHER`, by $k = 3$ passes and a majority vote. The classifier sees *only* the description—a proposed name, a definition, and a one-sentence summary—plus the category names and definitions, and not the task, the trajectory, or the attribution evidence, which stops it from quietly re-deriving attribution. In an ablation on 350 cases, broader context changed 22.6% of labels, and hand adjudication of every diverging case found the change was not reliably an improvement, so we kept the minimal interface.

Evidence regimes and counting. Visual evidence differs by benchmark era, so we distinguish three regimes—text only, final screenshot, step-localized—record which each pass used, and never pool across them. Attribution is per trace by majority vote; category distributions are per classification candidate, of which a trace contributes up to k.

5 Experimental Setup

We evaluate on a held-out corpus of 3,825 traces spanning 28 agent–model–benchmark populations (Table 2). Four unmodified agent CLIs—Qwen Code, Claude Code, OpenClaw, and Hermes Agent (Qwen Team, 2026; Anthropic, 2026; OpenClaw Contributors, 2026; Nous Research, 2026)—each run with Qwen 3.6 Flash or GLM 5.3 Flash (Alibaba Cloud, 2026; Zeng et al., 2026) across three backends: VirtualBox for OSWorld-1, nested QEMU/KVM in Docker for OSWorld-2, and Chromium/CDP

Table 2: Held-out execution corpus: tasks frozen before execution, the agent–model cells each was run under, and traces obtained. Task selection and the accounting of attempts that produced no usable trace are in Appendix A.

Benchmark	Upstream tasks	Frozen tasks	Agent–model cells	Traces
OSWorld-1	369	269	8	1,932
OSWorld-2	108	30	8	219
ClawBench-1	153	152	6	903
ClawBench-2	130	129	6	771
Total			28	3,825

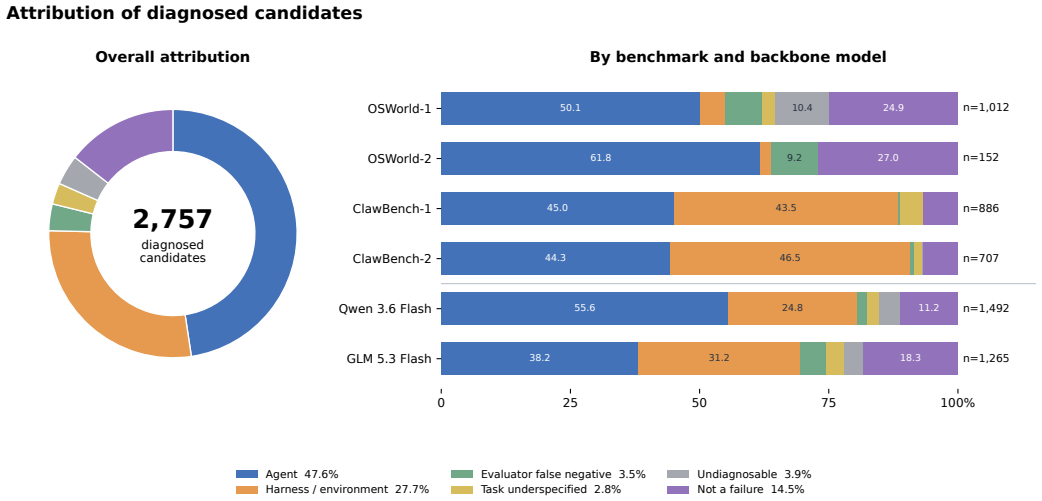

Figure 2: Causal attribution of the 2,757 diagnosed failures, pooled over all 28 populations (left) and separately by benchmark and backbone model (right). Numbers inside bars show percentages for segments of at least 8%; values are reported in Table 11.

with the benchmark's interceptor for ClawBench. Agent policies are not rewritten, and every upstream instruction and evaluator is used unchanged. Task subsets were frozen before any agent ran; OSWorld-1 keeps 269 of 369 tasks after near-duplicate removal. We record *execution outcome* separately from *task success*: reaching a step or time budget is a constrained outcome, while only the benchmark-native evaluator decides success, so the judge explains a benchmark-reported failure but never redefines it (Appendix A).

We report Krippendorff's α over nominal labels within one population at a time: *category* α varies only the Stage-3 pass with the description held fixed, *attribution* α the whole Stage-2 diagnosis. High α shows consistency, not correctness, so we measure the latter against people with Cohen's κ: two annotators independently labeled a stratified 118-instance package against the frozen taxonomy, with the pipeline's labels hidden and without seeing each other's answers (Appendix D.6).

6 Results

6.1 RQ1: Where do CUA failures originate?

Table 11 shows where the 2,757 diagnosed failures come from. The agent accounts for 47.6%; the rest divide among the harness or environment (27.7%), episodes that are not failures on inspection (14.5%), and a tail of undiagnosable traces, evaluator false negatives, and underspecified tasks. These shares are conditional on the judge specification that produced them, which we quantify in Section 6.4; the ranking of sources is stable across the prompt versions we tested. Per-population results are in Table 12.

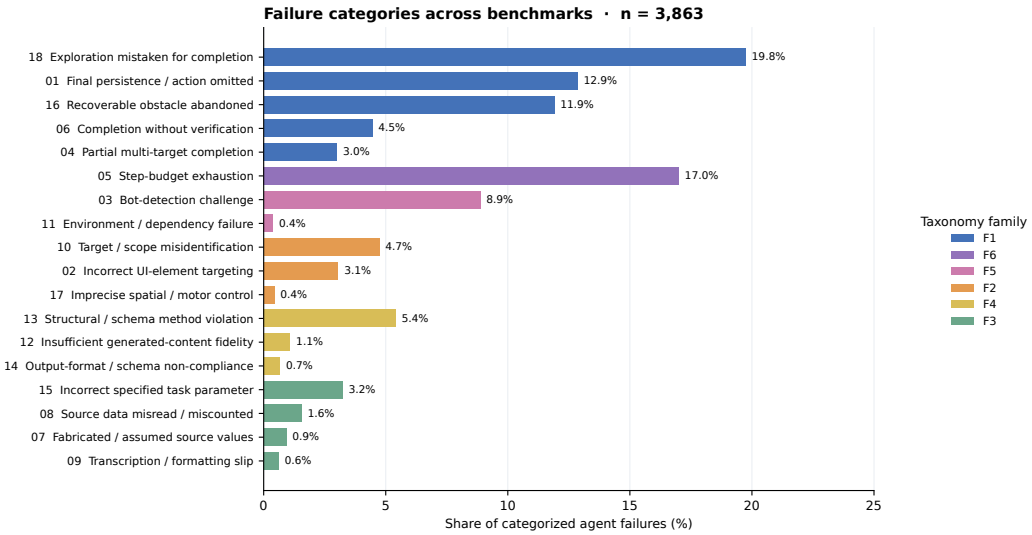

Figure 3: Distribution of the 3,863 categorized agent failures across the 18 frozen categories, pooled over the held-out corpus. Category IDs and families follow Table 1; benchmark-level shares appear in Figure 8.

The environment decides how much of a failure belongs to the agent. On OSWorld the agent dominates (50% and 62%) and the harness is rarely at fault (1–5%); on ClawBench the two are nearly level in both generations. The desktop sandbox removes most infrastructure failures, whereas live websites add dynamic pages, site-side errors, and unreachable resources. Anti-automation blocks are not among these harness attributions: the frozen scheme files those as *agent* failures (Section 4). Evaluator false negatives run the other way, concentrating on OSWorld (7.3–9.2% against under 1%), where state-checking scripts can reject a valid final state.

The backbone shifts attribution, not only the success rate. Qwen 3.6 Flash failures are agent-attributed 55.6% of the time against 38.2% for GLM 5.3 Flash. Because attrition is not uniform across the shared task set, we recomputed the contrast on the 438 task×scaffold cells where both backbones produced a diagnosed trace; the gap is essentially unchanged (11.6 points against 12.6 unmatched). The scaffold matters as much as the model: on OSWorld-1 with GLM the agent share ranges from 19.6% under Claude Code to 43.9% under OpenClaw.

Finding. A benchmark failure is not an agent failure. A large share of benchmark-negative episodes lies outside the agent, and on live websites the harness is responsible almost as often. The absolute split moves with the judge's prompt; the ordering does not.

6.2 What do agent failures look like?

Table 1 gives the composition of the 3,863 categorized agent failures. CUAs most often fail at *stopping correctly*: incomplete termination is the largest family overall (52.0%) and on every benchmark, and its largest category is an agent that finishes a preparatory phase and declares the task done (19.8% overall, 35.5% on OSWorld-2's long-horizon tasks). With step-budget exhaustion (17.0%) and omitted final actions (12.9%), these three cover just under half of all agent failures (49.7%), while grounding and data errors are rare (8.2% and 6.3%).

Each environment has its own failure signature. On OSWorld, failures of method and output structure are common (15.5% and 14.5%), mostly methods that violate a structural or schema constraint of the target file or application; they are almost absent on ClawBench ($< 2\%$). ClawBench instead shows three live-web patterns: bot-detection challenges (14–15%), omitted final commits (13–20%), and premature abandonment of recoverable obstacles (14–18%). The last two are the last-mile hesitation ClawBench's authors observed by manual inspection (Zhang et al., 2026b), here measured across 12 populations under one frozen vocabulary.

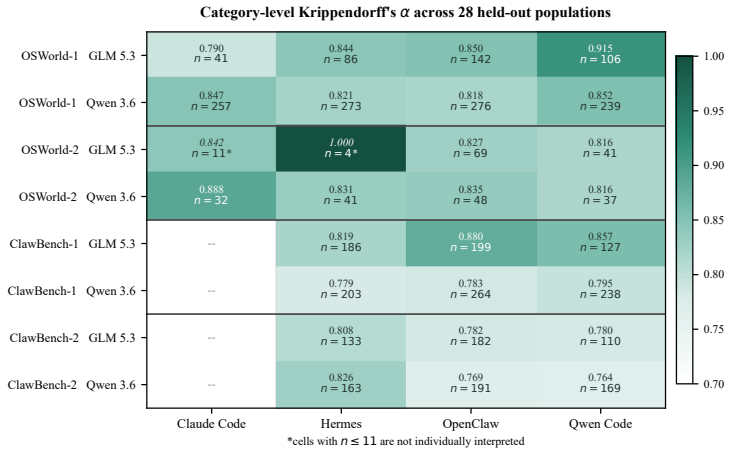

Figure 4: Category-level Krippendorff's α for all 28 held-out populations, by benchmark, backbone, and scaffold. ClawBench has no Claude Code cell. Cells with $n \leq 11$ are not individually interpreted. Numeric values and per-cell n are in Table 15.

Finding. Agents fail at stopping, not at clicking. Premature termination, budget exhaustion, and abandoning recoverable obstacles dominate; fine-grained grounding errors are comparatively rare.

6.3 RQ2: Does the taxonomy transfer?

On the construction corpus, category-level reliability is $\alpha = 0.840$ (95% bootstrap CI $[0.828, 0.853]$, $n = 1{,}907$). Figure 4 reports α for each held-out population, diagnosed independently; two of the four scaffolds and the GLM 5.3 Flash backbone were never observed during construction. Transfer here means to unseen traces, scaffolds, backbones and benchmark generations, not to an unseen model family or a full official benchmark.

Reliability holds across scaffolds, backbones, and benchmark generations. Against the conventional 0.80 bar for drawing conclusions, 20 of the 28 populations clear it and the remaining eight sit in the tentative band above 0.667. Seven of those eight are ClawBench cells, where the categories that most often trade places describe genuinely adjacent situations. OSWorld-2's longer-horizon task set stays inside the OSWorld-1 band (0.816–0.888 against 0.790–0.915), and ClawBench drops only slightly (pooled $\alpha = 0.805$, $n = 2{,}165$).

Pipeline labels track blind human annotation. On the 118 scored instances, the pipeline agrees with the first annotator at $\kappa = 0.694$ and with the second at $\kappa = 0.712$, while the annotators agree with each other at $\kappa = 0.765$. The pipeline's labels are therefore close to, but not interchangeable with, a trained annotator's. The two are not exposure-matched: annotators saw the task instruction and attribution evidence alongside the diagnosis, whereas Stage 3 classifies from the description alone, so these κ values measure agreement with human judgment rather than a matched test of the same interface.

Long-horizon traces separate the annotators from the pipeline, not from each other. On a separately scored 40-instance OSWorld-2 extension, agreement with the pipeline falls to $\kappa = 0.413$ and 0.544, while the annotators agree with each other *more* strongly than anywhere else in the study ($\kappa = 0.807$). The taxonomy therefore holds up on long-horizon traces—two trained annotators apply it to each other's satisfaction—and it is the pipeline that departs from both. Pooling both packages ($n = 158$) gives $\kappa = 0.624$, 0.671, and 0.777; disagreements concentrate on boundaries the definitions already flag as adjacent.

Finding. A taxonomy frozen on one corpus transfers to unseen scaffolds, backbones, and environment types: 20 of 28 populations reach $\alpha \geq 0.80$ on conditional Stage-3 category reliability. Its labels agree with blind human judgment nearly as closely as two annotators agree with each other, though the annotators judged from more evidence than the classifier receives.

Table 3: Sensitivity of causal attribution to three infrastructure choices, each against the judge's own resampling noise. *Verdict change* is the fraction of traces whose trace-level attribution differs between the two settings; *direction* counts agent→harness against harness→agent flips, measured from the first-named setting to the second. Direction counts for the visual-evidence rows are measured on the full corpus (Appendix D.5); the localization re-test is stratified by distance from the end of the trace in Appendix C.

Intervention	Comparison	n	Verdict change	Direction
Tool-call normalization	First call vs. all calls	134	18.7% [13.0, 26.1]	7 vs. 6
	Resampling only	134	7.5% [4.1, 13.2]	–
	Net shift, 5 groups pooled	628	+15.3 pp [5.8, 24.7]	balanced
Judge prompt version	Pre/post revision, random	26	+19.2 pp agent	–
	Pre/post revision, blocker	14	+21.4 pp agent	0 vs. 3
	Resampling only (old prompt)	26	+0.0 pp agent	–
Visual evidence	Text only vs. step-localized	54	16.7% [9.0, 29.0]	256 vs. 42
	Text only vs. final screenshot	51	21.6% [12.5, 34.6]	141 vs. 0
	Final vs. step-localized	31	9.7% ($\kappa = 0.753$)	$p = 0.499$
	re-run, derived steps	211	15.6% ($\kappa = 0.511$)	11 vs. 14

6.4 RQ3: Does the diagnostic infrastructure change the answer?

A diagnosis depends on what the judge is shown and how it is asked. We vary three settings that pipelines usually leave implicit, each against the judge's own resampling noise (Table 3).

Tool-call normalization adds variance beyond sampling noise. Our production normalizer logs only the first of several tool calls in a turn. Preserving all of them changes the verdict for 18.7% of traces against 7.5% for re-running the judge on identical traces; pooled over five groups ($n = 628$) the net shift is 15.3 points (95% CI $[5.8, 24.7]$). It has no preferred direction, so truncation behaves like added noise rather than bias.

The judge's prompt is the largest single effect. We revised the diagnosis prompt once during the study, to derive evidence before writing the attribution, require a named mechanism for any `agent` verdict, and state that an anti-automation block is an agent failure unless the blocking mechanism itself was broken. Because the corpus was diagnosed over a window spanning that change, 87.3% of stored diagnoses ($18{,}497$ of $21{,}186$) predate it. We re-diagnosed the same traces under both versions with the trace, sampling parameters, and k held fixed. Re-running the *old* prompt reproduces the stored agent share on 26 random traces exactly (73.1%), bounding resampling drift at essentially zero; the *current* prompt raises it to 92.3%, and on 14 traces naming an anti-automation blocker the pattern repeats from 78.6% to 100%. A twenty-point move from one prompt revision exceeds either other intervention, and it is the only one a reader cannot inspect from released labels. We release the prompt version and its content hash with every pass-level record.

Visual evidence matters mainly through its presence. Diagnosing from text alone rather than with a step-localized screenshot changes 16.7% of verdicts in the matched-pair subsample ($n = 54$), and adding only a final screenshot shifts a similar share (21.6%, $n = 51$); the direction is measured on the full corpus below. Localizing it to the suspect step then changes little: on 211 traces with mechanically derived steps, final-screenshot and step-localized judges agree 84.4% of the time, equally so whether the frames sit at the end of the trace or a quarter back (84.5% against 84.3%), which rules out the obvious confound that both were showing the same frame. At $n = 211$ this bounds rather than excludes an additional localization effect in these ClawBench conditions, and does not establish equivalence.

Withholding screenshots biases attribution toward the agent. Unlike normalization, removing visual evidence moves verdicts in one direction. Going from a text-only to a step-localized pass, 256 traces moved from `agent` to `harness_or_environment` against only 42 the other way ($p = 1.3 \times 10^{-38}$), and the final-screenshot comparison is more one-sided still (141 against 0). The asymmetry holds corpus-wide: over $2{,}859$ traces the agent share falls by 7.9 points, from 67.1% to 59.2%, once screenshots are attached, and the drop tracks how much the screen carries (-10.1 points on ClawBench against -0.7 on OSWorld-2). Evidence bearing on whether the environment

rather than the agent was responsible—a dead page, a site-side error, an unresponsive control—is often visible only on screen. We report the direction as measured rather than claiming each flip is individually correct (Appendix D.11).

Finding. The judge's own configuration is part of the measurement. Normalization, visual evidence, and the prompt each shift attribution beyond sampling noise—the prompt most of all—and a text-only judge systematically *over*-attributes failures to the agent.

7 Discussion

For benchmark and harness builders. The near-even agent/harness split on ClawBench means a live-web success rate partly measures site behavior and harness robustness; publishing attribution next to the score would show how much of a leaderboard gap belongs to the agents.

For agent developers. The dominance of termination failures suggests that verifying before stopping, and persisting on recoverable obstacles, deserve as much attention as perception and grounding; model and scaffold should be reported as a pair.

For diagnostic pipelines. Text-only and vision-grounded judges disagree in a consistent direction, and the prompt moves attribution further than either, so an attribution is incomplete without its evidence regime, normalization convention, and judge specification.

8 Limitations

Our human validation set (118 instances, plus a separately scored 40-instance OSWorld-2 extension) estimates aggregate but not per-category agreement for the rarest categories, and OSWorld-2 cells are small. The step-localization re-test covers only ClawBench, so it bounds any localization effect rather than excluding one. We study two mid-tier backbones, so absolute success rates are not comparable to frontier-model leaderboards. Category counts are candidate-level at $2.38\times$ the trace count, which leaves the family ordering unchanged but makes the rank of the two largest categories unit-dependent (Appendix D.10). Finally, the judge is addressed through a floating model alias, so re-running the pipeline will not reproduce these labels token for token, though every statistic is reproducible from the retained outputs.

Scope of the reported numbers. Two qualifications bound these results. The attribution distribution in Section 6.1 is a property of the judge specification that produced it: 87.3% of stored diagnoses predate the mid-study prompt revision, so absolute shares would be more agent-weighted under the current prompt throughout. Relative comparisons hold the prompt fixed and are unaffected. The α values in Section 6.3 are *conditional* Stage-3 category reliability, holding the description fixed; measured end to end agreement is $\alpha = 0.60$ and attribution reliability 0.57–0.69 (Appendix D.12).

9 Conclusion

Knowing that a computer-using agent failed is different from knowing why. We built infrastructure that keeps the two apart: a taxonomy of 6 families and 18 categories frozen before evaluation, and a judging pipeline that attributes each failure to a system component before classifying the agent-caused ones. Across 28 held-out populations, a large share of benchmark-negative episodes lies outside the agent, agent failures are dominated by how agents stop rather than where they click, and the taxonomy transfers to unseen scaffolds and backbones. Normalization, evidence regime, and judge specification belong in every reported attribution.

AI use statement

We used generative AI tools in three ways. **As the object of study**, the evaluated agents are built on LLM backbones. **As the measurement instrument**, the judging pipeline uses an LLM to diagnose attribution and classify agent failures, and an LLM labeled the remaining construction-corpus failures against the frozen taxonomy; every call runs under a fixed prompt version and a recorded model identifier. **As research assistance**, AI assistants suggested code for adapters and orchestration

utilities, helped debug runtime failures, and assisted with language editing, accepted only after inspection against source and traces. We have reviewed all AI-assisted work: AI-suggested code was verified against the source and the traces before use, and all AI-assisted text was checked against the underlying results. We did not use generative AI to formulate the research questions, design the experiments, or determine the taxonomy's structure and category boundaries, which were fixed by human open coding. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Ethics statement

This study runs agents in desktop applications and on live websites, which creates risks of unintended external actions, credential disclosure, and incidental capture of user or website data. We retained ClawBench's interception mechanism so that consequential final requests are observed without being submitted. Test accounts and benchmark-provided environments were used where required, and all credentials were supplied only at runtime and excluded from manifests, images, and traces. Video recording was disabled. Released artifacts are screened for secrets and personal data and include only material permitted by the applicable terms. The taxonomy describes agent behavior, not user competence.

Reproducibility statement

The release is assembled from artifacts retained during execution and judging. For each executed population it provides the Harbor execution ledger, the frozen specification it ran against, and the frozen task manifests. On the judging side it contains the three stage prompts and their revision history, the frozen taxonomy, every pass-level diagnosis and classification output rather than only aggregates, and the human-annotation packages with their answer keys. Each headline number is produced by a named script that reads only released artifacts (Appendices A–D); release follows the benchmark, software, and website terms in the Ethics Statement.

References

Alibaba Cloud. Qwen3.6-flash api reference. Model documentation, 2026. URL https://help.aliyun.com/en/model-studio/qwen3-6-flash.

Anthropic. Claude code, 2026. URL https://github.com/anthropics/claude-code.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026.

Google DeepMind. Gemini 3.5 flash-lite model card. Model card, 2026. URL https://deepmind.google/models/model-cards/gemini-3-5-flash-lite/.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

Nous Research. Hermes agent, 2026. URL https://github.com/NousResearch/hermes-agent.

OpenClaw Contributors. Openclaw, 2026. URL https://openclaw.ai/.

Qwen Team. Qwen code, 2026. URL https://github.com/QwenLM/qwen-code.

Mehil B Shah, Mohammad Mehdi Morovati, Mohammad Masudur Rahman, and Foutse Khomh. Characterizing faults in agentic ai: A taxonomy of types, symptoms, and root causes. *arXiv preprint arXiv:2603.06847*, 2026.

Lin Shi, Haowei Lin, Zixuan Zhu, Xiaoyue Zhou, Xiang Li, Xiangning Lin, Yaxuan Deng, Han Xu, Yuangang Li, Shanda Li, et al. Harbor adapters and harbor-index: Infrastructure and a curated meta-dataset for large-scale agentic evaluation. *arXiv preprint arXiv:2609.04298*, 2026.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. *Zenodo*, 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, et al. Osworld2. 0: Benchmarking computer use agents on long-horizon real-world tasks. *arXiv preprint arXiv:2606.29537*, 2026.

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.

Yuwen Zhai, Runze Li, Liang Wang, Nian Shi, Liwu Xu, Wei Zhang, Ran Lin, Bo Xu, and Benlei Cui. Guide: Interpretable gui agent evaluation via hierarchical diagnosis. *arXiv preprint arXiv:2604.04399*, 2026.

Weijia Zhang, Kunlun Zhu, Zeyi Liu, Yinting Chen, Tianyi Ma, Jiateng Liu, Jiaxun Zhang, Bingxuan Li, Xiangru Tang, Heng Ji, et al. Cuadebug: Diagnosing and repairing computer-use agent failures. *arXiv preprint arXiv:2608.02643*, 2026a.

Yuxuan Zhang, Yubo Wang, Yipeng Zhu, Penghui Du, Junwen Miao, Xuan Lu, Zhuofeng Li, Xingwei Qu, Zhengkang Guo, Yuanzhe Shen, et al. Clawbench: Can ai agents complete everyday online tasks? *arXiv preprint arXiv:2604.08523*, 2026b.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A Execution Framework: Setup, Configuration, and Lifecycle

This appendix follows an experiment from its frozen specification to a validated trace manifest. We first state which codebase is responsible for which part of the system, then describe task preparation, runtime images, per-trial lifecycles, agent launch, accounting, recovery, and the handoff to annotation. Machine-local paths and secrets are treated as configuration, not scientific inputs; the reproducibility package represents them with portable variables and secret-free templates.

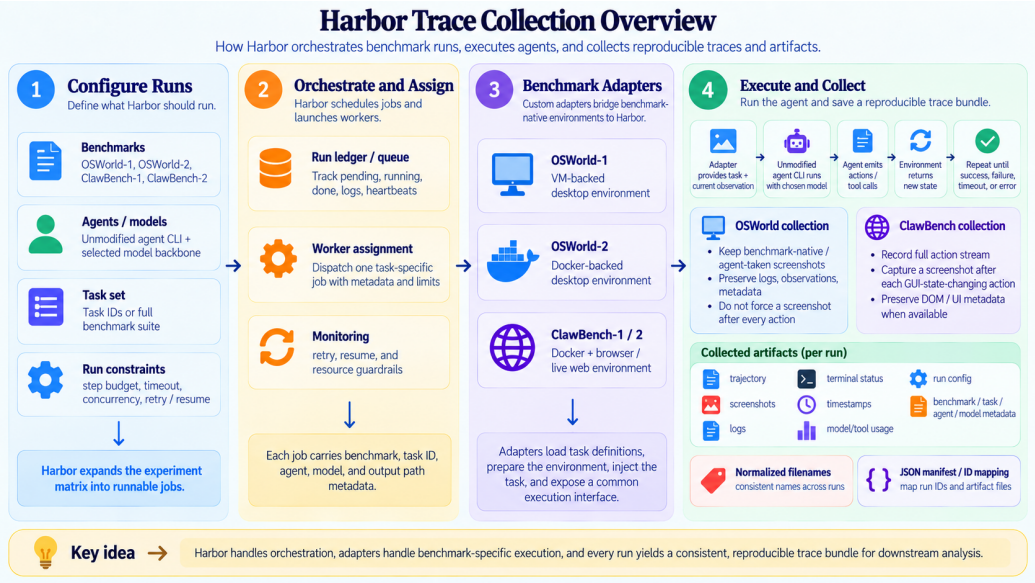

Figure 5: Trace-collection workflow. Harbor freezes and schedules task–agent–model runs; benchmark adapters prepare the OSWorld-1 VirtualBox VM, OSWorld-2 Docker/QEMU guest, or ClawBench browser runtime. Unmodified agent CLIs then interact with the prepared environment while trajectories, screenshots, evaluator results, and execution metadata are collected and validated into trace bundles for downstream annotation.

A.1 Responsibility Boundaries

Four components produce the traces we analyze.

- **Harbor** supplies the task–environment–agent–trial abstraction, the interface for launching installed agents, the execution lifecycle, and the canonical trajectory representation (Team, 2026).
- **OSWorld and OSWorld-2** supply the desktop environments, task instructions, setup operations, assets, control APIs, and state-based evaluators (Xie et al., 2024; Yuan et al., 2026).
- **ClawBench** supplies task definitions, containerized runtime services, Chromium/CDP integration, the interceptor that blocks consequential submissions, and its native recording and evaluation artifacts (Zhang et al., 2026b).
- **The judging pipeline** (Appendix C) reads committed traces after the fact. It never controls an agent or modifies a benchmark environment.

Our own code sits at the boundaries between these systems. It converts each benchmark's task format into a common representation, implements the environments that host the agents, configures agent launch, schedules and monitors execution, normalizes traces, collects telemetry, and handles recovery. Upstream instructions and evaluators are used unchanged and agent SDKs are not modified. Where an upstream file needed an adapter hook, the change is tracked by commit hash and included in the artifact diff.

A.2 VERSION FREEZING

A run whose code, images, or prompts drift during execution cannot be reproduced. Before its first assignment, each paper run therefore freezes the following fields into its plan and ledger: benchmark release and task manifest; the runtime image it starts from; agent CLI name and version; provider-facing model identifier and label; prompt text and role assignment, tool policy, cache policy, output cap, step cap, and timeout; and task IDs, the agent–model product, repetition policy, and node count. Resume and retry read this frozen specification rather than the live configuration file, so later edits affect only genuinely new runs.

All agent images pin Qwen Code 0.19.9, Claude Code 2.1.220, OpenClaw 2026.7.1-2, and Hermes Agent 0.19.1. Table 4 lists the versions used in the study.

Table 4: Agent and model versions used for the execution runs.

Component	Version or identifier
Qwen Code agent	Version `0.19.9`
Claude Code agent	Version `2.1.220`
OpenClaw agent	Version `2026.7.1-2`
Hermes agent	Version `0.19.1`
Qwen 3.6 Flash model	`qwen/qwen3.6-flash` (OpenRouter)
GLM 5.3 Flash model	`z-ai/glm-5.3-flash` (OpenRouter)
Judge model	`gemini-3.5-flash-lite`

A.3 CONFIGURATION AND SECRET HANDLING

Portable defaults live in a single file, `harbor/environment/config.json`. It names the OSWorld task and asset locations, the VirtualBox executable, VM image, and node directory, the Docker images and ports, the agent–model matrix, output limits, step and time budgets, the retry schedule, the screenshot policy, and the prompt-cache policy. Machine-specific paths are overridden by a platform configuration or documented environment variables without editing the shared defaults.

Secrets never appear in the configuration, task manifest, VM image, Docker image, trace, or ledger. Provider keys and the optional email and remote-control credentials are read from a protected environment file or entered with hidden input, and the runner passes each attempt only the values it needs. Publication tooling scans text and JSON artifacts for key patterns before release.

A.4 OSWORLD-1 TASK CURATION

We selected a subset of the 369 official OSWorld-1 tasks that keeps the benchmark's diversity while removing trivial and near-duplicate tasks:

1. Parse all 369 task IDs and instructions with their application category.
2. Embed each instruction with `google/embeddinggemma-300m`.
3. Within each category, cluster instructions with agglomerative clustering (complete linkage, cosine distance, threshold 0.4). The frozen artifact records 75 clusters.
4. Review each cluster under one rubric: keep tasks that are challenging and meaningfully distinct (GUI reasoning, navigation, state tracking, multi-step interaction, precision, or recovery); drop trivial, near-duplicate, or operationally ambiguous tasks.
5. Validate every retained item against its source cluster, rejecting invented IDs, altered instructions, duplicates, and malformed JSON.
6. Merge selections in a stable category/cluster/ID order into the final 269-task manifest.
7. Hash the source corpus, cluster artifact, selection artifact, and final manifest, and produce a per-cluster retention table.

Curation changes which tasks are in the study, never a task's initial state or reward function, and was completed before any agent ran. A second researcher reviewed the retained set against the rubric.

A.5 OSWORLD-2 TASK SELECTION AND RELEASE PREPARATION

OSWorld-2 contains 108 long-horizon, multi-application tasks. We excluded eight tasks that require human interaction during execution and one that requires unavailable Google Drive access. These exclusions are recorded in a skip manifest and are never counted as agent failures. Selection from the remaining 99 tasks used task descriptions, not agent results. An AI-assisted pass first ordered tasks into easy, medium, and hard by the interaction and cross-application work they describe. The rubric then required ten tasks per difficulty level and at least three per represented task category, favoring long-horizon workflows across browser, office, communication, shell and developer tools, file management, media editing, scientific visualization, research, and CAD/3D work. Two researchers applied the rubric independently and arrived at the same 30 tasks. The ordered task-ID map was frozen with the ledger before execution. This is a purposeful diagnostic subset, not a random sample.

All OSWorld-2 components come from one supported release. Before execution, the runner checks that the code tag, gated task classes and assets, website suffix or self-hosted website revision, and provider image are mutually consistent. Assets are resolved from a local snapshot rather than a floating remote branch, and a private GitLab deployment supports the workflows that need it.

A.6 RUNTIME IMAGES

ClawBench image. We build one image that extends the benchmark runtime and installs all four pinned CLIs, instead of installing them per trial. Build validation checks each executable and version, the runtime server's dependencies, Chromium/CDP availability, and the interceptor health endpoint. Trial containers inherit this immutable layer, which avoids repeated downloads and version drift. The deployment uses no host bind mounts; adapter files are copied into the build context.

OSWorld-2 nested-QEMU image. OSWorld-2's upstream Docker wrapper boots a QCOW2 guest disk under QEMU/KVM. To start every trial from a guest with all four agents installed, we booted the release-matched Ubuntu disk in a temporary node, installed and verified the agents, powered the guest off cleanly, and committed a checkpoint. We then flattened the writable `/boot.qcow2` overlay and its backing `/System.qcow2` into a single disk and copied it into a clean wrapper image, so the image does not depend on a backing layer that could later disappear. Validation ran from a fresh container with a fresh storage volume and checked the guest API, screenshot capture, file upload, all four CLI versions, and a short end-to-end agent run. The final image was exported as a compressed archive.

A.7 PER-TRIAL LIFECYCLES

OSWorld-1 on VirtualBox. Each logical node is set up once and then restored per trial. Setup imports the VM image as an independent VM, assigns resources and a NAT control-port rule, and creates an `initial` snapshot. A warm-up step starts the VM headlessly, waits for the OSWorld control API, installs or verifies the four CLIs and computer-use support, validates the screenshot endpoint and agent versions, and saves a benchmark-specific warm snapshot. At each trial, the worker takes exclusive ownership of a node, restores the warm snapshot, configures its host port, starts the VM, and waits for readiness, retrying if VirtualBox reports that a previous session is still unlocking. Task setup runs under its own timeout so it does not consume the agent's budget. Task files are uploaded through the guest API, an initial screenshot is captured after a settle period, and the agent begins. Afterwards the native evaluator runs on the final guest state and artifacts are downloaded. Because every trial restores the same snapshot, no attempt inherits another's state. Interrupt and remote-stop signals propagate from the coordinator to every worker.

OSWorld-2 on nested QEMU/KVM. This backend runs the guest inside a nested QEMU/KVM virtual machine and requires Linux with `/dev/kvm`. For each worker, the coordinator allocates non-overlapping loopback ports for the guest control API, CDP, noVNC, and an optional media service, and creates a uniquely named container with the pinned runtime image, `/dev/kvm` and `NET_ADMIN`, no host bind mounts, a fresh anonymous `/storage` volume, the configured CPU and memory, and labels identifying its paper, matrix, node, and attempt. A container can report "running" before the guest is usable, so the worker waits until the guest screenshot endpoint responds. The upstream disk exposes 50 GiB but its root partition initially occupies about 29.5 GiB; before task

setup, the adapter grows the partition, resizes the ext4 filesystem online, clears safe installer caches, and checks a free-space threshold, since some tasks otherwise run out of disk. It then verifies the computer-control MCP endpoint, uploads assets, runs the official task setup, and clears setup-time desktop notifications just before the agent starts. Notification services stay enabled, so notifications caused by the agent remain part of the environment. The agent receives the official instruction and the public guest password needed for legitimate in-guest operations. After evaluation, cleanup removes exactly the container, volume, and network this attempt created, without a global prune that could disturb other jobs. Logs, guest diagnostics, and preflight output of failed trials are retained.

ClawBench. For each task, the adapter produces a Harbor task directory with the instruction, environment, agent step, and an artifact-copy step. The environment starts the benchmark runtime server and its Chromium instance, and is considered ready only when both the runtime/interceptor status endpoint and CDP discovery succeed within a 60-second probe window, separate from the agent's budget. Because agents expect the browser under different names, the adapter exports the same CDP URL under each alias they use. The system instruction requires the agent to use the existing Chromium session rather than a second browser or direct API/SMTP calls, explains how completion is verified, and notes that a final submission may be intercepted. Only task-specific content is sent as the user message, and first- and second-generation instructions remain distinct even when they share a policy prefix.

With restricted tools enabled, each agent receives the browser surface it normally uses: Playwright MCP for Qwen Code and Claude Code where configured, the Hermes CDP browser toolset, and OpenClaw's native browser configuration. Read-only access to task files is kept; shell access, package installation, code editing, Git, delegation, messaging, and unrelated web tools are removed when they cannot contribute to browser-based execution. A preflight check validates the live CDP endpoint and each agent's required tools before the first provider request. This restriction applies to ClawBench only. During execution, the interceptor records relevant requests and blocks consequential final submissions. An official positive interception can end the run early and is stored as its own evidence; legacy non-matching placeholders are left unscored. Collection never assigns an unofficial reward.

A.8 Agent launch, budgets, and accounting

Launch and routing. The coordinator creates one run item per task–agent–model combination. The adapter launches the native CLI non-interactively with the frozen model ID, endpoint, prompt roles, environment variables, and output limit, leaving the agent's reasoning and tool loop untouched. Exit status is propagated through pipes so a failed CLI cannot appear successful because its logging wrapper exited cleanly. Each attempt has a unique session identity, and requests within an attempt keep a stable provider affinity so the agent's context is not split across backend routes. The adapter collects the CLI's native session and transcript files and converts observable model turns into the common trajectory format. Terminal-event validation distinguishes a normal exit, missing telemetry, context overflow, provider error, and external interruption.

Budgets. An LLM step is one provider call (one model turn), which may contain zero, one, or several tool calls. Caps are enforced on LLM steps, and both counts are reported. Table 5 lists the budgets. They are maxima, not targets: a native success signal ends the trial immediately, and an agent that exits early is recorded as `execution_completed`, not as a success. Hitting a step, time, or idle limit is reported as a constrained outcome rather than a software failure.

Table 5: Execution budgets per benchmark.

Benchmark	LLM-step cap	Agent timeout	Browser idle guard
OSWorld-1	150	50 min	–
OSWorld-2	500	180 min	–
ClawBench-1	100	30 min	5 min
ClawBench-2	100	30 min	5 min

Prompt caching and cost. Where the CLI permits, prompts are arranged as a stable prefix followed by an append-only history, and tool definitions and policy text stay fixed within an attempt. Qwen-

compatible routes use explicit ephemeral cache markers at supported boundaries; GLM keeps a stable session route and relies on the provider's implicit prefix caching. The cache lifetime is five minutes, which consecutive calls within an attempt normally fall inside. Each terminal summary records input, output, and cache-read tokens, with the cache ratio defined as cache-read over input tokens. Cost comes from the provider's generation-level cost when available, otherwise from token accounting with a versioned price catalog, and is otherwise left unavailable; estimates are labeled as such. Because other devices may draw on the same OpenRouter balance, balance changes serve only as a consistency check. Provider generation and session identifiers are logged so anomalies can be reconciled without storing the API key.

A.9 Screenshots and trace layout

Screenshot policy. ClawBench video recording is disabled, and screenshots are captured only when they are likely to matter: once after setup and before the first action; after any tool classified as able to change GUI state, following a settle delay; never twice for a turn that already contains an agent-requested screenshot; never after read-only, planning, logging, or non-browser tools; and once before teardown and evaluation. Images receive short monotonic integer names, which avoids path-length failures on Windows. The image index records each image's number, relative path, source actor, turn and tool-call IDs, capture reason, action name, request and completion timestamps, delay, and status. Recorder images stay outside the model's conversation; only screenshots the agent explicitly requests are sent to the model.

Canonical trace. The trace directory is keyed by benchmark version, agent, model, and task ID, and attempt metadata lives in the ledger. A complete trace contains: **identity** (paper, matrix, and attempt IDs; task ID; benchmark release; agent, model, and provider; node; timestamps); **inputs** (system and user prompts, tool schema and policy, task-file manifest); **trajectory** (normalized messages, model turns, tool calls and results, error events, native transcript references); **browser/desktop** (actions, intercepted requests, screenshot index and images); **evaluation** (native reward, partial metrics, evaluator logs, interception evidence); **execution** (terminal status, task-success status and source, step and tool counts, duration, exception, limits, stop reason); and **usage** (input, output, and cache tokens, exact or estimated cost, generation and session IDs). Large native request logs are stored once, transient work directories are deleted after their content is committed, and agent screenshots are moved rather than copied into the recorder tree.

A.10 Ledger, recovery, and monitoring

Planned versus executed attempts. The frozen task $\times$ agent–model matrix defines $4{,}078$ planned attempts: $2{,}152$ on OSWorld-1 (269×8), 240 on OSWorld-2 (30×8), 912 on ClawBench-1 (152×6), and 774 on ClawBench-2 (129×6). Of these, $3{,}825$ produced a usable normalized trace and form the held-out trace corpus (Table 2); the remaining 253 (6.2%) produced none—a VM or container fault, a provider or harness error exhausting its retry budget, or a trace that failed normalization—and are excluded entirely rather than counted as failures. Because an attempt that yields no trace is not an agent outcome, including it would inflate the failure rate with infrastructure attrition. Per-benchmark attrition is 10.2% on OSWorld-1, 8.8% on OSWorld-2, 1.0% on ClawBench-1, and 0.4% on ClawBench-2; the desktop-VM backends account for nearly all of it.

Ledger. Only the master process writes the SQLite ledger; workers report assignment and completion events to it. An entry is marked running with its node and attempt ID before assignment, and a completed trace is staged, validated, and atomically moved to its canonical location before the entry is marked terminal.

Resume and retry. `resume` reloads the frozen plan and schedules entries that are still queued or were interrupted. `retry-mode` schedules only outcomes configured as retryable (agent, provider, or harness errors, and context overflow); it never requeues step, time, or idle limits or completed entries. Each retry receives a new attempt ID and a clean environment; the ledger keeps the full lineage and the canonical trace points to the latest accepted attempt. A portable resume mode rewrites only runtime paths, so a trace tree and ledger can move between Windows and Linux mid-experiment without becoming a new experiment.

Health checks. Preflight validation checks configuration types, file existence, task counts, duplicate IDs, provider credentials, backend availability, image existence, port collisions, and agent versions. Runtime checks cover guest or container readiness, screenshot and control endpoints, CDP and browser tools, and required benchmark services. Failures are serialized into the trace and ledger. Provider errors use jittered bounded retries, and new assignments pause during a shared provider backoff. A missing required terminal event is classified as `telemetry_missing`, never as success.

Monitoring and controlled stop. For multi-day runs, the coordinator emits structured summaries and periodic email snapshots of progress, active assignments, per-run tokens, cost, and duration, and host resources. Two independent mechanisms can stop a run: a polled remote stop control that halts new assignments and tears down owned containers, and a credit guard that triggers the same path when the provider balance falls below a floor. Both are authenticated before a long run begins.

Commands. Each benchmark is launched through a matrix runner, with Windows/PowerShell and Linux/Bash wrappers over the same Python implementation, taking a provider, backend, task set, and paper/node identifier. Paper IDs, node counts, and task lists are frozen in the ledger rather than hard-coded. Appending `--resume` resumes unfinished work, and `--retry-mode` retries eligible failures. The full scripts are included in the code release.

A.11 Analysis handoff and reproducibility controls

Before judging, a validation pass checks that every terminal ledger entry has exactly one canonical trace, every image index entry points to an existing file, every model turn has consistent step and usage accounting, evaluator artifacts identify their benchmark, and no credential pattern appears in the data. Only then is a content-addressed analysis manifest created. The judge reads only this manifest, which records the trace-schema version, benchmark and execution identifiers, and the locations of trajectory, evaluator, and image-index artifacts. Every judgment joins back to the ledger by an immutable attempt ID, and the paper's tables are generated from these joined records.

Together, these mechanisms enforce eight controls: denominators come from the ledger, not directory counts; exact and estimated costs are never mixed; constrained outcomes are not counted as infrastructure failures; native ClawBench outcomes and judge outcomes are stored separately; skip manifests and purposeful subsets stay explicit; public traces are scanned for secrets and checked against benchmark and site licenses; backend smoke tests exercise reset, asset setup, all four launchers, native evaluation, trace persistence, and cleanup; and resume and retry preserve the frozen plan and record every attempt under its own lineage.

B Taxonomy Construction and Failure Distributions

B.1 Construction corpus and procedure

The construction corpus contains $1{,}712$ traces from OSWorld-1, OSWorld-2, and an earlier ClawBench release, covering Claude, GPT, and Qwen backbones. It shares no traces with the held-out corpus and was collected with earlier harness versions, different backbone models, and only two of the four held-out scaffolds. Automated induction over pooled failure descriptions produced 93, 16, and 32 categories in three replicate runs and was discarded.

The final taxonomy was built by grounded-theory coding by two researchers, who read randomly sampled agent-attributed failure descriptions, assigned open codes, and refined category boundaries by merging and splitting codes. Coding stopped at saturation, when 50 consecutive newly sampled instances produced no new category. The hierarchy was then frozen before any held-out trace was diagnosed, so the held-out results test a scheme that was already fixed. The remaining agent-attributed construction-corpus failures were labeled against the frozen hierarchy by Gemini 3.5 Flash-Lite (Google DeepMind, 2026). Of these $2{,}253$ failures, 11 (0.49%) were emitted as `OTHER`, which serves as an out-of-sample coverage check on the saturation sample.

B.2 Label semantics

The hierarchy has two levels. The 18 categories (Table 1) are the unit of analysis, and the six families group them. Families carry immutable identifiers: `family_01_incomplete_termination`, `family_02_grounding_targeting`, `family_03_data_value_errors`, `family_04_method_output_structure`, `family_05_external_blockers`, and `family_06_process_inefficiency`. A candidate that fits no category is emitted as `OTHER` rather than forced into an incorrect label.

Boundary observations. Two boundaries of the frozen scheme are worth knowing when reading the distributions. A non-spatial programmatic keypress failure commonly files under category 17 (imprecise interactive spatial or motor control), and an application-level preference-file edit commonly files under category 11 (environment or dependency setup failure). Both are properties of the frozen boundaries; changing either would require a new taxonomy version and a full reclassification.

B.3 Failure distributions

Figure 6 plots per-population reliability; the per-population distributions themselves are tabulated below. All figures are generated directly from the released per-population results.

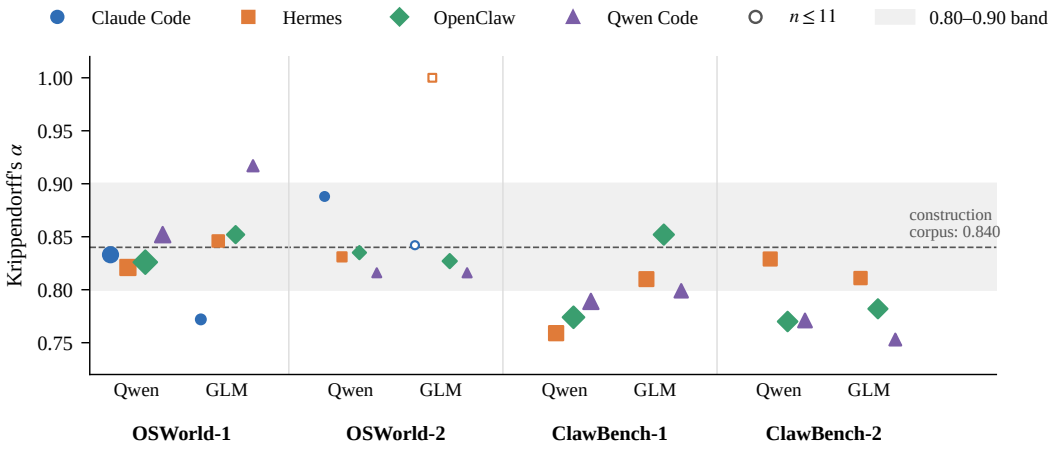

Figure 6: Category-level Krippendorff's α for all 28 held-out populations (values in Table 15). Marker size grows with the number of classified candidates; hollow markers denote cells with $n \leq 11$. The dashed line is the construction-corpus value (0.840) and the shaded band spans 0.80–0.90.

B.4 Per-population failure counts

Tables 6–9 give raw family and category counts for each of the 28 held-out populations. Family and category IDs follow Table 1. Each row's n sums the 18 frozen categories and excludes `OTHER`. Because one attributed episode can yield more than one classified candidate, n need not equal the failing-episode denominators in Table 12.

C Judging Pipeline Details

The pipeline runs strictly after execution. It reads only the manifest from Appendix A.11, never interacts with an environment, and never modifies a benchmark reward (Figure 9).

C.1 Annotation units

The pipeline keeps four questions apart: the *execution outcome* (whether the software ran to completion, errored, timed out, or hit a context limit); the *task outcome* (the benchmark evaluator's score, which the pipeline reads but never changes); the *causal attribution* (which component was primarily

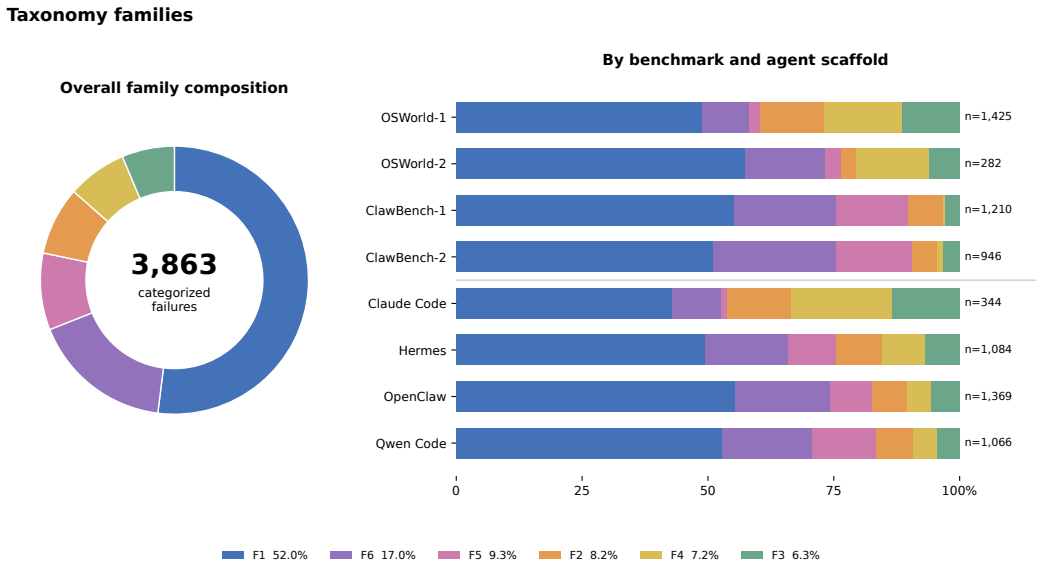

Figure 7: Family composition of the 3,863 categorized agent failures, pooled overall (left) and separately by benchmark or agent scaffold (right). Scaffold rows pool over applicable benchmarks and both backbones; ClawBench has no Claude Code run.

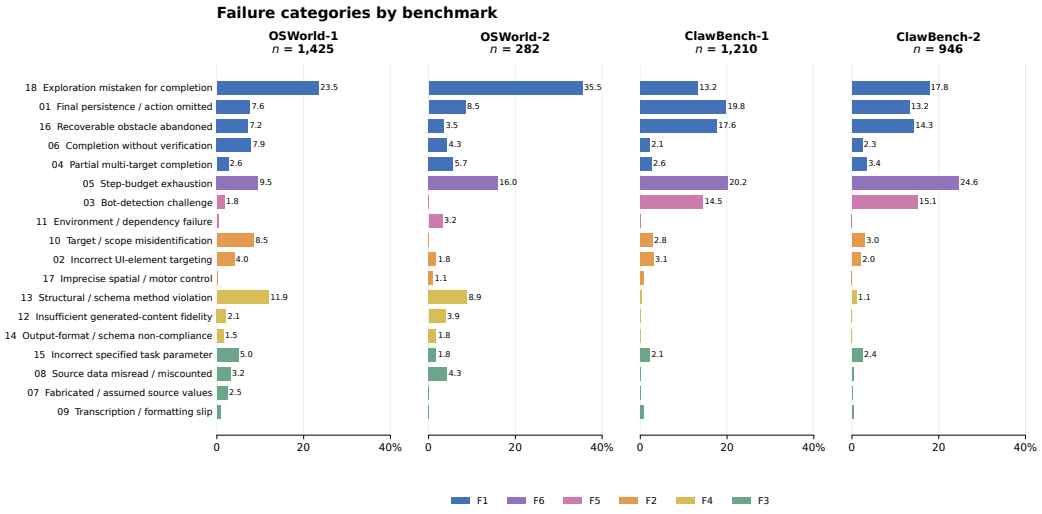

Figure 8: Category shares (%) of categorized agent failures by benchmark, pooled over scaffolds and backbones. Category order and family colors match Figure 3.

responsible for a failure); and the *failure category* (assigned only when the attribution is `agent`). The unit of annotation is one attempt, identified by its attempt ID. Attribution is assigned to the whole episode, with individual steps used as supporting evidence, and reliability is computed over episodes.

C.2 Input contract and normalization

Each annotated item contains, where available, the task instruction, benchmark outcome, execution status, normalized model turns, tool calls and results, native-transcript references, evaluator artifacts, and the indexed screenshots of the attempt. Normalization assigns step indices in code, so the text diagnosis, image lookup, classification record, and released artifact all refer to the same step (Figure 10). It never discards the native trace: it builds a separate judge-facing representation with

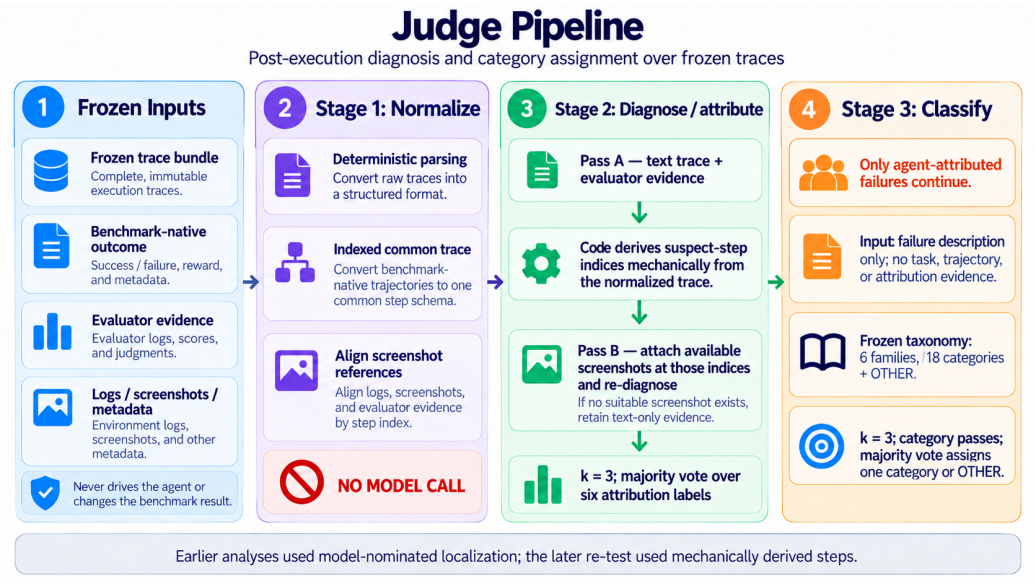

Figure 9: Post-execution judging pipeline. Deterministic normalization precedes three-pass causal attribution; only agent-attributed failures proceed to description-only classification under the frozen taxonomy. The mechanically derived suspect steps shown here are the current implementation and the basis of the $n = 211$ localization re-test; earlier visual-evidence contrasts (Section 6.4) used steps nominated by the text pass.

references back to the source messages and files, and every batch records the schema version and normalization-code revision. Experimental variants, such as preserving every tool call in a turn instead of only the first, receive their own configuration identifiers so their verdicts can be compared with production without overwriting it.

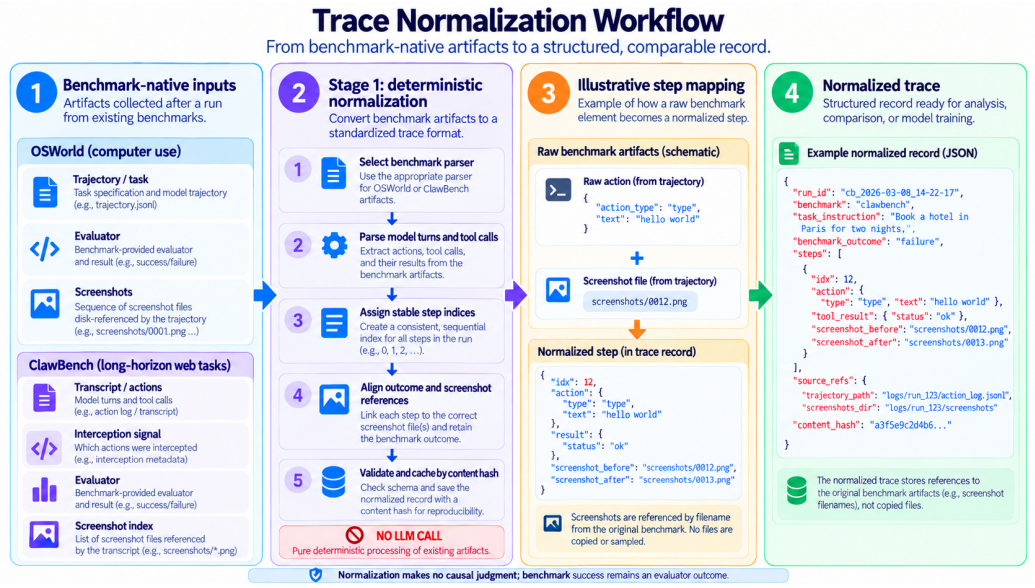

Figure 10: Illustrative trace-normalization workflow. Benchmark-specific parsers map native trajectories, evaluator outcomes, and screenshot references into an indexed common record without a model call. The example fields and values are schematic, not a literal exported trace.

C.3 THREE STAGES

All calls use the model string `gemini-3.5-flash-lite` under a frozen output schema and a fixed prompt version. This is a floating alias rather than a dated snapshot; the identifier it resolved to is recorded in the `model` field of every Stage 2 and Stage 3 output.

Stage 1: normalization. Benchmark-native trajectories are converted into the common step representation by deterministic parsing and indexing only.

Stage 2: causal diagnosis. Every trace the benchmark reports as failed receives $k = 3$ independent passes at temperature 0.7. Within a pass, the judge first examines the text trajectory and evaluator evidence, then re-diagnoses with available screenshots at suspect steps. Earlier visual analyses used steps nominated by the text pass; the later localization re-test derived them mechanically from the normalized trace, as shown in Figure 9. Each pass returns a closed-set attribution (Table 10), a rationale grounded in the evidence, references to suspect steps, and, for agent attributions, a factual failure description for Stage 3. The trace label is the majority vote. All pass-level outputs are kept, including unanimous ones, so sampling variability and evidence-dependent changes can be audited.

Stage 3: frozen-taxonomy classification. Only agent-attributed outputs are classified. The classifier receives the factual failure description and the candidate category names and definitions, but not the task instruction, the trajectory, or the attribution evidence. Withholding this context prevents the classifier from re-performing attribution and forces its decision to depend on the description and the category boundaries. Classification uses $k = 3$ passes at temperature 0.3 and a majority vote; the released record keeps every pass-level label, the aggregate category, its family, and any `OTHER` outcome. In an ablation over 350 cases, broader trajectory context changed 22.6% of labels ($79/350$). All 79 diverging cases were hand-adjudicated against the task, the evidence, and the frozen definitions: 39 (49.4%) were regressions, 28 (35.4%) improvements, and 12 (15.2%) ambiguous. Added context therefore changes roughly one candidate in five without reliably improving it, so the minimal interface was kept.

C.4 VISUAL-EVIDENCE REGIMES

Available visual evidence differs across benchmark eras. OSWorld and re-instrumented ClawBench traces provide screenshots localized to the suspect step; legacy ClawBench traces provide only a terminal screenshot; and some controlled analyses withhold all images. We therefore distinguish text-only, final-screenshot, and step-localized regimes and record the regime of every pass. A terminal screenshot is never relabeled as step-localized evidence, and a missing image is never inferred from a neighboring step. Results from different regimes are compared as sensitivity analyses and never pooled.

C.5 AGGREGATION AND RELIABILITY

Attribution agreement asks whether repeated diagnosis passes choose the same responsible component. Category reliability asks whether repeated classification passes apply the same one of the 18 labels to an agent-failure description; we measure it with Krippendorff's α over nominal labels, which corrects observed agreement for agreement expected by chance. Every α is computed within a single stated population; we never pool across scaffolds, backbones, benchmark versions, or corpus eras. Reliability measures reproducibility under the frozen judge specification, not correctness: passes could agree on a label that is systematically wrong. Human adjudication is therefore kept as a separate validation layer.

D ADDITIONAL RESULTS

D.1 PER-POPULATION ATTRIBUTION

Table 12 gives the attribution distribution for each of the 28 held-out populations, from which the pooled rows of Table 11 are summed. All twelve ClawBench populations were executed under our framework and are distinct from the construction-era ClawBench runs.

D.2 CONSTRUCTION-ERA ATTRIBUTION

Before the held-out protocol was adopted, an earlier generation of OSWorld-1, OSWorld-2, and ClawBench runs was diagnosed with the same three-stage pipeline. These runs predate the held-out collection protocol and its frozen agent–model matrix, and ClawBench was then scored in two separate stages, reported both individually and merged under the benchmark's two-stage AND criterion. Table 13 reports them on the same basis as Table 12; they are never pooled with the held-out corpus. On the merged ClawBench population, category-level reliability is $\alpha = 0.851$ ($n = 219$).

D.3 OSWORLD-2 DETAIL

Of OSWorld-2's 219 normalized traces, 152 contained a diagnosable failure and were diagnosed at $k = 3$ (456 samples). At the sample level, 281 (61.6%) were agent-attributed, 113 `not_a_failure`, 47 `evaluator_false_negative`, 11 `harness_or_environment`, and 4 `undiagnosable`. At the trace level, 94 of 152 (61.8%) are agent-attributed. The 284 agent-attributed candidates (one sample can yield more than one) comprise 282 non-`OTHER` category instances—the OSWorld-2 column of Table 1—and 2 `OTHER`, reaching category-level $\alpha = 0.865$.

D.4 TOOL-CALL NORMALIZATION

Table 14 reports the net attribution shift from preserving all tool calls, beyond matched resampling noise, for each of the five test groups pooled in Section 6.4. A directionality test (exact binomial against a 0.5 null) finds no significant asymmetry in any group (p from 0.21 to 1.0); agent→harness and harness→agent flips are balanced, for example 7 versus 6 in the primary group and 11 versus 12 in Replication 1. Tool-call truncation therefore behaves as direction-agnostic noise on top of judge sampling variance.

D.5 VISUAL EVIDENCE

Final and step-localized screenshots agree on 90.3% of verdicts ($\kappa = 0.753$, $n = 31$), and moving from a final to a step-localized screenshot produces no significant further change ($p = 0.499$). In a controlled test where only terminal visual evidence was available, adding the final screenshot to text changed 21.6% of verdicts (95% CI $[12.5, 34.6]\%$, $n = 51$), the same order as the text-only versus step-localized shift (16.7%). The presence of any visual evidence, not its granularity, is what matters at these sample sizes.

The shifts are also directional. Repeating the text-only versus step-localized comparison on the full corrected corpus (n = 1,072), 256 traces moved from `agent` to `harness_or_environment` and 42 moved the other way (exact binomial $p = 1.3 \times 10^{-38}$). The final-screenshot comparison is more one-sided ($n = 357$; 141 versus 0, $p = 7.2 \times 10^{-43}$), and the same direction holds in the smaller multimodal step-localized and final-screenshot partitions (27 versus 10, $p = 0.008$; 8 versus 0, $p = 0.008$). Adding visual evidence therefore moves verdicts from `agent` toward `harness_or_environment`, and withholding it moves them back toward the agent. Much of the evidence bearing on whether the environment rather than the agent was responsible—a dead page, a site-side error, an unresponsive control—is visible only on screen, so a text-only judge sees an agent acting without effect and charges the agent. Under the frozen convention an anti-automation block is not such evidence: it remains an `agent` failure (category 03). This contrasts with the direction-agnostic normalization effect above.

D.6 HUMAN VALIDATION

The validation package initially contained 143 stratified instances. A later audit found rendering defects in part of the visual package; the 25 affected instances were marked stale, with their content and reasons preserved, and were not re-scored. Two annotators reviewed the remaining 118 instances independently, with pipeline labels hidden and without access to each other's answers. Each record showed the task instruction, the attribution evidence, and the diagnosis under review. Annotators selected one or more of the frozen taxonomy's labels (or `OTHER`), wrote a short rationale, and rated their confidence on a five-point scale; all scoring is at the 18-category level. Across both packages each annotator recorded 183 verdicts in total (the 143 prepared for the original package plus the

40 of the extension), of which $118 + 40$ are scored here. The second annotator recorded multiple categories on 39 of those 183 verdicts and the first on none; every such verdict is scored on its first listed category, the most conservative available rule (scoring on the last listed category instead would raise that annotator's agreement with the pipeline from $\kappa = 0.712$ to 0.838, and counting a match whenever any listed category is correct would raise raw agreement to 87.3%). We report Cohen's κ, the conventional chance-corrected agreement statistic for two raters, with binomial intervals on the raw agreement proportion rather than on κ. The pipeline agrees with the first annotator at $\kappa = 0.694$ ($84/118$, $[0.624, 0.786]$), with the second at $\kappa = 0.712$ ($86/118$, $[0.642, 0.801]$), and the annotators agree with each other at $\kappa = 0.765$ ($92/118$, $[0.697, 0.845]$). The separately scored 40-instance OSWorld-2 extension gives $\kappa = 0.413$ ($18/40$) and 0.544 ($23/40$) against the pipeline but 0.807 ($33/40$) between the annotators—the strongest inter-annotator agreement anywhere in the study, so the divergence there is the pipeline's rather than the taxonomy's. Pooling both packages ($n = 158$) gives 0.624, 0.671, and 0.777. All figures are recomputed directly from the annotation files by the released recomputation script. We also ran an auxiliary LLM ensemble as an additional diagnostic baseline; it is not counted as human validation. Ensemble outputs, human labels, pipeline labels, and any adjudicated resolution are stored as separate fields.

D.7 Data-quality audit and correction

Manual audits of construction-era candidate failures found upstream environment and harness artifacts recorded as agent failures. In an initial sample, 32.9% of recorded failures (95% CI $[25.6, 41.0]$) stemmed from environment or harness defects, reaching 49.2% ($[36.8, 61.6]$) in web environments versus 5–15% in OS environments. Checking all $6{,}154$ agent-attributed candidates across the five construction-era populations against the taxonomy's agent-attributed precedent identified 492 misattributed instances (8.0%), which were relabeled rather than discarded. Separately, of 346 candidates flagged by network-interception signals, only 48 (13.9%) were genuine completed actions. With all corrections applied, reliability rises slightly (ClawBench $\alpha = 0.804$, $n = 2{,}162$; aggregate $\alpha = 0.833$, $n = 5{,}663$, from 0.829 before correction), consistent with the corrections improving label quality.

This $5{,}663$-candidate aggregate spans the five construction-era populations of Table 13, a broader population than the $n = 1{,}907$ figure in Section 6.3, which is the frozen classifier's inter-pass reliability on the construction corpus alone. We report them separately and never pool them.

D.8 Annotation record and provenance

Each released annotation record joins to execution by an immutable attempt ID and carries the benchmark, task, agent, model, and schema identifiers; the judge model, prompt, and sampling parameters; pass-level and aggregate attribution and category labels with their evidence; and links to any human or ensemble review, stored alongside rather than over the automated output. A post-freeze taxonomy correction requires a new version and a full reclassification run, never a silent edit to the cases that motivated it.

D.9 Per-population category reliability

Table 15 gives the category-level Krippendorff's α for each of the 28 held-out populations summarized in Section 6.3 and plotted in Figure 6.

D.10 Counting-unit sensitivity

D.11 Attribution reliability and the agent/environment boundary

Attribution reliability is measured by treating a trace's k diagnosis samples as raters over that trace. The image-informed pass—the one production uses wherever it exists—assigns a unanimous attribution to 73.6% of traces, with $\alpha = 0.57$ on ClawBench, 0.69 on OSWorld-1, and 0.57 on OSWorld-2. All three sit below the conventional 0.80 threshold.

Two things qualify that. First, α is depressed by how concentrated the labels are: on OSWorld-2 the image-informed pass is unanimous on 91.9% of traces yet scores $\alpha = 0.566$, because 93.5% of its

labels are `agent` and chance agreement is correspondingly high. Unanimity and α should be read together. Second, and against expectation, visual evidence does not make attribution more stable. On the $2{,}494$ traces carrying both passes, unanimity *falls* from 77.8% under the text-only pass to 73.6% once screenshots are attached, and on ClawBench from 75.2% to 68.4% with α roughly flat (0.596 to 0.572). Screenshots change *which* component is blamed—systematically, and away from the agent—without making the judge more certain. The text-only pass is the more self-consistent of the two and the less accurate one.

The boundary is specified but applied unevenly. Category 03 covers being blocked by CAPTCHAs, WAF access-denied pages and bot-behaviour heuristics, with worked examples defined by how the agent responds—halting on first encounter, abandoning the target site for a substitute, burning the step budget on bypass attempts, or trying and being blocked on every attempt. The diagnosis prompt states the same rule and instructs the judge to default to `agent`, reserving `harness_or_environment` for mechanisms that were themselves broken. Among traces whose samples disagree, `agent` versus `harness_or_environment` accounts for 56.6% of all disagreement. On a hand-checked set of six traces blocked by an unambiguous anti-automation system (reCAPTCHA on two sites, an Amazon image challenge, an Imperva IP-level block, a Cloudflare challenge, and a CAPTCHA-gated submission), the judge assigned `agent` to three and `harness_or_environment` to the other three, with no distinguishing feature in the evidence. External blockers are cited in the evidence of 173 of $1{,}454$ agent-attributed traces (11.9%), concentrated on live web (21–36% of ClawBench's agent-attributed traces against 0–2% on OSWorld).

Mixed-regime traces. Each of a trace's k passes escalates to screenshots independently, only if that pass nominated a suspect step with indexed imagery, so a trace's votes need not all come from the same evidence regime: 47.7% of traces are decided entirely by image-informed passes, 36.0% entirely by text-only passes, and 16.3% by a mixture. Since the two regimes disagree systematically rather than randomly, a mixed trace's verdict depends partly on how many of its passes happened to escalate. Scoring the traces that carry at least one image-informed pass under a consistent image-informed rule instead changes the attribution of 212 of $2{,}859$ traces (7.4%). We report the production rule as specified and record that 7.4% as the sensitivity of attribution to this choice.

D.12 Diagnosis groundedness

Stage 3 classifies each candidate from its generated failure description alone. A post-hoc audit of all 183 validation-set diagnoses and a 60-trace random corpus sample found that 22% name a causal mechanism more specific than the cited evidence strictly supports—most often asserting a particular UI or tooling mechanism where the trace establishes only the resulting outcome. Reclassifying corrected descriptions changed the category for 51% of affected candidates, against a 22% test–retest baseline measured on unchanged text, implying roughly 6% of categorized candidates and 4% of family assignments are sensitive to this. The effect on aggregate shares is diffuse and partly self-cancelling, and does not approach the 35-point margin separating the largest family from the second largest.

Measured end to end—independently diagnosing and then classifying the same trace—agreement is $\alpha = 0.60$ pooled over $1{,}265$ traces (54% exact category agreement), which is the appropriate figure for the full pipeline rather than for its final stage alone. Attribution reliability is lower still: no benchmark reaches 0.80 under either pass. The text-only pass ranges from $\alpha = 0.692$ to 0.736, and the image-informed pass from 0.566 to 0.689. Part of that gap is structural, since attribution has six labels against eighteen with one label dominant, so chance agreement is high. We do not claim the attribution labels are as settled as the category labels, and note that the pass with the higher self-consistency is also the one that assigns more failures to the agent, so self-consistency here is not evidence of correctness.

Stability of the free-text fields. Holding the trace fixed and re-running the image-informed pass three times, the trace-level attribution is identical in 80% of cases when each repetition selects its own screenshots and 90% when every repetition is given the same mechanically derived set, on a random sample of 39 traces. Over the same repetitions the diagnosed *root-cause step* agrees on only 48% and 38% of traces respectively, and the generated `proposed_name` is softer still: no trace received the same name in all three repetitions under either arm. An earlier run on 14 traces deliberately selected

as the hardest cases in the corpus put attribution at 79% against 77% and root-cause agreement at 36% against 67%. At these sample sizes neither difference is resolvable, and we draw only the conclusion both support: attribution is reproducible at roughly four traces in five, root-cause localization at roughly two in five, and the proposed name never. None of these free-text fields enters a reported statistic, but they should not be read as stable descriptions of a trace.

D.13 Worked examples at the category boundaries

The distribution tables say how often each label is used but not what applying it looks like. The five traces below are drawn from the held-out corpus and were also used in a blind attribution study, so each has an independent reading alongside the pipeline's. They are chosen to sit on the boundaries that Section 6.1 identifies as hardest.

An unambiguous agent failure. On a ClawBench appointment-booking task the agent spent roughly fifty steps alternating between re-extracting the same PDF and clicking around a mail tab, never opened the requirements email, and terminated having booked nothing. Nothing external intervened and the site behaved normally; the failure is entirely in the agent's control flow. This is the shape most `agent` verdicts take—not a wrong click, but a loop that never converges.

An unambiguous harness failure. On a ClawBench retail task the trace ends abruptly with `terminated_by: harness_error` immediately after the first add-to-cart click, with the second item never attempted. The agent was mid-task and making orderly progress; the run was cut off by infrastructure. `harness_or_environment` is reserved for exactly this—the episode ended for reasons no agent action could have altered.

An evaluator false negative. On a Ticketmaster pricing task the agent opened the event page, set the quantity to two, and read back the price tiers—the task as instructed. The evaluator's interceptor requires a URL matching `-tickets-`, a segment that the playoff event URLs on this site never contain, so the check could not fire however well the agent performed. Two traces in our sample share this signature, which makes it a property of the instrumentation rather than of either agent.

The anti-automation boundary, decided against intuition. On a Cineplex ticketing task every path returned an Imperva "Access Denied — Error 16" page, an IP-reputation block with no challenge to solve. The intuitive reading is `harness_or_environment`: nothing the agent could type or click clears an IP ban. The frozen scheme nonetheless files anti-automation blocks as *agent* failures, classified by the agent's response (category 03), and reserves `harness_or_environment` for mechanisms that were themselves broken. We record the label the scheme dictates. This is the single boundary where the convention most often surprises readers, and it accounts for a large share of the disagreement reported in Section 6.1.

A case the scheme does not cleanly resolve. On a job-search task the target site had been sunset and redirects to a different company, so the task is not achievable as written. The evidence supports `harness_or_environment` (an unreachable resource), `task_underspecified` (not achievable as specified), and—because the agent stopped to ask the user rather than acting—`agent`. We report such cases under the rule the definitions give rather than adjudicating them individually, and note them here because the taxonomy's boundaries are genuinely indistinct in this corner rather than merely applied inconsistently.

Table 6: Family-level failure counts per agent–model–benchmark population (held-out corpus). Columns follow family order 01–06 (Table 1): Incomplete/premature termination, Grounding/targeting, Data/value errors, Method/output structure, External blockers, Process inefficiency.

Population	n	01	02	03	04	05	06
OSWorld-1 Qwen 3.6 Flash / Claude Code	261	110	39	38	43	2	29
OSWorld-1 Qwen 3.6 Flash / Hermes	273	110	37	45	56	7	18
OSWorld-1 Qwen 3.6 Flash / OpenClaw	276	115	50	33	34	8	36
OSWorld-1 Qwen 3.6 Flash / Qwen Code	241	114	27	21	33	4	42
OSWorld-1 GLM 5.3 Flash / Claude Code	41	23	4	5	8	0	1
OSWorld-1 GLM 5.3 Flash / Hermes	85	50	4	7	14	3	7
OSWorld-1 GLM 5.3 Flash / OpenClaw	141	98	13	9	19	2	0
OSWorld-1 GLM 5.3 Flash / Qwen Code	107	75	7	4	14	5	2
OSWorld-2 Qwen 3.6 Flash / Claude Code	32	10	0	2	18	2	0
OSWorld-2 Qwen 3.6 Flash / Hermes	41	19	0	2	10	7	3
OSWorld-2 Qwen 3.6 Flash / OpenClaw	48	24	1	3	6	0	14
OSWorld-2 Qwen 3.6 Flash / Qwen Code	36	20	6	1	1	0	8
OSWorld-2 GLM 5.3 Flash / Claude Code	10	5	1	1	0	0	3
OSWorld-2 GLM 5.3 Flash / Hermes	5	0	0	3	2	0	0
OSWorld-2 GLM 5.3 Flash / OpenClaw	69	54	0	3	2	0	10
OSWorld-2 GLM 5.3 Flash / Qwen Code	41	30	0	2	2	0	7
ClawBench-1 GLM 5.3 Flash / Hermes	184	78	18	5	2	32	49
ClawBench-1 GLM 5.3 Flash / OpenClaw	199	123	2	5	0	33	36
ClawBench-1 GLM 5.3 Flash / Qwen Code	126	73	5	4	0	27	17
ClawBench-1 Qwen 3.6 Flash / Hermes	201	123	26	4	3	14	31
ClawBench-1 Qwen 3.6 Flash / OpenClaw	264	161	10	9	0	29	55
ClawBench-1 Qwen 3.6 Flash / Qwen Code	236	111	21	8	0	40	56
ClawBench-2 GLM 5.3 Flash / Hermes	133	61	3	2	1	26	40
ClawBench-2 GLM 5.3 Flash / OpenClaw	181	82	6	8	0	19	66
ClawBench-2 GLM 5.3 Flash / Qwen Code	110	61	0	2	0	34	13
ClawBench-2 Qwen 3.6 Flash / Hermes	162	96	11	6	5	14	30
ClawBench-2 Qwen 3.6 Flash / OpenClaw	191	103	15	8	3	23	39
ClawBench-2 Qwen 3.6 Flash / Qwen Code	169	79	12	5	1	27	45

Table 7: Category counts per population, OSWorld-1 (IDs follow Table 1).

Population	n	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18
Qwen 3.6 Flash / Claude Code	261	19	12	0	5	29	29	5	9	5	25	2	3	33	7	19	14	2	43
Qwen 3.6 Flash / Hermes	273	16	10	6	14	18	14	16	10	3	27	1	10	38	8	16	28	0	38
Qwen 3.6 Flash / OpenClaw	276	24	18	5	4	36	18	4	8	3	32	3	4	30	0	18	31	0	38
Qwen 3.6 Flash / Qwen Code	241	14	13	4	8	42	30	6	4	0	13	0	7	25	1	11	12	1	50
GLM 5.3 Flash / Claude Code	41	12	1	0	0	1	3	0	5	0	3	0	0	8	0	0	3	0	5
GLM 5.3 Flash / Hermes	85	9	2	3	1	7	0	2	2	0	2	0	0	14	0	3	4	0	36
GLM 5.3 Flash / OpenClaw	141	13	1	2	5	0	15	2	5	0	12	0	4	12	3	2	6	0	59
GLM 5.3 Flash / Qwen Code	107	2	0	5	0	2	3	0	2	0	7	0	2	10	2	2	4	0	66

Table 8: Category counts per population, OSWorld-2 (IDs follow Table 1).

Population	n	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18
Qwen 3.6 Flash / Claude Code	32	2	0	0	3	0	2	0	2	0	0	2	6	10	2	0	0	0	3
Qwen 3.6 Flash / Hermes	41	3	0	0	3	3	3	0	1	0	0	7	0	8	2	1	8	0	2
Qwen 3.6 Flash / OpenClaw	48	10	0	0	5	14	2	0	1	0	0	0	1	5	0	2	2	1	5
Qwen 3.6 Flash / Qwen Code	36	5	4	0	2	8	0	0	1	0	0	0	0	0	1	0	0	2	13
GLM 5.3 Flash / Claude Code	10	0	1	0	1	3	4	0	1	0	0	0	0	0	0	0	0	0	0
GLM 5.3 Flash / Hermes	5	0	0	0	0	0	0	0	3	0	0	0	2	0	0	0	0	0	0
GLM 5.3 Flash / OpenClaw	69	3	0	0	0	10	1	0	2	0	0	0	2	0	0	1	0	0	50
GLM 5.3 Flash / Qwen Code	41	1	0	0	2	7	0	0	1	0	0	0	0	2	0	1	0	0	27

Table 9: Category counts per population, ClawBench-1/2 (IDs follow Table 1).

Population	n	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18
CB-1 GLM 5.3 Flash / Hermes	184	31	9	32	3	49	2	0	0	0	6	0	0	2	0	5	22	3	20
CB-1 GLM 5.3 Flash / OpenClaw	199	50	1	33	3	36	3	0	0	3	1	0	0	0	0	2	40	0	27
CB-1 GLM 5.3 Flash / Qwen Code	126	21	1	27	3	17	1	0	0	2	4	0	0	0	0	2	41	0	7
CB-1 Qwen 3.6 Flash / Hermes	201	29	14	14	5	31	6	0	0	0	7	0	1	2	0	4	51	5	32
CB-1 Qwen 3.6 Flash / OpenClaw	264	64	3	29	15	55	5	0	0	2	7	0	0	0	0	7	37	0	40
CB-1 Qwen 3.6 Flash / Qwen Code	236	44	9	40	2	56	9	0	0	2	9	0	0	0	0	6	22	3	34
CB-2 GLM 5.3 Flash / Hermes	133	25	2	26	4	40	0	0	0	1	1	0	0	1	0	1	16	0	16
CB-2 GLM 5.3 Flash / OpenClaw	181	28	2	19	6	66	3	1	0	2	4	0	0	0	0	5	17	0	28
CB-2 GLM 5.3 Flash / Qwen Code	110	19	0	34	4	13	1	0	0	0	0	0	0	0	0	2	13	0	24
CB-2 Qwen 3.6 Flash / Hermes	162	17	5	14	4	30	9	0	0	0	6	0	0	5	0	6	32	0	34
CB-2 Qwen 3.6 Flash / OpenClaw	191	20	8	23	7	39	8	0	2	0	7	0	0	3	0	6	34	0	34
CB-2 Qwen 3.6 Flash / Qwen Code	169	16	2	27	7	45	1	0	1	1	10	0	0	1	0	3	23	0	32

Table 10: Closed-set causal-attribution labels assigned in Stage 2.

Label	Operational meaning
`agent`	The trace supports an error in the agent's reasoning, grounding, action, data use, method, verification, or termination behavior.
`evaluator_false_negative`	The trace supports task completion, but the benchmark evaluator reports failure.
`harness_or_environment`	The failure is primarily caused by the execution harness, unavailable or malfunctioning environment state, website behavior, a dependency, or infrastructure.
`task_underspecified`	The instruction or success condition does not determine a unique, verifiable requirement strongly enough to assign a cause.
`undiagnosable`	The trace and evaluator evidence are insufficient to identify the responsible component.
`not_a_failure`	The episode should not have entered the failure set, or the evidence establishes that no relevant failure occurred.

Table 11: Causal attribution of benchmark-reported failures, pooled (% of diagnosed failures, trace-level majority vote over $k = 3$ passes; 95% Wilson CIs for the two largest sources). Benchmark rows pool that benchmark's populations; backbone rows pool all benchmarks.

	n	Agent	Harness/Env.	Eval. FN	Underspec.	Undiag.	Not a failure
OSWorld-1	1,012	50.1 [47,53]	4.7 [4,6]	7.3	2.6	10.4	24.9
OSWorld-2	152	61.8 [54,69]	2.0 [1,6]	9.2	0.0	0.0	27.0
ClawBench-1	886	45.0 [42,48]	43.5 [40,47]	0.3	4.4	0.1	6.7
ClawBench-2	707	44.3 [41,48]	46.5 [43,50]	0.7	1.6	0.3	6.7
Qwen 3.6 Flash	1,492	55.6 [53,58]	24.8 [23,27]	2.1	2.2	4.1	11.2
GLM 5.3 Flash	1,265	38.2 [36,41]	31.2 [29,34]	5.1	3.4	3.7	18.3
All	2,757	47.6 [46,49]	27.7 [26,29]	3.5	2.8	3.9	14.5

Table 12: Attribution distribution per held-out population (% of diagnosed failures; 95% Wilson CIs for agent and harness/environment). Populations are never pooled across scaffold, backbone, or benchmark.

Benchmark	Backbone	Scaffold	n	Agent	Harness/Env.	Eval. FN	Underspec.	Undiag.	Not a failure
OSWorld-1	Qwen 3.6 Flash	Claude Code	152	61.2 [53,69]	2.0 [1,6]	3.3	4.0	12.5	17.1
OSWorld-1	Qwen 3.6 Flash	Hermes	136	71.3 [63,78]	5.1 [3,10]	2.9	2.2	8.8	9.6
OSWorld-1	Qwen 3.6 Flash	OpenClaw	170	58.2 [51,65]	5.9 [3,10]	2.9	1.2	10.6	21.2
OSWorld-1	Qwen 3.6 Flash	Qwen Code	142	59.2 [51,67]	2.1 [1,6]	6.3	2.8	7.8	21.8
OSWorld-1	GLM 5.3 Flash	Claude Code	97	19.6 [13,29]	8.2 [4,15]	18.6	4.1	10.3	39.2
OSWorld-1	GLM 5.3 Flash	Hermes	102	27.5 [20,37]	6.9 [3,13]	8.8	2.0	7.8	47.1
OSWorld-1	GLM 5.3 Flash	OpenClaw	114	43.9 [35,53]	5.3 [2,11]	12.3	2.6	15.8	20.2
OSWorld-1	GLM 5.3 Flash	Qwen Code	99	37.4 [28,47]	4.0 [2,10]	10.1	2.0	9.1	37.4
OSWorld-2	Qwen 3.6 Flash	Claude Code	18	66.7 [44,84]	0.0 [0,18]	22.2	0.0	0.0	11.1
OSWorld-2	Qwen 3.6 Flash	Hermes	25	56.0 [37,73]	4.0 [1,20]	8.0	0.0	0.0	32.0
OSWorld-2	Qwen 3.6 Flash	OpenClaw	21	71.4 [50,86]	4.8 [1,23]	0.0	0.0	0.0	23.8
OSWorld-2	Qwen 3.6 Flash	Qwen Code	15	80.0 [55,93]	0.0 [0,20]	0.0	0.0	0.0	20.0
OSWorld-2	GLM 5.3 Flash	Claude Code	14	21.4 [8,48]	7.1 [1,31]	28.6	0.0	0.0	42.9
OSWorld-2	GLM 5.3 Flash	Hermes	15	6.7 [1,30]	0.0 [0,20]	13.3	0.0	0.0	80.0
OSWorld-2	GLM 5.3 Flash	OpenClaw	24	95.8 [80,99]	0.0 [0,14]	4.2	0.0	0.0	0.0
OSWorld-2	GLM 5.3 Flash	Qwen Code	20	70.0 [48,85]	0.0 [0,16]	5.0	0.0	0.0	25.0
ClawBench-1	Qwen 3.6 Flash	Hermes	149	47.0 [39,55]	45.6 [38,54]	0.0	3.4	0.0	4.0
ClawBench-1	Qwen 3.6 Flash	OpenClaw	151	57.0 [49,65]	37.1 [30,45]	0.0	2.6	0.7	2.6
ClawBench-1	Qwen 3.6 Flash	Qwen Code	149	51.0 [43,59]	39.6 [32,48]	0.0	2.7	0.0	6.7
ClawBench-1	GLM 5.3 Flash	Hermes	148	41.9 [34,50]	44.6 [37,53]	0.7	4.0	0.0	8.8
ClawBench-1	GLM 5.3 Flash	OpenClaw	142	44.4 [36,53]	43.0 [35,51]	0.0	4.2	0.0	8.4
ClawBench-1	GLM 5.3 Flash	Qwen Code	147	28.6 [22,36]	51.0 [43,59]	1.4	9.5	0.0	9.5
ClawBench-2	Qwen 3.6 Flash	Hermes	122	45.1 [37,54]	48.4 [40,57]	0.0	0.0	0.0	6.6
ClawBench-2	Qwen 3.6 Flash	OpenClaw	122	51.6 [43,60]	41.8 [33,51]	0.0	1.6	0.0	4.9
ClawBench-2	Qwen 3.6 Flash	Qwen Code	120	45.0 [36,54]	43.3 [35,52]	1.7	2.5	0.0	7.5
ClawBench-2	GLM 5.3 Flash	Hermes	114	38.6 [30,48]	51.8 [43,61]	0.0	0.9	0.9	7.9
ClawBench-2	GLM 5.3 Flash	OpenClaw	116	52.6 [44,61]	36.2 [28,45]	0.9	1.7	0.9	7.8
ClawBench-2	GLM 5.3 Flash	Qwen Code	113	31.9 [24,41]	58.4 [49,67]	1.8	2.6	0.0	5.3

Table 13: Attribution distribution for construction-era populations (% of diagnosed failures; 95% Wilson CIs for agent and harness/environment).

Population	n	Agent	Harness/Env.	Eval. FN	Underspec.	Undiag.	Not a failure
OSWorld-1	722	25.1 [22,28]	1.5 [1,3]	66.3	0.3	0.0	6.8
OSWorld-2	317	81.1 [76,85]	3.2 [2,6]	11.7	0.3	0.0	3.8
ClawBench, stage 1	156	19.9 [14,27]	30.1 [23,38]	44.2	2.6	0.0	3.2
ClawBench, stage 2	156	32.1 [25,40]	39.7 [32,48]	25.0	1.3	0.6	1.3
ClawBench, merged (two-stage AND)	137	56.2 [48,64]	30.7 [24,39]	4.4	2.9	0.0	5.8

Table 14: Net attribution shift induced by tool-call normalization beyond matched judge resampling noise, per test group.

Test group	Net shift (points)	Standard error	n
Primary	11.19	4.06	134
Replication 1	26.52	4.86	132
Replication 2	16.39	4.81	122
Replication 3	17.27	4.54	120
Replication 4	5.83	4.61	120
Pooled	15.3 [5.8, 24.7]		628

Table 15: Category-level Krippendorff's α for every held-out population (n: classified failure candidates). ClawBench has no Claude Code cell. *Cells with $n \leq 11$ have wide confidence intervals and are not individually interpreted.

Benchmark	Backbone	Claude Code	Hermes	OpenClaw	Qwen Code
OSWorld-1	Qwen 3.6 Flash	0.847 (257)	0.821 (273)	0.818 (276)	0.852 (239)
	GLM 5.3 Flash	0.790 (41)	0.844 (86)	0.850 (142)	0.915 (106)
OSWorld-2	Qwen 3.6 Flash	0.888 (32)	0.831 (41)	0.835 (48)	0.816 (37)
	GLM 5.3 Flash	0.842* (11)	1.000* (4)	0.827 (69)	0.816 (41)
ClawBench-1	Qwen 3.6 Flash	–	0.779 (203)	0.783 (264)	0.795 (238)
	GLM 5.3 Flash	–	0.819 (186)	0.880 (199)	0.857 (127)
ClawBench-2	Qwen 3.6 Flash	–	0.826 (163)	0.769 (191)	0.764 (169)
	GLM 5.3 Flash	–	0.808 (133)	0.782 (182)	0.780 (110)

Table 16: Sensitivity of the reported distribution to the counting unit. *Candidate* is the unit used throughout the paper: each of a trace's k diagnosis samples yields its own classification candidate. *Trace* collapses those siblings to one row per trace by majority vote (12.0% of traces have no sibling majority and are resolved alphabetically as a last resort). The candidate total here is marginally below the 3,863 of Table 1 because this analysis re-applies the current trace-level attribution filter to every candidate before counting. Families are stable; among categories only the top two exchange rank.

	Candidate	Trace	Δ
n	3,859	1,619	2.38×
F1 Incomplete or premature termination	52.0	48.7	−3.4
F6 Process inefficiency	17.0	18.5	+1.5
F5 External blockers	9.3	11.0	+1.7
F2 Grounding and targeting	8.2	8.7	+0.5
F4 Method and output structure	7.1	6.9	−0.3
F3 Data and value errors	6.3	6.2	−0.1
18 Preparatory phase mistaken for completion	19.8	15.4	−4.4
05 Step-budget exhaustion	17.0	18.5	+1.5
01 Final action omitted	12.9	13.4	+0.5
16 Premature abandonment of recoverable obstacle	11.9	12.4	+0.5
03 Bot-detection challenge	8.9	10.6	+1.7
13 Method violates structural constraint	5.4	5.3	−0.2