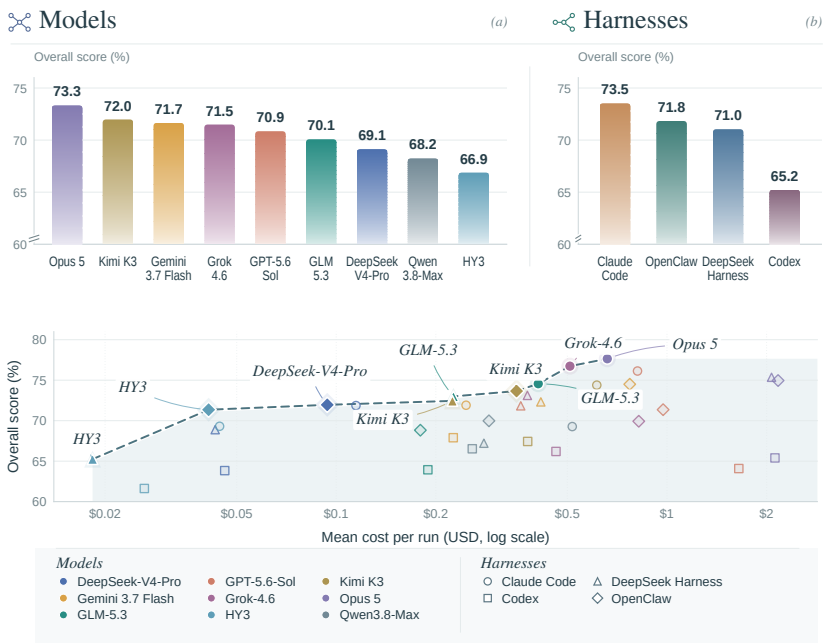
TRIAL BY TRACE: BENCHMARKING AI AGENT HARNESSES FOR RELIABILITY AND FAILURE ATTRIBUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents are increasingly used to solve real-world problems, and their effectiveness depends on both the underlying models and the harnesses that support their operation. However, most existing benchmarks focus on model capabilities and overall agent performance, with limited attention to the evaluation of agent harnesses. To address this gap, we introduce **TraceTrial-Bench**, which organizes harness reliability into seven components covering context, execution, governance, lifecycle, observability, tools, and verification. We construct 95 expert-level, mechanism-oriented tasks with executable or state-based ground truth to support semantically equivalent comparisons while preserving the native behavior of each harness. We design a **Trajectory Tracing (TT) module** to capture agent execution traces and introduce **a two-layer evaluation framework** that separately assesses agent performance and harness reliability. In addition, we design a **Multi-source Evidence Acquisition (MEA) module** to provide evidence from the agent context, harness, and execution environment for LLM-based evaluation, and develop a **Failure Attribution (FA) module** that uses explicit mechanism chains to support attribution at the run level. Across 4 harnesses, 9 models, and 95 tasks, our evaluation of 3,420 runs yields macro-averaged harness scores ranging from 59.8% to 72.0%, while 31.1% of runs achieve task success despite failing at least one harness criterion. TraceTrial-Bench provides an auditable and reproducible foundation for comparing agent harnesses, diagnosing their failures, and studying how harness design interacts with model behavior. Code is available at: `https://anonymous.4open.science/r/Trace_anoy-F901`.



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.

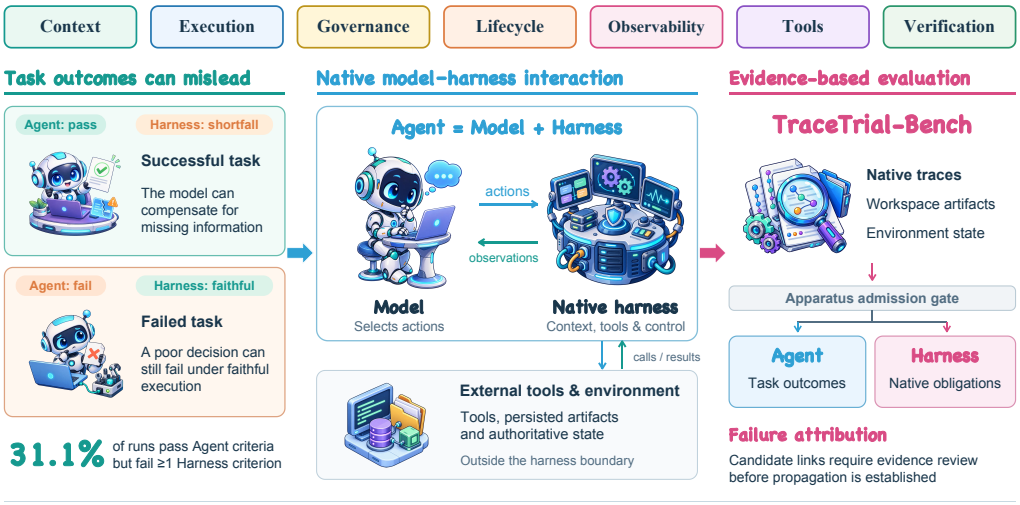


Figure 1: **Background and motivation.** Agent performance depends on both the model and its harness, but task outcomes alone cannot distinguish model errors from harness shortfalls. TraceTrial-Bench uses native execution traces, artifacts, and environment state to assess agent performance and harness reliability separately and support evidence-based failure attribution.

1 Introduction

Large language models increasingly support agents that use tools, execute commands, modify files, and interact with external environments (Wang et al., 2024; Yao et al., 2022; Schick et al., 2023; Wang et al., 2025). Their behavior depends on more than the underlying model: instructions, context management, tool interfaces, persistent state, and execution controls determine which information reaches the model and how its decisions affect the environment (Sumers et al., 2023; Packer et al., 2023; Wu et al., 2023; Mei et al., 2024). The *agent harness* connects these components. Evaluating its reliability is therefore necessary for understanding the complete agent system (Li et al., 2026; Ning et al., 2026).

Most agent benchmarks measure whether a model or complete system finishes a task, using final answers, tests, or environment states (Merrill et al., 2026; Deng et al., 2025; Barres et al., 2025). Such outcomes combine model capability with harness behavior. As Figure 1 illustrates, a successful task can conceal lost information, misleading tool results, or lifecycle violations; a failed task can occur even when the harness faithfully executes the model's decisions. Outcome scores alone cannot localize these differences to context management, tool communication, or execution control (Barke et al., 2026; Liu et al., 2026). Recent harness evaluations examine configuration effects and safety (Yao et al., 2026; Bai et al., 2026), motivating complementary tests of specific native harness obligations.

Therefore, we introduce **TraceTrial-Bench**, which follows the principle of "Trial by Trace": assessing harness behavior through auditable execution evidence and external state. Building on the harness taxonomy (Li et al., 2026), we organize reliability into seven components: context, execution, governance, lifecycle, observability, tools, and verification. The benchmark contains 95 mechanism-oriented tasks with executable or state-based ground truth. Cross-harness adapters preserve native interfaces and execution behavior while maintaining equivalent task semantics.

Specifically, the Trajectory Tracing (TT) module reconstructs native execution records, and the Multi-source Evidence Acquisition (MEA) module assembles criterion-specific evidence from the agent context, harness, and environment. Two scored layers separately assess Agent outcomes and Harness obligations after measurement-apparatus checks. The Failure Attribution (FA) module connects failed criteria through predeclared mechanism chains, distinguishing automatic candidate activation from evidence-validated propagation. Across four harnesses and nine models, the 3,420 selected runs show different leaders for task success and harness satisfaction, demonstrating the value of reporting both.

The distinction matters for both diagnosis and improvement. A low task score alone cannot tell a harness developer whether to change context handling, tool communication, or execution control. Conversely, a high task score can reward a model that compensates for an unreliable interface without revealing that the interface failed its contract. Recording the model-visible information together with independent environment state makes these cases inspectable. TraceTrial-Bench therefore retains the evidence supporting each criterion verdict and separates measured shortfalls from stronger claims about defects and their consequences.

Our contributions are: (i) a seven-component, 95-task benchmark targeting native harness mechanisms; (ii) a provenance-preserving tracing and evidence pipeline supporting separate Agent and Harness assessment; and (iii) an auditable attribution framework and a complete model–harness comparison with released tasks, criteria, configurations, and traces. The present study contains one selected execution per task–model–harness cell; it characterizes observed behavior rather than repeated-run stability.

2 Related Works

Agent harnesses. Harness research addresses context, tools, state, verification, and permissions (Zhong & Zhu, 2026; Barbaste et al., 2026). ACE, ACDL, and SkillAlign improve context and interfaces (Zhang et al., 2026; Peleg Pelc et al., 2026; Ren et al., 2026); LongHorizon-Harness and Crab address continuity and recovery (Ma et al., 2026; Wu et al., 2026); AgentBound and SafeHarness enforce policies and safeguards (Bühler et al., 2026; Lin et al., 2026b). HARBOR, retrospective optimization, and SHAPER optimize configurations, skills, or harness code (Sengupta & Wang, 2026; Pan et al., 2026; Wang et al., 2026a), while HarnessFix diagnoses and patches implementation flaws (Chen et al., 2026). Interpreting improvements requires matched budgets and held-out evaluation (Wang et al., 2026b). TraceTrial-Bench supplies mechanism-level criteria and evidence while preserving native harness behavior.

Configuration search, skill evolution, and implementation repair target different system components. TraceTrial-Bench preserves native execution strategies and records obligation-specific evidence to examine whether task-level gains coincide with more reliable harness behavior.

Agent evaluation and failure analysis. Terminal-Bench and SWE-Bench Pro assess verifiable task outcomes (Merrill et al., 2026; Deng et al., 2025). Harness-Bench compares model–harness configurations under shared conditions (Yao et al., 2026); HarnessAudit and HarnessRisk examine execution fidelity and adversarial risks (Liu et al., 2026; Bai et al., 2026). TraceTrial-Bench complements these evaluations with separate Agent and Harness criteria, exposing successful deliveries that coexist with harness shortfalls.

Who&When localizes responsible agents and error steps (Zhang et al., 2025); AgentRx diagnoses constraint violations (Barke et al., 2026); REFLECT uses interventions and controlled replay (Lin et al., 2026a). Our attribution framework connects failed criteria through declared mechanisms and source-specific evidence. Automatically activated links identify candidates for review rather than establish propagation.

3 Preliminaries

Agent and harness boundary. We use $\mathrm{Agent} = \mathrm{Model} + \mathrm{Harness}$. The model selects actions; the harness constructs context, exposes tools, transports calls and results, maintains state, and enforces execution and permission controls (Wei et al., 2026; Zhong & Zhu, 2026). External tools, environments, artifacts, evaluators, and measurement apparatus lie outside this boundary (Figure 1). A scaffold describes prompts and workflows; a runtime supplies resources. Our boundary includes their orchestration and control functions, but excludes external tool behavior. Harness reliability concerns faithful information delivery and execution semantics, without guaranteeing task success.

Run and evidence. For harness h, model m, task t, and execution instance r, we record

$$R_{h,m,t,r} = (\tau, \Delta W, G, V_H, V_A, L_{\mathrm{act}}), \tag{1}$$

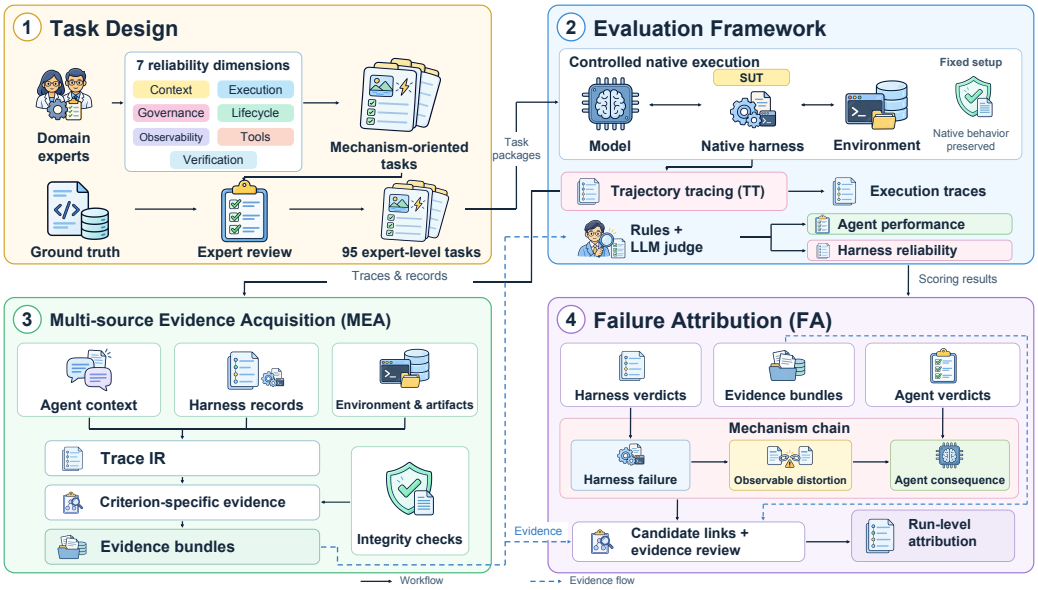


Figure 2: **TraceTrial-Bench overview.** Mechanism-oriented tasks exercise native harnesses; trajectory tracing and multi-source evidence support separate Agent and Harness scores and evidence-based failure attribution. The figure's evidence-extraction stage corresponds to the MEA module.

where τ is the normalized trace, ΔW the workspace change, G the apparatus-admission state, V_H, V_A the Harness and Agent verdict vectors, and L_{act} activated candidate links. Run identity binds the task source, resolved configuration, container image, archive, and artifact digests; the reported population retains one selected execution per cell. Environment state establishes what occurred, artifacts establish persisted deliverables, and native model-facing records establish what the model could observe. Provider-request evidence supports only captured structured tool declarations, not uncaptured request bodies. These sources answer different questions and cannot substitute for one another.

Levels of claim. A criterion verdict reports satisfaction under selected evidence; confirming a harness defect additionally requires reconciliation with authoritative sources. An activated link identifies failed endpoints and a matching activation condition. Establishing propagation further requires evidence of intermediate distortion, consistent event order, and review of competing explanations. Apparatus failure is a measurement limitation, not a harness defect.

4 TRACETRIAL-BENCH

TraceTrial-Bench tests the native harness while controlling task semantics, model configuration, initial state, and declared resource limits. Figure 2 summarizes the design. Detailed contracts and implementation rules appear in Appendices A–E.

4.1 TASK SUITE AND CROSS-HARNESS EXECUTION

The suite covers 40 mechanism-level properties through 95 expert-level tasks: 15 Context, 27 Execution, 9 Governance, 19 Lifecycle, 7 Observability, 17 Tools, and 1 Verification task. Context tests information preservation and isolation; Execution tests process semantics and result delivery; Governance tests permissions and boundaries; Lifecycle tests state transitions and recovery; Observability tests faithful diagnostic channels; Tools tests interface and transport fidelity; Verification tests agreement between completion claims and authoritative outcomes. Tasks are assigned by the obligation being measured, not their application domain.

Table 1: **Task contracts and evidence boundaries.** Each task combines a Harness obligation with an Agent deliverable and an independent truth anchor. The detailed catalog is in Appendix A.3.

Contract element	Fixed before execution	Evaluation role
Task and pressure	Prompt, fixtures, mechanism trigger, and resource limits	Exercise a declared native behavior within a realistic task.
Agent score sheet	Required actions and final deliverables	Check task completion from artifacts and authoritative state.
Harness score sheet	Native obligation, observation boundary, and evidence selector	Check faithful information delivery and execution behavior.
Oracle and provenance	Private state, workspace baseline, and run identity	Separate environment truth from model-facing claims.

Each task specifies a mechanism trigger, expected behavior, observation boundary, and executable or state-based truth anchor. Examples include oversized outputs requiring recovery, nonzero exits paired with success-shaped artifacts, pending jobs, facts crossing session boundaries, and private values available only through an interaction channel. A task package fixes the prompt or turn sequence, workspace, tools, budgets, deliverables, verifier, and two score sheets. Private oracle state and baseline snapshots remain outside the agent's writable workspace. Review checks solvability, mechanism reachability, verifier agreement, hidden leakage, and equivalent semantics. The release includes 46 single-turn, 45 fixed multi-turn, and four simulator-driven tasks; its L3 designation describes mechanism pressure rather than model reasoning intensity.

Adapters for Claude Code, Codex, DeepSeek Harness, and OpenClaw launch each native system and normalize observations without replacing prompt construction, tool transport, context policy, permissions, or session behavior. Normalization may change representation but cannot introduce capabilities, reveal private information, or alter pass conditions. Task-scoped containers provide isolation, while resolved configurations preserve effective budgets and invocation details. Appendix B specifies adapter boundaries and eligibility.

Coverage and measurement boundaries. Mechanism membership requires a reproducible operating condition and an observable verdict boundary. Coverage is tracked separately for task assignment, native capability availability, mechanism activation, and evidence sufficiency. A task can be present in the manifest without every selected execution exercising its target path. Similarly, an unavailable named tool is not automatically a defect if the native harness supplies an equivalent usable path. If an adapter cannot preserve the task's domain semantics or required evidence, the corresponding comparison is non-comparable. Appendix B.4 details this distinction.

4.2 Trajectory Tracing and Multi-source Evidence Acquisition

Trajectory Tracing The TT module collects source-specific event streams and native session records, preserving raw observations that cannot be safely normalized. The Trace Intermediate Representation (Trace IR) supplies a common envelope with schema version, sequence index, timestamp, run identity, and event type. Tool events retain call identifiers, arguments, results, and supported error, exit, duration, and truncation fields. Native identifiers correlate calls and results; ambiguous or unmatched observations remain explicitly unlinked. Parent, session, and turn identifiers preserve execution lineage when exposed by the source.

Multi-source Evidence Acquisition MEA combines these traces with private simulator ledgers, workspace differences, provider-request tool declarations, and execution/security probes. Sources retain provenance and digest bindings. Structured provider declarations document submitted tools, while full request bodies and authorization material are excluded. A native result and an external process outcome can therefore be compared without confusing what occurred with what the model received.

Each semantic criterion declares deterministic selectors over eligible event types, fields, literal conditions, and return limits. Agent criteria primarily select changed artifacts; Harness criteria select trajectories and probes. Bounded evidence packets preserve matching entries, relevant text win-

dows, scanned/matched/returned counts, and truncation indicators. Full archives remain available for audit. Empty required selections are recorded explicitly: failure to establish satisfaction is not independently proof of a defect. Appendix C gives the schema, source precedence, and integrity controls.

4.3 Two-Layer Evaluation

Apparatus admission checks container lifecycle, archive integrity, trace scoreability, applicable probes, and private oracles before scoring. It is a gate, not a third score. Deterministic verifier results are fixed first. Semantic Agent rubrics are then judged individually, followed by Harness rubrics using their own evidence packets. Completed Agent verdicts supply downstream context, without serving as evidence of a harness failure.

The released protocol finalizes each criterion to a binary verdict. Semantic Harness rubrics retain four diagnostic states: unsupported (0), supported (1), not exercised (2), and malfunction (3). States 0 and 3 finalize to failure; state 1 finalizes to pass. State 2 is outcome-conditioned: it passes only when all Agent criteria pass. Every finalized item remains in its layer denominator. This explicit dependence means that the two score sheets are distinct but not statistically independent. The analysis reports operational criterion satisfaction rather than confirmed-defect rates.

For layer $X \in \{A, H\}$, let p_X of n_X items pass. The combined score is

$$S = 50\frac{p_A}{n_A} + 50\frac{p_H}{n_H}. \tag{2}$$

Agent task success requires all Agent criteria to pass; a combined pass requires all criteria in both layers. Macro rates average per-run fractions, while micro rates pool criterion counts. We preserve these measures separately, together with original verdicts, Judge records, and evidence coverage. Appendix D and Appendix E.3 provide full scoring rules.

Aggregation and denominator policy. For an evaluated run population $\mathcal{R}$ with measured Harness items, the two reported Harness rates are

$$M_H^{\text{micro}} = \frac{\sum_{r\in\mathcal{R}} p_{H,r}}{\sum_{r\in\mathcal{R}} n_{H,r}}, \qquad M_H^{\text{macro}} = \frac{1}{|\mathcal{R}|}\sum_{r\in\mathcal{R}} \frac{p_{H,r}}{n_{H,r}}. \tag{3}$$

The micro rate weights each finalized criterion equally; the macro rate weights each run equally. Because task criterion counts differ, both are needed to distinguish broad run-level behavior from the influence of larger score sheets. Related scenarios are not independent repetitions of a mechanism. Reported tables identify their task and run population, and evidence-quality statistics remain separate from performance aggregates.

4.4 Failure Attribution

A task may declare a mechanism chain $\ell = (h_\ell, a_\ell, \mu_\ell)$ connecting a Harness criterion, an Agent criterion, and an intermediate information, action, or state distortion. Let F_H and Z_A be the raw zero-scored Harness and Agent criteria. A candidate activates when both endpoints fail and its declared activation condition holds. Activation identifies an evidence path for review; it does not establish that the proposed distortion occurred.

More precisely, task-specific declarations define the runtime candidate pool $\mathcal{C}_t^{\text{run}}$. With activation predicate $q_\ell(R)$, the automatically recorded set is

$$L_{\text{act}}(R) = \{\ell \in \mathcal{C}_t^{\text{run}} : h_\ell \in F_H,\ a_\ell \in Z_A,\ q_\ell(R) = 1\}. \tag{4}$$

Some tasks do not declare a Harness-to-Agent endpoint pair. Absence of a link therefore does not establish the absence of a harness contribution. Likewise, a run may contain both linked and unlinked Agent failures, so candidate-link categories need not partition all failed criteria.

Manual validation reconciles both endpoint verdicts with their authoritative sources, checks event order and observable intermediate distortion, and considers alternative explanations. Only validated links support propagation claims. Runtime labels and optional bypass adjustments are retained as implementation outputs; the reported experiments use raw verdicts consistently. Crossing the

Table 2: **Aggregate results (%).** All harnesses use 855 matched model–task cells. Overall averages unrounded Agent Score and Harness Macro equally, matching the first-page plots. Agent Pass requires all Agent criteria; Clean requires both layers. Score and Macro average run fractions; Micro pools Harness items. Full results appear in Appendix G.

	Agent		Harness			
Harness	Pass	Score	Micro	Macro	Clean	Overall
Claude Code	**44.0**	**75.9**	71.5	71.2	9.1	**73.5**
Codex	35.8	70.7	60.2	59.8	2.8	65.2
DeepSeek Harness	40.0	73.3	69.5	68.8	9.8	71.0
OpenClaw	38.2	71.7	**71.9**	**72.0**	**11.7**	71.8
All harnesses	39.5	72.9	68.3	67.9	8.4	70.4

empty/nonempty status of F_H and Z_A gives four operational states: jointly clean, Harness-only shortfall, Agent-only shortfall, and joint shortfall. Harness-only shortfall denotes successful delivery despite failed Harness criteria; defect confirmation and causal recovery require additional evidence. The full decision rules appear in Appendix E.3.

5 Experiments

5.1 Experimental Setup

We evaluate Claude Code, Codex, DeepSeek Harness, and OpenClaw with nine model groups: DeepSeek-V4-Pro, Gemini 3.7 Flash, GLM-5.3, GPT-5.6-Sol, Grok-4.6, HY3, Kimi K3, Opus 5, and Qwen3.8-Max. Reasoning settings follow the archived configurations. The September 5, 2026 archive contains 3420 selected executions: 855 per harness and 95 per model–harness block. Every cell has a final combined score and nonempty binary verdicts for both layers; no score is imputed. Selected executions include replacements from recovery, not independent repetitions.

Runs execute in Docker with task-specific wall-time ceilings of 120–5,400 seconds, 1–120 user-input turns, and 5–300 task-tool calls. These are configured limits, not observed counts. All final semantic evaluations use DeepSeek-V4-Flash. Per-criterion output limits vary: 4,096 tokens for 1,966 evaluations, 8,192 for 1,452, and 32,000 for two. Final combined-score files supply the verdicts; root execution-status records are not score authority. Scheduling, recovery, and scoring-budget variation limit causal interpretations of harness differences.

All 3,420 archived final evaluations enter the descriptive analysis once, using stored values; auxiliary flags do not remove cells or adjust raw failures. Agent Score and Harness Macro average run-level pass fractions; Harness Micro pools criterion counts. Agent Pass requires all Agent criteria, while Clean requires both layers to pass fully. Candidate-link incidence is reported separately from validated causality. Resource estimates use recorded token usage and model-specific prices published on the providers' official websites, excluding Judge and discarded-run costs. Protocol, accounting, and reproducibility details appear in Appendix F.

5.2 Overall Outcomes and Harness–Model Interaction

Table 2 reports aggregate outcomes. Claude Code leads Overall Score (73.5%), matching the first-page plots, and Agent task success (44.0%) and Score (75.9%). OpenClaw leads Harness Macro (72.0%), Micro (71.9%), and Clean (11.7%). Harness Macro spans 12.2 percentage points, from Codex (59.8%) to OpenClaw (72.0%), compared with 8.2 points for Agent success. Micro and macro Harness rates produce the same ordering.

The same mismatch also appears within individual models. For Gemini 3.7 Flash, DeepSeek Harness has the highest Agent success (44.2%), whereas OpenClaw has the highest Harness Macro (74.8%). Qwen3.8-Max instead places OpenClaw first on both measures. A single model-independent harness ranking would miss these differences. Fixed-model comparisons retain the

Table 3: **Harness Micro by reliability dimension (%).** All four harnesses use the same tasks and nine models within each row. Coverage is uneven; Verification contains one task. Full pooled and Agent outcomes appear in Table 36.

Dimension	Tasks	Claude Code	Codex	DeepSeek Harness	OpenClaw
Context	15	**74.5**	62.0	64.3	61.6
Execution	27	64.7	51.5	65.3	**77.1**
Governance	9	72.3	74.2	68.8	**80.5**
Lifecycle	19	71.3	53.0	**71.7**	65.9
Observability	7	**88.7**	76.5	86.8	85.0
Tools	17	**71.5**	65.6	**71.5**	70.2
Verification	1	93.7	**95.2**	82.5	88.9

same 95 tasks and therefore expose interactions that are obscured by averaging over all model groups.

Across the population, 14,979/20,016 Agent items (74.8%) and 19,242/28,188 Harness items (68.3%) pass. Mean per-run scores are 72.9% and 67.9%, respectively. All-criteria success is substantially stricter: 1,351 runs (39.5%) pass every Agent item, and 286 (8.4%) pass both layers. The first-page plots summarize model–harness performance and standardized cost; the complete numerical comparison is in Table 35.

Fixed-model comparisons show a recurring harness effect with model-dependent magnitude. Codex has the lowest Harness Macro for every model; OpenClaw leads six groups, and Claude Code leads GLM-5.3, GPT-5.6-Sol, and Grok-4.6. Within-model Harness Macro ranges span 10.0–16.2 points. Agent success varies differently: Grok-4.6 spans 33.7% with OpenClaw to 55.8% with Claude Code, while Opus 5 spans 30.5% with Codex to 53.7% with Claude Code. These contrasts describe the collected configurations without isolating a single design choice.

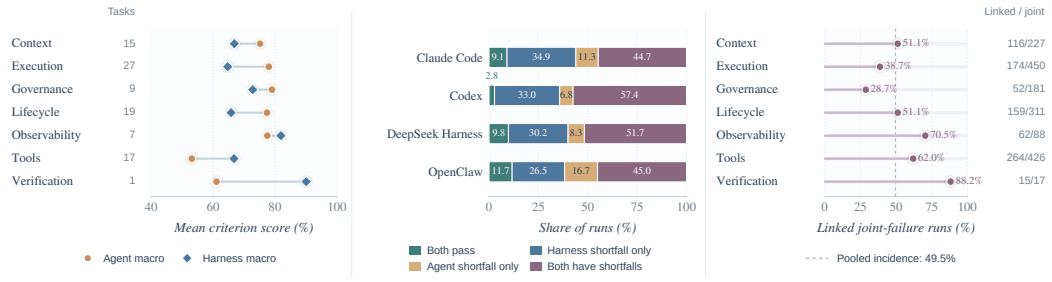
Recorded resource use. At the frozen model-specific prices, mean estimated Agent costs per selected run are $0.436 for Claude Code, $0.595 for Codex, $0.446 for DeepSeek Harness, and $0.633 for OpenClaw. Mean total recorded tokens are 578.8k, 617.2k, 372.7k, and 680.0k, respectively. Token counts alone do not determine normalized cost because model prices and available cache counters differ. These estimates describe recorded usage under the retained executions; they are not historical invoices or a controlled throughput comparison. Judge calls, discarded executions, storage, and tool charges are excluded. The full model-level resource columns and accounting policy remain in the appendix.

5.3 Reliability Dimensions and Failure States

Execution has the lowest pooled Harness Micro (64.6%), followed by Lifecycle (65.5%) and Context (65.6%). Observability reaches 84.3%; Verification reaches 90.1% but contains only one task. Execution also shows a large cross-harness spread: 51.5% for Codex versus 77.1% for OpenClaw. Context changes the aggregate ordering, with Claude Code at 74.5% and OpenClaw at 61.6%. Full per-dimension results appear in Table 36.

Figure 3(a) shows that the layer gap changes direction across components. Execution averages 78.0% Agent Score versus 64.7% Harness Macro, and Lifecycle 77.4% versus 65.8%. Tools reverses this gap (53.2% versus 66.8%). At the property level, subagent result aggregation reaches 72.2% Agent success despite 46.2% Harness satisfaction, while cross-turn fact retention reaches 95.8% versus 48.4%. These observations demonstrate why task success alone cannot characterize execution reliability.

All-criteria Agent success adds another perspective. Execution achieves 49.2% Agent success, the highest among the multi-task dimensions, despite its lowest pooled Harness Micro. Tools reaches 69.7% Harness Micro but only 21.4% Agent success over 612 runs. The single Verification task has 8.3% Agent success despite a 90.1% Harness rate. Neither direction of discrepancy can be



(a) Layer-wise performance (b) Outcome composition (c) Candidate-link incidence

Figure 3: **Reliability and candidate links on the 3,420-run grid.** (a) Mean per-run layer scores by dimension. (b) Four criterion-verdict states over 855 runs per harness. (c) Runs with a candidate link among joint-failure runs, with linked/joint-failure counts and the pooled 49.5% rate. Candidate activation is not validated causality; Verification contains one task.

interpreted from one layer alone: criterion-level fidelity and complete task delivery are different measured requirements.

No one of the 40 properties reaches 100% Harness Micro across all nine models and four harnesses. Network restriction is lowest at 38.9%, but contains one scenario. Tool exposure reaches 45.8% Harness satisfaction and no all-criteria Agent successes across its two scenarios. These narrow populations support targeted diagnosis, while limiting claims about the entire mechanism family. The case studies preserve paired native trajectories so that readers can inspect concrete observations alongside aggregate rates.

The raw criterion intersections contain 286 jointly clean runs, 1,065 Harness-only shortfalls, 369 Agent-only shortfalls, and 1,700 joint shortfalls. Thus 31.1% of runs achieve Agent success despite at least one failed Harness criterion. This is operational absorption, not automatic confirmation of a defect or successful recovery mechanism. Claude Code has the most Harness-only runs; OpenClaw has the most Agent-only and jointly clean runs (Figure 3(b)).

Among the 1,700 joint-shortfall runs, 842 (49.5%) contain at least one stored candidate link, totaling 1,633 activated endpoint pairs. Of 2,069 Agent-failure runs, 1,773 (85.7%) retain at least one failed Agent criterion not reached by a link. Linked and unlinked failures can coexist. Conditional activation is lower for Execution and Governance than for Tools and Observability (Figure 3(c)); these rates do not estimate causal-link precision. Table 37 gives complete counts, and Appendix H presents five paired trajectory studies.

Interpretation limits. One selected execution per cell cannot establish repeated-run stability. Uneven property coverage, source visibility, evidence selection, and Judge-budget variation constrain generalization. Scores and automatically activated links remain operational outputs; controlled interventions and broader validation are needed for stronger causal claims. The current evaluation does not estimate evidence-selector recall, Judge agreement, or candidate-link precision over the full corpus. Evidence review strengthens individual diagnoses but requires expert effort. These validation gaps remain important when using benchmark feedback to guide automated harness repair. Appendix J discusses these limitations and future work.

6 Conclusion

TraceTrial-Bench evaluates native agent harnesses through 95 mechanism-oriented tasks across seven reliability components. Trajectory tracing and multi-source evidence support separate Agent and Harness scores, while explicit mechanism chains guide auditable failure analysis. Across 3,420 selected runs, Harness Macro differs by 12.2 percentage points and the leaders in task success and harness satisfaction differ. Frequent successful deliveries with failed Harness criteria show that outcome-only evaluation cannot characterize harness reliability. The released tasks, configurations, traces, and verdicts support evidence-based comparison and diagnosis of harness behavior.

AI USE STATEMENT

We used generative AI tools for:

- Generating synthetic benchmark task data.

We have not used generative AI tools for:

- Helping develop theoretical models or conceptual frameworks.
- Proposing or refining hypotheses.
- Designing or providing feedback on research methodology or experiments.
- Implementing methods.
- Supporting qualitative and thematic data analysis.
- Interpreting results.

The following tasks are not applicable to this work:

- Formulating mathematical claims.
- Providing critical ingredients for proving mathematical claims.
- Assisting in the writing of proofs.
- Assisting with translation.
- Cleaning and reformatting datasets.

Additionally, we used generative AI tools for:

- Creating and editing scripts for aggregating existing experiment records.
- Discovering research topics or identifying gaps.
- Editing the manuscript to improve clarity and readability.

We have reviewed all AI-assisted work. Generated data were checked for format validity, generated code was tested by the authors, and references suggested by AI tools were checked for accuracy. For manuscript editing, AI tools provided revision suggestions, while the authors retained responsibility for the writing. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

The anonymous repository is linked in the abstract. Appendices A–D document tasks, adapters, evidence, and scoring; Appendix F.4 identifies the inputs needed to recompute the complete results in Appendix G.

REFERENCES

Yajing Bai, Jinhao Duan, Jie Peng, Xianfeng Wu, Sijia Liu, Song Wang, and Tianlong Chen. Harnessrisk: A lifecycle-oriented benchmark for agent harness safety. *arXiv preprint arXiv:2608.17597*, 2026.

Paul Barbaste, Tristan Darrigol, Germain Vu, and Tom Wiltberger. Harness engineering: Anatomy, architecture, and evolution of coding agents–a source-code study of eleven systems. *arXiv preprint arXiv:2609.00006*, 2026.

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. Agentrx: Diagnosing ai agent failures from execution trajectories. *arXiv preprint arXiv:2602.02475*, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Christoph Bühler, Matteo Biagiola, Luca Di Grazia, and Guido Salvaneschi. Agentbound: Securing execution boundaries of ai agents. *Proceedings of the ACM on Software Engineering*, 3(FSE): 2141–2164, 2026.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From failed trajectories to reliable llm agents: Diagnosing and repairing harness flaws. *arXiv preprint arXiv:2606.06324*, 2026.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Mohammed Mehedi Hasan, Hao Li, Gopi Krishnan Rajbahadur, Bram Adams, and Ahmed E Hassan. Model context protocol (mcp) tool descriptions are smelly! towards improving ai agent efficiency with augmented mcp tool descriptions. *arXiv preprint arXiv:2602.14878*, 2026.

Junjie Li, Xi Xiao, Yunbei Zhang, Chen Liu, Lin Zhao, Xiaoying Liao, Yingrui Ji, Janet Wang, Jianyang Gu, Yingqiang Ge, et al. Agent harness engineering: A survey. *OpenReview preprint*, 2026.

Xiaofeng Lin, Yingxu Wang, Tung Sum Thomas Kwok, Daniel Guo, Sahil Arun Nale, Charles Fleming, and Guang Cheng. Reflect: Intervention-supported error attribution for silent failures in llm agent traces. *arXiv preprint arXiv:2606.09071*, 2026a.

Xixun Lin, Yang Liu, Yancheng Chen, Yongxuan Wu, Yucheng Ning, Yilong Liu, Nan Sun, Shun Zhang, Bin Chong, Chuan Zhou, et al. Safeharness: Lifecycle-integrated security architecture for llm-based agent deployment. *arXiv preprint arXiv:2604.13630*, 2026b.

Chengzhi Liu, Yichen Guo, Yepeng Liu, Yuzhe Yang, Qianqi Yan, Xuandong Zhao, Wenyue Hua, Sheng Liu, Sharon Li, Yuheng Bu, et al. Auditing agent harness safety. *arXiv preprint arXiv:2605.14271*, 2026.

Ziyu Ma, Hailang Huang, Shun Zou, Yong Wang, Shidong Yang, Yiming Hu, Fei Wei, and XiangXiang Chu. Longhorizon-harness: Advancing long-horizon agents for real-world tasks. *arXiv preprint arXiv:2608.01964*, 2026.

Kai Mei, Xi Zhu, Wujiang Xu, Wenyue Hua, Mingyu Jin, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. Aios: Llm agent operating system. *arXiv preprint arXiv:2403.16971*, 2024.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference. *arXiv preprint arXiv:2606.05922*, 2026.

Noga Peleg Pelc, Gal A Kaminka, and Yoav Goldberg. A language for describing agentic llm contexts. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pp. 85–102, 2026.

Shuo Ren, Xiaomian Kang, and Jiajun Zhang. Skillalign: Aligning skill interfaces for llm-based agents. *arXiv preprint arXiv:2609.07255*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Biswa Sengupta and Jinhua Wang. Harbor: Automated harness optimization. *arXiv preprint arXiv:2604.20938*, 2026.

Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*, 2023.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of computer science*, 18(6):186345, 2024.

Peidong Wang, Zhiming Ma, Ying Chang, Xufang Luo, Xiaocui Yang, Shi Feng, Yuqing Yang, and Dongsheng Li. Self-evolving embodied agents via skill-harness evolution. *arXiv preprint arXiv:2608.11350*, 2026a.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*, 2026b.

Tianxin Wei, Ting-Wei Li, Zhining Liu, Xuying Ning, Ze Yang, Jiaru Zou, Zhichen Zeng, Ruizhong Qiu, Xiao Lin, Dongqi Fu, et al. Agentic reasoning for large language models. *arXiv preprint arXiv:2601.12538*, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Tianyuan Wu, Chaokun Chang, Lunxi Cao, Wei Gao, and Wei Wang. Crab: A semantics-aware checkpoint/restore runtime for agent sandboxes. *arXiv preprint arXiv:2604.28138*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, et al. Harness-bench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, et al. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025.

Hailin Zhong and Shengxin Zhu. Ai harness engineering: A runtime substrate for foundation-model software agents. *arXiv preprint arXiv:2605.13357*, 2026.

APPENDIX

A BENCHMARK TAXONOMY AND TASK SUITE

A.1 HARNESS RELIABILITY TAXONOMY

The taxonomy adapts the ETCLOVG decomposition (Li et al., 2026) to observable harness responsibilities in the 95-task release. A property belongs in the benchmark when the harness mediates it, a task can exercise it under comparable conditions, and its outcome can be checked against execution evidence. The seven primary components are

$$\mathcal{D}_H = \{C, E, G, L, O, T, V\}, \tag{5}$$

Each task contributes to one primary component and one mechanism-level property. Secondary tags describe interactions without duplicating observations in component totals. Table 4 defines the responsibilities and their scope.

Table 4: The seven components of the TraceTrial-Bench harness reliability taxonomy.

Component	**Harness responsibility**	**Representative properties in the final task suite**
Context (C)	Manage the information made available to the model and preserve task-relevant content across context reduction, turns, sessions, and agent boundaries.	Truncation disclosure, rolling-compaction fidelity, cross-turn fact retention, session-resume fidelity, persistent-memory behavior, and subagent context isolation.
Execution (E)	Execute requested actions and return process state, output, and failure information without changing or concealing their operational meaning.	Exit-code fidelity, standard-error capture, timeout behavior, shell-state persistence, background-process supervision, timeout propagation, stale-artifact handling, and PTY termination.
Governance (G)	Apply permissions, safety policies, and environmental restrictions consistently while preserving a clear relationship between declared and enforced behavior.	Execution governance, approval and guardrail consistency, network restrictions, destructive-action control, and secret or credential redaction.
Lifecycle (L)	Coordinate state and control flow across turns, sessions, delegated work, recovery operations, and resource boundaries.	Model-call budgets, submitted-versus-confirmed state, subagent result aggregation, pending-child handling, workflow handoff, goal revision, and session recovery.
Observability (O)	Expose sufficient and accurate information for the model or operator to identify failures, distinguish their sources, and reconstruct important execution events.	Tool-error relay, harness-error and refusal notification, event provenance, diagnostic state, and auditable completion information.
Tools (T)	Expose external capabilities through stable model-facing interfaces and faithfully transport tool definitions, arguments, results, and interaction requests.	Tool availability, naming and registration, argument transport, structured-result fidelity, skill-result relay, dynamic tool discovery, and structured user-clarification channels.
Verification (V)	Support evidence-based confirmation of execution progress and completion, and distinguish verified results from unsupported success claims.	Claim-to-execution consistency, citable completion receipts, disclosure of incomplete work, and consistency between reported values and produced artifacts.

Ownership follows the tested responsibility: tool declarations and argument transport belong to Tools; process outcomes belong to Execution; child context belongs to Context; child completion

belongs to Lifecycle. Verification concerns harness support for checking progress, while the independent oracle belongs to the measurement apparatus.

Component and property counts appear in Table 9.

A.2 Task Design and Benchmark Populations

Each task couples a harness mechanism with a practical deliverable. The designer specifies a trigger, an expected behavior, an observation boundary, and an oracle before writing the scenario. Examples include oversized outputs, masked exit failures, pending jobs, session handoffs, and structured clarification. Private or one-time values make correct delivery depend on the intended interaction; tasks that test recovery provide a bounded recovery route.

The task package fixes the prompt or turn sequence, initial workspace, tools, budgets, required outputs, verifier, and two score sheets. Cross-harness adapters preserve this contract while retaining native execution behavior (Appendix B). A missing target capability can be a measured outcome; an adapter that changes the task semantics creates a different comparison.

The release population is the 95 unique paths in `lean95.manifest`. The construction groups in Table 5 describe task origins and receive no separate scoring weights.

Table 5: Construction groups represented in the final 95-task release.

Construction group	**Tasks**	**Design purpose**
Obstruction and attribution cases	28	Expose harness-created obstacles, available recovery paths, and the possible effect of those obstacles on task completion.
Shared-mechanism comparison cases	27	Exercise mechanisms that are broadly available across harnesses so that comparisons focus on differences in implementation quality.
Capability-gap cases	21	Examine situations in which harnesses differ in whether they provide a usable path for clarification, delegation, approval, context handling, or lifecycle control.
Facet-balancing supplement	19	Improve the representation of under-covered mechanisms and execution surfaces, particularly coding-oriented and terminal-oriented scenarios.
Total	95	Authoritative release population defined by `lean95.manifest`.

The protocol distinguishes task construction from execution and evaluation:

$$\begin{aligned}\text{Authored} \rightarrow \text{Audited} \rightarrow \text{Cross-harness eligible} \\ \rightarrow \text{Executed} \rightarrow \text{Apparatus valid} \rightarrow \text{Measured} \rightarrow \text{Attributable}.\end{aligned} \tag{6}$$

Authored candidates are not performance denominators. A run attaches a harness, model, and execution instance to a released task; its final evaluation supplies criterion verdicts and candidate links. The reported population contains one selected final evaluation per task–model–harness cell. The prospective protocol stages above do not add exclusions to the full-grid analysis in Appendix F.3.

A.3 Task Cards, Oracles, and Hard Gates

Task cards summarize the contract, execution conditions, and verification sources defined by the task files. All released packages provide a prompt, turn specification, runtime profile, workspace, executable verifier, and separate Harness and Agent score sheets. Table 6 gives the normalized documentation fields; the archived task package remains authoritative.

Table 6: Normalized task-card fields for the released benchmark tasks.

Field	Primary source	Content
Task identity	`task.yaml`, release manifest	Task ID, title, released path, and task revision.
Scenario and target	`task.yaml`	Application domain, primary scenario, target component, and mechanism-level property.
Agent-visible contract	`prompt.md`, `turns.json`	User instructions, task goal, operational rules, working boundaries, and expected response.
Conversation protocol	`turns.json`, `runtime/execution.json`	Single-turn, fixed multi-turn, or simulator-driven interaction, including any session boundary or replay condition.
Initial state	`workspace/`, simulator configuration	Files, directories, services, records, and controlled state available when execution begins.
Available interfaces	`task.yaml`, runtime profile, tool manifests	Native tools, task-specific tools, business services, process controls, and other interfaces required by the task.
Target pressure	`task.yaml`, `isolation.yaml`, simulator configuration	The controlled condition used to exercise the target behavior, such as truncation, timeout, pending work, state recovery, schema conflict, or policy denial.
Milestones and restrictions	Prompt, turn schedule, simulator, verifier	Required intermediate states and actions that would bypass or invalidate the intended task path.
Deliverables	`expected_outputs`, prompt, verifier	Required files or responses, together with their paths, formats, schemas, and completeness conditions.
Resource budget	`task.yaml`, resolved run configuration	Maximum user-input turns, tool calls, and wall-clock time, together with any task-specific limit.
Execution contract	`runtime/execution.json`, probes	Execution mode, runtime requirements, process behavior, and evidence that must be collected during the run.
Oracle and verification	Private truth, simulator state, trace evidence, `score.py`	Authoritative facts, expected states, criterion definitions, and the executable verifier entry point.

Agent-visible inputs are separated from private truth, simulator state, and verifier logic. For example, `C1c-escalation-export-L3-001` requires one complete ticket export and bounded recovery requests against the same snapshot to obtain twelve escalation codes. The private ledger records the issued codes and requests; the verifier compares them with delivered tool results and the final JSON mapping. This binds task success to the information channel under test without relying on the Agent's account of its own actions.

Oracle sources. An oracle may be a truth file, protected fixture, simulator ledger, process outcome, or executable condition. Table 7 lists these forms.

Table 7: Main oracle sources used by TraceTrial-Bench.

Oracle source	Examples	Evaluation role
Private structured truth	Identifiers, codes, receipts, mappings, expected hashes	Supports exact comparison between required values and final deliverables.

Oracle source	Examples	Evaluation role
Simulator state and ledger	Accepted requests, issued values, state transitions, call counts	Records what reached the controlled service independently of the model-visible result.
Workspace and artifact state	File contents, database rows, manifests, output schemas	Determines whether required changes occurred and whether protected inputs remained unchanged.
Execution state	Exit status, standard output, standard error, signals, timeouts, process state	Establishes the actual outcome of commands and long-running operations.
Trace and probe evidence	Call-result identity, message delivery, session boundaries, child completion, policy decisions	Determines what the harness delivered to the model and how execution events were represented.

Dynamic values are taken from the state of the corresponding run. Oracle checks establish factual outcomes; assigning responsibility additionally requires the layer-specific evidence and attribution rules in Appendix D.

Milestones and prohibited events. A correct final artifact may still violate an interaction contract, for example by repeating a one-time operation or reading private state. Required actions, ordering constraints, and forbidden shortcuts are encoded in prompts, turn schedules, score sheets, and verifiers. These checks preserve the distinction between Agent actions, harness-mediated behavior, and collection failures.

Admission and hard gates. Ordinary criteria measure individual requirements; runner hard gates can override an execution-time verifier result; apparatus checks assess whether the evidence is usable. Table 8 documents these runner records. The reported analysis uses archived final criterion scores without applying a new gate or score override.

Table 8: Runner-level admission and hard-gate policy, distinct from the final-score aggregation used in the present analysis.

Condition	Evidence	Treatment
Apparatus admission failure	Missing or inconsistent run identity, execution record, required trace, oracle evidence, or verifier result	Admission is withheld before scoring. This protocol state is not an additional exclusion from the finalized archive analyzed in this report.
Task-specific prohibited event	Workspace state, simulator record, process evidence, or normalized trace	The relevant task criterion fails. The event does not override the whole score unless an explicit policy gives it hard-gate status.
Simulator evidence integrity	Private simulator manifest and its integrity result	For an otherwise scoreable simulator-backed run, failed integrity triggers a score-zero gate. If integrity cannot be established because of an apparatus failure, the run is invalid and the score override is not applied.
Non-simulator tool-call ceiling	Task-level `max_tool_calls` and a complete structured trace	If the trace proves that the limit was exceeded and no gateway prevented the extra calls from taking effect, the effective score is set to zero.

Table 9: Primary component coverage in `lean95.manifest`.

Component	Tasks	Share	Properties
Context	15	15.8%	6
Execution	27	28.4%	9
Governance	9	9.5%	4
Lifecycle	19	20.0%	8
Observability	7	7.4%	2
Tools	17	17.9%	10
Verification	1	1.1%	1
Total	95	100.0%	40

Table 10: Scenario and execution coverage of the 95 released tasks.

Facet	Category	Tasks	Share
Scenario surface	Workflow	59	62.1%
	Terminal	22	23.2%
	Coding	14	14.7%
Conversation form	Single-turn	46	48.4%
	Fixed multi-turn	45	47.4%
	Simulator-driven	4	4.2%
Execution mode	Native	93	97.9%
	Docker	2	2.1%
Task intensity	L3	95	100.0%

Condition	Evidence	Treatment
Simulator request ceiling	Immutable task limit and simulator-side request count	Requests beyond the limit are rejected before they can affect business state. The excess is recorded as a budget violation and is not charged again through a second score-zero gate.
Wall-clock and conversation limits	Resolved invocation timeout, turn schedule, and execution status	The limits are enforced during execution. The resulting termination state is retained as run evidence.

Validation and revisions. Task validation covers structural consistency, oracle and verifier behavior, trigger reachability, solvability within budget, and shortcuts that could bypass the target mechanism. Cross-harness review checks the same task contract and observable effects. Validation assets are summarized below.

A.4 Benchmark Coverage

Taxonomy coverage. Table 9 counts each task once under its primary component. Half of the 40 properties have only one or two scenarios, and Verification has one task. Related surface variants are not independent repetitions of a mechanism.

Scenario and execution coverage. Table 10 summarizes the task surfaces and interaction forms. Native execution means use of the harness's native interfaces inside the benchmark container; it does not mean execution outside Docker. The L3 label describes task pressure, not model reasoning intensity.

Oracle and evidence coverage. The artifact categories in Table 11 overlap and reflect task-specific oracle and evidence requirements.

Table 11: Task metadata, oracle, probe, and validation coverage.

Artifact or declaration	Tasks	Share
Executable verifier and both score sheets	95	100.0%
Structured `expected_outputs` field	83	87.4%
Private `truth.json`	66	69.5%
Explicit `oracle.json`	19	20.0%
Isolation specification	68	71.6%
Business simulator declaration	44	46.3%
Execution probe declaration	73	76.8%
Security probe declaration	3	3.2%
Dedicated `selftest.py`	57	60.0%

The score sheets contain 783 Harness and 556 Agent criteria: 909 programmatic rules and 430 semantic rubrics in total. Broader limits of this stress-task population and its grading procedure are consolidated in Appendix J.

B Harness Configurations and Cross-Harness Adapters

B.1 Harness Selection and Configuration

The four harnesses support repeated model–tool interaction, configurable model access, isolated task execution, and observable native activity. Table 12 identifies the configured packages and interfaces. Resolved configurations are retained with each run.

Table 12: Harness versions and controlled execution paths used in TraceTrial-Bench.

Harness	Version	Controlled execution path	Dockerfile
OpenClaw	`2026.6.9`	Native OpenClaw runtime with an isolated loopback Gateway	`docker/harnessbench/openclaw.Dockerfile`
Codex	`0.145.0`	Native Codex CLI through a loopback LiteLLM gateway	`docker/harnessbench/codex.Dockerfile`
Claude Code	`2.1.220`	Native Claude Code CLI through a loopback LiteLLM gateway	`docker/harnessbench/claudecode.Dockerfile`
DeepSeek Harness	`0.1.0-rc.6`	Official SDK with the native JSON-RPC agent runtime	`docker/harnessbench/deepseek.Dockerfile`

Codex and Claude Code use a local LiteLLM gateway; OpenClaw uses its native Gateway; DeepSeek Harness uses its SDK and JSON-RPC runtime. Docker isolates the task workspace and services while native harness interfaces control execution. Multi-turn tasks continue through the same native session unless the task declares a session boundary.

The model registry binds reporting aliases to provider endpoints. The adapter records the resolved configuration and preserves native defaults unless a task, measurement requirement, or compatibility condition requires an override. Reporting groups and the selected execution population are defined in Appendix F.1.

Configuration boundary. Common task contracts, initial states, and external limits are benchmark-controlled. System instructions, context policy, built-in tools, permissions, and session behavior remain native. Gateways and adapters provide transport and observation; they must not solve the task, repair model decisions, or supply a missing native capability.

B.2 Native Capability Discovery and Harness Profiles

Capability profiles distinguish documented support, exposure to the model, exercise in a run, and behavior verified against external state. Static configuration or the absence of a tool call alone cannot establish availability. Table 13 lists adapter collection contracts: A denotes formal-run support and T task-specific support, not successful behavior in every run.

Table 13: Measurement-facing capability contracts of the evaluated harnesses.

Capability or evidence contract	OpenClaw	Codex	Claude Code	DeepSeek Harness
Native tool calls and correlated tool results	A	A	A	A
Provider-boundary tool inventory with an explicit completeness status	A	A	A	A
Terminal completion, timeout, process containment, and background-process evidence	A	A	A	A
Native context-compaction boundary observation	A	A	A	A
Task-scoped business tools, service simulators, and MCP interfaces	T	T	T	T
Native child delegation and child-lifecycle observation	T	T	T	T
Asynchronous child dispatch, child-return boundary, and pushed completion	T	T	T	T
Approval, policy, principal-lineage, and managed-path evidence	T	T	T	T

Native evidence paths. OpenClaw combines Gateway results with session JSONL and native parent–child relationships. Codex combines JSONL execution events with thread-resume and session records. Claude Code combines stream-JSON with session archives and native tool-call identifiers. DeepSeek Harness uses persistent JSON-RPC notifications across turns. Task-specific contracts enable observation of compaction, permissions, and asynchronous child return where the runtime exposes them.

B.3 Adapter Boundary and Canonicalization

Adapters launch the selected harness, expose common task tools through native extensions, collect source records, and normalize them into Trace IR. Benchmark-supervised timeouts, probes, and container events remain distinguishable from native harness actions. Native session continuation is used for multi-turn tasks; repeated exported history is merged only with supporting identities and content agreement.

Table 14 summarizes the event families, and Table 15 identifies their native inputs. Appendix C.3 supplies the detailed schema.

Table 14: Main event types produced by the cross-harness adapters.

Event type	Normalized meaning
`model_input`	A model-visible input or request surface, including available messages and tool definitions when they can be observed safely.
`tool_inventory`	The structured tools exposed at a provider-request boundary, together with source, visibility, and completeness information.
`assistant_delta`	A native fragment of streamed assistant output.
`assistant_message`	A complete or terminal assistant message produced by the harness.
`tool`	A native tool call and its correlated result, including identifiers, arguments, status, output, and error information when available.
`token_usage`	Native or provider-level input, output, reasoning, and cache-token counts, with unavailable fields kept distinct from zero.

Event type	Normalized meaning
`context_compaction`	A native context-compaction boundary and its replacement summary when the harness exposes one.
`subagent_lifecycle`	The creation, start, return, completion, or failure of a child agent, together with parent-child identity information.
`policy_event`	A native approval, denial, permission, sandbox, or policy decision associated with an attempted action.
`native_session_archive`	The location, identity, and completeness status of an archived native session or notification stream.
`run_started`, `run_finished`, and `run_failed`	Benchmark lifecycle events that record process status without being treated as model or harness actions.

Table 15: Native evidence sources and their adapter mappings.

Harness	Native evidence source	Adapter mapping
OpenClaw	Gateway-backed CLI output and session JSONL records	Parses the structured CLI result and recovers missing trajectory information from the session bound to the run. Parent and child sessions remain linked through native session identifiers.
Codex	Native `exec` JSONL stream and session archives	Maps command execution, MCP calls, collaboration events, assistant messages, and usage records. Native thread identifiers are retained for continuation and child-session discovery.
Claude Code	Native stream-JSON output and session archives	Matches tool calls and results through native identifiers, removes duplicate stream records, and retains messages, reasoning output, usage, compaction, and child-agent evidence.
DeepSeek Harness	SDK results and persistent JSON-RPC notifications	Reconstructs the model-visible sequence from incremental notifications and retains tool interactions, compaction, subagent lineage, usage, and policy events across turns.

Canonicalization rules. The following rules preserve source meaning while allowing a common representation.

1. **Native content is preserved.** Prompts, assistant messages, tool arguments, tool outputs, errors, and compaction summaries are not rewritten for semantic consistency. Secret values may be redacted, but the surrounding record and its structure remain unchanged.
2. **Tool calls and results are matched conservatively.** Native tool-call identifiers are used whenever they are available. A call without an observable result remains in the trace with a missing result rather than being removed or assigned a successful outcome. An unmatched result is also retained as incomplete evidence.
3. **Native identifiers remain recoverable.** Session, message, tool-call, and child-agent identifiers are preserved in their native form. If a harness reuses the same tool-call identifier across conversation turns, the merged trace assigns a turn-scoped normalized identifier and stores the original value as `native_tool_call_id`.
4. **Event order follows the strongest available native order.** Stream order, native sequence numbers, and session-record order take priority over adapter observation timestamps. The final trace receives a contiguous sequence number for inspection. Source sequence numbers and timestamps remain attached to imported events.
5. **Distinct principals are not flattened.** Actions performed by the main agent and actions performed by child agents retain separate identities. Parent-child relationships are recorded explicitly, and a child action is not reassigned to the parent merely because both appear in the same session archive.
6. **Unknown and damaged records remain visible.** A valid but unrecognized native event is retained as a generic native or raw event. If a stream contains malformed structured output,

the affected record is preserved and the structured-trace status is downgraded. The adapter does not silently discard the parsing failure.

7. **Missing information is not reconstructed without evidence.** An adapter does not generate hidden reasoning, omitted tool results, approval decisions, or lost context. A native compaction event records only the summary and boundary exposed by the harness. It does not restore messages removed by the compaction process.

8. **Apparatus evidence remains separate.** Process timeouts, container enforcement, provider-request records, and task-tool transport logs are labeled according to their actual source. They may support evaluation or attribution, but they are not converted into native harness events.

For Codex command events, normalization removes only the fixed outer shell wrapper added by the evaluated executor. Agent-supplied pipes, redirections, and nested wrappers remain unchanged, and the full native command remains available. Task-tool transport may change its envelope but not arguments, responses, state effects, or error meaning.

B.4 Semantic Equivalence and Eligibility

This subsection defines prospective comparison requirements. It does not introduce an additional filter on the archived results in Appendix G. Equivalent paths preserve the task goal, information, action opportunities, state effects, and evaluation conditions while leaving harness-controlled differences observable.

1. **Goal equivalence.** The model receives the same task objective, required deliverables, constraints, and success conditions. Formatting changes required by a native input interface must not add, remove, or clarify task information.

2. **Initial-state equivalence.** The workspace, files, service records, credentials, environment variables, and simulated external state are initialized from the same task specification. Harness-specific configuration files may differ, but they must not contain task answers or additional problem-solving information.

3. **Task-tool equivalence.** Benchmark-supplied tools must represent the same task-level operations. Their required inputs, returned information, errors, and effects on external state must remain equivalent even when the harness uses a different transport mechanism.

4. **Action-effect equivalence.** The same valid task action must produce the same externally observable state change. An adapter must not execute an operation on behalf of the model, repeat a failed operation automatically, or convert a rejected operation into a successful one.

5. **Observation equivalence.** Tool results, errors, completion signals, and service responses must preserve the information returned by the task environment. Harness-specific presentation may differ, but the adapter must not summarize, expand, or repair the information before it reaches the model.

6. **Lifecycle equivalence.** Turn limits, wall-clock limits, model-call limits, process boundaries, and session-reset rules must follow the same task specification. A native continuation, compaction, background process, or child-agent mechanism must not be replaced with a stronger benchmark-controlled alternative.

7. **Policy equivalence.** Managed paths, approval requirements, prohibited actions, and task-specific access restrictions must apply to the same operations. A benchmark safety backstop may prevent damage, but it cannot be counted as evidence that the harness itself enforced the policy.

8. **Evidence equivalence.** The evidence required by the task oracle must be observable with comparable strength. A structured native event cannot be replaced by an adapter-generated statement that the event probably occurred. When one harness exposes weaker evidence, the affected comparison is restricted or marked non-comparable.

Pair-level eligibility. Let R_t be the capabilities needed to administer and observe task t, C_h the harness's measurement contract, $S_{t,h}$ semantic equivalence, and $O_{t,h}$ the required observation path. Then

$$E_{t,h} = \mathbb{I}\left[R_t \subseteq C_h \wedge S_{t,h} \wedge O_{t,h}\right], \tag{7}$$

A target capability need not be present when its absence is the behavior under test. A pair is unsupported if the test cannot be administered or observed, and non-comparable if the adapted path changes its semantics.

Run-level validity. For an eligible pair, let $A_{t,h,r}$ denote apparatus health and $I_{t,h,r}$ source and archive integrity. Protocol admission is

$$V_{t,h,r} = E_{t,h} \wedge A_{t,h,r} \wedge I_{t,h,r}. \tag{8}$$

Admission concerns evidence integrity, not task success. A model action that bypasses a measurement channel is annotated separately; it does not erase a finalized zero. The full-grid reporting policy is specified in Appendix F.3.

Examples of non-comparable substitutions are:

- replacing a structured clarification channel with an adapter-generated answer
- replacing asynchronous child execution with a sequential runner-controlled command
- reconstructing lost context from the full transcript after native compaction
- treating an outer container denial as a native harness approval decision
- converting plain assistant text into a tool call that the model did not issue
- supplying a task result through the initial prompt instead of the required tool interface
- inferring a native lifecycle event only from the final task artifact

A collector or archive failure can make an observation inconclusive. A harness timeout or absent event remains measurable when the observation path itself is intact. Table 16 summarizes the protocol distinctions.

Table 16: Treatment of eligibility and validity outcomes.

Status	Meaning	Statistical treatment
Eligible	A semantically valid and measurable native execution path is available.	Included in the cross-harness eligible population.
Unsupported	The benchmark lacks a valid administration or observation path; this is distinct from an absent target capability.	Outside the prospective protocol population; not an additional filter on the current archive.
Non-comparable	A possible adapter path would change task semantics, native behavior, or evidence strength.	Report the protocol limitation; the current analysis retains its declared full grid.
Measurement bypass	An Agent action obscured a target Harness property.	Preserve raw verdicts and annotate the affected criteria; adjust only the optional attribution set.
Apparatus-invalid	The benchmark could not execute, collect, or verify the run as specified.	Withhold protocol admission; do not infer a Harness defect. No new archival exclusion is applied in this study.
Measured failure	The pair was eligible, the apparatus was valid, and the observed behavior failed one or more criteria.	Included as a failure in the applicable denominator.

Matched comparisons. For a prospective study using protocol-based admission, the matched population for two harnesses is

$$\mathcal{D}_{h_1,h_2} = \{(t, m, r) \mid E_{t,h_1} = E_{t,h_2} = 1, V_{t,h_1,r} = V_{t,h_2,r} = 1\}. \tag{9}$$

The present study instead retains all finalized cells in its fixed archive, with 855 model–task cells per harness. Each model–task cell contributes one selected execution; the population does not constitute independent repeated trials of each cell.

C Evidence Acquisition and Trace IR

C.1 Evidence Sources and Acquisition Architecture

The collection pipeline preserves native harness records, provider-boundary metadata, task-tool ledgers, environment state, and apparatus records. Table 17 specifies their producers, visibility, and uses. The runner captures an initial workspace, collects observations during execution, and archives the final workspace and source records for offline evaluation.

Table 17: Evidence layers collected for each benchmark run.

Evidence layer	Producer	Model visibility	Examples and primary use
Native harness evidence	Harness runtime	Mixed	Assistant messages, native tool calls and results, session records, compaction events, child-agent events, policy decisions, and native usage reports. Used to reconstruct harness behavior.
Provider-boundary evidence	Request recorder	Mixed	Model-visible tool declarations, request counts, numeric timing observations, and provider usage fields. Used to confirm the request surface and derive comparable efficiency measurements.
Task-tool evidence	Task tools and service simulators	Responses are visible, transport records are observer-only	Tool requests, returned values, errors, committed operations, and service-state transitions. Used to verify the effects of model-initiated actions.
Environment-state evidence	Workspace and task environment	Depends on task permissions	Baseline and final files, workspace differences, output artifacts, and final external-service state. Used to determine whether required results were produced.
Apparatus evidence	Benchmark runner and container runtime	Not visible	Process status, exit code, timeout, image identity, resource limits, archive status, cleanup status, and collector health. Used to determine whether the run was measured correctly.

Visibility and authority. A task-tool ledger establishes what a service received, produced, or committed. Only a correlated model-facing record establishes what the model received through the harness. Likewise, a workspace or simulator snapshot establishes external state, while an assistant message records a claim about that state. Observer-only records, including private truth and apparatus logs, are not treated as information available to the Agent.

The provider recorder retains structured tool declarations and numeric request metadata rather than full prompts or conversation content. A complete inventory supports a claim about the structured tool surface at that request boundary; it does not describe every possible native action interface. Native traces contain only exposed reasoning or assistant content, without reconstructing hidden reasoning.

Artifacts and partial collection. The run archive stores normalized events in `events.json`, a derived overview in `summary.json`, native records, workspace changes, and applicable service or probe evidence. The root `result.json` indexes execution artifacts; final scoring authority is specified in Appendix F.2. Collected records remain available after errors or timeouts, and malformed or unsupported native records are retained as raw evidence. Missing events are never synthesized to complete a trajectory.

C.2 Trace Reconstruction and Source Precedence

Reconstruction parses native records, binds them to the run and session, correlates exchanges, merges resumed turns, and preserves supported ordering. Native identities and explicit relationships take precedence over text similarity or timestamp proximity.

Calls, turns, and principals. Tool calls and results are matched by native call identifiers. Repeated exports are merged only when identities and semantic content agree; unmatched calls or results remain incomplete. Multi-turn events retain `conversation_turn`, `source_event_seq`, and `source_event_timestamp`. If a later turn reuses an identifier for a different call, the normalized identifier becomes turn-scoped while `native_tool_call_id` preserves the source value.

Child sessions retain their own principal and a typed parent relationship. Spawn, return, and completion records establish cross-session dependencies; local event order is preserved within each principal. Background jobs are linked through exposed job or process identifiers. External completion and delivery to the parent or model are distinct observations.

Context and ordering. Persisted history is not necessarily active model context. Native model-input records, when available, establish the messages delivered at a step; compaction records establish only the boundary and summary actually exposed. Timestamps may describe creation, observation, or export and cannot establish cross-session causality by themselves. The final contiguous `seq` index is a presentation order, not a stronger causal claim.

Claim-specific precedence. Table 18 identifies the preferred source for each question. Conflicting lower-precedence observations remain in the archive: for example, a native success message and a simulator's failed transaction describe different boundaries and may reveal a delivery problem. Ambiguous session bindings instead require integrity review.

Table 18: Claim-specific source precedence used during trace reconstruction.

Claim	Preferred evidence	Fallback and limitation
Tool availability	Complete model-visible provider-boundary inventory	An observed native call proves presence only. It cannot establish that other tools were absent.
Model-visible context	Native model-input or request-surface record	Persisted session history may support reconstruction, but it does not prove that every message remained active.
Native tool invocation	Native tool-call event with a stable identifier	A task-service request proves that an operation reached the service, but it may not identify the responsible principal.
Tool-result delivery	Native tool-result or later model-input record containing the result	A service response proves that the service produced a result, but not that the model received it.
External state change	Simulator ledger, authoritative service state, or final workspace	Tool output and assistant text describe the claimed effect but do not override authoritative state.
Compaction, subagent, or policy event	Typed native event bound to the correct session or principal	Assistant text may support interpretation but cannot create the missing native event.
Model-request count	Complete provider-request sequence	Per-call native usage may provide a derived count. Turn-level or run-level aggregates do not.

Claim	Preferred evidence	Fallback and limitation
Token usage	Complete per-request provider usage	Native per-call usage is used when provider evidence is incomplete. Run aggregates remain labeled as aggregate evidence.
Timing	Provider-boundary request and response observations	Native CLI stream timing is used only when its scope and coverage are known.
Process completion and timeout	Process supervisor and container lifecycle	A harness message may explain the stop condition but does not override the observed process status.
Apparatus validity	Container lifecycle, archive manifest, integrity checks, and collector status	Harness output cannot establish that the measurement apparatus completed correctly.

Provider usage replaces a native aggregate only when request coverage is complete. Timing and token quantities must cover the same requests before they are combined. Partial records may support an external-state or action claim without supporting result delivery or attribution; the criterion's evidence contract determines what can be concluded.

C.3 Trace IR Schema and Event Dictionary

Trace IR separates a common envelope from event-specific payloads. Tables 19 and 20 define the required fields and core events. Optional identities and values are populated only when supported by native evidence.

Table 19: Common fields in the Trace IR event envelope.

Field	Type	Definition
`schema_version`	integer	Trace IR schema version. The current release uses version 1.
`seq`	positive integer	Contiguous event position in the final reconstructed trace.
`timestamp`	string	UTC timestamp assigned when the normalized event is written. This is an observation timestamp rather than a native execution timestamp.
`run_id`	string	Identifier shared by all events and artifacts belonging to one benchmark run.
`event_type`	string	Canonical type that determines the event-specific payload.
`source`	string or absent	Native stream, session archive, provider boundary, or apparatus component from which the event was obtained.
`conversation_turn`	positive integer or absent	Benchmark-managed user-input turn to which the event belongs.
`session_id`	string or absent	Native harness session, thread, or conversation identifier.
`turn_id`	string or integer or absent	Native identifier for a model or harness turn.
`step_id`	string or integer or absent	Native identifier for an internal model step when the harness exposes one.
`principal_id`	string or absent	Identity of the main agent or child agent responsible for the event.
`parent_principal_id`	string or absent	Identity of the parent principal when the event belongs to a child agent.
`principal_role`	string or absent	Role of the responsible principal, normally `main` or `child`.
`source_event_seq`	integer or absent	Original sequence number before a turn or session trace was merged.
`source_event_timestamp`	string or absent	Observation timestamp copied from the source trace during merging.
`source_timestamp`	string or absent	Timestamp supplied directly by the native harness record.
`raw_event`	JSON value or absent	Native source record associated with the normalized event after required redaction.

The normalized `timestamp` records collection time; source timestamps remain in `source_timestamp` or `raw_event`. Run, session, turn, step, and principal identifiers locate each observation without collapsing concurrent agents into one execution stream.

Table 20: Core Trace IR event types and payload fields.

Event type	Main payload fields	Meaning
`model_input`	`model`, `provider`, `messages`, `tools`, `request_params`	A model invocation surface exposed by the native harness. The completeness of messages and tools depends on the source.
`tool_inventory`	`model_visible`, `complete`, `scope`, `tools`, `request_count`, `inventory_fingerprint`	Structured tool declarations observed at the provider boundary. Each tool contains a name, type, description, and input schema.
`assistant_delta`	`delta` or `content_delta`, `reasoning_delta`, `content_kind`, `fragment_seq`	A fragment of streamed assistant text or reasoning content exposed by the harness.
`assistant_message`	`message_id`, `content`, `content_kind`, `reasoning`, `model`, `finish_reason`	A complete assistant message or complete reasoning block. A reasoning block is not treated as the final answer.
`tool`	`tool_call`, `tool_result`	A native tool call and its correlated result. The result may be null when no matching native result was observed.
`token_usage`	`usage`, `usage_scope`, `usage_aggregate`, `reported_cost_usd`	Token usage associated with a model call, turn, complete run, or internal harness operation.
`context_compaction`	`native_event_type`, `native_event_id`, `native_event_key`, `trigger`, `summary`, `summary_present`, `notice_visible_to_model`	A native context-compaction, pruning, or replacement boundary.
`subagent_lifecycle`	`action`, `status`, `stop_reason`, principal fields, `boundary_kind`, `pending_child_ids`	Creation, dispatch, start, return, completion, or failure of a native child agent.
`task_completion`	`status`, `content`, `model_visible_content`, principal fields, `parent_reactivated`	A native child or delegated task result that reaches a completion boundary.
`policy_event`	`policy_kind`, `decision`, `reason_code`, `approval_id`, `tool_call_id`, `tool_name`, `reason_visible_to_model`, `execution_started`	A native approval, permission, sandbox, or policy decision.
`native_event`	`native_event_type` or `native_method`, `status`	A valid native record that does not map to a more specific core event.
`native_diagnostic`	`diagnostic_type`, diagnostic-specific fields	A native diagnostic that may explain a process result but does not itself represent a model action.

Tool exchanges. A `tool` event contains a `tool_call` with its identity, name, and arguments, and a correlated `tool_result` when available. Result fields preserve content, source-indicated error status, integer exit code, duration, and known truncation. Error status is not inferred from negative wording. A missing result is represented as

$$\texttt{tool_result} = \texttt{null}. \tag{10}$$

This differs from a successfully returned empty string.

Usage scope. Usage retains whether it describes an individual call or an aggregate:

- `call` for one model request
- `turn` for a native harness turn
- `run` for a complete execution aggregate
- `internal` for a harness operation such as a compaction call

Cached input is included in total input, and reasoning output in total output; neither is added again. Missing usage remains unavailable, and an aggregate is not expanded into estimated calls.

Compaction, child lifecycle, and policy. Compaction records retain a native identity or record hash, the exposed replacement summary, and whether a summary was present. A visible compaction notice does not establish preservation of all relevant context. Child lifecycle events retain action, principal, and parent identities; a child completion does not establish parent reactivation without a corresponding native observation. Policy events retain the native decision, supported reason, visibility, and whether execution began. An outer-container denial remains apparatus evidence rather than a native harness policy decision.

Support and extension events. Table 21 documents collection and orchestration events, which are not counted as model actions.

Table 21: Lifecycle, orchestration, and evidence-support events.

Event type	Main payload fields	Meaning
`run_started`	`harness`, `model`, `cwd`, `argv`, `isolation`	Start of the harness execution or task protocol.
`run_finished`	`exit_code`, `trace_complete`, `summary_path`	Normal end of the observed execution process.
`run_failed`	`error_type`, `error_message`, `exit_code`, `timed_out`	Abnormal termination, timeout, unsupported protocol, or infrastructure failure.
`conversation_started`	`mode`, `planned_turns`, `max_turns`	Start of a benchmark-managed multi-turn conversation.
`conversation_turn_started`	`turn_index`, `input`, `user_input`	Delivery of one benchmark-managed user turn.
`conversation_turn_finished`	`turn_index`, `status`, `native_session_id`	Completion status of one conversation turn.
`conversation_simulator_decision`	`turn_index`, `done`, `next_message`	Decision produced by a task-defined conversation simulator.
`conversation_finished`	`status`, `turns_completed`, `native_session_id`	End of the complete benchmark-managed conversation.
`session_boundary`	previous and new session identifiers, `before_turn`, `workspace_preserved`, `history_replayed`	Explicit transition from one native session to another.
`context_handoff`	`action`, `before_turn`, `model_visible`, history count and hash	Benchmark-controlled delivery of permitted prior context at a declared session boundary.
`provider_request_count`	`count`, `source`	Native or provider-supported count of model requests when exposed as a trace event.
`native_session_archive`	`status`, `format`, `artifact`, `sha256`, `size_bytes`, `redacted`	Metadata for an archived native session or notification stream.
`trace_export`	`status`, `format`, `path`, `redacted`, export-specific checks	Status of a native trace-export operation.
`raw_stdout`	`text`, `stream_seq`, `parseable`, `parse_error`	Standard output that could not be represented by a more specific event.

Event type	Main payload fields	Meaning
raw_stderr	text, stream_seq, severity	Standard error retained for diagnosis after redaction.

Task-declared memory and lifecycle protocols may additionally use:

- session_created
- episode_injected
- fresh_session_completed
- delegated_parent_created
- delegated_child_created
- delegated_child_completed
- final_prompt_completed
- app_server_notifications

These receipts retain native request, session, child, or episode identities and content hashes. They supplement the core events only for the corresponding protocol.

Mapping and missing values. Table 22 summarizes normalization. Missing fields, explicitly unavailable values, measured zeros, and empty successful outputs retain their distinct meanings. The complete native record remains associated with its canonical event.

Table 22: General mapping from native records to Trace IR events.

Native record category	Trace IR event	Mapping rule
Request start or turn start	model_input	Preserve the exposed model, messages, tools, parameters, and native turn identity.
Structured provider tool declaration	tool_inventory	Normalize name, type, description, and input schema while retaining completeness and visibility fields.
Streamed text or reasoning fragment	assistant_delta	Preserve the text fragment, content kind, fragment order, and source timing.
Completed assistant content	assistant_message	Preserve message identity, visible content, content kind, and finish reason.
Tool-use start followed by result	tool	Match through the native call identifier and retain both source records.
Tool-use start without result	tool	Preserve the call and set tool_result to null.
Native usage record	token_usage	Preserve token fields and record whether their scope is a call, turn, run, or internal action.
Native compaction or pruning record	context_compaction	Preserve the boundary, replacement summary, session identity, and model-visibility status.
Child-agent lifecycle record	subagent_lifecycle	Preserve child identity, parent identity, action, status, and return boundary.
Delegated result delivered to a parent	task_completion	Preserve the child result and whether the parent was reactivated.
Approval, permission, or sandbox record	policy_event	Normalize the decision while retaining the complete native policy record.
Valid but unrecognized native record	native_event	Preserve the source type and complete raw record without assigning unsupported semantics.
Malformed or non-JSON stream content	raw_stdout or raw_stderr	Preserve the original text and record parsing or severity information.

C.4 Evidence Quality, Integrity, and Privacy

Table 23 defines the quality checks and reporting dimensions. Structural validity, criterion coverage, and support for attribution are separate properties; the table is a measurement specification, not a claim that every dimension has a corpus-level estimate in this study.

Table 23: Evidence-quality dimensions used by TraceTrial-Bench.

Dimension	Question	Main checks
Completeness	Were the records required for the intended measurement captured?	Trace termination, structured-event availability, session coverage, call-result coverage, provider-request coverage, and required probe artifacts.
Fidelity	Does the normalized event preserve the meaning and content of its native source?	Native identifiers, arguments, outputs, errors, status fields, raw records, and allowed normalization rules.
Source traceability	Can a normalized claim be traced to a native record or an authoritative state artifact?	Source labels, session and principal identities, source sequence numbers, artifact paths, and hashes.
Integrity	Are the stored artifacts complete, unchanged, and bound to the correct run?	File-set manifests, SHA-256 values, run identity, image identity, path safety, lifecycle checks, and evidence-schema validation.
Privacy	Were credentials and private benchmark materials excluded from released evidence?	Field-level redaction, output-tree scans, path scans, provider-record restrictions, and separation of private verifier inputs.
Replayability	Can the run be inspected and scored again without rerunning the agent?	Preserved workspace, trace, task state, scoring-input manifest, frozen configuration, and verifier-gate checks.
Bounded overhead	Did evidence collection remain within declared time, memory, output, and storage limits?	End-to-end timing, peak memory, captured output size, archive limits, and collector status.

Completeness and scoreability. The `trace_complete` field records whether the configured collection path reached its expected end, not whether all harness internals were observable. `structured_trace_status` records whether usable structured events were extracted. Table 24 summarizes these states. The current archive inclusion policy appears in Appendix F.3; criterion handling appears in Appendix D.2.

Table 24: Run-level evidence-quality states.

State	Condition	Treatment
Complete	Structured extraction succeeded, the normal acquisition path finished, and all evidence required by the reported measurements passed validation.	All supported criteria may be evaluated.
Partial	Some valid structured or state evidence was preserved, but the run ended early or one or more expected sources were incomplete.	Check admission and retain coverage limitations. Admitted criteria follow the fixed scoring rules.
Unavailable	No usable structured trajectory was recovered, although raw output or final artifacts may remain.	Check whether required-source admission fails. Do not silently remove finalized trace criteria from a score.
Integrity failure	Evidence cannot be bound to the run, does not match its manifest, contains an unsafe artifact, or fails a required schema check.	The affected measurement is rejected and may make the run apparatus-invalid.

Integrity and release boundary. Archive manifests bind file paths, sizes, and hashes to the run configuration and source identities. Task-specific probes and service ledgers retain their own provenance. These checks detect archive changes and incorrect bindings; they do not establish the truth of every source record. Permitted representation changes are listed in Appendix B.3.

Private truth and verifier inputs are separated from the Agent's writable workspace. The export policy excludes credentials and full provider request bodies and requires redaction of sensitive archived content. Offline scoring reads preserved evidence and verifies a workspace copy; it does not replay the Agent or guarantee reproduction of its decisions.

D Layered Evaluation and Failure Attribution

D.1 Evaluation Layers and Criterion Catalog

The Agent layer scores task actions and deliverables; the Harness layer scores the mediated execution behavior. Apparatus checks establish evidence integrity without contributing a third score (Table 25). The two verdict vectors remain separate, with one explicit dependency: Harness state 2 uses the completed Agent outcome.

Table 25: Evaluation layers used in TraceTrial-Bench.

Layer	Role	Main evaluation target
Agent	Scored, 50 points	Correctness, completeness, and task compliance of the agent's actions, claims, and final deliverables.
Harness	Scored, 50 points	Faithful delivery of context, tools, execution results, state, errors, permissions, and lifecycle events.
Apparatus	Admission gate	Integrity of traces, probes, runner metadata, private evidence, and archived artifacts used by the evaluator.

The task score sheets are `verifier/harness_score.yaml` and `verifier/agent_score.yaml`. A criterion specifies its identifier, type, statement, and required evidence:

$$c = \langle id, layer, type, statement, evidence, component, trigger, role \rangle. \tag{11}$$

Table 26: Fields used to describe an evaluation criterion.

Field	Meaning
`id`	A stable identifier that is unique across the two score sheets of a task.
`layer`	The scored layer to which the criterion belongs, either Harness or Agent.
`type`	The evaluation method, either a deterministic rule or an evidence-bounded rubric.
`statement`	A direct description of the condition that must hold for the criterion to pass.
`evidence`	The event types, fields, keywords, and matching rules used to select relevant evidence.
`component`	The harness taxonomy component examined by the criterion.
`trigger`	The condition under which a mechanism-specific criterion becomes applicable.
`role`	Whether the criterion contributes to a core metric or provides diagnostic information.

A *rule* uses deterministic checks of task state, artifacts, or exact trace properties. A *rubric* requires semantic judgment of a bounded evidence packet. Taxonomy ownership and task construction are defined in Appendix A.

Score composition. Let P_H, P_A be passed criterion counts and N_H, N_A the corresponding totals. Each layer contributes up to 50 points to the combined score:

$$S_H = 50 \times \frac{P_H}{N_H}, \tag{12}$$

$$S_A = 50 \times \frac{P_A}{N_A}, \tag{13}$$

$$S_{\text{total}} = S_H + S_A. \tag{14}$$

These contributions differ from the separately reported layer percentages, $100P_H/N_H$ and $100P_A/N_A$. A combined pass requires $S_{\text{total}} = 100$; Agent task success requires only $P_A = N_A$.

D.2 Verdict Rules and Evaluation Procedure

Deterministic rules are binary. Semantic Harness rubrics may return diagnostic states 2 or 3 before binary resolution, as specified in Table 27. State 2 passes only when all Agent criteria pass; state 3 resolves to zero.

Table 27: Criterion states and their final scoring treatment. Every finalized criterion remains in its layer denominator.

State	Meaning	Binary resolution
0: Unsupported	A criterion is violated or its selected evidence does not establish satisfaction.	Fail (0).
1: Supported	The selected evidence establishes satisfaction.	Pass (1).
2: Not exercised	A semantic Harness criterion concerns a native mechanism that is absent or whose path was not reached.	Pass only if all Agent criteria pass; otherwise fail. Retain the original state.
3: Malfunction	A semantic Harness criterion concerns a mechanism that exists and was reached but malfunctioned.	Fail (0), independently of the Agent outcome. Retain the original state.

Deterministic results are fixed first, followed by Agent rubrics and then Harness rubrics. Evidence selectors specify event types, fields, matching conditions, and return limits. Each semantic judgment retains its packet, prompt, response, verdict, and rationale. Agent verdicts provide downstream context and resolve state 2, but do not independently establish a Harness defect.

A required selector with no evidence receives zero and an empty-selection record. All finalized criteria remain in their layer denominator, including bypass-affected zeros. These rules describe operational criterion satisfaction; Appendix F.2 specifies the archived Judge and final-score source, and Appendix F.3 fixes the reporting population.

D.3 Failure Attribution and Evidence Requirements

Attribution separates raw failures, activated candidate links, confirmed Harness defects, and validated propagation. For finalized verdict vectors V_H, V_A, define

$$F_H = \text{fail}(V_H), \qquad Z_A = \text{fail}(V_A), \qquad Z_H = F_H \setminus B, \tag{15}$$

Here B contains Harness criteria whose observation was bypassed by an Agent action. Bypass annotations leave scores unchanged. The optional runtime summary uses Z_H; the reported contingency uses raw failures F_H. Membership in the confirmed defect set $D_H \subseteq Z_H$ additionally requires criterion-specific source reconciliation.

Candidate activation and validation. A declared mechanism $\ell = (h_\ell, a_\ell, \mu_\ell)$ connects Harness and Agent criteria through an intermediate information, action, or state distortion. The runtime activates it when the raw endpoints fail and its condition holds:

$$L_{\text{act}}(R) = \{\ell \in \mathcal{C}_t^{\text{run}} \mid h_\ell \in F_H,\ a_\ell \in Z_A,\ q_\ell(R) = 1\}. \tag{16}$$

Activation identifies a candidate for review. Promotion to L_{val} requires confirmed endpoints, observable intermediate distortion, supported event order, and consideration of competing explanations (Table 28). Candidate descriptors are not calibrated probabilities.

Table 28: Evidence to examine when validating a candidate Harness-to-Agent mechanism.

Stage	Required question	Typical evidence
Harness endpoint	Does the criterion zero correspond to a confirmed Harness defect?	Reconciled native records, probes, and authoritative state.
Intermediate distortion	What information, action, or state was lost, changed, delayed, or misreported?	Correlated field comparisons, messages, status records, or state transitions.
Agent exposure	Did the distorted signal reach the Agent before the relevant decision?	Model-facing delivery, call identity, and native event order.
Agent response	What decision, action, or omission followed?	Tool calls, unsupported claims, or recovery attempts.
Outcome effect	Which Agent criterion and authoritative task state were affected?	Deliverables, private oracle facts, or environment state.

Stored labels and interpretation. Table 29 contains seven decision rules and six stored labels; rules 4 and 6 both produce `agent_caused`. An outcome issue denotes $Z_A \neq \emptyset \vee B \neq \emptyset$.

Table 29: Ordered runtime attribution rules. The first matching rule determines the stored label.

Rule	Condition	Stored label
1	Apparatus admission fails.	`inconclusive_device`
2	Either layer has no measured pass/fail criterion.	`inconclusive_unmeasured`
3	$Z_H \neq \emptyset$, an outcome issue exists, and $L_{\mathrm{act}} \neq \emptyset$.	`harness_caused`
4	$Z_H \neq \emptyset$, an outcome issue exists, and $L_{\mathrm{act}} = \emptyset$.	`agent_caused`
5	$Z_H \neq \emptyset$, all Agent criteria pass, and $B = \emptyset$.	`harness_defect_absorbed`
6	$Z_H = \emptyset$ and an outcome issue exists.	`agent_caused`
7	Neither a Harness zero nor an outcome issue remains.	`clean`

The label `harness_caused` records candidate activation, not validated propagation; `agent_caused` does not assert that the Harness has no failed criteria. Linked and unlinked Agent failures can coexist. Operational absorption means $F_H \neq \emptyset$ with $Z_A = \emptyset$; confirmed defect absorption additionally requires a relevant endpoint in D_H. A specific recovery claim also needs an observed recovery action and verified final state. Population-level validation limits are consolidated in Appendix J.

E Pipeline Illustrations and Metric Reference

This appendix provides the pipeline diagrams, compact reference tables, and metric equations supporting Section 4. Appendices A–D give the corresponding task, adapter, evidence, and evaluation specifications.

E.1 Run Representation and Benchmark Design

For harness h, model m, task t, and execution instance r, the run record is

$$R_{h,m,t,r}=(\tau, \Delta W, G, V_H, V_A, L_{\mathrm{act}}), \tag{17}$$

Here τ is the normalized trace, ΔW the workspace change, G the apparatus state, V_H, V_A the finalized verdict vectors, and L_{act} the activated candidate links. Figure 4 shows how these records connect the native harness to the measurement pipeline.

Taxonomy and tasks. Figure 5 and Table 30 summarize the released taxonomy. Task packages bind each tested mechanism to a prompt, initial state, oracle, and two score sheets; Appendix A gives the construction and coverage details.

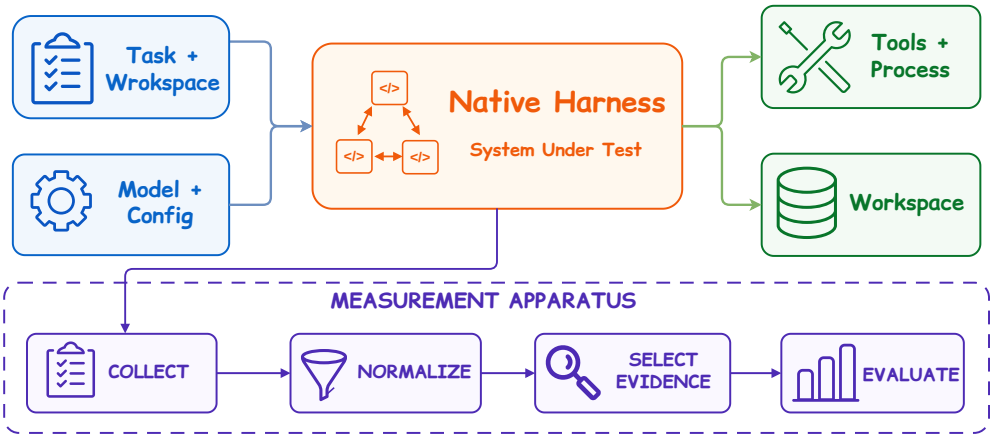


Figure 4: **TraceTrial-Bench system boundary and implemented pipeline.** The native harness is the SUT. Dashed arrows denote evidence acquisition; solid arrows denote task execution or subsequent analysis. The evaluator consumes normalized evidence only after run-identity and apparatus checks.

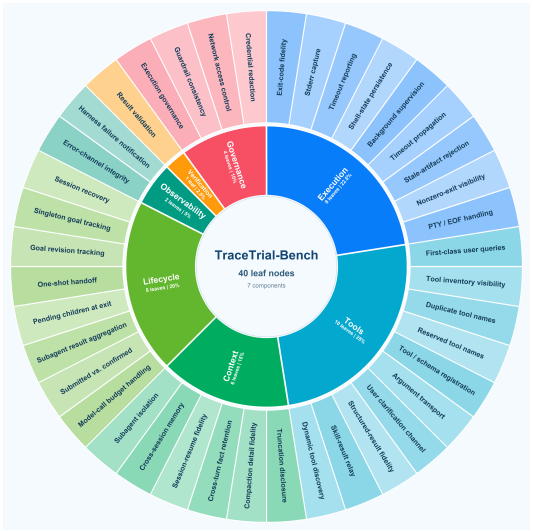


Figure 5: Distribution of the 40 mechanism-level reliability properties in TraceTrial-Bench across seven harness components. The inner ring shows each component's share of the properties, while the outer ring lists individual properties, with each leaf assigned equal weight.

Table 30: **Top-level Harness dimensions and coverage in the formal evaluation set.** Task counts refer to assignments in the 95-task evaluation population.

Dimension	Validation direction and content	Tasks
Context (C)	Preservation and disclosure of task-critical information across reduced outputs, session resumption, and persistent-memory boundaries.	15
Execution (E)	Faithful process semantics and result delivery, including timeout, exit-status, shell-state, background-supervision, and post-tool propagation behavior.	27
Governance (G)	Enforcement and communication of execution approvals, file-versus-exec guardrails, credential redaction, and untrusted-content boundaries.	9
Lifecycle (L)	Correct state transitions across model-call budgets, delayed confirmation, session recovery, and idempotent cursor-based retry.	19
Observability (O)	Model-visible and replayable error, refusal, and hook channels, together with persistent step logs and attributable sources.	7
Tools (T)	Validity and fidelity of inventories, names, schemas, arguments, typed and malformed results, structured transport, routing, skill relay, and dynamic discovery.	17
Verification (V)	Agreement between reported or delivered completion and authoritative simulator, artifact, or external outcome state.	1
Total		**95**

Native execution and apparatus. The adapters preserve native execution semantics under common task contracts (Appendix B). Task-scoped containers provide the initial workspace and declared services, while collectors preserve native and external observations. Apparatus records remain separate from harness behavior.

E.2 Trace Collection and Evidence Packets

The TT module maps exposed native activity to Trace IR. Table 31 provides a compact schema view; Appendix C.3 documents the fields. Figure 6 illustrates why call identity, external execution, and model-facing result delivery must remain distinguishable.

Table 31: Canonical event families used by the current tracer. Optional fields are emitted only when the native source supports them.

Family	Representative event types	Principal normalized content
Lifecycle	`run_started`, `run_finished`, `run_failed`	Run identity, native session, exit state, timeout, and trace-completeness boundary.
Model surface	`model_input`, `tool_inventory`, `assistant_message`, `assistant_delta`	Messages or declarations observed at a supported request boundary, assistant content, model identity, and finish reason.
Action/ observation	`tool`	Correlated call ID, tool name, arguments, model-facing result content, error/status fields, timing, and the native call/result record.
Stateful control	`context_compaction`, `subagent_lifecycle`, `policy_event`	Compaction boundary, parent/child identity, completion delivery, approval, or policy transitions when exposed natively.
Accounting/ archive	`token_usage`, `native_session_archive`, `raw_stdout`, `raw_stderr`	Usage, archive provenance and digest, or raw material that could not be normalized safely.

Criterion evidence. The MEA module selects bounded evidence packets according to each score sheet. Table 32 summarizes the sources and Figure 7 shows their path to the Judge. Source precedence and reconstruction rules are centralized in Appendix C.2.

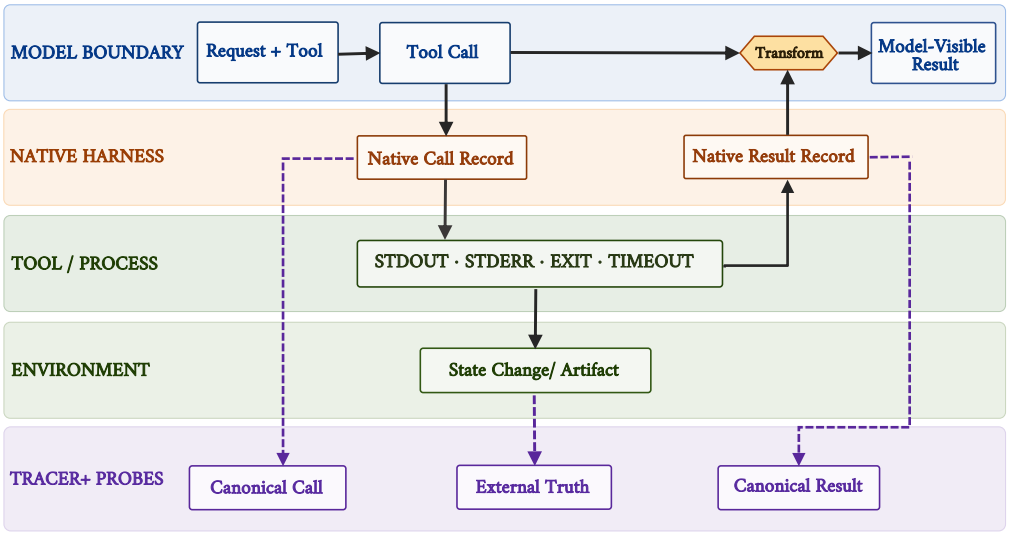


Figure 6: **Cross-boundary reconstruction of a tool action.** The model-facing result, native record, external execution state, and normalized event are related but not interchangeable. Solid arrows show execution; dashed arrows terminate in separately sourced, identity-aligned evidence.

Table 32: Evidence sources and their implemented epistemic scope.

Source	Collected by	Primary supported claim	Corroborating source for
Simulator ledger and final state	Runner-owned private service	Exact requests, issued values, irreversible state, and oracle truth	Model-visible presentation of the corresponding result
Workspace baseline and changed artifacts	Persisted scoring inputs and post-run workspace	Delivered file existence, structure, content, and unauthorized changes	Trajectory and tool evidence explaining their production
Native normalized trajectory	Harness-specific adapter	Model-facing calls/results when the native source exposes them, lifecycle, actor lineage, and final messages	External process or service truth
Provider-request evidence	Runner-side recorder	Successful structured tool declarations, request status, timing, and usage	Native delivery and subsequent environment effects
Execution/security probes	Runner-owned post-tool observers	Process disposition, timeout, path, approval, and actor observations independent of prose reports	Model visibility through a correlated native result
Agent claims	Final messages and output artifacts	What the agent asserted or delivered	Harness behavior and external truth through independent evidence

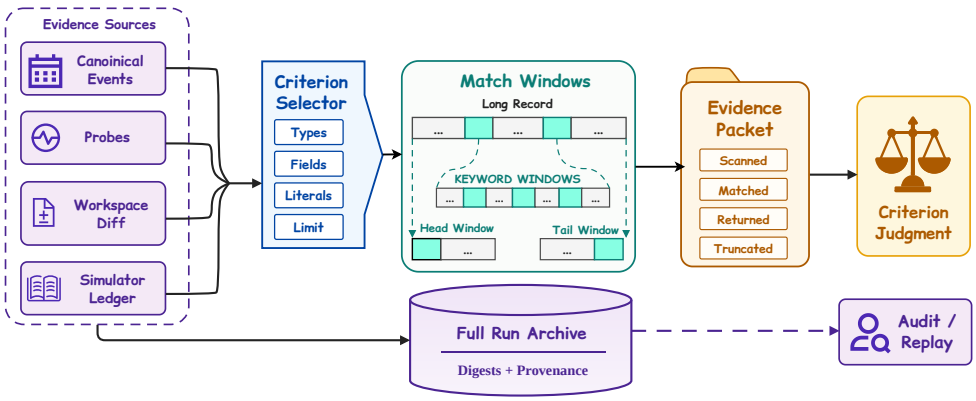


Figure 7: **Implemented evidence-packet construction.** Selection is deterministic and criterion-specific, while the complete archive remains available for audit and replay.

E.3 Evaluation and Attribution Reference

Table 33 summarizes the raw semantic states and their binary resolution. The evaluation order and evidence requirements are specified in Appendix D.2.

Table 33: **Judge states and their binary resolution.** Deterministic criteria use states 0–1; semantic Harness criteria may additionally use states 2–3. Every finalized criterion remains in the layer denominator.

State	Scope	Judge-side meaning	Final binary treatment
0 Unsupported	Agent or Harness	The criterion is violated, or the selected evidence does not establish satisfaction.	Finalize as fail (0).
1 Supported	Agent or Harness	The selected evidence establishes that the criterion is satisfied.	Finalize as pass (1).
2 Not exercised	Harness rubric only	The native mechanism is absent, or the run did not reach the relevant execution path.	Resolve from the Agent outcome: pass when all Agent criteria pass, and fail otherwise; retain state 2.
3 Malfunction	Harness rubric only	The mechanism exists and was reached, but the Harness itself malfunctioned.	Finalize as Harness fail (0), independent of the Agent outcome; retain state 3.

For n_X finalized items and p_X passes in layer $X \in \{H, A\}$, the equally weighted combined score is

$$S = 50\frac{p_H}{n_H} + 50\frac{p_A}{n_A}. \tag{18}$$

Figure 8 illustrates the evaluation flow. Apparatus admission is a protocol check; the archive population used for this study is defined separately in Appendix F.3.

Failure sets. The equations below collect the attribution notation from Appendix D.3. Raw Harness failures are F_H; bypass annotations B affect only the optional adjusted set Z_H, leaving scores unchanged:

$$F_H = \text{fail}(V_H), \qquad Z_H = F_H \setminus B, \qquad Z_A = \text{fail}(V_A). \tag{19}$$

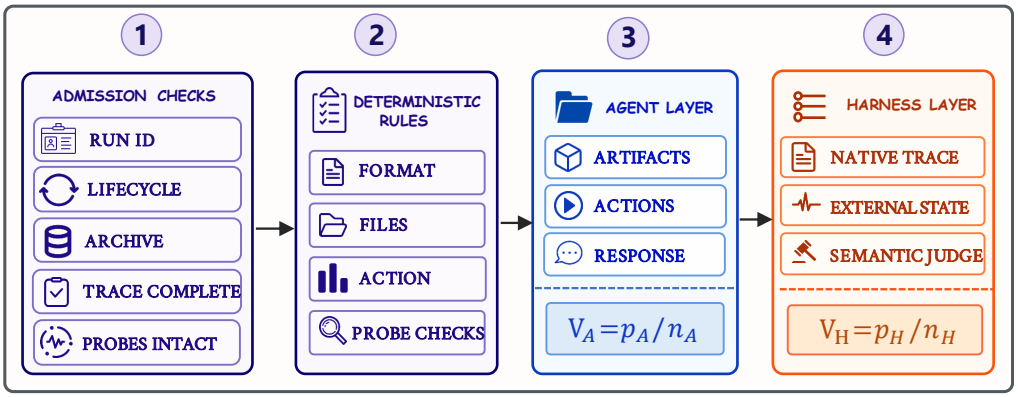


Figure 8: **Admission and layered-evaluation order.** Agent results are completed before Harness semantic judging; the two layers retain distinct evidence packets and verdict vectors.

Mechanisms and activation. A task-declared candidate connects criterion endpoints through an intermediate distortion:

$$\ell = (h_\ell, a_\ell, \mu_\ell), \tag{20}$$

For runtime candidate pool $\mathcal{C}_t^{\text{run}}$ and declared condition $q_\ell(R)$, activation is

$$L_{\text{act}}(R) = \{\ell \in \mathcal{C}_t^{\text{run}} \mid h_\ell \in F_H,\ a_\ell \in Z_A,\ q_\ell(R) = 1\}. \tag{21}$$

The failed Agent criteria reached by these links form

$$Z_{A,\text{linked}} = \{a \in Z_A \mid \exists \ell \in L_{\text{act}} : a_\ell = a\}. \tag{22}$$

Table 34 is the compact decision reference for stored labels. Their interpretation and the additional requirements for confirmed defects D_H and validated propagation L_{val} appear in Appendix D.3.

E.4 Aggregate Metrics

Let $\mathcal{R}_A$, $\mathcal{R}_H$, and $\mathcal{R}_{\text{attr}}$ be the reporting populations for Agent, Harness, and attribution metrics. In this study all three are the complete selected archive. Agent task success requires every Agent criterion to pass:

$$M_A = \frac{1}{|\mathcal{R}_A|} \sum_{R \in \mathcal{R}_A} \mathbb{I}[Z_A(R) = \emptyset], \tag{23}$$

Harness satisfaction is reported as both micro and macro pass rates:

$$M_H^{\text{micro}} = \frac{\sum_{R \in \mathcal{R}_H} p_H(R)}{\sum_{R \in \mathcal{R}_H} n_H(R)}, \qquad M_H^{\text{macro}} = \frac{1}{|\mathcal{R}_H|} \sum_{R \in \mathcal{R}_H} \frac{p_H(R)}{n_H(R)}. \tag{24}$$

The micro rate weights criteria equally; the macro rate weights runs equally. Both use finalized verdicts, including the prescribed resolution of state 2. Failure-state proportions use the raw sets F_H and Z_A. In particular, operational absorption is $F_H \neq \emptyset$ with $Z_A = \emptyset$, without bypass adjustment. Candidate-link incidence and unlinked Agent failures are reported separately under the aggregation policy in Appendix F.3.

Table 34: **Runtime-label decision table.** Rows are evaluated from top to bottom, and the first matching row determines the stored label. A "Harness zero" means $Z_H \neq \emptyset$; an "outcome issue" means $Z_A \neq \emptyset \vee B \neq \emptyset$. The labels summarize operational evidence states rather than validated causal conclusions.

Order	First matching run state	Stored label	Operational meaning
1	Apparatus admission fails ($G.\text{ok} = \text{false}$).	`inconclusive_device`	Exclude the run from Harness–Agent attribution.
2	Either layer has no measured pass/fail criterion.	`inconclusive_unmeasured`	At least one evaluation layer is unmeasured.
3	Harness zero and outcome issue, with $L_{\text{act}} \neq \emptyset$.	`harness_caused`	A candidate Harness-to-Agent mechanism is activated; validation is reported separately through L_{val}.
4	Harness zero and outcome issue, with $L_{\text{act}} = \emptyset$.	`agent_caused`	The outcome issue is unlinked; concurrent Harness zeros remain recorded on the Harness axis.
5	Harness zero, but all Agent criteria pass and no bypass is recorded.	`harness_defect_absorbed`	The Agent delivery is complete despite an operational Harness zero; D_H determines defect confirmation.
6	No Harness zero remains, but an outcome issue is present.	`agent_caused`	The Agent-side zero or measurement bypass is assigned to the Agent axis.
7	Neither a Harness zero nor an outcome issue is present.	`clean`	No measured criterion is zero and no bypass is recorded.

F Experimental Protocol and Reproducibility

F.1 Systems and Experimental Population

The study compares Claude Code, Codex, DeepSeek Harness, and OpenClaw on the 95-task archive imported on September 5, 2026. The nine model reporting groups are DeepSeek-V4-Pro, Gemini 3.7 Flash, GLM-5.3, GPT-5.6-Sol, Grok-4.6, HY3, Kimi K3, Opus 5, and Qwen3.8-Max. Original model identifiers and resolved configurations remain in the run records; the task taxonomy is defined in Appendix A.1.

The factorial population is

$$4 \text{ Harnesses} \times 9 \text{ models} \times 95 \text{ tasks} \times 1 \text{ selected execution} = 3{,}420 \text{ cells}. \tag{25}$$

Each root task–model–harness directory in `outputs/traces` contributes one selected execution. Nested tool or simulator records are not additional runs. This yields 855 cells per harness and 95 tasks per fixed model–harness comparison.

F.2 Execution and Final Evaluation

Tasks execute in Docker at `/workspace` through native harness interfaces. Task packages specify prompts, fixtures, tools, deliverables, verifiers, and budgets. Effective wall-time ceilings range from 120 to 5,400 seconds; task definitions allow 1–120 user-input turns and 5–300 task-tool calls. These are limits, not observed usage.

The archived semantic Judge is `deepseek-v4-flash`. The analysis preserves final verdicts and performs no new Agent execution or judging.

Performance is read from `artifacts/llm_judge/combined_score.json`, which contains both layer verdicts and activated candidate links. The root `result.json` supplies execution identity and metadata. Identity, binary criterion values, and consistency with the 50/50 formula are checked before aggregation.

F.3 Inclusion and Aggregation

Population. All 3,420 finalized evaluations enter once with their stored criterion values. The general eligibility and admission protocols in Appendix B.4 are not additional retrospective filters on this archive.

Scores and failure states. Agent task success, layer macro scores, and Harness micro scores follow Appendix E.4. Each task contributes once to its primary taxonomy dimension. Failure-state tables cross the empty/nonempty status of raw F_H and Z_A, producing four mutually exclusive states. No bypass adjustment is applied to the reported scores or contingency.

Candidate-link counts use stored endpoint pairs after final scoring. For dimension d, let J_d be jointly failing runs and $L_d \subseteq J_d$ runs with a stored link. Conditional incidence is $100|L_d|/|J_d|$; the pooled rate is $100 \sum_d |L_d| / \sum_d |J_d|$. Each run contributes once, regardless of link count. Linked and unlinked Agent failures may occur in the same run.

Resource accounting. Token columns report mean valid recorded usage in thousands. Estimated Agent cost uses the same model-specific prices published on the providers' official websites, frozen on September 5, 2026, across harnesses. Input accounting separates ordinary input, cache reads, and cache writes where counters are available; output includes reasoning tokens without adding them again. These standardized estimates exclude Judge calls, discarded executions, and non-model charges.

F.4 Reproducibility

Outcome aggregates can be recomputed from the final score files, their task and model identities, and each task's primary component. Native traces, resolved configurations, task packages, and scoring

evidence support inspection of individual verdicts. Resource estimates additionally require usage records and the frozen price catalog. The anonymous repository is linked in the abstract; the tables in Appendix G report the complete selected population. Interpretation limits are consolidated in Appendix J.

G Complete Results and Additional Analyses

G.1 Overall and Fixed-Model Comparisons

Table 35 reports all 3,420 selected evaluations, first pooled by harness and then within each model group. Mean Agent and Harness macro scores are 72.9% and 67.9%, respectively, giving a mean combined score of 70.4. Agent task success is 39.5%, while 8.4% pass every criterion in both layers.

Table 35: **Model × Harness main results.** Each fixed-model block evaluates 95 tasks per Harness; the aggregate pools nine models. Outcome columns are percentages: Pass requires all Agent criteria; Agent Score and Harness Macro average run-level fractions; Micro pools criteria. Clean requires both layers to pass; Link records candidate links; Abs. denotes Harness zeros without Agent zeros; Layered is the 50/50 mean. Resource columns average valid recorded usage: input, output, and total Tokens in thousands (k), and estimated Agent cost in USD at the frozen LLM Gateway prices; Judge costs are excluded. Within each block, **bold** and underlining mark the highest and second-highest displayed outcomes, with ties sharing marks. Link, Abs., resources, and the All Harnesses row are unranked.

	Agent layer		Harness layer		Attribution evidence			Layered	Agent resources (mean/run)			
Harness	**Pass**	**Score**	**Micro**	**Macro**	**Clean**	**Link**	**Abs.**	**Score**	**Input (k)**	**Output (k)**	**Total (k)**	**Est. cost ($)**
All models (9 models; 855 runs per Harness)												
Claude Code	**44.0**	**75.9**	71.5	71.2	9.1	22.7	34.9	**73.5**	569.4	9.5	578.8	0.436
Codex	35.8	70.7	60.2	59.8	2.8	28.1	33.0	65.2	606.9	10.2	617.2	0.595
DeepSeek Harness	40.0	73.3	69.5	68.8	9.8	21.8	30.2	71.0	361.5	11.2	372.7	0.446
OpenClaw	38.2	71.7	**71.9**	**72.0**	**11.7**	26.0	26.5	71.8	670.8	9.2	680.0	0.633
All Harnesses	39.5	72.9	68.3	67.9	8.4	24.6	31.1	70.4	552.1	10.0	562.1	0.528
DeepSeek-V4-Pro (95 tasks per Harness)												
Claude Code	**43.2**	**72.5**	71.6	71.2	8.4	22.1	34.7	71.9	603.5	12.9	616.5	0.115
Codex	30.5	68.4	59.9	59.3	0.0	27.4	30.5	63.8	580.1	13.0	593.1	0.046
DeepSeek Harness	37.9	69.9	68.6	67.9	10.5	25.3	27.4	68.9	325.0	11.4	336.4	0.043
OpenClaw	38.9	72.0	**71.9**	**71.9**	**11.6**	25.3	27.4	**72.0**	605.0	10.8	615.8	0.094
Gemini 3.7 Flash (95 tasks per Harness)												
Claude Code	40.0	74.1	70.4	69.7	8.4	21.1	31.6	71.9	590.7	10.3	601.0	0.247
Codex	38.9	73.8	62.5	62.0	7.4	23.2	31.6	67.9	622.2	13.1	635.3	0.226
DeepSeek Harness	**44.2**	**74.6**	70.8	70.0	8.4	25.3	35.8	72.3	496.0	11.6	507.6	0.416
OpenClaw	38.9	74.2	**75.1**	**74.8**	**10.5**	26.3	28.4	**74.5**	975.9	10.6	986.5	0.772
GLM-5.3 (95 tasks per Harness)												
Claude Code	43.2	**76.0**	**73.7**	**73.1**	9.5	21.1	33.7	**74.6**	639.3	13.9	653.2	0.408
Codex	28.4	68.6	59.1	59.3	2.1	28.4	26.3	63.9	627.1	8.2	635.3	0.189
DeepSeek Harness	**44.2**	75.5	71.4	70.5	9.5	17.9	34.7	73.0	380.6	14.5	395.1	0.226
OpenClaw	41.1	67.5	69.7	70.1	**12.6**	25.3	28.4	68.8	606.1	10.2	616.3	0.180
GPT-5.6-Sol (95 tasks per Harness)												
Claude Code	**44.2**	**79.6**	**72.8**	**72.6**	11.6	21.1	32.6	**76.1**	559.5	4.7	564.2	0.814
Codex	41.1	71.8	57.6	56.4	3.2	25.3	37.9	64.1	1427.8	10.9	1438.7	1.649
DeepSeek Harness	36.8	73.2	71.3	70.4	11.6	22.1	25.3	71.8	298.3	2.6	300.9	0.361
OpenClaw	42.1	71.5	71.1	71.2	**14.7**	29.5	27.4	71.4	606.7	5.1	611.8	0.974
Grok-4.6 (95 tasks per Harness)												
Claude Code	**55.8**	**81.6**	**71.9**	**71.9**	**10.5**	18.9	45.3	**76.7**	531.1	7.2	538.3	0.509
Codex	44.2	72.9	59.6	59.5	5.3	21.1	38.9	66.2	292.6	4.2	296.9	0.462
DeepSeek Harness	43.2	75.6	71.4	70.6	**10.5**	18.9	32.6	73.1	315.7	11.9	327.6	0.378
OpenClaw	33.7	70.8	68.5	69.0	8.4	24.2	25.3	69.9	541.6	4.4	546.0	0.822
HY3 (95 tasks per Harness)												
Claude Code	**32.6**	69.5	69.7	69.2	6.3	30.5	26.3	69.3	798.9	15.0	813.9	0.044
Codex	30.5	64.2	59.3	59.0	1.1	35.8	29.5	61.6	415.9	13.7	429.6	0.026
DeepSeek Harness	28.4	64.8	65.8	65.7	6.3	30.5	22.1	65.2	247.0	11.2	258.2	0.018
OpenClaw	29.5	**70.0**	**72.7**	**72.7**	**11.6**	29.5	17.9	**71.4**	659.5	18.5	678.0	0.041

Continued on next page

Table 36: **Reliability by taxonomy dimension.** Criterion pass rates and Agent task success on the full 3,420-run grid (%; higher is better).

	Coverage		Harness Micro (%)				Pooled (%)	
Dimension	**Tasks**	**Runs/H**	**Claude Code**	**Codex**	**DeepSeek Harness**	**OpenClaw**	**H Micro**	**Agent Pass**
Context	15	135	**74.5**	62.0	64.3	61.6	65.6	45.2
Execution	27	243	64.7	51.5	65.3	**77.1**	64.6	49.2
Governance	9	81	72.3	74.2	68.8	**80.5**	74.0	27.2
Lifecycle	19	171	71.3	53.0	**71.7**	65.9	65.5	43.6
Observability	7	63	**88.7**	76.5	86.8	85.0	84.3	43.3
Tools	17	153	**71.5**	65.6	**71.5**	70.2	69.7	21.4
Verification	1	9	93.7	**95.2**	82.5	88.9	90.1	8.3
All tasks	95	855	71.5	60.2	69.5	**71.9**	68.3	39.5

Note. Runs/H counts runs per Harness: nine models per task. Within each row, **bold** with shading marks the highest Harness Micro rate and underlining the second-highest; ties at the displayed precision share marks. Pooled H Micro aggregates criterion counts across Harnesses; Agent Pass requires all Agent criteria. Pooled columns are not ranked. All tasks pools the full grid, not the seven displayed dimension rates. Each task belongs to one dimension; Verification contains one scenario.

Table 35 (continued)

	Agent layer		Harness layer		Attribution evidence			Layered	Agent resources (mean/run)			
Harness	**Pass**	**Score**	**Micro**	**Macro**	**Clean**	**Link**	**Abs.**	**Score**	**Input (k)**	**Output (k)**	**Total (k)**	**Est. cost ($)**
Kimi K3 (95 tasks per Harness)												
Claude Code	**49.5**	**78.2**	71.1	70.6	**11.6**	21.1	37.9	**74.4**	358.1	4.8	362.9	0.613
Codex	42.1	74.0	61.3	60.9	2.1	28.4	40.0	67.4	573.9	7.7	581.6	0.379
DeepSeek Harness	43.2	74.4	71.5	70.5	10.5	16.8	32.6	72.4	281.7	6.1	287.8	0.225
OpenClaw	40.0	73.8	**73.6**	**73.5**	10.5	22.1	29.5	73.7	580.8	5.7	586.6	0.351
Opus 5 (95 tasks per Harness)												
Claude Code	**53.7**	**83.4**	71.8	71.9	7.4	16.8	46.3	**77.6**	379.4	3.7	383.1	0.660
Codex	30.5	70.3	60.8	60.5	2.1	34.7	28.4	65.4	482.8	7.6	490.4	2.121
DeepSeek Harness	49.5	79.3	72.3	71.4	**13.7**	17.9	35.8	75.4	551.5	16.9	568.5	2.071
OpenClaw	41.1	76.1	**73.8**	**73.9**	11.6	23.2	29.5	75.0	822.3	7.7	830.0	2.168
Qwen3.8-Max (95 tasks per Harness)												
Claude Code	33.7	68.3	70.5	70.2	8.4	31.6	25.3	69.3	663.8	12.6	676.4	0.517
Codex	35.8	**72.1**	61.4	61.0	2.1	28.4	33.7	66.5	436.7	13.4	450.2	0.258
DeepSeek Harness	32.6	71.8	62.5	62.6	7.4	21.1	25.3	67.2	357.9	14.2	372.1	0.279
OpenClaw	**38.9**	69.0	**70.6**	**70.9**	**13.7**	28.4	25.3	**70.0**	638.8	9.7	648.5	0.290

Claude Code leads Agent task success and the combined score; OpenClaw leads Harness satisfaction. DeepSeek Harness ranks second on Agent success and third on Harness macro score. Micro and macro Harness rates give the same aggregate ordering.

The fixed-model blocks expose interactions hidden by the aggregate. OpenClaw leads Harness macro performance for six models, while Claude Code leads for GLM-5.3, GPT-5.6-Sol, and Grok-4.6. For Gemini 3.7 Flash, DeepSeek Harness instead leads Agent success. Thus a model's best observed task outcome need not occur under the harness with its highest Harness score.

G.2 Taxonomy Dimensions and Failure States

Table 36 uses each task's primary component. Execution has the lowest pooled Harness micro rate (64.6%), followed by Lifecycle (65.5%) and Context (65.6%). The ordering varies by component: OpenClaw leads Execution, whereas Claude Code leads Context. Verification's high Harness rate covers only one task.

Table 37: **Failure states and candidate-link activation.** Results across Harnesses and primary dimensions on the full 3,420-run grid.

		Outcomes		Criterion states (runs)				Candidate links	
Group / Harness	**Runs**	**Agent Pass**	**H Micro (%)**	**Clean**	**H-only**	**A-only**	**Both**	**Linked**	**Link/Both (%)**
Full grid									
All Harnesses	3420	1351	68.3	286	1065	369	1700	842	49.5
Claude Code	855	376	71.5	78	298	97	382	194	50.8
Codex	855	306	60.2	24	282	58	491	240	48.9
DeepSeek Harness	855	342	69.5	84	258	71	442	186	42.1
OpenClaw	855	327	71.9	100	227	143	385	222	57.7
Primary dimension									
Context	540	244	65.6	29	215	69	227	116	51.1
Execution	972	478	64.6	76	402	44	450	174	38.7
Governance	324	88	74.0	14	74	55	181	52	28.7
Lifecycle	684	298	65.5	72	226	75	311	159	51.1
Observability	252	109	84.3	50	59	55	88	62	70.5
Tools	612	131	69.7	42	89	55	426	264	62.0
Verification	36	3	90.1	3	0	16	17	15	88.2

Note. Entries are run counts except the two percentage columns. Agent Pass requires every Agent criterion; H Micro pools Harness items. Clean means both layers pass; H-only, A-only, and Both indicate which layers contain a zero. These four states partition each row. H-only is operational absorption. Linked counts runs with at least one candidate link and is a subset of Both; Link/Both conditions on joint failure (— if none). Blocks are overlapping views, not additional runs. Verification contains one task. The full grid contains 1,633 activated endpoint pairs; 1,773/2,069 Agent-failure runs (85.7%) have at least one failed Agent criterion not reached by a link. Linked and unlinked criteria may coexist; candidate links do not establish causation.

The layer contrast also varies by mechanism. Subagent result aggregation has 46.2% Harness satisfaction and 72.2% Agent success; cross-turn fact retention has 48.4% and 95.8%, respectively. These comparisons motivate separate outcome and execution measures rather than using either as a substitute for the other.

Failure intersections and links. Table 37 partitions raw criterion failures and reports candidate-link activation separately. Claude Code has the most H-only runs; OpenClaw has the most A-only and jointly clean runs. Of 1,700 jointly failing runs, 842 contain a stored link. Another 1,065 runs pass all Agent criteria despite at least one Harness zero.

Interpretation. Candidate-link incidence over all runs and conditional incidence among joint failures answer different questions. Neither measures causal-link precision. The detailed attribution definitions are in Appendix D.3, and the study's limitations are in Appendix J.

G.3 Task-Level Score Heatmaps

Figure 9 shows the overall grid, followed by one panel per harness. All panels use the same task and model order and 0–100 scale. Cells show the equally weighted Agent/Harness score.

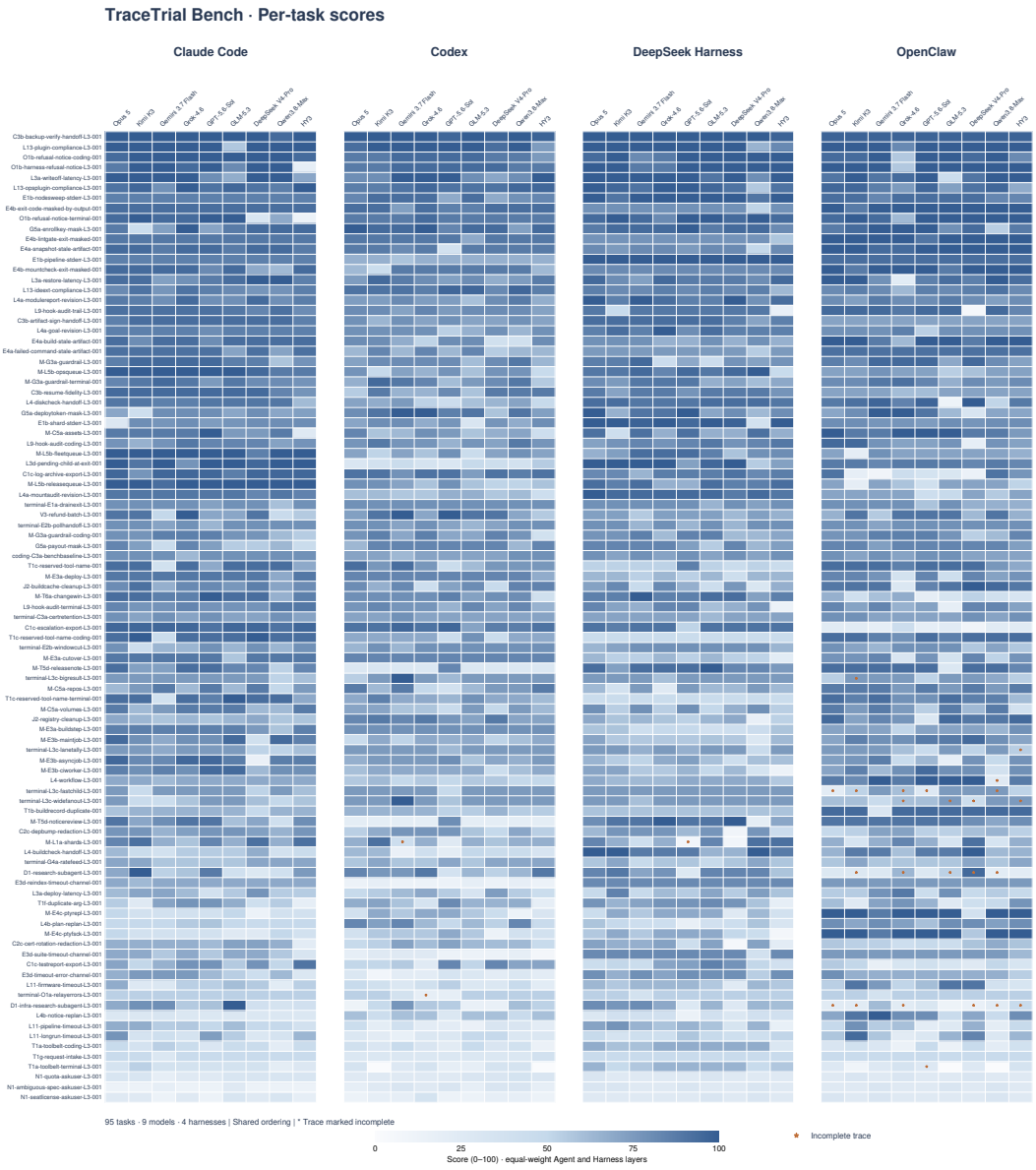


Figure 9: Task-level score overview: 95 tasks, nine models, and four Harnesses (3,420 selected runs). Shared ordering supports comparisons across Harnesses; color denotes the 50/50 layered score.

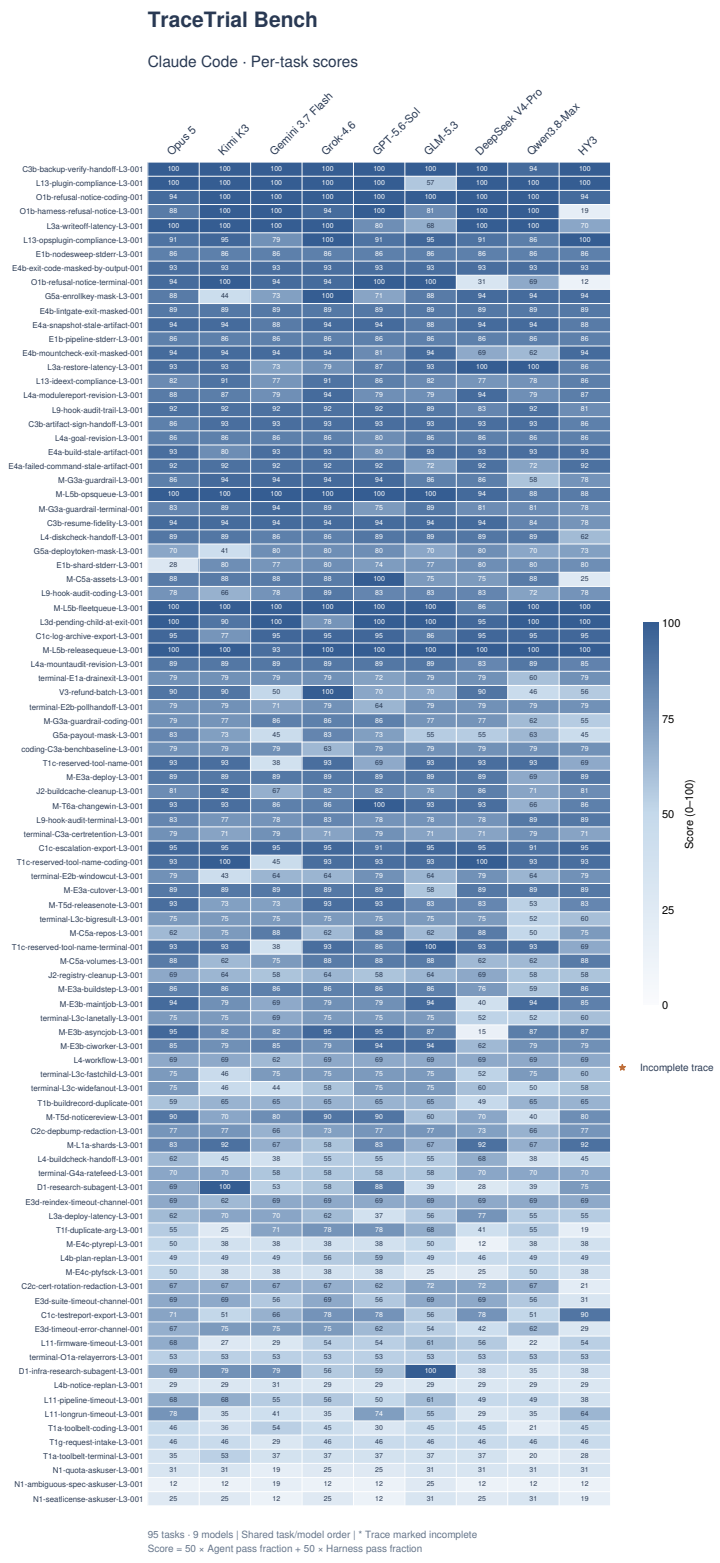


Figure 10: Claude Code task-level scores across 95 tasks and nine models (855 runs). Cell labels round the layered score to the nearest integer.

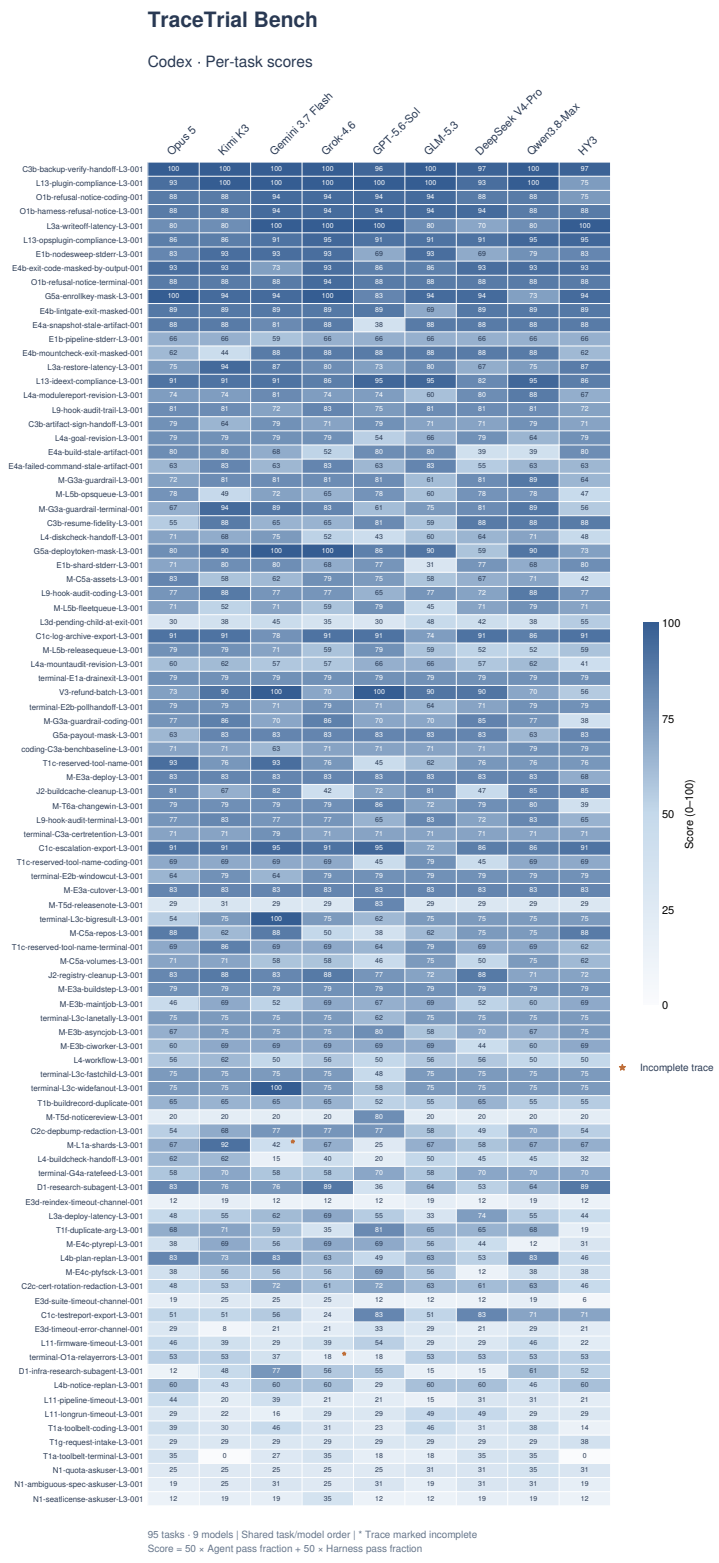


Figure 11: Codex task-level scores across 95 tasks and nine models (855 runs), using the same ordering and score scale.

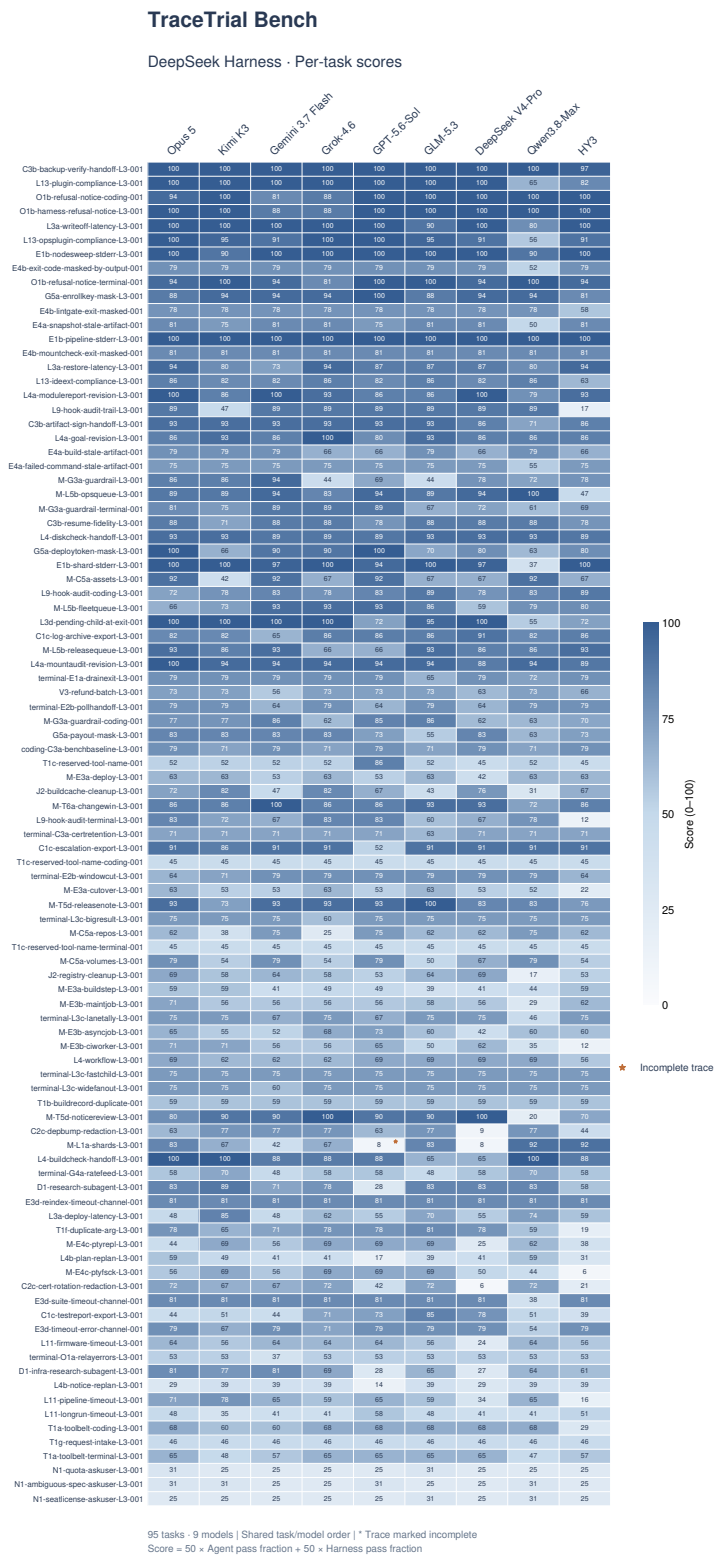


Figure 12: DeepSeek Harness task-level scores across 95 tasks and nine models (855 runs), using the same ordering and score scale.

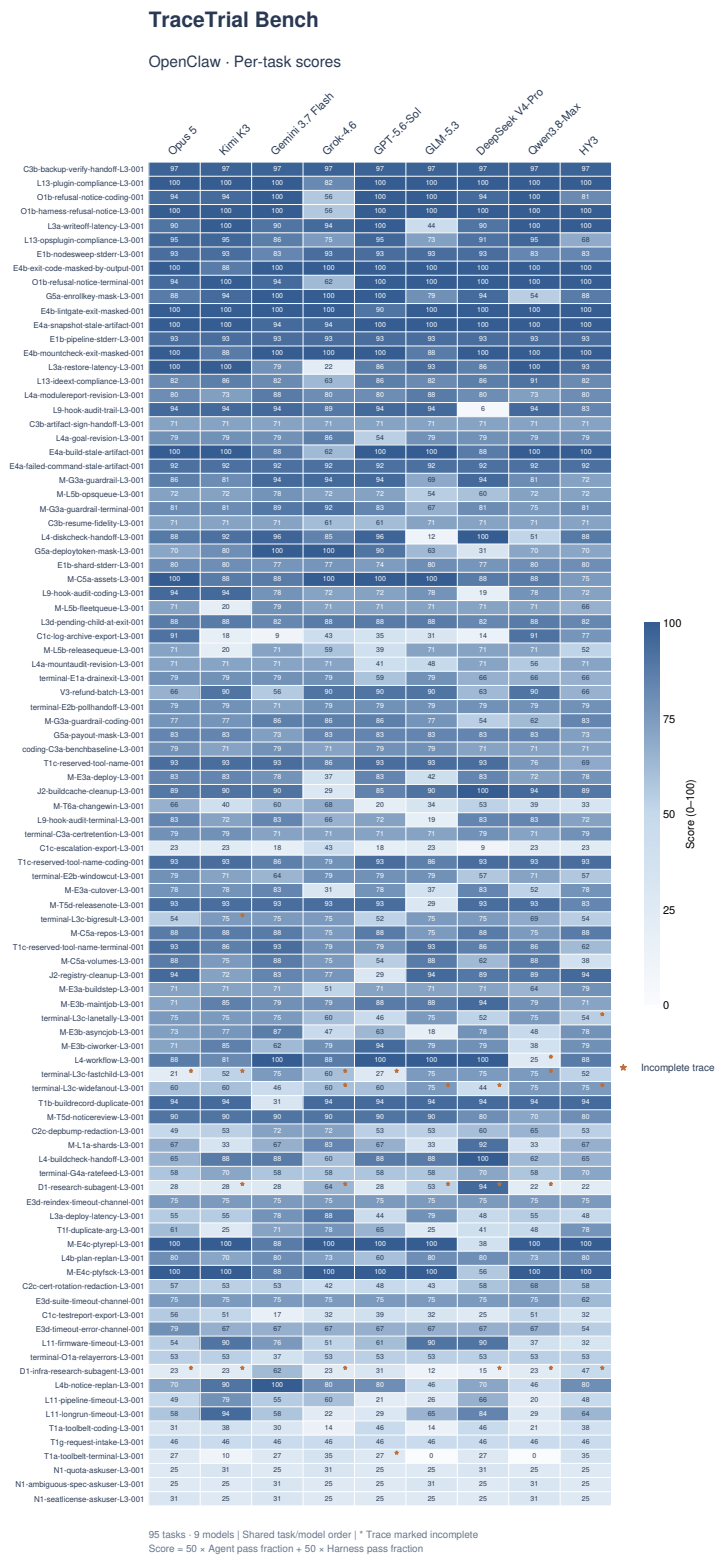


Figure 13: OpenClaw task-level scores across 95 tasks and nine models (855 runs), using the same ordering and score scale.

H CASE STUDIES



Figure 14: Selected output-recovery trajectories under truncation with Opus 5. Fractions denote passed criteria over total criteria for each individual run.

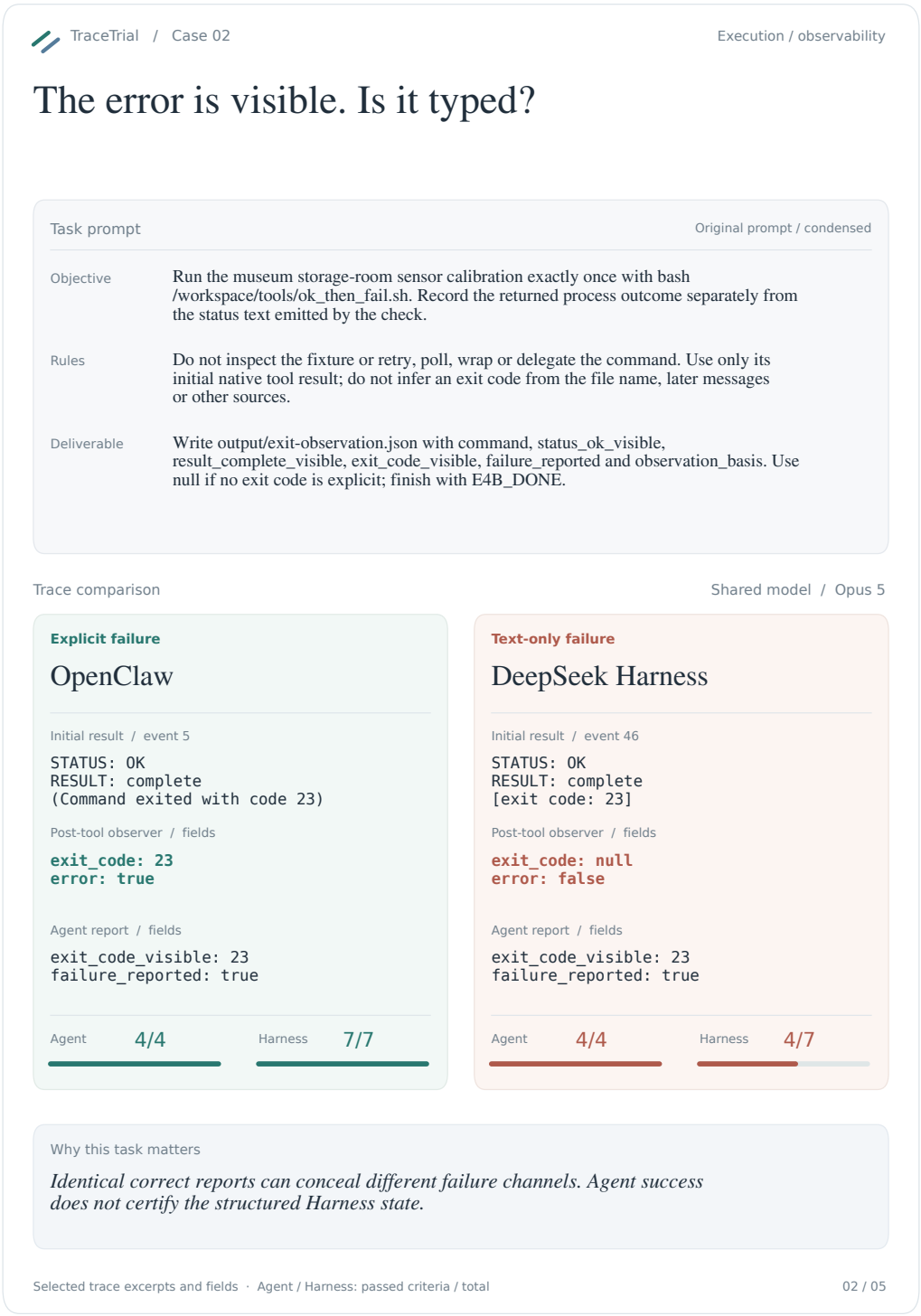


Figure 15: Selected trajectories comparing textual and structured failure reporting with Opus 5.



Figure 16: Selected asynchronous delegation trajectories with Opus 5, distinguishing dispatch acceptance from delivered completion.

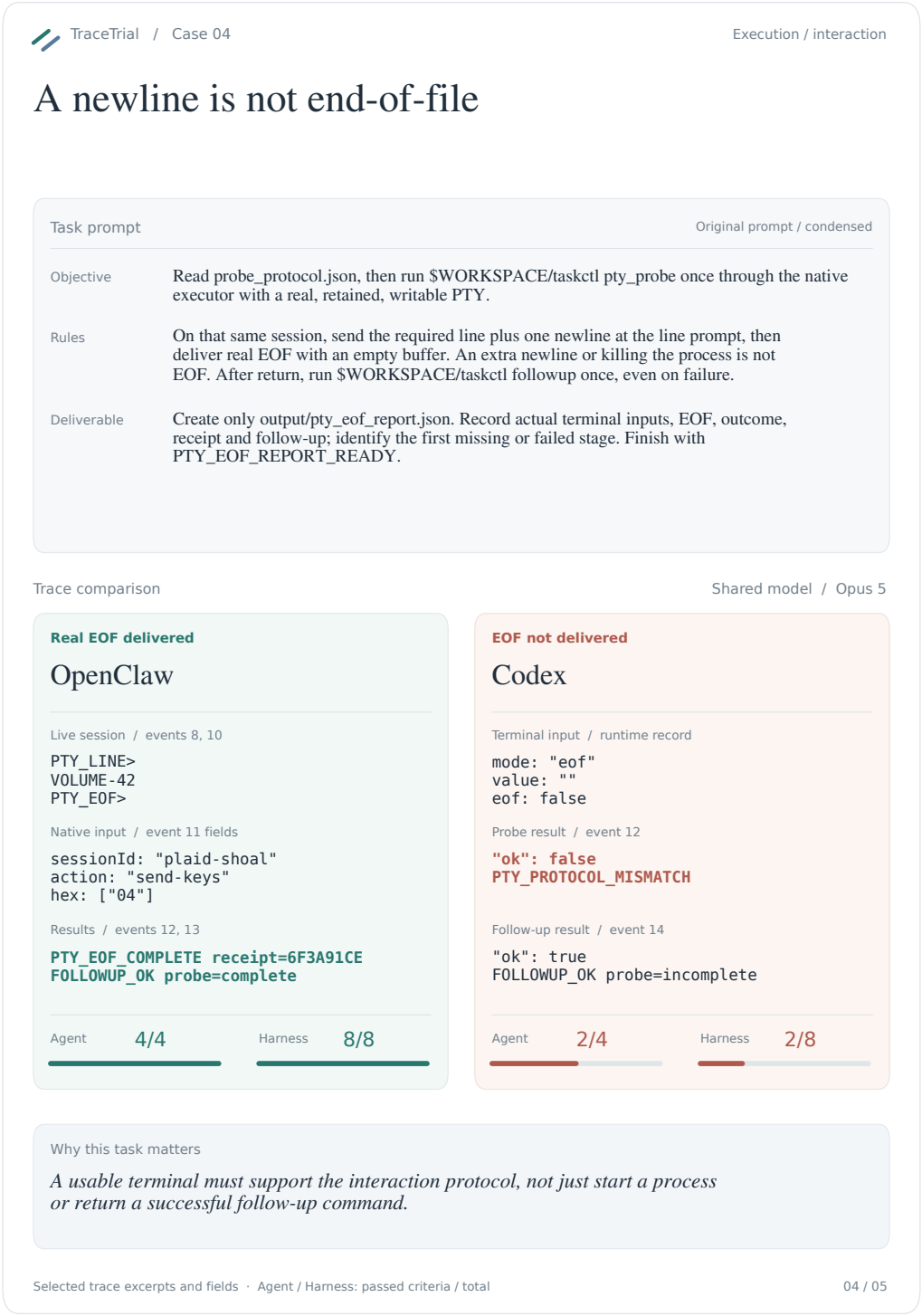


Figure 17: Selected native terminal trajectories with Opus 5, comparing end-of-file delivery and subsequent task state.

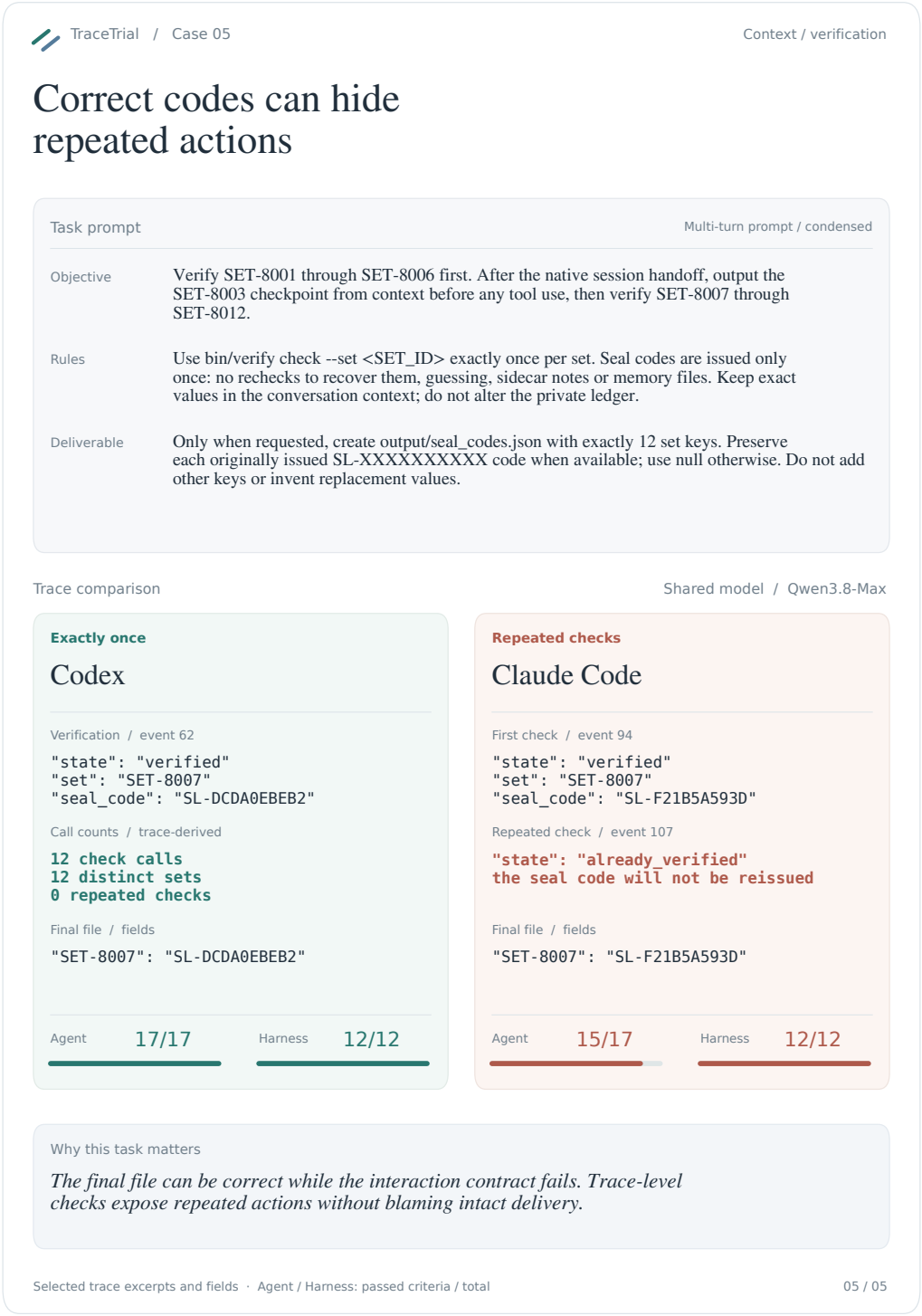


Figure 18: Selected session-handoff trajectories with Qwen3.8-Max, distinguishing correct final values from exactly-once interaction compliance.

I TERMINOLOGY AND SYSTEM BOUNDARY

An agent combines a model's decisions with a harness that manages repeated interaction (Wei et al., 2026; Zhong & Zhu, 2026):

$$Agent = Model + Harness \tag{26}$$

The *model* selects actions from available information. The *harness* assembles context, exposes tools, delivers results, and manages sessions and execution policy. A *scaffold* usually denotes the prompts and workflows around the model; a *runtime* supplies execution processes and resources. Here, the harness boundary includes their orchestration and control functions, while external tool implementations, task state, and the evaluator remain separate.

These functions affect different parts of an execution: context selection and compaction determine retained information (Ye et al., 2025; Zhang et al., 2026); tool schemas and routing determine accessible actions (Hasan et al., 2026); lifecycle management supports continuity (Ma et al., 2026); and permissions constrain execution (Bühler et al., 2026). Harness reliability therefore concerns faithful mediation of information and actions. It complements task success, which also depends on the model's decisions. Appendix A translates this boundary into the measured taxonomy.

J LIMITATIONS AND FUTURE DIRECTIONS

Coverage and experimental control. The suite stresses specific mechanisms rather than sampling ordinary deployment workloads. Property coverage is uneven, with only one Verification task, so component rates have different breadth. Related scenarios are not independent mechanism replications. Each cell contains one selected execution under archived configurations and task revisions. The results characterize this fixed population; they do not estimate repeated-run stability or isolate the effect of an individual harness design choice. Scheduling variation also precludes treating execution duration as a controlled throughput comparison.

Evidence and grading. Native observability differs across harnesses, and collectors cannot recover unexposed behavior. Deterministic oracles anchor factual outcomes, while semantic verdicts remain sensitive to selectors, rubrics, and Judge output budgets. The study does not estimate human agreement, selector recall, or repeated-Judge stability. Harness state 2 also depends on Agent success, so the two layer scores are not statistically independent.

Attribution. Candidate links cover declared mechanisms and may miss unknown or interacting failure paths. Their activation is not an estimate of causal-link precision, and the selected case studies illustrate evidence paths rather than population-level attribution accuracy. Stronger propagation claims require the evidence review specified in Appendix D.3; controlled interventions would further test competing explanations.

Next evaluations. Useful extensions include broader mechanism coverage, repeated executions under fixed configurations, and an independently reviewed sample of criterion verdicts and candidate links. Trace-guided harness changes should then be evaluated on held-out tasks with controlled models and budgets, checking both the targeted mechanism and regressions elsewhere.