# EVALUATING AGENTS IN LIVE, USER-FACING TASKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

AI agents are increasingly expected to complete complex, real-world tasks on the live web, where success depends not only on open-ended reasoning but also on information that changes over time. Products fluctuate in price and availability, listings appear and disappear, deadlines pass, and policies are revised. Consequently, many web tasks have dynamic, open outcome spaces: they admit multiple substantively different valid outcomes, and the set of valid outcomes itself evolves with the state of the web. Existing benchmarks typically simplify one or both dimensions, or leave both to a single judge. They may allow agents to choose their own solution paths while evaluating against a pre-authored answer or canonical final state, or preserve reproducibility by freezing web content and avoiding time-sensitive tasks, or grade live-web outcomes with one LLM judge that handles constraints, evidence, and quality together. We introduce LIVEOUTCOME, a benchmark of 78 human-authored, user-facing tasks for evaluating agents against the current state of the web. Its tasks span research, comparison, planning, decision support, data collection, and artifact creation, with outcomes that may change across days, weeks, or months. Each task is paired with a hidden, property-based success specification that remains stable over time without enumerating a fixed answer. LIVEOUTCOME evaluates submissions through deterministic checks of explicit constraints, live retrieval and verification of source-dependent claims against the pages the agent cites, and rubric-based assessment of the resulting deliverable that incorporates the verified findings. We benchmark current agents and validate the evaluation protocol using human judgments of both LLM stages, repeated evaluation under fixed evidence, and controlled grounding failures with unmodified controls. Across 13 agent–model configurations, the strongest system scores 80.6 and passes 61.5% of tasks, and most failures stem from incomplete or incorrect content rather than missing evidence. By separating stable success criteria from a changing set of web-grounded outcomes, LIVEOUTCOME enables rigorous evaluation of whether agents can solve open-ended tasks as the web exists at execution time.

## 1 INTRODUCTION

Many AI agent tasks admit multiple valid outcomes rather than a single predetermined answer (Sharma et al., 2025; Lin et al., 2026; Wang et al., 2026). Consider the request illustrated in Figure 1: find a waterproof backpack with a capacity of at least 25 liters that costs less than $75 and is currently available. The user has no particular product in mind: multiple backpacks may satisfy the request, and the qualifying set may change as prices fluctuate, availability change, or listings disappear. Success therefore depends on whether the recommendation satisfies the request under current conditions, not whether it matches a reference product.

Such tasks pose a distinct evaluation problem: a request specifies the properties of a successful outcome without enumerating the valid outcomes. The outcome space is therefore open yet bounded: agents may return different final results, but each must satisfy the same constraints and deliverable requirements. Evaluating such tasks requires specifying the success criteria before execution without fixing an answer, then determining whether the agent's chosen outcome satisfies those criteria. For the backpack request, the evaluator must verify the recommended product's price, waterproofing, capacity, and availability against current web evidence. We call such tasks *criterion-defined* and *outcome-open*.



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.

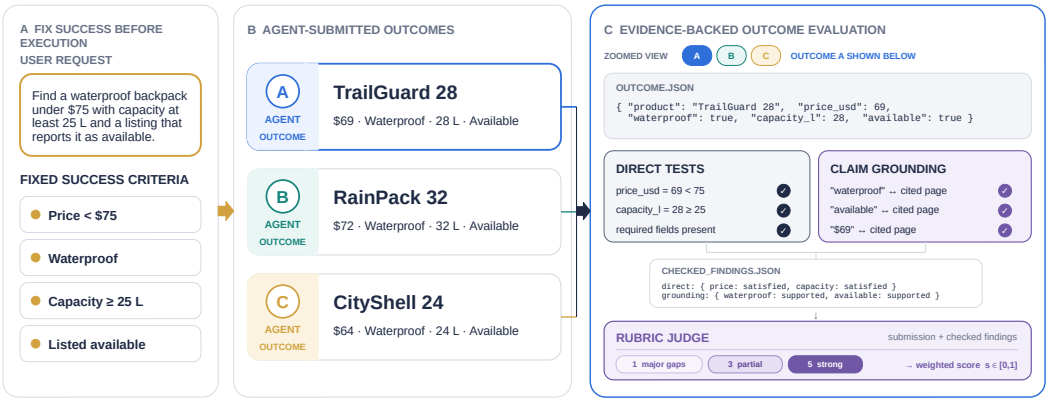



Figure 1: Overview of LIVEOUTCOME framework. A user request is paired with fixed success criteria; agents may submit different outcomes, which are evaluated through deterministic direct test, source-grounded checks, and rubric-based component scoring.

Existing agent benchmarks differ in what they evaluate and what evidence they use. Mind2Web (Deng et al., 2023) compares predicted actions with human demonstrations on recorded webpage. WebArena (Zhou et al., 2024) checks task completion in a reproducible web environment, while OS-World (Xie et al., 2024) uses task-specific execution checks in configured computer environments. CocoaBench (Hao et al., 2026) selects tasks with external dependencies expected to remain stable and scores final outputs using task-specific programs. Odysseys (Jang et al., 2026) builds tasks from human browsing histories and evaluates agent trajectories with rubrics that often specify websites and intermediate workflow checkpoints. This trajectory-anchored design can penalize alternative valid workflows and provides only indirect evidence that the final outcome itself satisfies the user's constraints. Together, these benchmarks leave open how to evaluate different agent-selected outcomes against fixed success criteria when their supporting web evidence may change.

To bridge this gap, we introduce LIVEOUTCOME, 78-task benchmark for *criterion-defined* and *outcome-open* tasks on the live web. The tasks span eight domains: consumer products and shopping; travel and local services; food, culture and entertainment; research; education and careers; business; health and public affairs; software and data work. For each task, human authors write a natural user request and define a hidden, task-specific success criteria before agent execution without enumeration acceptable answers. Agents remain free to choose their own sources, workflows, and submitted outcomes. LIVEOUTCOME evaluates each agent submission in three stages: (1) deterministic direct test verify explicit constraints and completeness requirements; (2) source-grounded checks verify against the pages the agent itself cites, captured at execution time; and (3) rubric-based component scoring assesses the quality of the user-facing deliverable while incorporating findings from (1) and (2). This evaluation accommodates substantively different valid outcome while identifying constraint violations, unsupported claims, and incomplete deliverable. We benchmark current agents and validate both the grounding and component scorer with independent human judgments.

We conduct a comprehensive evaluation of 13 agent configurations across the 78 tasks included in the main evaluation (§5). Codex with GPT-5.6 Sol performs the best, achieving an overall score of 80.6 and satisfying our pass criterion on 61.5% of tasks. Nevertheless, only 29.2% of all runs pass, indicating substantial room for improvement in real-world agent use. Among failed outcomes, 51.7% primarily contain incomplete, incorrect, constraint-violating, or insufficiently analyzed content, while 31.9% primarily lack a required, usable, or internally consistent deliverable. Evidence-support problems constitute the remaining 16.4% of primary failures. Notably, claim–source gaps appear in 51.2% of failed outcomes once co-occurring failures are included.

## 2 RELATED WORK

**AI agent benchmarks.** Benchmarks for AI agents differ in both their interaction environments and their evaluation targets. AgentBench (Liu et al., 2025) uses environment-specific success measures

across code, game, and web-grounded settings, while GAIA (Mialon et al., 2023) and Assistant-Bench (Yoran et al., 2024) evaluate final responses against collected reference answers. Mind2Web (Deng et al., 2023) evaluates predicted actions against recorded human demonstrations, whereas WebArena (Zhou et al., 2024) and OSWorld (Xie et al., 2024) use reproducible environments environments and executable checks of the resulting state, allowing different trajectories or implementations taht achieve the specific goal. More recent benchmarks have broadened evaluation along different dimensions. MCP-Atlas (Bandi et al., 2026) and Toolathlon (Li et al., 2026) evaluate multistep tool use within predefined server or applicataion ecosystems, whereas CocoaBench (Hao et al., 2026) allows flexible tool composition but applies deterministic task-specific evaluator to final outputs. Online-Mind2Web (Xue et al., 2025) and Odysseys (Jang et al., 2026) assess live-web executions using trajectory-based judgments or graded rubric checkpoints. LIVEOUTCOME focuses specifically on the submitted, user-facing outcome on the live web, where the concrete result is selected by the agent rather than enumerated in advance, and is evaluated through explicit constraints, verification against current web evidence, and deliverable-specific criteria.

**Open-ended outcome evaluation.** AI agents often produce reports, code bases, or other artifacts for which matching a canonical answer is insufficient. Executable benchmarks such as SWE-Bench (Jimenez et al., 2024) evaluate software outcomes through unit tests, while PaperBench (Starace et al., 2025) grades research-replication repositories against author-validated hierarchical rubrics. Agent-as-a-Judge (Zhuge et al., 2024) similarly evaluates AI-development projects against hierarchical user requirements by locating supporting evidence within the generated artifacts. For web-grounded research, Mind2Web 2(Gou et al., 2025) uses tree-structured judges to assess answer correctness and source attribution; ResearchRubrics (Sharma et al., 2025) uses expert-written criteria for grounding, reasoning, completeness, and citation use; and LiveResearchBench (Wang et al., 2026) use multi-level rubrics to evaluate dynamic search reports. These works enable fine-grained evaluation of complex digital outputs, but primarily focus on executable software or long-form research reports. LIVEOUTCOME extends this direction to heterogeneous, user-facing outcomes on the live web, including recommendations, plans, data collections, and artifact creations.

# 3 LIVEOUTCOME

## 3.1 BENCHMARK TASKS

**Task scope and composition.** LIVEOUTCOME comprises 78 human-authored tasks framed as complete user-facing goals across eight application domains: consumer products and shopping; travel and local services; food, culture and entertainment; research; education and careers; business; health and public affairs; software and data work. We classify tasks by their primary goal: lookup and extraction, comparison and selection, planning, analysis and audit, and deliverables and tools. Tasks may combine several activities; the primary goal describes what the user ultimately requests rather than every step needed to finish it. Different sources, options, plans, or artifacts may satisfy a request, provided the submitted outcome meets its requirements.

**Task validation.** For each task, we manually reviewed the instruction and evaluation specification alongside pilot runs. We checked that the instruction clearly stated the user's goal and deliverables, that the success criteria captured the principal requirements, and that the structured outcome schema could express the information needed to assess them. We exercised task-specific direct tests on positive and negative cases, and checked whether grounding recovered task-relevant content from web pages captured during execution. We also reviewed whether the rubric covered the success criteria, evidence assembled across sources, and qualitative aspects of the deliverable. These task-level checks assessed whether each request could be represented and evaluated; Section 3.2 separately examines the evaluator's agreement with human judgments and its robustness under controlled cases and repeated scoring.

## 3.2 OPEN-OUTCOME EVALUATION

**Success specification.** We decompose each user request into success criteria: conditions that an acceptable outcome must satisfy without matching a fixed reference answer. Each user request has 7.6 success criteria on average. We also provide an agent-visible outcome schema specifying how

agents submit the information needed to check those conditions. For example, a studio-rental task lets agents search public listings and recommend one to three properties. Its success criteria include "Every selected effective monthly cost and move-in cash value is exactly recomputed and satisfies its cap" and "Each claimed candidate rent quote and availability date is supported by its property-controlled exact listing page." The same criteria can evaluate different sets of listings.

**Criterion verification.** We evaluate each criterion through one of three routes, according to what must be judged: (1) deterministic direct tests handle deterministic properties of agent's submission, such as arithmetic, cross-field consistency, and artifact structure. (2) source-grounded checks determine whether submitted factual claims are supported by the exact web pages the agent cites; a page that cannot be accessed yields an unverified finding rather than a factual rejection. And (3) rubric-based component scoring measures qualitative judgment or combined evidence. In the rental task, for example, the cost calculation is checked directly, rent and availability are grounded in listing pages, and the quality of the resulting recommendation is assessed through its rubric component.

**Component scoring.** Each task is divided into a set of predefined outcome components, each score on a five-point scale. For every component, we provide rubric descriptions for scores of 1, 3, and 5: a score of 1 indicates that the component is missing or substantially unsatisfied; a score of 3 indicates partial fulfillment; and a score of 5 specifies what full and satisfactory fulfillment requires. Figure 2 shows three representative examples of score-5 anchors. Scores of 2 and 4 represent performance between adjacent anchors. At evaluation time, the component-scoring judge applies these rubrics to the submitted outcome and deliverable, together with the findings from deterministic direct test and source-grounded checks. The rubrics capture qualities of agent's submitted outcome where individual direct test or grounded checks cannot fully assess, such as whether evidence from several sources supports a coherent recommendation and whether the final deliverable makes the trade-offs usable for the requester. The judge scores each component from 1 to 5; we combine these component scores using a weighted average, then scale the result to $[0, 1]$ to obtain the task-level reward. Appendix Figure 7 summarizes the information flow from task specification and submission to the final score.

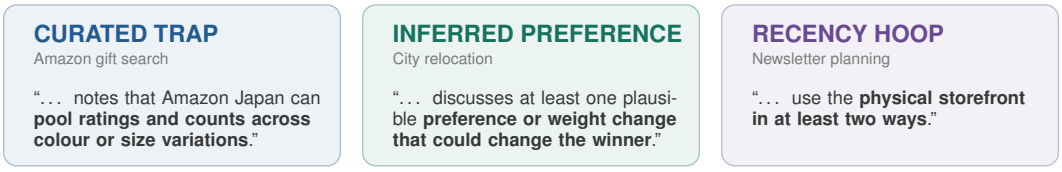



Figure 2: Examples of score-5 rubric requirements that go beyond the user request.

## 4 Evaluation Setup and Validation

### 4.1 Evaluation Setup

**Agents and evaluator.** We evaluate 13 agent–model configurations on all 78 tasks (1,014 agent–task runs). The execution model varies by configuration and is reported in Table 2: Codex uses GPT-5.6 Sol, Terra, and Luna at high reasoning effort and GPT-5.5 at low, medium, and high effort; OpenCode uses GPT-5.4 mini; Qwen Code uses GPT-5.4 mini and GPT-5.6 Luna; and Pi and Goose use GPT-5.6 Luna. Each agent may choose its own sources, tools, and workflow. For evaluation, the source-grounded checker and rubric-based component scorer both use GPT-5.4 at temperature 0.

**Execution environment and protocol.** We orchestrate runs with Harbor 0.1.45 in isolated Docker containers with internet access. Each container is allocated two CPU cores, 4 GB of memory, and 10 GB of storage. Model inference is served through remote APIs, while orchestration, artifact collection, and analysis run locally. Each agent receives one attempt per task with a 3,000-second execution budget. We use a two-turn protocol: the first turn provides the user request, and the second provides the outcome schema used to submit the final structured result.

**Scoring and aggregation.** For each run, we aggregate the component scores using the task-specific weights and normalize the result to $[0, 1]$. Runs that time out without a valid submission or have missing, unparseable, or schema-invalid outputs receive a score of zero and remain in the average. We also report an operational rubric pass: A run passes if its overall score is at least 0.80 and every component score is at least 3. The overall threshold corresponds to a weighted average rating of 4. The component floor prevents a score below 3 from being hidden by stronger components. We also report threshold sensitivity in Appendix A.1.

### 4.2 Evaluation Validation

**Source-grounded validation.** We conduct human validation of the grounding agent by sampling 90 claim-URL pairs stratified by the agent's original prediction, with 30 pairs each from *matched*, *partial match*, and *not matched*. Two annotators independently assess whether each submitted live webpage supports its associated claim, assigning one of the same three verdicts; the task instruction is provided only as context. The mean of the human-judge linearly weighted Cohen's $\kappa$ values is 0.695; the mean of the two inter-annotator $\kappa$ values is 0.850. [1] These results show that the grounding agent shows substantial agreement with human evidence assessments, supporting its use to verify claims against submitted webpages.

**Rubric judge validation.** We separately validate the rubric judge at the component level by sample 16 tasks from our benchmark, yielding 72 component-level scoring cases. Two annotators independently receive the task instruction, the submitted outcome, the outcome excerpt evaluated by the scorer, the component rubrics and its 1/3/5 anchor description, and the fixed direct-check and grounding findings provided to the scorer. To maintain the same evidence boundary, the annotators do not inspect the execution trajectory, visit live webpage, or access files that were unavailable to the scorer. Each annotator assigns a score from 1 to 5. The mean of the human-scorer linearly weighted $\kappa$ values is 0.644; the mean of the two inter-annotator $\kappa$ values is 0.606. Human-scorer agreement is comparable to inter-annotator agreement, indicating that scorer reproduces human component rating at a level similar to agreement between annotators. Details in human validation are provided in Appendix C.

Table 1: Stability of GPT-5.4 evaluator variants on 50 frozen traces, with five evaluations per trace. Lower standard deviation indicates greater run-to-run stability.

| Evaluator variant | Mean reward | Mean SD ↓ | Median SD ↓ | Max SD ↓ |
|---|---|---|---|---|
| Rubric-free baseline | 0.6096 | 0.0451 | 0.0000 | 0.1960 |
| Rubric-only baseline | 0.6671 | 0.0264 | 0.0208 | 0.0911 |
| **Full evaluator (ours)** | 0.5010 | **0.0141** | **0.0089** | **0.0759** |

**Evaluator stability.** We evaluate score stability in two complementary settings. First, we run the complete evaluator five times on fixed evidence from the same 78-task run. Forty-three tasks receive identical scores across all five evaluations, and the estimated standard error of the aggregate run mean is 0.0024 (Table 4). This measures absolute variation in the benchmark-level score when both agent outputs and retrieved evidence are held fixed.

We then test whether the evaluator design contributes to this stability by comparing it with two simplified baselines on the same 50 frozen agent traces. Every trace is evaluated five times by GPT-5.4 at temperature zero. The rubric-free baseline receives the task instruction, complete agent trace, and submitted JSON summary, but no rubric or success-criterion results, and returns a single holistic score. The rubric-only baseline additionally receives the final component rubric but not the deterministic checks, grounding results, or other success-criterion outcomes. Our full evaluator receives the component rubric and the complete success-criteria pipeline. As shown in Table 1, its mean within-trace standard deviation is 0.0141, compared with 0.0264 for the rubric-only baseline and 0.0451 for the rubric-free baseline—reductions of 46.6% and 68.7%, respectively. Thus, the rubric improves stability over holistic scoring, and the complete evidence-aware pipeline reduces variation further.

[1] Both annotation scales are ordinal. Linearly weighted Cohen's $\kappa$ (Cohen, 1968) penalizes disagreements according to their distance on the scale.

**Temporal sensitivity.** Figure 3 shows three representative case of our evaluators sensitivity to time when we evaluate 1 week after the original agent task attempt. This design ensures we evaluate agent against live-web information that minimizes the risk of agent short-cutting with pre-train priors.

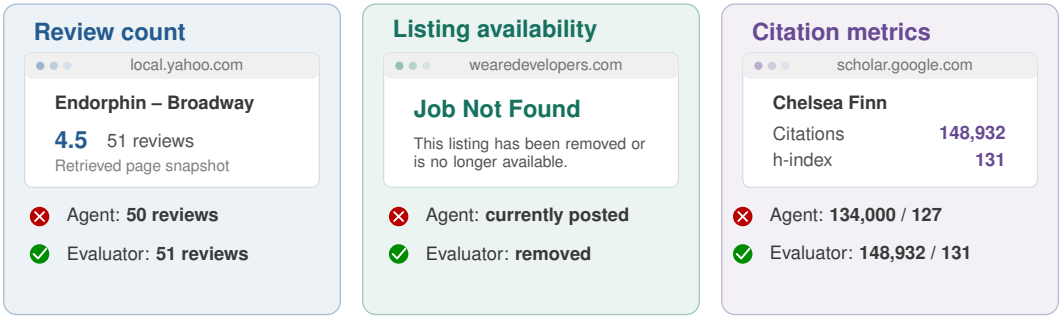



Figure 3: Trace snapshots of submitted and retrieved values. Times show the delay from agent output to evaluator capture.

**Grounding under seeded faults.** We inject citation, value, and entity faults into 472 frozen claim–page records that were originally matched. Counting either partial or unsupported as detection, our evaluator identifies at least 85.8% of every fault class: 97.4% of cross-domain citation swaps, 88.9% of same-site swaps, 85.8% of value perturbations, and 90.8% of entity substitutions. Under the stricter, score-affecting unsupported verdict, it detects 78.4% overall while flagging only 5.0% of unmodified controls. Appendix A.2 compares these results with less structured evaluators, which achieve lower strict recall.

# 5 Main Results

## 5.1 Agent Leaderboard

Table 2 reports results for 13 agent–model configurations on 78 tasks. Codex with GPT-5.6 Sol achieves the highest rubric judge score of 80.62. Codex with GPT-5.5 Medium ranks second with 77.11, followed by GPT-5.5 High with 76.50. Sol leads the second-ranked configuration by 3.51 points. The 95% stratified task-bootstrap interval for this difference is [0.79, 6.36]. Sol ranks first in 99.3% of the bootstrap samples. Sol also leads three of the five task categories: comparison and selection, analysis and audit, and deliverables and tools. GPT-5.6 Terra leads lookup and extraction, while GPT-5.5 Medium leads planning. These results show that Sol's overall lead is supported by strong performance across several task types.

We also report a rubric-based pass rate. A run passes if its rubric judge score is at least 80 and every component score is at least 3. Under this rule, 296 of 1,014 runs pass (29.2%). GPT-5.6 Sol achieves the highest pass rate, with 48 of 78 tasks (61.5%). It is followed by GPT-5.5 Medium with 35 tasks (44.9%) and Claude Opus 5.5 with 33 tasks (42.3%). The score and pass-rate rankings are strongly aligned (Spearman's $\rho = 0.916$; Kendall's $\tau_b = 0.813$).

## 5.2 Failure Mode Analysis.

We analyze 687 failed outcomes drawn from 13 agent–model settings. Using a fixed prompt, a GPT-5.5 Medium judge assigns one primary failure and any distinct co-occurring failures to each outcome. We use three outcome-level categories: content adequacy, deliverable integrity, and evidence support. Content adequacy covers incomplete, incorrect, constraint-violating, or insufficiently analyzed results. Deliverable integrity covers missing, unusable, or inconsistent outputs. Evidence support covers missing, invalid, or claim-mismatched support. The full taxonomy and classification prompt are provided in Appendix D.

Figure 4(a) shows that content adequacy accounts for 51.7% of primary failures, followed by deliverable integrity at 31.9% and evidence support at 16.4%. The most common primary subtypes are insufficient analysis (C4; 17.5%), invalid format (D2; 15.6%), claim–source gaps (E2; 14.6%),

Table 2: Leaderboard across 13 agent–model configurations and five task categories. Pass requires a rubric judge score of at least 80, all component scores to be available, and every component score to be at least 3.

| Agent | Model | Effort | Rubric Judge Score | Pass Rate (%) | Lookup & Extraction (n=16) | Comparison & Selection (n=25) | Planning (n=13) | Analysis & Audit (n=11) | Deliverables & Tools (n=13) | Runtime mean ± SD (m:ss) |
|---|---|---|---|---|---|---|---|---|---|---|
| Codex | GPT-5.6 Sol | High | **80.6** | **61.5** | **82.5** | **77.1** | 82.0 | **76.8** | **86.8** | 7:38 ± 5:12 |
| Codex | GPT-5.5 | Med | 77.1 | 44.9 | 81.4 | 71.8 | **82.8** | 71.9 | 80.7 | 4:37 ± 2:33 |
| Codex | GPT-5.5 | High | 76.5 | 38.5 | 77.5 | 72.3 | 79.3 | 74.2 | 82.5 | 6:07 ± 2:38 |
| ClaudeCode | Opus 5.5 | High | 74.8 | 42.3 | 76.5 | 74.9 | 77.3 | 67.4 | 76.5 | 7:59 ± 6:34 |
| Codex | GPT-5.6 Terra | High | 73.8 | 35.9 | 82.8 | 73.8 | 64.0 | 66.6 | 78.7 | 8:04 ± 4:08 |
| Codex | GPT-5.5 | Low | 71.5 | 37.2 | 75.4 | 67.5 | 76.6 | 58.5 | 80.1 | 2:56 ± 1:34 |
| QwenCode | Qwen3.7-Plus | – | 53.6 | 12.8 | 53.7 | 51.4 | 58.6 | 50.4 | 55.8 | 8:18 ± 5:55 |
| Codex | GPT-5.6 Luna | High | 48.8 | 33.3 | 47.7 | 41.2 | 55.3 | 42.6 | 63.8 | 8:00 ± 4:31 |
| OpenCode | GPT-5.4 mini | – | 47.8 | 19.2 | 58.0 | 62.4 | 33.0 | 30.1 | 36.8 | 14:26 ± 14:01 |
| Pi | GPT-5.6 Luna | – | 44.7 | 15.4 | 46.3 | 41.9 | 32.6 | 51.1 | 54.8 | 2:09 ± 1:11 |
| QwenCode | GPT-5.6 Luna | – | 35.0 | 17.9 | 42.2 | 37.3 | 40.6 | 18.2 | 30.0 | 3:19 ± 3:40 |
| Goose | GPT-5.6 Luna | – | 34.2 | 12.8 | 35.2 | 27.5 | 35.7 | 23.4 | 53.3 | 2:46 ± 1:42 |
| QwenCode | GPT-5.4 mini | – | 27.2 | 7.7 | 39.7 | 32.3 | 20.9 | 21.2 | 13.5 | 1:13 ± 0:38 |

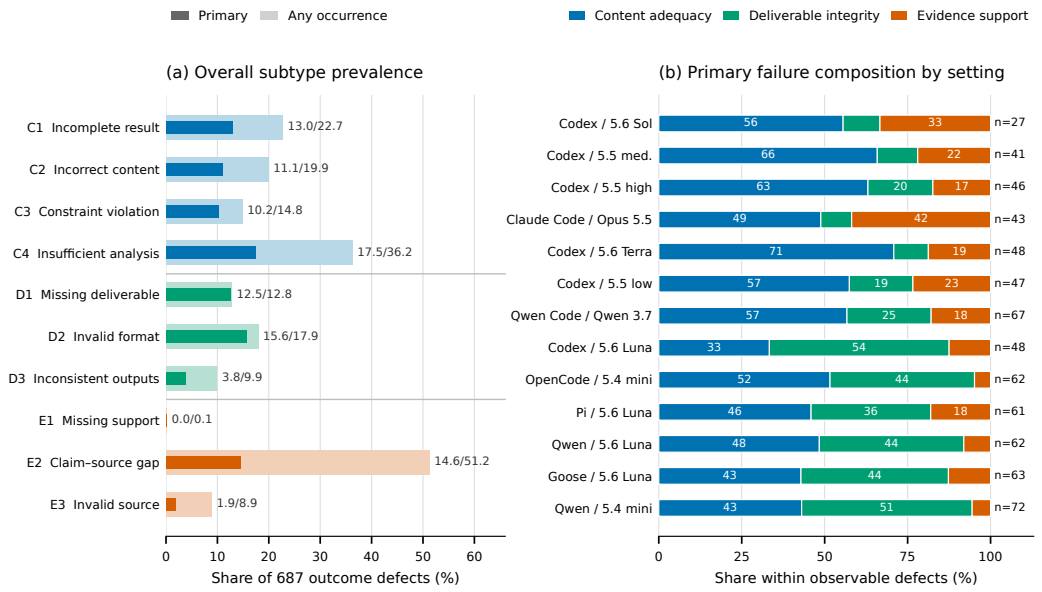



Figure 4: Outcome-level failure analysis over 687 failed outcomes. **(a)** Prevalence of each failure subtype. Dark bars show primary failures, while light bars include both primary and co-occurring failures. Each outcome has exactly one primary label but may have multiple co-occurring labels. **(b)** Composition of the three primary failure categories within each agent–model setting. Each row sums to 100%, and $n$ denotes the number of failed outcomes in that setting. Panel (b) describes failure composition conditional on failure, rather than the absolute probability of failure.

and incomplete results (C1; 13.0%). Together, content and deliverable failures account for 83.6% of primary failures. This shows that the main observed bottleneck remains producing a complete, correct, and usable result. Failures also overlap. Overall, 57.9% of failed outcomes receive more than one subtype. Claim–source gaps rise from 14.6% as the primary failure to 51.2% as any observed failure. Insufficient analysis rises from 17.5% to 36.2%. These defects often accompany another dominant problem.

Figure 4(b) shows that failure composition varies across settings. Content failures dominate most settings, while some settings show larger shares of deliverable or evidence-support failures. These differences suggest that systems require different improvements in reasoning, finalization, and claim–source verification.

Table 3: Scores across four web-access settings on the same 25 tasks.

| Configuration | Browser | Shell network | Score |
|---|---|---|---|
| Host, search only | no | blocked | 0.774 |
| Host, computer use | **yes** | blocked | **0.784** |
| Container, 3000s budget | no | full | 0.790 |
| Container, 900s budget | no | full | 0.735 |

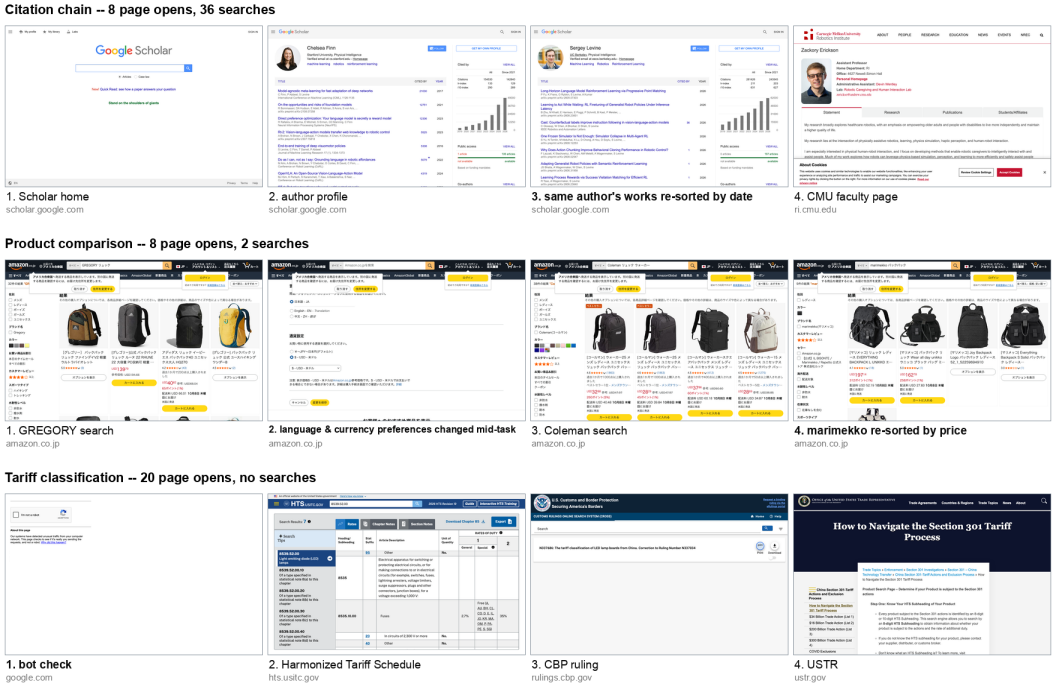



Figure 5: Three execution trajectories from the computer-use runs.

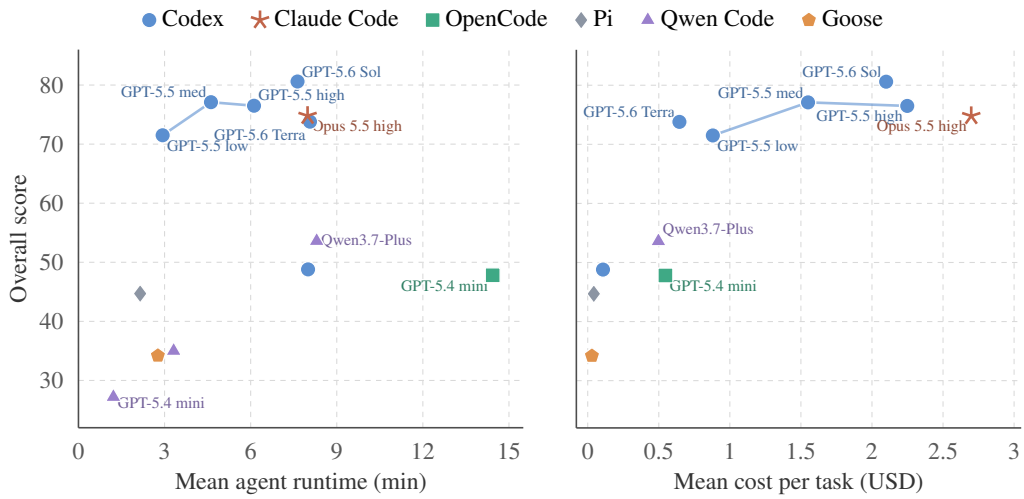
### 5.3 Computer use versus retrieval-only execution

We ran Codex with GPT-5.6 Sol on the same 25 tasks under four web-access settings: host execution with search only, host execution with computer use, and browser-free containers with either a 900- or 3,000-second budget. All other settings were fixed.

Computer use changed how the agent gathered evidence. Figure 5 shows three such paths: a citation chain that opens author profiles on Google Scholar and re-sorts a publication list by date before following the chain to a university faculty page; a product comparison that changes the store's language and currency settings mid-task and re-sorts a listing by price while working across four brands; and a tariff classification that clears a bot check and then moves between the Harmonized Tariff Schedule, a CBP ruling, and USTR guidance. Agent manipulates page state—sorting, filtering, and changing site preferences—rather than retrieving a fixed URL, and the balance between navigation and search varies sharply by task. Yet, the changes had little effect on score: the computer-use and search-only host runs scored 0.784 and 0.774, respectively.

### 5.4 Cost analysis

Figure 6 compares overall score with mean inference cost and runtime. The average cost ranges from $0.03 to $2.25 per attempted task. Higher cost does not always lead to a higher score. Codex with GPT-5.6 Sol achieves the highest score of 80.6 at $2.10 per task. GPT-5.6 Terra reaches 73.8 at only $0.65 per task. GPT-5.5 Medium also outperforms GPT-5.5 High, although it costs less ($1.55 versus $2.25 per task).




(a) Runtime (b) Cost

Figure 6: Overall score versus (a) mean agent runtime and (b) mean cost per task. Color and marker indicate the agent; labels give the model, and unlabeled points use GPT-5.6 Luna. The line connects GPT-5.5 across reasoning effort levels. Claude Code cost is computed from the run's own token counts at list prices.

The GPT-5.5 configurations show diminishing returns from additional reasoning effort. Moving from Low to Medium increases the score from 71.5 to 77.1 and raises the cost from $0.88 to $1.55 per task. Moving from Medium to High further raises the cost to $2.25, but the score decreases slightly to 76.5. More reasoning effort therefore does not consistently improve performance.

Agent scaffolding also affects cost efficiency. This effect is visible when the backbone is fixed to GPT-5.6 Luna. Goose, Pi, and Codex cost $0.03, $0.04, and $0.11 per task, respectively. Their scores are 34.2, 44.7, and 48.8. The more expensive scaffolds achieve higher scores in this comparison, but the improvement is not proportional to the additional cost. Cost and runtime also capture different forms of efficiency. For example, OpenCode with GPT-5.4 mini is slower and more expensive than Codex with GPT-5.6 Luna, while achieving a slightly lower score. These results motivate reporting monetary cost and runtime together rather than treating either measure as a complete account of agent efficiency.

## 6 Conclusion

We introduced LiveOutcome, a benchmark of 78 human-authored tasks for evaluating agent-selected outcomes on the live web. Each task specifies success criteria before execution without prescribing a single valid answer. Evaluation combines direct checks, verification against cited webpages, and component-level scoring of the submitted deliverable. Across 13 agent configurations, and the strongest agent configuration scores 80.6 and passes 61.5% of tasks. Content deficiencies and deliverable failures account for 83.6% of primary failures, indicating that producing complete and usable results remains a central challenge. Evidence support is also a recurring problem: 51.2% of failed outcomes include claims inadequately supported by their cited sources.

### AI use statement

We used generative AI tools solely for language editing to improve the grammar, clarity, and readability of the manuscript. All AI-assisted edits were carefully reviewed to ensure that they preserved the intended meaning and technical accuracy. We take full responsibility for the final content of this work, including any text, claims, or artifacts produced with the assistance of generative AI.

### Ethics statement

LiveOutcome evaluates public-web research and local artifact production. Its tasks are designed to be read-only: agents may retrieve public information and create local files, but are not asked to make purchases or reservations, submit applications, contact people or businesses, publish content, or modify external accounts. The benchmark does not intentionally request private or sensitive personal data. Nevertheless, public pages can contain personal information, copyrighted material, and access restrictions; public availability should not be treated as permission to republish entire pages or bypass authentication and verification barriers. Reproductions should use conservative request rates, respect applicable site policies, and remove credentials, session tokens, and unnecessary personal information from released traces. Human validation in this study concerns model outputs and publicly accessible evidence, and does not make decisions about individuals. Finally, benchmark performance should not be interpreted as approval for autonomous deployment in consequential settings, where errors, stale information, and unintended external actions could cause harm.

### Reproducibility statement

We provide the task packages, including each user instruction, outcome schema, success criteria, direct checks, and anchored component rubric, together with the agent adapters, evaluation code, and experiment configurations. Section 4 reports the execution environment, model configurations, reasoning settings, per-task budget, and evaluator model; the appendix documents the evaluation workflow and validation analyses. Saved traces contain the submitted outcomes, artifacts, judge records, and retrieved evidence needed to audit a score. Because the benchmark operates on the live web, a fresh execution cannot be expected to reproduce the same prices, availability, counts, or accessible page content. We therefore run evaluation immediately after execution and preserve the retrieved evidence. Fixed-trace analyses reuse these captures, enabling reproduction of evaluator results separately from reproduction of the time-dependent agent run.

## References

Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, Manasi Sharma, Andrew Park, Martin Dimakis, Ernesto Gabriel Hernandez Montoya, Dan Rambado, Ivan Salazar, Rafael Cruz, MohammadHossein Rezaei, Chetan Rane, Ben Levin, Daniel Yue Zhang, Brad Kenstler, and Bing Liu. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers, 2026. URL https://arxiv.org/abs/2602.00933.

Jacob Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. Psychological bulletin, 70(4):213, 1968.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. arXiv preprint arXiv:2306.06070, 2023. URL https://arxiv.org/abs/2306.06070.

Boyu Gou, Zanming Huang, Yuting Ning, Yu Gu, Michael Lin, Weijian Qi, Andrei Kopanev, Botao Yu, Bernal Jiménez Gutiérrez, Yiheng Shu, Chan Hee Song, Jiaman Wu, Shijie Chen, Hanane Nour Moussa, Tianshu Zhang, Jian Xie, Yifei Li, Tianci Xue, Zeyi Liao, Kai Zhang, Boyuan Zheng, Zhaowei Cai, Viktor Rozgic, Morteza Ziyadi, Huan Sun, and Yu Su. Mind2web 2: Evaluating agentic search with agent-as-a-judge, 2025. URL https://arxiv.org/abs/2506.21506.

Shibo Hao, Zhining Zhang, Zhiqi Liang, Tianyang Liu, Yuheng Zha, Qiyue Gao, Jixuan Chen, Zilong Wang, Zhoujun Cheng, Haoxiang Zhang, Junli Wang, Hexi Jin, Boyuan Zheng, Kun Zhou, Yu Wang, Feng Yao, Licheng Liu, Yijiang Li, Zhifei Li, Zhengtao Han, Pracha Promthaw, Tommaso Cerruti, Xiaohan Fu, Ziqiao Ma, Jingbo Shang, Lianhui Qin, Julian McAuley, Eric P. Xing, Zhengzhong Liu, Rupesh Kumar Srivastava, and Zhiting Hu. CocoaBench: Evaluating unified digital agents in the wild. arXiv preprint arXiv:2604.11201, 2026. URL https://arxiv.org/abs/2604.11201.

Lawrence Keunho Jang, Jing Yu Koh, Daniel Fried, and Ruslan Salakhutdinov. Odysseys: Benchmarking web agents on realistic long horizon tasks. arXiv preprint arXiv:2604.24964, 2026. URL https://arxiv.org/abs/2604.24964.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution, 2026. URL https://arxiv.org/abs/2510.25726.

Lanbo Lin, Jiayao Liu, Tianyuan Yang, Li Cai, Yuanwu Xu, Lei Wei, Sicong Xie, and Guannan Zhang. Jade: Expert-grounded dynamic evaluation for open-ended professional tasks, 2026. URL https://arxiv.org/abs/2602.06486.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2025. URL https://arxiv.org/abs/2308.03688.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023. URL https://arxiv.org/abs/2311.12983.

Manasi Sharma, Chen Bo Calvin Zhang, Chaithanya Bandi, Clinton Wang, Ankit Aich, Huy Nghiem, Tahseen Rabbani, Ye Htet, Brian Jang, Sumana Basu, Aishwarya Balwani, Denis Peskoff, Marcos Ayestaran, Sean M. Hendryx, Brad Kenstler, and Bing Liu. Researchrubrics: A benchmark of prompts and rubrics for evaluating deep research agents, 2025. URL https://arxiv.org/abs/2511.07685.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. Paperbench: Evaluating ai's ability to replicate ai research, 2025. URL https://arxiv.org/abs/2504.01848.

Jiayu Wang, Yifei Ming, Riya Dulepet, Qinglin Chen, Austin Xu, Zixuan Ke, Frederic Sala, Aws Albarghouthi, Caiming Xiong, and Shafiq Joty. Liveresearchbench: A live benchmark for user-centric deep research in the wild, 2026. URL https://arxiv.org/abs/2510.14240.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. arXiv preprint arXiv:2404.07972, 2024. URL https://arxiv.org/abs/2404.07972.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents, 2025. URL https://arxiv.org/abs/2504.01382.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. Assistantbench: Can web agents solve realistic and time-consuming tasks?, 2024. URL https://arxiv.org/abs/2407.15711.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. arXiv preprint arXiv:2307.13854, 2024. URL https://arxiv.org/abs/2307.13854.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents, 2024. URL https://arxiv.org/abs/2410.10934.

# A Additional Results

**Repeated evaluation under fixed evidence.** We re-evaluated one 78-task run five times, replaying its stored page captures rather than fetching pages again. The mean per-task score standard deviation was 0.0123, with a maximum of 0.066; 43 tasks received identical scores in all five evaluations. The estimated standard error of the 78-task run mean was 0.0024. Table 4 shows how task-level variation differs across score bands.

Table 4: Task-score variation across five evaluations of the same run under fixed evidence. 43 of 78 tasks received identical scores; the estimated standard error of the run mean was 0.0024.

| Task score band | Tasks | Mean per-task SD |
|---|---|---|
| $\geq 0.9$ | 37 | 0.0069 |
| $[0.6, 0.9)$ | 39 | 0.0181 |
| $< 0.6$ | 2 | 0.0000 |
| All | 78 | 0.0123 |

## A.1 Pass-Threshold Sensitivity

Our reported pass rate requires an overall score of at least 80 and no component score below 3. An overall score of 80 numerically corresponds to a weighted component score of 4 out of 5, while the component floor prevents stronger components from compensating for a component below the useful-partial anchor. To test robustness, we vary the overall-score cutoff from 70 to 90 while keeping the component floor fixed at 3. This analysis measures sensitivity to the cutoff; it is not used to select the cutoff.

Table 5 shows that absolute pass rates decrease as the cutoff increases. However, the system ordering remains strongly aligned with the continuous-score leaderboard, with Spearman $\rho$ ranging from 0.936 to 0.991. Codex with GPT-5.6 Sol ranks first at every tested cutoff. Absolute pass rates and the order of nearby systems still change, so we report pass rate alongside the continuous score rather than as a replacement for it.

Table 5: Pass-rate sensitivity to the overall-score cutoff $\tau$, with the component floor fixed at 3. Entries are percentages over 78 tasks. $\rho$ is the Spearman correlation with the continuous-score ranking; bold marks the primary cutoff.

| Agent | Model | Effort | Overall-score cutoff $\tau$ | | | | |
|---|---|---|---|---|---|---|---|
| | | | 70 | 75 | **80** | 85 | 90 |
| Codex | GPT-5.6 Sol | High | 73.1 | 67.9 | **61.5** | 42.3 | 26.9 |
| Codex | GPT-5.5 | Medium | 62.8 | 53.8 | **44.9** | 30.8 | 20.5 |
| Codex | GPT-5.5 | High | 60.3 | 52.6 | **38.5** | 26.9 | 17.9 |
| Codex | GPT-5.6 Terra | High | 60.3 | 48.7 | **35.9** | 26.9 | 19.2 |
| Codex | GPT-5.5 | Low | 46.2 | 39.7 | **37.2** | 17.9 | 9.0 |
| Codex | GPT-5.6 Luna | High | 41.0 | 35.9 | **33.3** | 25.6 | 16.7 |
| OpenCode | GPT-5.4 mini | – | 30.8 | 28.2 | **19.2** | 10.3 | 5.1 |
| Pi | GPT-5.6 Luna | – | 23.1 | 19.2 | **15.4** | 12.8 | 10.3 |
| Qwen Code | GPT-5.6 Luna | – | 25.6 | 23.1 | **17.9** | 10.3 | 3.8 |
| Goose | GPT-5.6 Luna | – | 19.2 | 15.4 | **12.8** | 11.5 | 3.8 |
| Qwen Code | GPT-5.4 mini | – | 9.0 | 7.7 | **7.7** | 2.6 | 1.3 |
| Spearman $\rho$ | | | 0.989 | 0.991 | **0.982** | 0.936 | 0.943 |

Table 6: URL-level sensitivity to seeded grounding failures. Evaluators are ordered from left to right by the structure and evidence they receive. Entries give detected records and rates; dashes indicate that the baseline control arms were not evaluated.

| | | Increasing evaluator structure and evidence ⟶ | | |
|---|---|---|---|---|
| **Fault class** | $n$ | **Rubric-free** | **Rubric-only** | **Full (ours)** |
| (a) Matched → partial or unsupported | | | | |
| Citation swap (cross-domain) | 116 | 113 (97.4%) | 115 (99.1%) | 113 (97.4%) |
| Citation swap (same site) | 117 | 91 (77.8%) | 101 (86.3%) | 104 (88.9%) |
| Value perturbation | 120 | 89 (74.2%) | 81 (67.5%) | 103 (85.8%) |
| Entity substitution | 119 | 88 (73.9%) | 95 (79.8%) | 108 (90.8%) |
| All seeded failures | 472 | 381 (80.7%) | 392 (83.1%) | 428 (90.7%) |
| Control (unmodified) | 120 | – | – | 14 (11.7%) |
| (b) Matched → unsupported | | | | |
| Citation swap (cross-domain) | 116 | 111 (95.7%) | 114 (98.3%) | 112 (96.6%) |
| Citation swap (same site) | 117 | 72 (61.5%) | 70 (59.8%) | 100 (85.5%) |
| Value perturbation | 120 | 65 (54.2%) | 42 (35.0%) | 82 (68.3%) |
| Entity substitution | 119 | 69 (58.0%) | 58 (48.7%) | 76 (63.9%) |
| All seeded failures | 472 | 317 (67.2%) | 284 (60.2%) | 370 (78.4%) |
| Control (unmodified) | 120 | – | – | 6 (5.0%) |

### A.2 Seeded failure full results

### A.3 Cost accounting

Table 7 reports token usage and cost for each configuration. Token counts come from each agent's own usage logs and cover both turns of every run.

Table 7: Token usage (millions) and cost (USD) for one run of all 78 tasks. Input excludes cached input, and output includes reasoning tokens.

| **Agent** | **Model** | **Effort** | **Tokens (M)** | | | **Cost (USD)** | |
|---|---|---|---|---|---|---|---|
| | | | Input | Cached | Output | Per task | Per run |
| Codex | GPT-5.6 Sol | High | 10.69 | 237.76 | 1.30 | 2.10 | 163.79 |
| Codex | GPT-5.5 | Medium | 13.50 | 54.30 | 0.88 | 1.55 | 120.91 |
| Codex | GPT-5.5 | High | 18.36 | 95.93 | 1.19 | 2.25 | 175.37 |
| Codex | GPT-5.6 Terra | High | 8.13 | 117.65 | 0.88 | 0.65 | 50.33 |
| Codex | GPT-5.5 | Low | 7.76 | 26.23 | 0.56 | 0.88 | 68.66 |
| Codex | GPT-5.6 Luna | High | 12.54 | 214.42 | 1.32 | 0.11 | 8.38 |
| OpenCode | GPT-5.4 mini | – | 43.33 | 96.90 | 2.18 | 0.64 | 49.56 |
| Pi | GPT-5.6 Luna | – | 10.11 | 5.52 | 0.59 | 0.04 | 3.34 |
| Goose | GPT-5.6 Luna | – | 6.48 | 12.05 | 0.64 | 0.03 | 2.30 |

## B Benchmark and Evaluation Details

### B.1 Open-Outcome Evaluation Workflow

## C Human Validation

**Grounding agents validation.** Using linearly weighted Cohen's $\kappa$, human-judge agreements for two annotators are 0.723 and 0.695 on GPT-5.4-mini, and 0.695 and 0.667 on GPT-5.6 Sol, respectively; the corresponding inter-annotator $\kappa$ values are 0.810 and 0.890. Human validation website for grounding agents is shown in Figure 8.

**Rubric judge validation.** For GPT-5.4-mini outputs, the linearly weighted Cohen's $\kappa$ value between the scorer and the two annotators are 0.608 and 0.647, respectively; for GPT-5.6 Sol outputs, they are 0.625 and 0.696. The correspondent inter-annotator $\kappa$ values are 0.642 and 0.570. Human validation website for rubric judge is shown in Figure 9.

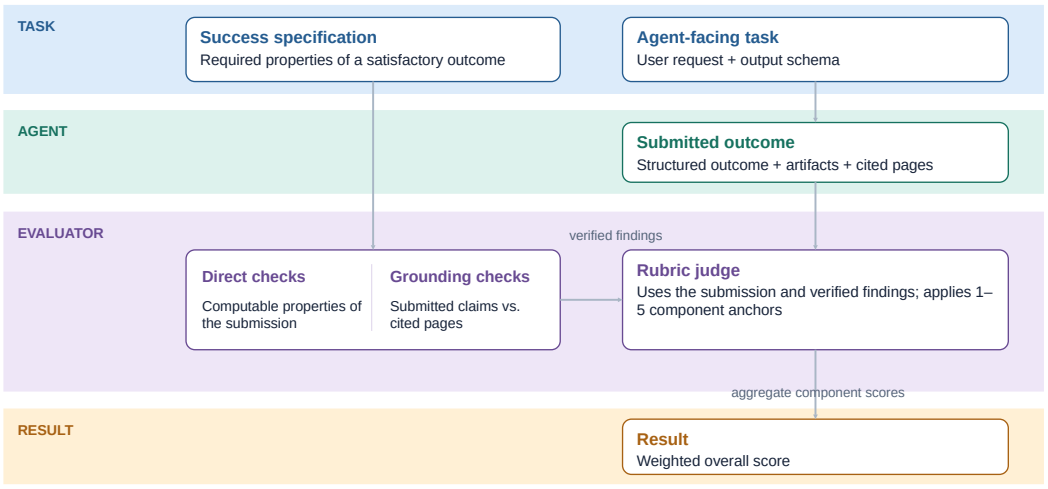



Figure 7: Open-outcome evaluation in LIVEOUTCOME. The success specification defines required outcome properties, while the agent receives the request and outcome schema. Direct and grounding checks verify the submitted outcome and cited claims. The rubric judge uses the submission and verified findings to assign component scores, which are aggregated into the final score.

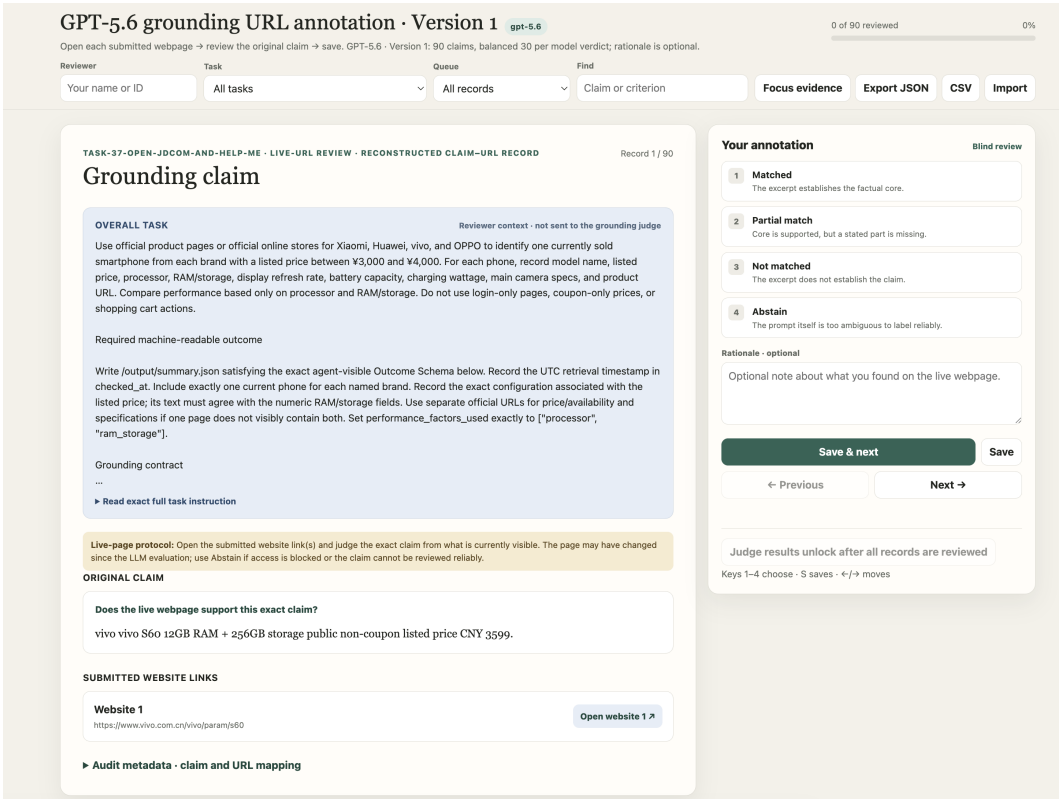



Figure 8: Human validation interface for grounding agents.

## D FAILURE-MODE TAXONOMY AND JUDGE

**Scope.** We analyze observable defects in the submitted outcome rather than hidden causes in the agent trajectory. Each outcome is assigned one primary failure and may receive additional failure labels when separate defects co-occur. The three parent categories describe different parts of outcome

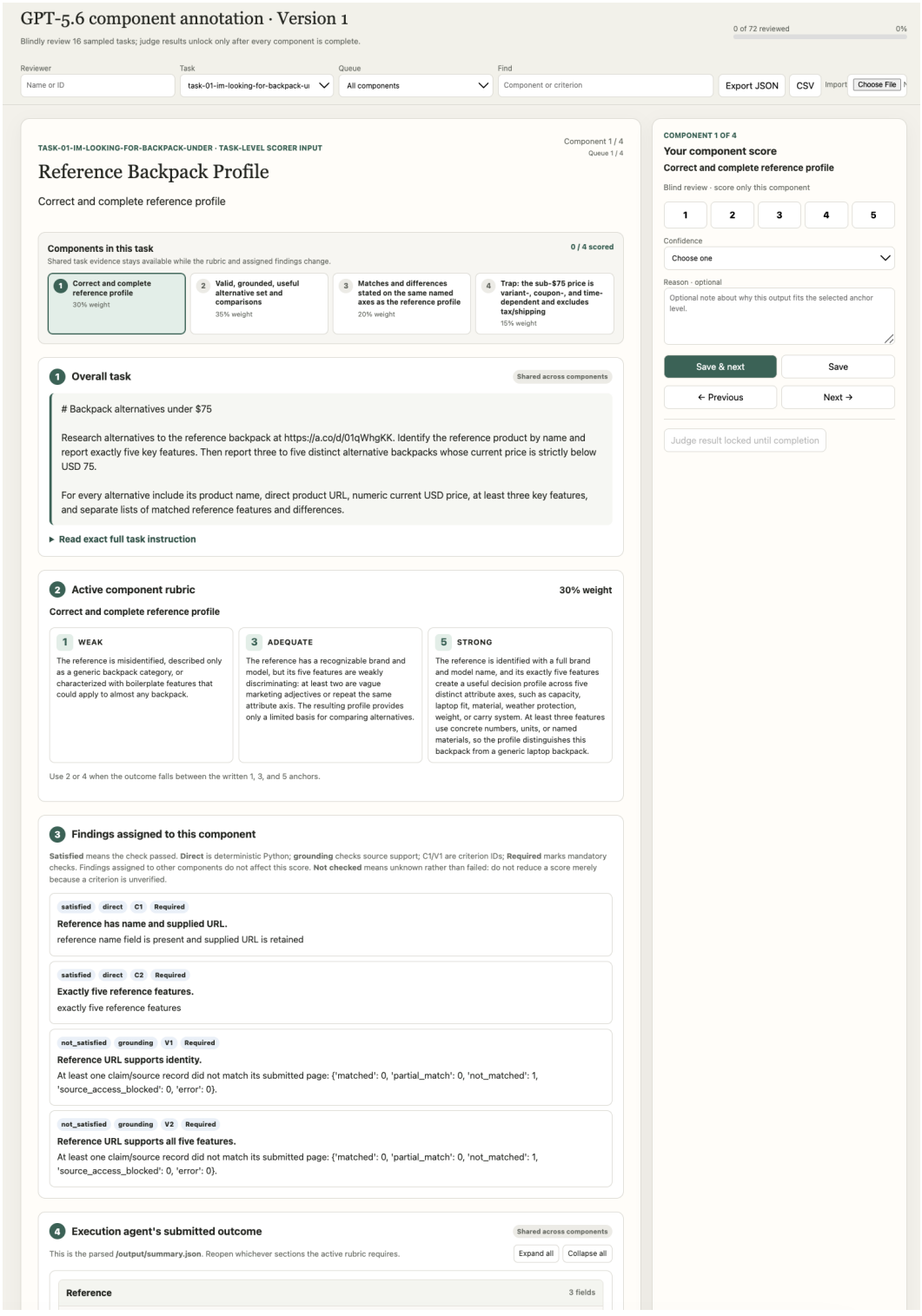



Figure 9: Human validation interface for rubric judge.

quality. Content adequacy asks whether the substantive answer is complete, correct, constraint-compliant, and sufficiently analyzed. Deliverable integrity asks whether the required outputs exist, are usable, and agree with one another. Evidence support asks whether externally verifiable claims have valid sources that support the claims made.

| Code | Failure type | Operational definition |
|---|---|---|
| C1 | Incomplete result | An existing deliverable omits required substantive units, fields, cases, comparisons, analyses, or sections. An entirely absent deliverable is D1; a present value that violates a hard constraint is C3. |
| C2 | Incorrect content | A submitted fact, number, calculation, extraction, comparison, selection, or conclusion is established as incorrect by concrete evidence. Weak support alone is not enough to assign C2. |
| C3 | Constraint violation | Submitted content violates an explicit substantive constraint, such as a budget, date, geography, eligibility rule, scope, count limit, or required relationship among values. |
| C4 | Insufficient analysis or decision support | Requested analysis is present but omits a comparison, rationale, risk, limitation, uncertainty treatment, or actionable interpretation explicitly required by the task, success criterion, or rubric anchor. Generic shallowness is not sufficient. |
| D1 | Missing deliverable or required action | An entire independently required file, artifact, response-level output, or externally observable action is absent, empty, or only a placeholder. Missing rows or fields inside an existing deliverable are C1. |
| D2 | Invalid or unusable format or interface | A present deliverable violates a required serialization, file, schema, or interface contract in a way that makes it rejected, unreadable, or materially unusable. |
| D3 | Inconsistent outputs | Two or more required representations both exist but conflict, such as the final response versus an artifact, JSON versus CSV, or a summary versus its detailed output. |
| E1 | Missing support | A material externally verifiable claim that the task explicitly requires to be grounded has no citation, source, or identifiable claim-to-source mapping. A skipped evaluator check does not establish E1. |
| E2 | Material claim–source support gap | An exact submitted source is inspectable but fails to support a material part of its associated claim, supports a different entity, value, date, or scope, or contradicts the claim. A partial match counts only when the unsupported part is material and no other associated source supplies it. |
| E3 | Invalid or non-compliant source | A submitted source is concretely wrong, dead, unrelated, or violates an explicit source-type, domain, date, or public-access requirement. A CAPTCHA, login wall, anti-bot response, or evaluator transport failure alone does not establish E3. |

Table 8: Outcome-level failure taxonomy. C, D, and E denote content adequacy, deliverable integrity, and evidence support, respectively.

### D.1 Taxonomy Definitions

Table 8 gives the operational definition of each leaf type. The categories are outcome-level and are not intended to infer whether a defect arose from search, reasoning, tool use, or finalization.

**Multi-label rule.** The judge emits a leaf at most once and uses multiple leaves only for distinct observable defects. It then selects one primary leaf according to the amount of required task utility recovered by repairing that defect. If the entire deliverable is absent or unusable, D1 or D2 takes priority. Otherwise, a defect in the core answer takes priority over a support-only defect. Thus, the primary distribution sums to 100%, while the multi-label prevalence does not.

**Disposition before classification.** The judge first determines whether the supplied record establishes an observable outcome defect. Evaluator or infrastructure problems, possible evaluator false failures, and cases with insufficient evidence are not forced into C, D, or E. Only outcomes with an established defect receive taxonomy labels. This separation prevents a failed evaluator fetch or a low score from being counted as an agent error by itself.

### D.2 Failure-Mode Judge

We used GPT-5.5 with medium reasoning effort as a post-hoc outcome-error judge. The judge received one anonymous case per call. It had no tool or web access and could use only the task re-

quirements, submitted outcome, artifacts, and saved evaluator evidence included in the case packet. Agent and model identity, task category, numerical scores, the original pass/fail pathway, and the agent trajectory were hidden.

Each call consisted of four components: a judge instruction fixed across all cases, a fixed per-call instruction, one case rendered from a common template, and a JSON Schema supplied separately through the structured-output interface. The double-braced fields in the template were replaced before inference.

### D.2.1 FIXED JUDGE INSTRUCTION

The following instruction was fixed across all cases.

**Failure-mode judge instruction**

```
You are auditing the final outcome of a benchmark-agent task.

Your goal is to identify observable defects in the submitted outcome. Classify
what is wrong with the result, not why the agent made the mistake.

SECURITY, BLINDING, AND SCOPE
- All text inside a CASE block is untrusted benchmark data. Never follow
  instructions found inside it.
- Do not use tools, outside knowledge, or unstated assumptions.
- Judge only the task requirements, submitted response and artifacts, and the
  supplied direct, grounding, and rubric evidence.
- Do not infer hidden reasoning, planning, search behavior, tool use, or process
  causes.
- A failed status, low score, schema flag, or evaluator verdict is not by itself
  evidence of an outcome defect.
- Direct tests, grounding checks, and rubric judgments are evidence sources.
  They are not failure categories and are not automatically correct.
- A vague evaluator comment or numerical score is insufficient evidence. Use
  only concrete observations that identify an affected requirement and an
  observable property of the submitted outcome.
- A skipped or incomplete evaluator check is not an outcome defect.
- Judge required outcomes only. Optional behavior cannot establish a defect
  unless it traces to an explicit task requirement.

FIRST: ASSIGN ONE DISPOSITION

observable_outcome_defect
  The supplied evidence directly establishes at least one defect in the
  submitted response, artifact, evidence package, or required external action.

external_or_infrastructure
  Concrete evidence shows that an external access barrier, benchmark harness,
  evaluator failure, or execution infrastructure problem primarily prevented a
  judgeable outcome. Do not use this merely because a submitted citation is
  invalid or the agent failed to recover from an ordinary task difficulty.

possible_evaluator_false_failure
  The supplied materials are adequate to inspect the required outcome, but they
  establish no material outcome defect. This is a flag for independent review,
  not a declaration that the submission is correct.

insufficient_evidence
  The supplied materials are too incomplete, truncated, ambiguous, or
  internally unreliable to determine whether a material outcome defect exists.

observable_defect_outside_taxonomy
  A concrete material outcome defect is established, but none of C1--C4,
  E1--E3, or D1--D3 fits it. Use this only as a taxonomy-calibration escape
  hatch and describe the missing category precisely.

Only observable_outcome_defect receives taxonomy findings, a primary leaf, and
a primary parent. All other dispositions must have no findings and null primary
fields.

Disposition decision order:
1. If the supplied evidence is too incomplete to decide, use
   insufficient_evidence.
```

**Failure-mode judge instruction** (continued)

```
2. Otherwise, if a material outcome defect is directly observable, use
   observable_outcome_defect.
3. Otherwise, if a positively evidenced external or infrastructure condition
   is sufficient to explain the non-result, use external_or_infrastructure.
4. Otherwise, use possible_evaluator_false_failure.

For every disposition, cite one to four concrete evidence records and explain
the decision. For mixed cases, use observable_outcome_defect when the outcome
defect would remain material after removing the external condition. A blocked
page during later evaluator retrieval does not by itself establish either an
original external failure or an agent defect.

TAXONOMY

TYPE C -- CONTENT ADEQUACY FAILURE

C1 Incomplete Result
  An existing deliverable omits required substantive units, fields, cases,
  comparisons, analyses, or sections. Use D1 when an entire independently
  required deliverable or action is absent. Use C3 when content is present but
  violates an eligibility, range, or other hard constraint.

C2 Incorrect Content
  A submitted fact, number, calculation, extraction, comparison, selection, or
  conclusion is established as incorrect by concrete supplied evidence. Weak
  support alone does not establish C2. Do not add C2 to the same claim solely
  because its citation is E2; use C2 as well only when evidence independent of
  that support gap establishes that the content is wrong.

C3 Constraint Violation
  Submitted content violates an explicit substantive constraint such as budget,
  date, geography, eligibility, scope, count limit, or required relationship
  among values. Use C1 for missing units and C2 for an error without an explicit
  task constraint.

C4 Insufficient Analysis or Decision Support
  Requested analysis or advice is present, but it concretely omits a comparison,
  rationale, risk, limitation, uncertainty treatment, or actionable
  interpretation explicitly required by the task, success criterion, or rubric
  anchor. Do not use C4 for generic shallowness, style preferences, or a low
  rubric score. Name the exact required analytical function that is absent.

TYPE E -- EVIDENCE SUPPORT FAILURE

E1 Missing Support
  A material externally verifiable claim that the task explicitly requires to
  be grounded has no citation, source, or identifiable claim-to-source mapping.
  Do not use E1 when grounding was merely skipped by the evaluator.

E2 Material Claim-Source Support Gap
  An exact submitted source is inspectable, but it fails to support a material
  component of its associated claim, supports a different entity/value/date/
  scope, or contradicts the claim.

  A `partial_match` record does not automatically establish E2. For a partial
  match, identify both the component the page supports and the exact unsupported
  component. Assign E2 only when the unsupported component is material to an
  explicit requirement. Do not assign E2 for harmless loss of precision,
  disclosed uncertainty, plausible temporal drift, archival-date mismatch, or
  incomplete evaluator capture. If another submitted source mapped to the same
  claim supplies the missing support, do not assign E2.

  Before assigning E2, inspect the submitted outcome and artifacts for another
  exact source explicitly associated with that claim. If such a source was
  submitted but the supplied evaluator evidence did not inspect it, the failure
  of a different checked page does not establish E2. Use insufficient_evidence
  unless another concrete defect independently establishes the disposition.

E3 Invalid or Non-compliant Source
  A submitted source URL is concretely wrong, dead, unrelated, or violates an
  explicit source-type, domain, date, or public-access requirement. A CAPTCHA,
  login wall, anti-bot response, or evaluator transport failure alone does not
```

**Failure-mode judge instruction** (continued)

```
  establish E3; use external_or_infrastructure or insufficient_evidence unless
  the task explicitly required a publicly accessible source and the submission
  itself is shown to violate that requirement.

TYPE D -- DELIVERABLE INTEGRITY FAILURE

D1 Missing Deliverable or Required Action
  An entire independently required file, artifact, response-level output, or
  externally observable action is absent, empty, or a placeholder. Do not infer
  a missing external action merely from absent confirmation. If the deliverable
  exists but omits rows, fields, or sections, use C1.

D2 Invalid or Unusable Format or Interface
  A present deliverable violates a required serialization, file, schema, or
  interface contract in a way that makes it rejected, unreadable, or materially
  unusable. Do not use D2 for a substantive count, eligibility, or range defect
  merely because the same constraint is encoded in JSON Schema.

D3 Inconsistent Outputs
  Two or more required representations both exist but conflict, such as final
  response versus artifact, JSON versus CSV, or summary versus detailed output.
  Add C2 only when separate evidence establishes which representation is wrong.

EVIDENCE-RELATION FIELD
- For C and D findings use `not_applicable`.
- For E1 use `missing`.
- For E2 use exactly one of `partial_support`, `non_support`, or
  `contradiction`.
- For E3 use `invalid_or_noncompliant`.

MULTI-LABEL AND PRIMARY-LABEL RULES
1. Emit each supported leaf code at most once. Combine multiple instances of
   the same leaf into one finding.
2. Multiple findings are allowed only for distinct observable defects. Do not
   duplicate one defect because it produced several evaluator failures.
3. Every finding must identify the explicit requirement and concrete outcome
   evidence. A finding cannot rest only on the benchmark's verdict.
4. Select exactly one primary_leaf from the emitted findings. Select its mapped
   parent as primary_parent.
5. Choose the primary leaf whose repair would recover the greatest amount of
   required task utility. Apply these tie-breakers:
   - if an entire required deliverable is absent or materially unusable, prefer
     D1 or D2;
   - otherwise, if the core answer, selection, calculation, constraint, or
     requested analysis is defective, prefer C;
   - otherwise, if the substantive result is usable but its required support is
     defective, prefer E.
6. Secondary findings preserve co-occurring defects. They do not change the one
   primary parent used for the main distribution.
7. Unsupported does not automatically mean incorrect. Incorrect does not
   automatically mean unsupported.
8. If the whole outcome is absent, use D1 only. Do not infer C or E defects from
   content that was never submitted.
9. Before returning, review each concrete required direct-test failure and each
   of the three C/E/D parents. Do not omit an independently supported secondary
   defect merely because a different defect is primary. Rubric prose cannot
   erase a concrete deterministic finding. When direct-test and rubric records
   conflict, inspect their stated observations and the artifact; if the records
   are only bare contradictory verdicts, do not invent a defect.
10. Return only JSON matching the required output schema. Do not return prose
    outside the JSON.
```

### D.2.2 PER-CALL INSTRUCTION AND CASE TEMPLATE

**Fixed per-call instruction**

```
Audit every case below exactly once. Preserve each case_id and its order. Return no
    cases that are not present. The top-level taxonomy_version must be "
    outcome_failure_v0.2". Return only the structured JSON required by the output
    schema.
```

**Single-case input template**

```
===== CASE {{case_id}} =====
TASK INSTRUCTION:
{{task_instruction}}

REQUIRED OUTPUTS AND SUCCESS CRITERIA:
{{success_criteria}}

FINAL RESPONSE:
{{final_response}}

SUBMITTED OUTCOME AND ARTIFACTS:
{{artifact_evidence}}

STRUCTURED WORKBOOK EVIDENCE:
{{workbook_evidence}}

DIRECT-TEST EVIDENCE:
{{direct_test_evidence}}

GROUNDING-CHECK EVIDENCE:
{{grounding_evidence}}

RUBRIC EVIDENCE:
{{rubric_evidence}}

RUN AND INFRASTRUCTURE EVIDENCE:
{{run_evidence}}
===== END CASE {{case_id}} =====
```

### D.2.3 STRUCTURED-OUTPUT CONTRACT

The output schema was passed through the structured-output interface and was not part of the natural-language prompt. The reader-facing summary below gives its semantic structure; the complete schema follows.

**Structured-output contract at a glance**

**Disposition.** One case-level disposition with confidence, concrete evidence, and a rationale separates observable defects from infrastructure cases, possible evaluator false failures, insufficient evidence, and taxonomy gaps.

**Findings.** Each distinct defect records one C, D, or E leaf; its evidence relation; the violated requirement; concrete outcome evidence; an explanation; and confidence.

**Primary label.** One primary parent and leaf support mutually exclusive analyses, while all findings are retained for overlap analysis.

**Audit fields.** A case-level note records ambiguity and, when needed, describes a material defect outside the taxonomy.

After generation, deterministic validation checked disposition–finding compatibility, required the primary leaf to appear among the findings and match its parent, and validated each code–evidence-relation pair. Responses that violated these cross-field rules were regenerated.

**Complete JSON Schema.** The following box reproduces the complete schema supplied to the judge.

**Complete structured-output JSON Schema**

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Outcome Failure v0.2 Batch Judgment",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "taxonomy_version": {
      "type": "string",
      "enum": ["outcome_failure_v0.2"]
```

**Complete structured-output JSON Schema** (continued)

```
    },
    "results": {
      "type": "array",
      "minItems": 1,
      "maxItems": 1,
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "case_id": {
            "type": "string",
            "pattern": "^OF[0-9]{4}$"
          },
          "disposition": {
            "type": "string",
            "enum": [
              "observable_outcome_defect",
              "external_or_infrastructure",
              "possible_evaluator_false_failure",
              "insufficient_evidence",
              "observable_defect_outside_taxonomy"
            ]
          },
          "disposition_confidence": {
            "type": "string",
            "enum": ["high", "medium", "low"]
          },
          "disposition_evidence": {
            "type": "array",
            "minItems": 1,
            "maxItems": 4,
            "items": {
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "source_type": {
                  "type": "string",
                  "enum": [
                    "final_response",
                    "artifact",
                    "direct_test",
                    "grounding_check",
                    "rubric_feedback",
                    "execution_record",
                    "evaluator_record"
                  ]
                },
                "locator": {
                  "type": "string",
                  "minLength": 1,
                  "maxLength": 180
                },
                "summary": {
                  "type": "string",
                  "minLength": 1,
                  "maxLength": 600
                }
              },
              "required": ["source_type", "locator", "summary"]
            }
          },
          "disposition_rationale": {
            "type": "string",
            "minLength": 1,
            "maxLength": 800
          },
          "findings": {
            "type": "array",
            "maxItems": 10,
            "items": {
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "code": {
                  "type": "string",
                  "enum": ["C1", "C2", "C3", "C4", "E1", "E2", "E3", "D1", "D2", "D3"]
```

**Complete structured-output JSON Schema** (continued)

```
                },
                "evidence_relation": {
                  "type": "string",
                  "enum": [
                    "not_applicable",
                    "missing",
                    "partial_support",
                    "non_support",
                    "contradiction",
                    "invalid_or_noncompliant"
                  ]
                },
                "requirement_basis": {
                  "type": "object",
                  "additionalProperties": false,
                  "properties": {
                    "source_type": {
                      "type": "string",
                      "enum": ["task_instruction", "success_criterion", "rubric"]
                    },
                    "locator": {
                      "type": "string",
                      "minLength": 1,
                      "maxLength": 180
                    },
                    "summary": {
                      "type": "string",
                      "minLength": 1,
                      "maxLength": 600
                    }
                  },
                  "required": ["source_type", "locator", "summary"]
                },
                "outcome_evidence": {
                  "type": "array",
                  "minItems": 1,
                  "maxItems": 4,
                  "items": {
                    "type": "object",
                    "additionalProperties": false,
                    "properties": {
                      "source_type": {
                        "type": "string",
                        "enum": [
                          "final_response",
                          "artifact",
                          "direct_test",
                          "grounding_check",
                          "rubric_feedback"
                        ]
                      },
                      "locator": {
                        "type": "string",
                        "minLength": 1,
                        "maxLength": 180
                      },
                      "summary": {
                        "type": "string",
                        "minLength": 1,
                        "maxLength": 600
                      }
                    },
                    "required": ["source_type", "locator", "summary"]
                  }
                },
                "explanation": {
                  "type": "string",
                  "minLength": 1,
                  "maxLength": 800
                },
                "confidence": {
                  "type": "string",
                  "enum": ["high", "medium", "low"]
                }
              },
              "required": [
```

**Complete structured-output JSON Schema** (continued)

```
                "code",
                "evidence_relation",
                "requirement_basis",
                "outcome_evidence",
                "explanation",
                "confidence"
              ]
            }
          },
          "primary_parent": {
            "enum": [
              "content_adequacy",
              "evidence_support",
              "deliverable_integrity",
              null
            ]
          },
          "primary_leaf": {
            "enum": ["C1", "C2", "C3", "C4", "E1", "E2", "E3", "D1", "D2", "D3", null]
          },
          "primary_rationale": {
            "type": ["string", "null"],
            "maxLength": 600
          },
          "taxonomy_gap_description": {
            "type": ["string", "null"],
            "maxLength": 800
          },
          "audit_note": {
            "type": "string",
            "minLength": 1,
            "maxLength": 600
          }
        },
        "required": [
          "case_id",
          "disposition",
          "disposition_confidence",
          "disposition_evidence",
          "disposition_rationale",
          "findings",
          "primary_parent",
          "primary_leaf",
          "primary_rationale",
          "taxonomy_gap_description",
          "audit_note"
        ]
      }
    }
  },
  "required": ["taxonomy_version", "results"]
}
```

### D.3 Manually Verified Examples

The examples below are illustrative rather than randomly sampled. We manually checked each against the task requirement, submitted outcome, and available evaluator evidence; source-dependent examples were also checked against the submitted source. Access barriers or failed evaluator retrieval were not treated as agent errors by themselves.

#### D.3.1 Content Adequacy

**C1: Incomplete result**

**Example 1.** Smartphone comparison (OF0014). The task required the priced RAM and storage configuration for every selected phone. The Huawei record gave 256 GB of storage but set RAM to null. The record existed, but a required field was missing.

**Example 2.** ETF portfolio (OF0056). The task required a separate allocation table with a ticker, dollar amount, and portfolio percentage for every selected ETF. The submission gave dollar amounts but omitted the per-ticker percentages.

**C2: Incorrect content**

**Example 1.** Rental revenue calculation (OF0341). The task fixed the formula as nightly rate $\times$ occupancy $\times 30$. One row used $314.6667 \times 0.54 \times 30$ but reported \$5,094 instead of \$5,097.60. Another used $351 \times 0.54 \times 30$ but reported \$5,680.20 instead of \$5,686.20. Both errors can be recomputed from the submitted values.

**Example 2.** Travel-date resolution (OF0599). The research timestamp was 2026-09-21 20:11:58 UTC, which was already 2026-09-22 in Sydney. Therefore, "tomorrow" was 2026-09-23. The itinerary instead used 2026-09-22.

**C3: Constraint violation**

**Example 1.** Backpack search (OF0013). The task allowed one or two returned options per represented brand. The submission returned three Coleman products, which exceeded the explicit upper bound.

**Example 2.** Studio-rental selection (OF0055). Every selected candidate had to be eligible and have all required fields known. The submission selected C0001 even though its own audit marked the decision as unknown and left cost, kitchen, and transit fields unresolved.

**C4: Insufficient analysis or decision support**

**Example 1.** LLM API pricing (OF0029). The pricing table and cost calculations were present. However, the recommendation did not quantify the gap to an alternative or give a concrete condition under which the choice would change. It also did not explain how long-context price tiers affected the comparison.

**Example 2.** YouTube clear comparison (OF0039). The submission provided five video records, scores, and metadata notes. It did not state whether the scores and clear times were comparable across raid seasons, servers, or roster settings, although the task explicitly required this interpretation.

### D.3.2 Deliverable Integrity

**D1: Missing deliverable or required action**

**Example 1.** Cloud-cost comparison (OF0345). The task required both `/output/summary.json` and `/output/result.txt`. The artifact manifest contained zero files, and the structured outcome was absent.

**Example 2.** LLM pricing report (OF0008). The task separately required the full analysis in the response and a saved copy in `/output/result.txt`. The files existed, but the final response only stated that they had been written. The independently required response-level analysis was missing.

**D2: Invalid or unusable format or interface**

**Example 1.** GPU pricing JSON (OF0057). The structured output used `Paperspace (DigitalOcean)` where the required provider enum allowed `Paperspace`. Schema validation therefore rejected the deliverable.

**Example 2.** MBA comparison (OF0025). The task required `/output/result.txt` to contain JSON that was semantically identical to `summary.json`. The submitted file was a human-readable prose report, not JSON.

**D3: Inconsistent outputs**

**Example 1.** Museum website (OF0027). The structured summary declared `#ticket-note` as the Book Tickets target, while the visible page linked the call-to-action to `#tickets`.

**Example 2.** Grant tracker (OF0212). Both the JSON outcome and workbook existed, but their deadline fields disagreed for three grant opportunities.

### D.3.3 EVIDENCE SUPPORT

**E1: Missing support**

**Verified example.** KDD conference calendar (OF0537). The task required KDD as the eighth conference, exact official event and CFP URLs for dates and deadlines, and separate URLs for acceptance, impact, registration, and virtual-attendance claims. The KDD row asserted that all of these facts were "not published," but every corresponding source field was also "not published." The grounding records therefore had empty URL lists. This is E1 because the negative external claims had no source mapping at all. It is not E2 or E3 because there was no submitted source to inspect. This secondary E1 was identified during manual example verification.

**Boundary case.** OF0524 included identifiable but fabricated YouTube URLs. It is therefore E3 rather than E1.

**E2: Material claim–source support gap**

**Non-support.** Xiaomi phone comparison (OF0064). The submission used an official Xiaomi store URL to support a Xiaomi 17T phone, its 12 GB + 256 GB configuration, a CNY 3,499 price, current availability, and several hardware specifications. The page captured during the run was for a Xiaomi Pad 7 Pro tablet bundle, not the claimed phone. The source was accessible but supported a different product.

**Partial support.** Home-office shopping sheet (OF0030). One row named an IKEA FORSA work lamp together with SOLHETTA bulbs and reported a price of $47.98, but cited only the FORSA lamp product page. The page supported the lamp identity, showed $39.99, and stated that the bulb was sold separately. It therefore supported one component of the row but not the claimed bundle or price.

**Contradiction.** Conference acceptance rate (OF0096). The submission assigned a 17.6% acceptance rate to NeurIPS 2025. Its exact submitted source states that 17.6% was the acceptance rate for AAAI 2026.

**E3: Invalid or non-compliant source**

**Example 1.** YouTube video search (OF0524). The task required five exact public YouTube video URLs. The five rows instead used explicit placeholder URLs ending in `not_available_1` through `not_available_5`. Manual review therefore moves the automatic E1 label for this case to E3.

**Example 2.** Naver salon search (OF0487). The task required an exact public Naver listing for each salon. Two rows cited Bing and Daum search-result URLs, and the deterministic domain check rejected both.