LONGWEBBENCH: FINE-GRAINED EVALUATION OF AGENTS ON LONG-HORIZON WEB TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agents powered by large language models are increasingly asked to complete long-horizon tasks that span hundreds of dependent interactions. Evaluating them calls for tasks that are long by construction, an environment that is controllable yet disrupts the agent as real runs are disrupted, and an evaluation that shows where a run falls short, but existing benchmarks rarely provide all three. To fill this gap, we introduce LongWebBench, a benchmark of long-horizon web data collection on generated, self-hosted websites. Each of its 85 tasks, spanning five difficulty levels, asks the agent to collect a complete dataset, such as every product of an online store with all its attributes, amounting to several hundred to a few thousand items spread over listing, detail, and login-protected pages. During collection, verification gates, expiring sessions, truncated responses, and obfuscated pages disrupt the run. Because the server records what it delivers, LongWebBench compares what a task requires, what the server delivered, and what the agent submitted, and it diagnoses each run at four stages, namely Interface Access, State Control, Data Acquisition, and Result Assembly. We evaluate ten agent systems on the benchmark and find that even the strongest remains far from complete collection and that rankings on the easiest level do not carry over to the hardest. The diagnosis further shows that agents fail mainly by omission, since most of the missing information was never delivered to them rather than misread, and that the backbone model shapes what is collected far more than the agent harness does. A stage-wise evaluation grounded in delivery records thus reveals what a single score hides and where agents need to improve.

1 INTRODUCTION

Many tasks now delegated to agents built on large language models, such as compiling a complete dataset from a website, cannot be finished in a few tool calls and instead require hundreds of steps, each building on those before it. Success in such tasks depends less on any single action than on sustaining progress over the whole run, and a run can fail at many points along the way. Evaluating such agents therefore calls for tasks that are *long by construction*, an environment that is *controllable yet disrupts the agent* as real runs are disrupted, and an evaluation that shows *where a run lost progress* rather than only whether it succeeded.

Existing benchmarks each address part of these requirements, and among the representative benchmarks in Table 1, we identify none that satisfies all three. Long-horizon web benchmarks lengthen tasks through dependent workflows or multi-site goals, yet they mainly judge the final outcome, and those built on the live web give up control over content that changes over time and may appear in pretraining data (Boisvert et al., 2024; Miyai et al., 2026; Jang et al., 2026). Controllable benchmarks host or generate their websites but rarely disrupt the agent during a run, while studies of perturbations apply them to shorter tasks with a single answer or state as the deliverable (Zhou et al., 2024; Yuan et al., 2026; Bai et al., 2026; Wang et al., 2026a). Process-aware and large-scale collection benchmarks score steps along a trajectory or the completeness of a final table, but they do not show whether the information behind a multi-record output was served during the run (Pan et al., 2024; Chung et al., 2026; Wong et al., 2026; Bohra et al., 2025). As a result, when an agent returns an incomplete dataset after a long run, current benchmarks cannot tell whether it never reached the data, lost access midway, or discarded data it had already collected.

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.

Table 1: Comparison with representative benchmarks along the three properties that LongWebBench combines. ✓ marks full support, ○ partial support, and – none. Appendix A defines each column.

	Long-horizon task		Controlled environment		Stage-wise evaluation	
Benchmark	Long chain	Full record set	Self-hosted	Perturbed	Process-level	Delivery-based
WebArena (Zhou et al., 2024)	–	–	✓	–	–	–
WorkArena++ (Boisvert et al., 2024)	✓	–	○	–	–	–
WebChoreArena (Miyai et al., 2026)	✓	○	✓	–	○	○
Odysseys (Jang et al., 2026)	✓	○	–	–	○	○
WebCanvas (Pan et al., 2024)	–	–	–	–	✓	–
StressWeb (Bai et al., 2026)	–	–	○	✓	○	–
WebStep (Chung et al., 2026)	–	–	○	–	✓	○
WideSearch (Wong et al., 2026)	○	✓	–	–	–	–
WebLists (Bohra et al., 2025)	○	✓	–	–	–	–
BaRA (Lee et al., 2026)	○	○	○	–	○	○
LongWebBench	✓	✓	✓	✓	✓	✓

To close this gap, we introduce **LongWebBench**, a benchmark of long-horizon web data collection that combines the three properties. First, its tasks are long by construction. Each task asks for a complete dataset of several hundred to a few thousand records whose fields are spread across listing, detail, and login-protected pages, and some records can be found only through links discovered during retrieval, so no single page or short loop completes the export and a run on the harder levels takes hundreds of agent steps. Second, its environment is controllable. We generate and host the websites ourselves, which keeps their content out of pretraining data and lets us place perturbations along the run, including verification gates, expiring sessions and cursors, truncated or stale responses, and varied page representations. The 85 tasks range from e-commerce catalogs to travel directories and form five difficulty levels that increase scale and perturbation density together. Third, its evaluation is stage-wise. Because the server records what it delivers during each run, we compare what the task requires, what the server delivered, and what the agent submitted. This comparison diagnoses each run at four stages, namely Interface Access, State Control, Data Acquisition, and Result Assembly, and thus separates fields without recorded delivery from fields delivered intact but missing or incorrect in the submission.

We evaluate ten agent systems on the full suite and analyze the delivery records of eight configurations, and three findings stand out. First, even the strongest system remains far from complete collection, and rankings on the easiest level do not carry over to the hardest. Second, agents fail mainly by omission rather than by misreading. Most missing information was never delivered to the agent, and as difficulty grows, the dominant gap shifts from unopened detail pages to records that were never reached. Third, the backbone model decides what an agent collects, whereas the harness changes the score by only a few points.

Our main contributions are as follows.

- **A long-horizon web benchmark in a controllable environment.** LongWebBench provides 85 generated, self-hosted data collection tasks across five difficulty levels, in which dependent discovery and dispersed fields make runs long and perturbations along the run test whether agents can sustain collection.
- **A stage-wise evaluation grounded in delivery records.** We link the private ground truth, the server's delivery records, and the final submission to diagnose Interface Access, State Control, Data Acquisition, and Result Assembly alongside the Overall score.
- **An empirical study of current agents.** Across ten systems and eight diagnostic configurations, we show that rankings on easy tasks do not carry over to hard ones, that agents fail mainly by omission as the dominant gap moves from unopened detail pages to unreached records, and that the backbone rather than the harness decides what is collected.

2 Related Work

Long-horizon web-agent benchmarks. WebArena and VisualWebArena host self-contained web applications, and REAL and WebForge simulate or generate websites for reproducibility (Zhou et al.,

2024; Koh et al., 2024; Garg et al., 2025; Yuan et al., 2026). WorkArena++, WebChoreArena, and Odysseys lengthen tasks through dependent workflows, memory-heavy objectives, or multi-site goals (Boisvert et al., 2024; Miyai et al., 2026; Jang et al., 2026). These benchmarks make tasks long but mainly check an end state, so they cannot show what an incomplete run missed.

Large-scale information seeking and extraction. WideSearch and DeepWideSearch fill tables through a search API and a page reader (Wong et al., 2026; Lan et al., 2025), WebLists and BaRA collect record sets from live sites within a few result pages or a fifty-page budget (Bohra et al., 2025; Lee et al., 2026), and web information extraction works on pages given in advance (Huang et al., 2024; Yang et al., 2026). These works share our focus on complete outputs, but their retrieval is mediated, bounded, or given, whereas LongWebBench requires end-to-end collection over hundreds of pages under changing access conditions.

Process-aware and stateful evaluation. AgentBoard, WebCanvas, and WebStep score subgoals, key nodes, or state transitions along a trajectory (Ma et al., 2024; Pan et al., 2024; Chung et al., 2026), while StressWeb, Open CaptchaWorld, ContractBench, and WAREX test perturbations, CAPTCHAs, stateful contracts, or transient faults (Bai et al., 2026; Luo et al., 2025; Wang et al., 2026a; Kara et al., 2025). They measure progress or gate passage, but not whether the information served during a run reaches a multi-record output. Among the benchmarks in Table 1, LongWebBench is the only one we identify that combines long-horizon tasks, a controllable environment with mid-run perturbations, and stage-wise diagnosis, although its synthetic sites do not reproduce the live-web settings of Odysseys, WebLists, or WideSearch. Appendix A discusses all three lines in detail.

3 The LongWebBench Benchmark

LongWebBench evaluates whether agents can sustain large-scale web collection over long chains of dependent requests. Its name reflects its three components, which Figure 1 outlines. Section 3.1 defines the *Long* task, Section 3.2 describes the *Web* environment that hosts it, and Section 3.3 presents the *Bench* evaluation, which compares each run with what the environment actually delivered.

3.1 Long: Long-Horizon, Full-Dataset Collection Tasks

Complete-dataset collection is an established setting for evaluating web agents (Wong et al., 2026; Bohra et al., 2025), and it lends itself to long horizons, because a complete dataset spreads over many pages that can be reached only through links, indexes, and access steps. We therefore instantiate long-horizon tasks as web data collection. Each task provides a natural-language objective, an entry URL, and an output schema, and asks for every target record with all required fields correct. The instructions disclose neither credentials, target values, nor the size of the target set, so the agent has to judge from the site itself when it is done. The agent may browse, issue HTTP requests, or write code, so choosing a strategy is part of the task. To cover varied sites, the 85 tasks derive from 20 archetypes, each pairing a site family, such as a catalog, directory, or dashboard, with a content domain, such as e-commerce, finance, or travel.

Dependent discovery and dispersed fields. A complete-collection objective alone does not guarantee a long horizon, since a short loop could export a single paginated listing. We therefore distribute the target data so that no single view completes the export, as Figure 1a shows. Each task contains 500–2,900 records, and from L2 onward, a growing share of them appears only through alternative indexes, tabs, or links on detail pages, until nearly half of the target records lie beyond the default listing at L5. Opaque identifiers in six L2 tasks and all L3–L5 tasks prevent enumeration, so the set of pages to visit grows during retrieval. Each record also has 9–12 fields spread across listing, detail, and protected views, which the agent has to join by record identifier. A run therefore follows newly revealed routes, opens each record, and tracks which fields every record still lacks, as Figure 2a illustrates. In our runs, a task on the harder levels takes hundreds of LLM turns with about one tool call each. Because each single-use cursor appears only in the response to the previous page, the cursor listings of L4–L5 also add chains of about a hundred dependent requests that cannot run in parallel, as Appendix B.4 details.

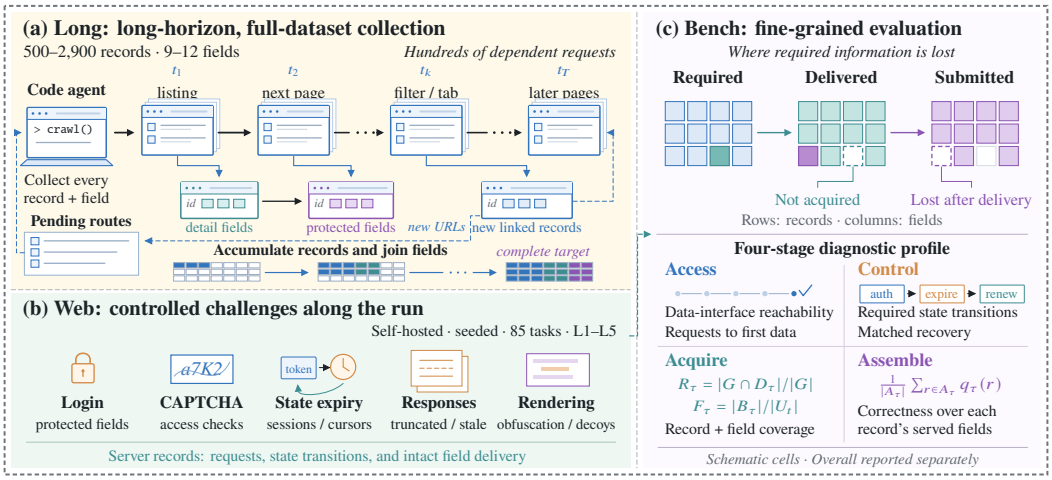


Figure 1: Overview of LongWebBench. (a) Each task asks for a complete dataset that the agent collects across many requests. (b) Self-hosted, seeded websites place controlled challenges along the run. (c) The evaluation compares required, delivered, and submitted information and reports a four-stage diagnostic profile.

3.2 Web: Controlled Challenges on Self-Hosted Websites

Self-hosted environments support reproducible web-agent evaluation (Zhou et al., 2024; Garg et al., 2025). To make these long tasks controllable, we host them on generated websites. Each website and its private ground truth are generated from one specification of domain, site family, rendering mode, discovery structure, and access rules, and a fixed seed sets the content, layout, and challenges. Shared record and field identifiers let the evaluator match server responses to the required information, and because the content exists only on the benchmark host, it cannot be recalled from pretraining data. Every run uses an isolated workspace and one tracked session, and the server records its requests, state transitions, and intact field deliveries for the evaluation of Section 3.3. Appendix B details the generation, the suite of 113,060 target records, and the construction tests.

Controlled challenges along the run. Hosting the sites also lets us disrupt the agent as long runs are disrupted in practice, where network faults and expiring state break valid workflows (Kara et al., 2025; Wang et al., 2026a). To this end, the benchmark server implements 29 protection mechanisms, and together with authentication, expiring cursors, and recoverable disruptions, they form five kinds of challenges along the collection path, as Figures 1b and 2b show. Login walls protect some fields, CAPTCHA-style and browser checks gate access to data pages, sessions and cursors expire, responses may arrive truncated or stale, and rendering varies through obfuscation and decoy records. Because these challenges arise mid-run, the agent needs to keep reaching the data, check each response, and restore lapsed state without losing earlier results. They simulate common web mechanisms rather than commercial anti-bot services.

Difficulty levels. To separate easy settings from hard ones, the 85 tasks form five composite difficulty levels that vary record scale, challenge density, and time limits together. L1 requires only public listing and detail retrieval, and each later level adds requirements until L5 combines all kinds of challenges at the largest scale, as Appendix B.3 details.

3.3 Bench: Evidence-Grounded Evaluation

A long run can fall short in qualitatively different ways. An agent may never reach a page, lose its session midway, open a page without extracting its fields, or collect the data and drop it from the submission. These failures call for different remedies, yet they leave the same trace in the final output, a missing record or field. A final score therefore cannot tell them apart, and process-level evaluations, which score subgoals or states along a trajectory (Ma et al., 2024; Chung et al., 2026), do not record



Figure 2: Examples from the generated sites. (a) A long-horizon collection chain on one site. (b) Challenges placed along the collection path. Appendix C.4 shows more sites and challenges.

which required information the environment served. Only the environment knows what it served. We therefore link three records of each run, namely what the task requires, what the server delivered, and what the agent submitted, so that every missing field can be located before or after delivery.

Linking required, served, and submitted information. To make this linkage possible, the website and the ground truth share record identifiers, and the server logs each run's data-bearing responses. We define *acquisition* as intact delivery recorded by the server and keep the best recorded state of each field across repeated deliveries. Recorded delivery thus splits the gap between required and submitted information into *coverage*, from required to served, and *fidelity*, from served to submitted, so an incomplete output can be traced to partial delivery or to losses after full delivery, as Appendix D.5 details.

Four stage diagnostics. Building on this linkage, we report four diagnostics for each run, one for each stage that a collection run passes through. Interface Access checks whether the agent reached a data interface, and a state- or budget-related refusal also counts because it proves reachability. State Control is a protocol-compliance verdict on whether the task's state requirements were met, including ordered transitions and recoveries that match the failed condition. Data Acquisition measures how much required information the server delivered, independently of the submission, and Result Assembly measures how faithfully the agent preserves it. Let G be the target identifiers, Φ_t the required fields, and $U_t = \{(r, f) : r \in G, f \in \Phi_t\}$ the required record–field pairs. For run τ, let $A_\tau = G \cap D_\tau$ be the target records among the delivered identifiers D_τ, $B_\tau \subseteq U_t$ the pairs served intact, and $q_\tau(r)$ the fraction of the served fields of $r \in A_\tau$ that are correct in its best matching submitted row, with zero for an omitted record. Then

$$\textsc{Acquire}_\tau = \text{mean}\left\{\frac{|A_\tau|}{|G|}, \frac{|B_\tau|}{|U_t|}\right\}, \qquad \textsc{Assemble}_\tau = \frac{1}{|A_\tau|}\sum_{r \in A_\tau} q_\tau(r), \tag{1}$$

where Acquire also averages in file coverage when a task requests files. With field-state logs, Assembly is scored only over served fields, so the same loss is not penalized twice. Legacy traces use the full-field fallback of Appendix D.4 and do not share this guarantee.

Overall score and aggregation. To compare systems on task completion, the canonical Overall score takes a weighted mean of record recall, field quality, precision, workflow coverage, authentication decisions, and request efficiency, renormalizes inapplicable weights, and scores unsubmitted tasks as zero. We report pass rates for Access and Control and mean scores for Acquisition and Assembly. Appendix D gives the weights, the recall strata, the status rules, and strict row fidelity, and Appendix E tests the evaluator on constructed inputs.

Table 2: Full-suite leaderboard on 85 tasks and a 0–100 scale, with the four stage diagnostics.

Model	Harness	Access ↑	Control ↑	Acquire ↑	Assemble ↑	Overall ↑ [95% CI]
Claude Opus 5	Claude Code	100.0	27.0	73.8	**82.4**	**70.1** [65.8, 73.7]
GPT-5.6 Sol	Codex	100.0	23.8	68.9	76.4	67.3 [63.9, 70.4]
Gemini 3.8 Flash High	Antigravity	100.0	27.0	64.5	78.1	66.7 [61.8, 71.6]
DeepSeek V4.1 Flash	Claude Code	95.3	16.9	**75.0**	77.6	64.8 [60.8, 69.3]
GPT-5.6 Luna	Codex	100.0	20.6	67.7	73.2	63.8 [59.9, 67.8]
GLM-5.3	Claude Code	98.8	12.7	73.0	69.1	63.5 [59.2, 68.1]
GPT-5.6 Terra	Codex	100.0	**32.3**	59.9	75.3	61.4 [57.1, 65.7]
DeepSeek V4 Pro	Claude Code	100.0	16.4	74.3	55.6	50.9 [42.4, 59.7]
Sonnet 4.6, low effort	Claude Code	100.0	17.5	60.0	50.8	45.2 [39.7, 51.2]
Qwen3.8-Flash	Claude Code	100.0	8.3	59.4	68.7	40.7 [33.4, 48.1]

4 EXPERIMENTS

Section 4.2 asks how far current agents are from complete collection and how this depends on difficulty and the harness, Section 4.3 locates where the missing information is lost, and Section 4.4 asks how submissions and resource use shape what a run returns.

4.1 EXPERIMENTAL SETUP

Models. Our backbones span the Claude (Anthropic, 2026a;b), GPT-5.6 (OpenAI, 2026), Gemini 3 (Google DeepMind, 2026), DeepSeek V4/V4.1 (DeepSeek-AI, 2026a;b), GLM-5 (GLM-5-Team, 2026), and Qwen3.8 (Qiu et al., 2026) model families. Ten backbones complete all 85 tasks, namely Claude Opus 5, Claude Sonnet 4.6 at low effort, GPT-5.6 Sol, Terra, and Luna, Gemini 3.8 Flash at high effort, DeepSeek V4.1 Flash and V4 Pro, GLM-5.3, and Qwen3.8-Flash.

Harnesses. Each backbone runs inside an agent harness, and we call the pair a system. The Claude, DeepSeek, GLM, and Qwen models run in Claude Code (Anthropic, n.d.b), the GPT-5.6 models in Codex (OpenAI, n.d.), and Gemini 3.8 Flash in Antigravity (Google, n.d.). To separate the backbone from the harness, we also run DeepSeek V4.1 Flash and GLM-5.3-Flash in Codex and Pi (Zechner, 2025) on the 58 L1–L3 tasks, next to their Claude Code runs.

Protocol. Every run receives an isolated workspace, one tracked session, and the same task prompt. Time limits are 15/20/30/60/120 minutes for L1–L5, and there is no token or step cap. A task without a submission scores zero, and we report the selected run for each task rather than the best of several attempts. Overall is the task macro mean on a 0–100 scale, and brackets give 95% percentile-bootstrap intervals over the 20 task archetypes (Efron, 1979). Because only locally run configurations have delivery records and traces, Sections 4.3 and 4.4 analyze the 525 archived runs of eight of them.

4.2 OVERALL PERFORMANCE

Even the strongest system is far from complete collection. In Table 2, Claude Opus 5 and GPT-5.6 Sol lead, followed by a cluster of systems whose intervals overlap, yet even the leaders remain far from complete collection. No system acquires more than three quarters of the required records and fields. At the bottom, weaker systems lose much of their score to tasks they never submit rather than to poor exports, as Qwen3.8-Flash submits on only about two thirds of the tasks. The stage columns also separate systems that Overall groups together. Access is near ceiling for every system, whereas Control passes in at most a third of applicable tasks, and the best Acquire and the best Assemble belong to different systems, DeepSeek V4.1 Flash and Opus 5.

The hardest levels decide the ranking. To see where the ranking comes from, Figure 3 breaks Overall down by level, and Appendix F.2 shows the other systems. Scores fall steeply with difficulty, but the systems do not fall alike. The four systems lie within seven points of each other at L1 but spread over 25 points at L5. Moreover, Opus 5 ranks last at L1 and first at L5, and L5 orders the four systems exactly as Overall does, so a suite of only easy tasks would rank them very differently. Because scale, challenges, and time limits change together, the level trend isolates no single factor.

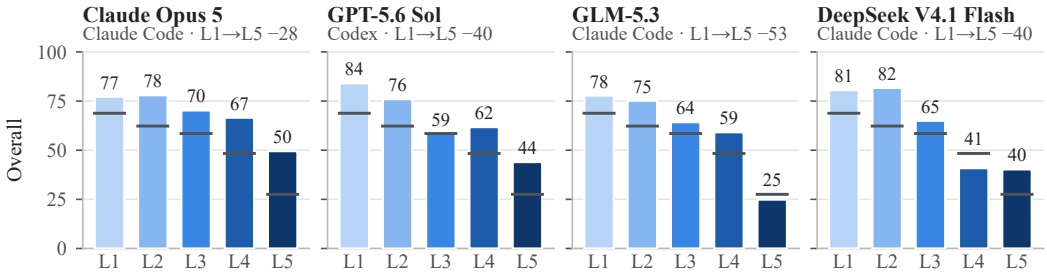


Figure 3: Overall by difficulty level for four systems, with grey ticks at the mean of all systems.

Table 3: Harness comparison on L1–L3. Δ is the paired Overall difference from Claude Code.

Backbone	Harness	Access ↑	Control ↑	Acquire ↑	Assemble ↑	Overall ↑ [95% CI]	Δ
DeepSeek V4.1 Flash	Claude Code	100.0	**25.0**	**90.0**	80.6	76.1 [71.6, 79.9]	–
	Codex	100.0	16.7	88.8	80.9	76.6 [72.4, 80.0]	+0.5
	Pi	100.0	22.2	89.1	**83.2**	**78.1** [74.4, 80.9]	**+2.0**
GLM-5.3-Flash	Claude Code	100.0	19.4	80.3	**60.4**	57.6 [49.2, 65.7]	–
	Codex	100.0	**25.3**	78.6	57.5	55.8 [47.6, 63.8]	−1.8
	Pi	100.0	18.8	**81.7**	60.2	**58.4** [49.8, 66.7]	**+0.8**

Overall scores are similar across harnesses. To isolate the harness, Table 3 holds the backbone fixed and varies the harness on the 58 L1–L3 tasks. For both backbones, the three harnesses stay within about three points of each other in Overall and Acquire, with overlapping intervals, whereas within each harness the two backbones differ by about 20 points. The backbone, not the harness, thus decides what an agent collects. Appendix F.3 also compares single- and multi-agent execution.

4.3 Where Incompleteness Arises

A low score can come from any stage of a run, so we next locate where the missing information is lost. To this end, we partition the required record–field pairs of the 524 profiled runs by recorded delivery and output correctness, and we split each stage into sub-components, as Appendix G details.

Long-horizon agents fail by omission, not by misreading. Figure 4 shows that only about half of the required pairs are delivered and returned correctly. Most of the rest is never delivered, and a smaller share is lost after delivery. The undelivered share also contains a few pairs that are correct in the output without recorded delivery, so it measures gaps in delivery-supported correctness rather than only missing output. Before delivery, agents mostly leave detail pages unopened or never reach a record, and after delivery, runs end without a submission or return a correct value in the wrong surface form. Substantively wrong values are rare, so agents seldom misread what they obtain. Figure 5 locates the gap in the pages that agents skip, as on L1–L3 every configuration reaches at least 85% of the records on the default listing but receives only 35–70% of the detail fields. For agents, the limit is therefore coverage, namely reaching every record and opening every detail page, rather than reading each page correctly. For evaluation, the correctness of returned values says little about completeness, and only the delivery records separate pairs never delivered from later losses.

As horizons grow, the bottleneck moves from fields to records. At L1, agents lose pairs mainly to unopened detail pages and formatting, whereas at L5 most pairs belong to records that were never delivered, as the left panel of Figure 4 shows. At L5, these unreached records split about evenly between the cursor-paged default listing and the records behind filters and search, at 30% and 33% of all pairs, and Appendix G.1 shows the same shift within the two configurations that ran every level. Discovery also separates the backbones most clearly. In Figure 5, DeepSeek V4.1 Flash finds 76–85% of the records behind filters and search in every harness, whereas GLM-5.3-Flash finds 21–37%. Similar scores can also hide different routes. Gemini 3.8 Flash and GLM-5.3 have overlapping intervals in Table 2, yet on L1–L3 Gemini recovers more cross-referenced records and far fewer records behind filters and search.

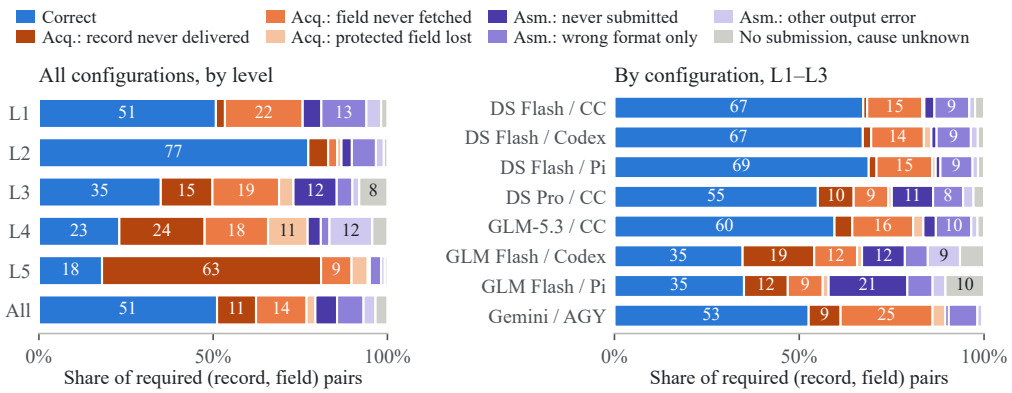


Figure 4: Delivery-supported correctness of required record–field pairs. Blue pairs are delivered and returned correctly, orange pairs lack recorded intact delivery, and purple pairs are lost after delivery.

		DeepSeek V4.1 Flash			DeepSeek V4 Pro	GLM-5.3	GLM-5.3-Flash		Gemini 3.8 Flash	All systems
		CC	Codex	Pi	CC	CC	Codex	Pi	AGY	L4–L5
	Access gates cleared	100	97	100	95	100	93	83	83	99
Control	Authentication	55	45	48	45	39	47	61	71	15
Control	API token	20	30	30	30	67	40	50	30	–
Control	Cookie chain	89	78	78	56	67	44	56	89	67
Control	URL session id	0	0	0	30	0	0	10	30	0
Acquire	Listing records	100	100	100	92	97	86	92	97	65
Acquire	Filter/search records	85	76	78	69	68	21	37	39	54
Acquire	Cross-referenced records	75	79	78	63	56	19	19	82	18
Acquire	Detail fields	63	65	62	70	55	52	59	35	22
Acquire	Protected fields	93	78	88	85	70	69	67	57	9
Assemble	Submitted at all	95	97	97	79	95	70	60	97	86
Assemble	Delivered values correct	85	84	86	81	82	67	75	83	65
Assemble	Records exactly right	24	22	29	15	14	0	5	21	9

Figure 5: Stage sub-components by configuration on a 0–100 scale, where darker cells are better.

Pass rates misstate what challenges cost. The Access and Control rows of Figure 5 show how often agents get through each challenge, and Table 4 adds what verification gates cost them. Nearly every gate on a data-bearing page is eventually followed by delivered records, although this does not establish recovery after a refusal. Timing checks and CAPTCHA-style challenges, however, always fire before the first data response and cost a median of 224 and 84 refused requests per run, half and a sixth of the task's request budget. A gate's high pass rate can therefore coexist with a substantial request cost. State protocols, in contrast, pass far less often and differ widely by family. On L1–L3, cookie chains pass in 44–89% of applicable runs, authentication in 39–71%, and URL session identifiers in at most 30%, and authentication falls to 15% on L4–L5, as Appendix G.3 details.

4.4 Progress and Resources over a Run

A run's outcome depends not only on what it collects but also on when it submits and what it spends along the way. We therefore measure prompt tokens, wall-clock time, and site requests per run, and we impute the tokens of runs stopped at the time limit from their elapsed time and level.

Late or missing submissions waste collected data. A run earns its score only through its submission, so data collected but never submitted count for nothing. In Figure 4, such runs cost DeepSeek V4 Pro and both GLM-5.3-Flash arms 11–21% of the required pairs on L1–L3. These three configurations also reach the time limit in 55–64% of their L1–L3 runs, and in the right panel of Figure 6, the curves of DeepSeek V4 Pro and GLM-5.3-Flash in Pi climb in steps at the 15-, 20-, and 30-minute limits, where their stopped runs end. Part of what separates systems is thus when they write and stop,

Table 4: Verification gates on data-bearing pages in the eight diagnostic configurations. A gate is *cleared* when the blocked page delivered records at any point of the run and *bypassed* when the same data came through another page. The last column relates refused requests to the request budget.

Gate family	Levels	Runs meeting it	Blocked a data page	Cleared	Bypassed	Not cleared	Met before first data	Median refused requests and share of budget
XHR header on data API	L1–L5	210	209	197 (94%)	11	1	3%	2 (1%)
Browser proof	L4–L5	36	36	36 (100%)	0	0	36%	6 (1%)
Client fingerprint, UA and headers	L2–L5	48	27	27 (100%)	0	0	29%	2 (1%)
Timing check	L3	23	23	22 (96%)	0	1	100%	**224 (50%)**
CAPTCHA-style challenge	L3–L5	48	34	33 (97%)	1	0	100%	**84 (16%)**

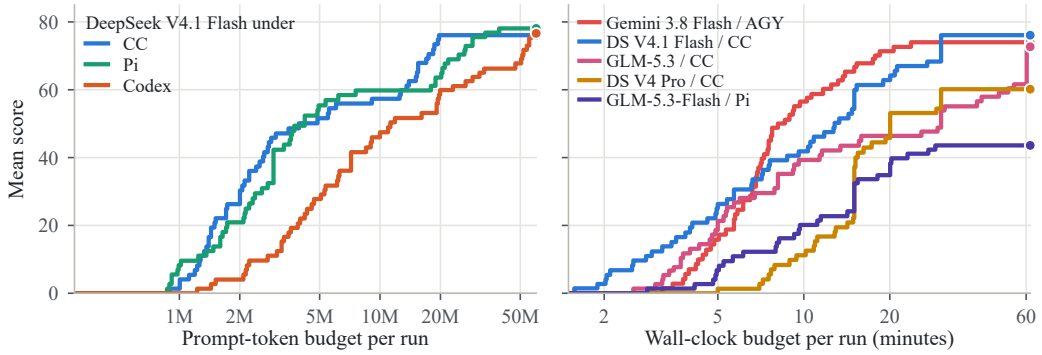


Figure 6: Retrospective mean L1–L3 final score under a budget, counting a run only when it finishes within a prompt-token threshold in the left panel or a wall-clock threshold in the right panel. The curves do not measure the answer held at an intermediate cutoff.

and only the delivery records distinguish this timing from what they collect. Appendix H.1 follows the answer held during a run for the two configurations whose harness scores every step.

A final score hides the resources spent to reach it. Beyond timing, systems with the same final score can differ widely under a budget. In the left panel of Figure 6, DeepSeek V4.1 Flash in Codex spends nearly three times as many tokens as in Claude Code, so it scores far lower under a small token budget although both end at the same score. In the right panel, Gemini 3.8 Flash leads DeepSeek V4.1 Flash by a wide margin after ten minutes although both end within two points. A final score therefore does not show which system is more efficient, and Appendix H.2 relates coverage to site requests, tokens, and time across all diagnostic runs.

5 Conclusion

We introduced LongWebBench, a benchmark that evaluates agents on long-horizon web data collection in a controllable environment and diagnoses each run stage by stage. Its generated, self-hosted websites make tasks long by construction and disrupt agents along the way, while the server's delivery records let the evaluation separate fields that were never delivered from fields delivered intact but lost in the submission. Across ten agent systems, even the strongest remains far from complete collection, and rankings on the easiest level do not carry over to the hardest. Agents fail mainly by omission, since most missing information is never delivered to them, and the dominant gap shifts from unopened detail pages to unreached records as difficulty grows. The backbone model, rather than the harness, decides what is collected. Verification gates rarely stop agents but can consume half of a task's request budget, and runs that end without a submission waste data they have already collected. These findings point agent development toward searching beyond the default listing, fetching every detail page, and submitting and normalizing the collected data before a run ends.

AI USE STATEMENT

We used GPT-5.6 Sol (OpenAI) to polish the manuscript, check grammar, assist with code implementation, and write code for visualizing the experimental results. We did not use generative AI for any other task that requires disclosure under the ICLR 2027 AI Policy for Authors, and the benchmark data are produced by our seeded generation programs. The authors reviewed and edited all AI-assisted text, checked the code and figures against the intended methods and results, and take full responsibility for the final manuscript and artifacts. Separately, GPT-5.6 Sol is evaluated in Section 4 as an object of study, under the same protocol as every other system.

REPRODUCIBILITY STATEMENT

Section 3 defines the tasks, websites, and evaluation. Appendix B details task generation from seeds (Appendix B.2), the level parameters (Appendix B.3), and the run conditions and task specification version (Appendix B.5). Appendix C describes the execution pipeline, agent prompt, and harness setup and versions, and Section 4.1 lists the evaluated configurations. Appendices D and E define all metrics and the integrity tests. The supplement includes the benchmark and evaluator code.

REFERENCES

Anthropic. Claude Opus 5 system card. Technical report, Anthropic, July 2026a. URL https://anthropic.com/claude-opus-5-system-card.

Anthropic. Claude Sonnet 4.6 system card. Technical report, Anthropic, February 2026b. URL https://anthropic.com/claude-sonnet-4-6-system-card.

Anthropic. Claude Agent SDK overview, n.d.a. URL https://code.claude.com/docs/en/agent-sdk/overview. Accessed 2026-09-26.

Anthropic. Claude Code overview, n.d.b. URL https://code.claude.com/docs/en/overview. Accessed 2026-09-26.

Haoyue Bai, Dong Wang, Long Chen, Bingguang Hao, Pengyang Shao, Yonghui Yang, Yicheng He, and Chenyi Zhuang. StressWeb: A diagnostic benchmark for web agent robustness under realistic interaction variability. *arXiv preprint arXiv:2604.16385*, 2026.

Arth Bhardwaj, Nirav Diwan, and Gang Wang. Beyond BeautifulSoup: Benchmarking LLM-powered web scraping for everyday users. *arXiv preprint arXiv:2601.06301*, 2026.

Arth Bohra, Manvel Saroyan, Danil Melkozerov, Vahe Karufanyan, Gabriel Maher, Pascal Weinberger, Artem Harutyunyan, and Giovanni Campagna. WebLists: Extracting structured information from complex interactive websites using executable LLM agents. *arXiv preprint arXiv:2504.12682*, 2025.

Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. WorkArena++: Towards compositional planning and reasoning-based common knowledge work tasks. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

William Brach, Francesco Zuppichini, Marco Vinciguerra, and Lorenzo Padoan. ScrapeGraphAI-100k: Dataset for schema-constrained LLM generation. *arXiv preprint arXiv:2602.15189*, 2026.

Jiwan Chung, JiHyuk Byun, Vibhav Vineet, and Seon Joo Kim. Where did it go wrong? process-level evaluation of web agents with semantic state tracking. In *Conference on Language Modeling*, 2026.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026a.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026b.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 2024.

B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979. doi: 10.1214/aos/1176344552.

Iliana Fayolle, Sihem Bouhenniche, Samuel Pélissier, Pierre Laperdrix, Clémentine Maurice, and Walter Rudametkin. On the internet, nobody knows you're an LLM bot: Unmasking web agents with multi-layer fingerprinting. *arXiv preprint arXiv:2606.30119*, 2026.

Div Garg, Diego Caples, Andis Draguns, Nikil Ravi, Pranav Putta, Naman Garg, Prannay Hebbar, Youngchul Joo, Jindong Gu, Charles London, Christian Schroeder de Witt, and Sumeet Motwani. REAL: Benchmarking autonomous agents on deterministic simulations of real websites. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4536.

GLM-5-Team. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Google. Antigravity CLI overview, n.d. URL https://antigravity.google/docs/cli/overview/. Accessed 2026-09-26.

Google DeepMind. Gemini 3.8 Flash. Model card, Google DeepMind, September 2026. URL https://deepmind.google/models/model-cards/gemini-3-8-flash/.

Wenhao Huang, Zhouhong Gu, Chenghao Peng, Jiaqing Liang, Zhixu Li, Yanghua Xiao, Liqian Wen, and Zulong Chen. AutoScraper: A progressive understanding web agent for web scraper generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 2371–2389. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.141.

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.

Su Kara, Fazle Faisal, and Suman Nath. WAREX: Web agent reliability evaluation on existing benchmarks. *arXiv preprint arXiv:2510.03285*, 2025.

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

Tian Lan, Bin Zhu, Qianghuai Jia, Junyang Ren, Haijun Li, Longyue Wang, Zhao Xu, Weihua Luo, and Kaifu Zhang. DeepWideSearch: Benchmarking depth and width in agentic information seeking. *arXiv preprint arXiv:2510.20168*, 2025.

Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Quentin Cappart, Graham Neubig, Ruslan Salakhutdinov, Nicolas Chapados, and Alexandre Lacoste. The BrowserGym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2025.

Soojeong Lee, Joseph Lee, Yongseong Cho, Sunjae Kim, Youngwoo Moon, and Kyungwoo Song. BaRA: Budget-constrained and reliable web data collection agent. *arXiv preprint arXiv:2607.00007*, 2026.

Eric Li and Jim Waldo. WebSuite: Systematically evaluating why web agents fail. *arXiv preprint arXiv:2406.01623*, 2024.

Yaxin Luo, Zhaoyi Li, Jiacheng Liu, Jiacheng Cui, Xiaohan Zhao, and Zhiqiang Shen. Open CaptchaWorld: A comprehensive web-based platform for testing and benchmarking multimodal LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3006.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Atsuyuki Miyai, Zaiying Zhao, Kazuki Egashira, Atsuki Sato, Tatsumi Sunada, Shota Onohara, Hiromasa Yamanishi, Mashiro Toyooka, Kunato Nishina, Ryoma Maeda, Kiyoharu Aizawa, and Toshihiko Yamasaki. WebChoreArena: Evaluating web browsing agents on realistic tedious web tasks. In *Conference on Language Modeling*, 2026.

OpenAI. GPT-5.6 system card. Technical report, OpenAI, July 2026. URL https://deploymentsafety.openai.com/gpt-5-6.

OpenAI. Codex CLI, n.d. URL https://learn.chatgpt.com/docs/codex/cli. Accessed 2026-09-26.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. WebCanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*, 2024.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, Fan Zhou, Hao Luo, Haofeng Huang, Haoran Lian, Haoyan Huang, Hongqing Chen, Jianwei Zhang, Jing Xu, Junjie Wang, Langshi Chen, Liangyu Wang, Linlang Jiang, Man Yuan, Minmin Sun, Peng Jin, Siqi Zhang, Siyu Wang, Xingzhang Ren, Yakai Wang, Yi Zhang, Yiming Dong, Yizhong Cao, Yubo Ma, Yunfei Mao, Bo Zheng, and Dayiheng Liu. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026.

Jicheng Wang, Yifeng He, Zili Wang, Hanwen Xing, Arkaprava De, and Hao Chen. ContractBench: Can LLM agents preserve observation contracts? *arXiv preprint arXiv:2605.17281*, 2026a.

Shoupeng Wang, Jiantao Qiu, Wuyang Zhang, and Conghui He. Co-Scraper: query-aware DOM pruning and reusable scraper synthesis for lightweight web data extraction. *arXiv preprint arXiv:2606.14821*, 2026b.

Ryan Wong, Jiawei Wang, Junjie Zhao, Li Chen, Yan Gao, Long Zhang, Xuan Zhou, Zuo Wang, Kai Xiang, Ge Zhang, Wenhao Huang, Yang Wang, and Ke Wang. WideSearch: Benchmarking agentic broad info-seeking. In *International Conference on Learning Representations*, 2026.

Zonglin Wu, Yule Xue, Yaoyao Feng, Xiaolong Wang, and Yiren Song. MCA-Bench: A multimodal benchmark for evaluating CAPTCHA robustness against VLM-based attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 38039–38047, 2026. doi: 10.1609/aaai.v40i44.41141.

Seungbin Yang, Jihwan Kim, Jaemin Choi, Dongjin Kim, Soyoung Yang, ChaeHun Park, and Jaegul Choo. LiveWeb-IE: A benchmark for online web information extraction. In *International Conference on Learning Representations*, 2026.

Peng Yuan, Yuyang Yin, Yuxuan Cai, and Zheng Wei. WebForge: Breaking the realism-reproducibility-scalability trilemma in browser agent benchmark. *arXiv preprint arXiv:2604.10988*, 2026.

Mario Zechner. What I learned building an opinionated and minimal coding agent, November 2025. URL https://mariozechner.at/posts/2025-11-30-pi-coding-agent/. Accessed 2026-09-26.

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. In *International Conference on Learning Representations*, 2024.

APPENDIX CONTENTS

A Extended Related Work **14**
- A.1 Long-Horizon and Compositional Web-Agent Benchmarks . . . 14
- A.2 Large-Scale Information Seeking and Web Data Extraction . . . 14
- A.3 Process-Aware Evaluation and Stateful Interaction . . . 15

B Benchmark Construction **16**
- B.1 Design Rationale with One Catalog Task . . . 16
- B.2 Instance Generation and Agent Interface . . . 17
- B.3 Difficulty Levels . . . 18
- B.4 Horizon Length . . . 19
- B.5 Run Conditions and Reproducibility . . . 19

C Conducting a Benchmark Run **21**
- C.1 Execution Pipeline . . . 21
- C.2 What the Agent Receives . . . 21
- C.3 Harnesses and Isolation . . . 22
- C.4 What the Agent Sees . . . 23
- C.5 Anatomy of One Run . . . 24

D Evaluation Metrics **28**
- D.1 Notation and Value Matching . . . 28
- D.2 The Overall Score . . . 28
- D.3 Sensitivity of Overall to Its Scoring Rules . . . 31
- D.4 Stage Diagnostics . . . 31
- D.5 Delivery–Output Accounting . . . 32

E Evaluation Integrity Tests **34**

F Additional Leaderboard Results **35**
- F.1 Notes on the Main-Text Tables and Figures . . . 35
- F.2 Results by Difficulty Level . . . 35
- F.3 Single- versus Multi-Agent Execution . . . 35

G Additional Fine-Grained Diagnostics **37**
- G.1 Sub-Components and the Loss Partition . . . 37
- G.2 Verification Gates in Access . . . 37
- G.3 Protocol Families in Control . . . 38
- G.4 Acquire and Assemble by Level . . . 39

H Additional Efficiency Results **41**
- H.1 Progress within a Run . . . 41
- H.2 Resource Use and Outcomes . . . 43

I Limitations **44**

A Extended Related Work

Section 2 positions LongWebBench against three lines of work only briefly. This appendix discusses each line in more depth, namely long-horizon web-agent benchmarks in Appendix A.1, large-scale information seeking and extraction in Appendix A.2, and process-aware evaluation of stateful interaction in Appendix A.3. Because the discussion refers to the columns of Table 1, we define them first.

Columns of Table 1. The comparison groups six properties under the three components of LongWebBench. For the task, a *long chain* needs many dependent interactions across many pages or steps, and a *full record set* asks for every target record rather than one answer. For the environment, a *self-hosted* benchmark hosts or generates its sites, which keeps tasks reproducible and their content fixed, and a *perturbed* benchmark deliberately disrupts access, state, or responses while the agent works. For the evaluation, *process-level* scoring locates where a run lost progress instead of scoring only the final outcome, and *delivery-based* scoring compares the output with what the environment served during the run.

A.1 Long-Horizon and Compositional Web-Agent Benchmarks

Task completion and workflow complexity. Most web-agent benchmarks ask whether an agent completes a task, and they check its answer or the resulting application state. WebArena, for example, hosts self-contained replicas of e-commerce, forum, code-hosting, and content-management applications (Zhou et al., 2024), and VisualWebArena extends this setting to visually grounded tasks (Koh et al., 2024). WorkArena evaluates knowledge-work operations on an enterprise platform (Drouin et al., 2024). Building on these operations, WorkArena++ constructs workflows in which later steps depend on earlier outcomes, which calls for planning and reasoning across operations (Boisvert et al., 2024). WebChoreArena further introduces memory-heavy tasks that draw on many pages (Miyai et al., 2026), and Odysseys composes long, multi-site tasks on the live web and evaluates them with rubrics (Jang et al., 2026). To gain control over the environment, REAL and WebForge use deterministic simulations or generated websites (Garg et al., 2025; Yuan et al., 2026), and BrowserGym provides a common interface for many such benchmarks (Le Sellier De Chezelles et al., 2025). Across these settings, the evaluation thus checks whether the required answer, form, or application state was produced.

The specific challenge of sustained acquisition. Data collection differs from these tasks, because its deliverable is a set of records whose completeness no single page can establish. To complete it, an agent needs to discover further retrieval paths, maintain the state that these paths require, and retain the information it has already collected. As a result, the final collection may omit records or fields even when every individual operation succeeds. Whereas workflow evaluation can inspect end states or checkpoints, collection evaluation also has to account for information that goes missing across the run. The relevant requirement is therefore complete record and field coverage over dependent interactions, rather than task length alone.

Position. WorkArena++ and WebChoreArena already contain long and demanding tasks. Building on them, LongWebBench focuses on the completeness of a target set of records, fields, and files, together with evidence of which items the environment served. It lets us examine an incomplete output against both the task's target set and the information served during the run.

A.2 Large-Scale Information Seeking and Web Data Extraction

Information seeking with set-valued outputs. A related line of work evaluates structured collections rather than single answers. WideSearch asks agents to fill tables through a search API and a page reader and scores row- and item-level F1 against human-curated gold tables (Wong et al., 2026), and DeepWideSearch extends this setting with deeper per-item search (Lan et al., 2025). Similarly, WebLists asks agents to extract complete record sets under a given schema from fifty live company sites. It scores precision and recall, limits each task to five result pages, and permits termination on CAPTCHA or network errors (Bohra et al., 2025). These benchmarks share our focus on collection completeness and on record- and field-level output quality. Their access settings differ, however,

because a search API or reader mediates retrieval, or retrieval is bounded to a few pages and may stop at the first access error.

Extraction versus end-to-end acquisition. Web information extraction, in contrast, mainly evaluates content from pages that are already given to the system. SWDE-style datasets and AutoScraper evaluate field extraction and scraper executability on offline page sets (Huang et al., 2024). Co-Scraper and ScrapeGraphAI-100k study scraper synthesis and schema-constrained extraction (Wang et al., 2026b; Brach et al., 2026), while LiveWeb-IE extracts query-conditioned values from supplied URLs on consenting live sites (Yang et al., 2026). BaRA additionally requires page discovery on static synthetic sites and on live sites screened for bot challenges, and it scores a valid-link ratio under a fifty-page budget without authentication (Lee et al., 2026). End-to-end collection asks for more, since the agent needs to find retrieval paths, traverse listings and detail views, and handle access restrictions while it builds the dataset. Evaluating this process thus requires evidence of which target records were served, which neither the supplied pages nor the submitted collection can provide on its own.

The measurement problem. Because collection involves all of these steps, an incomplete collection can lose information at different points. Information may never have been served, or it may have been served and then omitted or misrepresented in the output. Output-only scoring, page budgets, and link ratios cannot tell these cases apart. To address this, we link environment-side delivery records to the target set and the submission, and Section 3.3 describes the resulting evidence model and scoring rules.

A.3 Process-Aware Evaluation and Stateful Interaction

Intermediate progress and execution evidence. Process-aware benchmarks look beyond final completion to measure intermediate progress. AgentBoard evaluates progress toward annotated subgoals (Ma et al., 2024), and WebCanvas matches human-annotated key nodes, such as URLs and elements, along live-web trajectories (Pan et al., 2024). WebSuite instruments interface components to attribute failures (Li & Waldo, 2024). StressWeb instead perturbs layout, markup, and interaction semantics and evaluates DOM checkpoints recorded by its harness (Bai et al., 2026). More recently, WebStep records semantic state transitions to separate exploration from execution and to locate the first divergent step (Chung et al., 2026). These measurements thus capture goal progress, intermediate actions and states, or, in WebStep, the information exposed before a single commit. Our comparison differs from all of them, since we evaluate a multi-record submission against the target information served across the run.

Access barriers and state requirements. Other work studies access barriers and stateful interaction directly. Open CaptchaWorld and MCA-Bench evaluate CAPTCHA solving as a standalone task (Luo et al., 2025; Wu et al., 2026). Fayolle et al. (2026) place self-hosted honeysites behind user-agent filters, CAPTCHAs, proof-of-work, and commercial bot management, and they use server logs to check whether an agent submits a form. Bhardwaj et al. (2026) group single-page scraping targets by authentication and anti-bot controls. For stateful APIs, ContractBench tests expiring tokens, session-cookie chains, cursors, rate-limit windows, and signed requests with server-log rewards (Wang et al., 2026a), and WAREX injects transient HTTP faults into existing benchmarks (Kara et al., 2025). In sustained collection, these conditions matter because they decide whether retrieval can continue, so an agent needs both to reach the interface and to maintain or recover the state that later requests need. We therefore examine access and state requirements while the agent collects a multi-record dataset, rather than treating gate passage or protocol compliance as the final deliverable.

Measurement objective of LongWebBench. In summary, prior work already covers set-valued outputs, access barriers, state requirements, and process-level evidence. LongWebBench combines a large multi-record deliverable with sustained access conditions and with per-record and per-field delivery records linked to the submission. To this end, we generate and host the websites, so that the target set, interaction rules, and delivery evidence align, which enables the delivery-conditioned diagnosis of Section 3.3. The same choice, however, limits realism, because Odysseys, WebLists, and WideSearch offer broader or live-web settings that our synthetic sites do not reproduce.

B BENCHMARK CONSTRUCTION

Sections 3.1 and 3.2 describe the tasks and the environment only as far as the results require. To make the benchmark reproducible and its design choices checkable, this appendix gives the construction details. We first explain the design with one task in Appendix B.1 and then describe how an instance is generated and what the agent is told in Appendix B.2. Appendix B.3 gives the parameters of the five levels, Appendix B.4 measures how long the tasks are, and Appendix B.5 states the run conditions and how runs are reproduced.

B.1 DESIGN RATIONALE WITH ONE CATALOG TASK

A single task shows most directly how the design makes collection long. Figure 7 follows a catalog task, in which 233 of the 1,060 target records are absent from the default listing and six of these can be found only through links on other products' detail pages. Each record also has 12 required fields, five from listings, five from detail pages, and two from an authenticated view. Completing the export therefore requires following the additional discovery routes and joining the three views by record identifier.

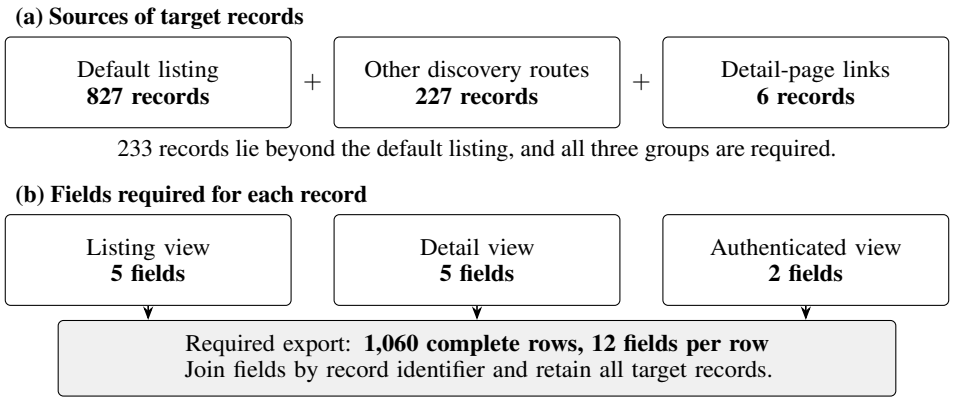


Figure 7: The catalog task. (a) The target set includes records from the default listing, other discovery routes, and detail-page links. (b) Required fields are distributed across three views. The counts and field allocations are available to the evaluator but are not given in the agent's instructions.

Dependent discovery and dispersed fields. Because the default listing omits some records, paging through it alone misses them. The agent thus needs to follow newly discovered routes and keep track of those it has already visited, guided by the site's navigation cues but without knowing the target count. Retrieving an identifier from a listing also does not provide its detail or protected fields. A listing-only crawl therefore leaves these fields missing even when it covers most target identifiers, so the agent has to track which fields each collected record still lacks, also when it automates retrieval.

Challenges along the horizon. Long runs also face conditions that change over time. Access obtained early in a run may lapse before collection is complete, and the agent then needs to restore the required state before resuming, without discarding earlier results. Under recoverable disruptions, an agent that treats a truncated listing or an empty page as the end of a listing, or a stale detail response as a complete record, leaves target information uncollected. Because seeded layouts, field-label aliases, and delivery formats vary, a parser written for one page template does not suffice, and decoy records call for validating retrieved content before it enters the output. In selected tasks, restrictions also depend on the agent's request behavior, and construction tests confirm that valid retrieval paths remain available.

Generated content and pretraining exposure. Collecting from existing public websites would compromise both control and validity. Their content and access policies change over time, and their pages may appear in pretraining corpora, so an agent could reproduce records or site structure from memory instead of collecting them, and a final score could not tell the two apart. To avoid this,

LongWebBench generates record values, identifiers, and credentials from the task seed and serves them only from the benchmark host. They thus correspond to no existing website and can be obtained only by interacting with the site. Because the content is a function of the specification and the seed, new instances can be generated from unreleased seeds if the released ones enter training data. As a further safeguard, the evaluator reports correct output values without recorded delivery separately from delivery-supported values, as Appendix D.5 describes.

B.2 Instance Generation and Agent Interface

To make every task reproducible from a compact specification, we generate each instance from its difficulty level ℓ and within-level index i, with content seed $10{,}000\ell + i$. We first select a domain–site pair and a permitted rendering mode and then set record volume, pagination, discovery paths, authentication, and recoverable disruptions. Because the same specification defines both the served records and the private evaluation data, the site stays consistent with its evaluation. Seeded layouts, visual identities, and field-label aliases then vary the presentation, and required protections are combined with a seeded sample of optional protections that is fixed when the run starts. Content and presentation are therefore reproducible, and each task can be rebuilt from its specification and seed, whereas sessions, adaptive responses, and timed transitions remain dynamic. In total, the suite covers 113,060 target records in 20 archetypes, which combine five content domains with five site families. Construction tests check that retrieval paths are reachable and that no answers leak through instructions, templates, or site decoration, as Appendices B.5 and E describe.

The instructions are designed to state the task without revealing its answers. The agent receives an objective, an entry URL, session-tracking instructions, authentication guidance, and an output template with field names and types, which Appendix C.2 shows in full. The template uses placeholder values, and the instructions state neither target values nor target counts. Likewise, the authentication guidance explains where to look for credentials without disclosing them. The agent submits structured records and, when a task requests them, file claims, provenance, and failure reports.

Construction requirements. To check that every construction serves a purpose, Table 5 links each property that a long collection run needs to sustain to the construction that enforces it, the shortcut that the construction rules out, and the evidence with which the evaluator measures the remaining gap. Record volume and dispersed fields implement the first two rows, and the challenges of Section 3.2 implement the other three. The rows are not fully separate, however, because session expiry can affect both state satisfaction and delivered coverage, and representation changes can affect both acquisition and assembly.

Table 5: Construction requirements, the constructions that enforce them, and the evidence used to measure the remaining gap.

Required property	Construction and ruled-out shortcut	Evidence of the remaining gap
Sustained coverage	Many records with dispersed fields, so one correct page cannot complete the export.	Target versus served record and field coverage.
Dependent discovery	Alternative indexes, tabs, and detail links, so the default listing is insufficient.	Delivery by discovery stratum and recorded route visits.
State continuity	Authentication, expiry, and cursors, so initial access need not persist.	Ordered state transitions and matching recovery events.
Response checking	Truncation, empty pages, and degraded fields, so a successful response may be incomplete.	Intact field states, repeated delivery, and unmet requirements.
Representation variation	Varying layouts, aliases, and delivery formats, so a fixed page template is insufficient.	Served field coverage and output fidelity under the configured presentation.

Research-workflow tasks. Beyond the 85 mechanism tasks, the suite contains three research-workflow tasks, a partner catalog, a member directory, and a dashboard statement, which test research practice rather than collection mechanisms. They additionally ask for downloaded files, source

provenance, and reports of expected failures, which the extension dimensions of Table 11 score, and they prescribe an order of steps, such as logging in before accepting a license. Because their outputs follow different rules, we keep them separate from the 85-task suite and leave them out of the reported results.

B.3 Difficulty Levels

To let difficulty grow in a controlled way, the five levels of Section 3.2 add collection requirements step by step. Table 6 summarizes their composition, and Table 7 adds their interaction parameters. Every level requires listing routes and detail views, because four or five required fields per record appear only in detail views, and the 65 non-public tasks also require two protected fields. Authentication may be optional at entry, deferred until a protected step, required at entry, or subject to expiry and renewal, so its position in the workflow depends on the task policy.

Table 6: Task composition by level. *Beyond listing* is the fraction of target records absent from the default listing, including detail-linked records. Time limits are 15/20/30/60/120 minutes for L1–L5.

Level	Tasks	Records / task	Beyond listing	Added collection requirements
L1	20	500–860	0%	Public listing and detail retrieval.
L2	20	800–1,250	10%	Alternative discovery routes, authentication, and protected fields.
L3	18	1,000–1,540	22%	Detail-linked records and asynchronous or embedded data.
L4	15	1,500–2,220	35%	Recoverable disruptions, expiring cursors, and browser rendering.
L5	12	2,000–2,900	45%	Expiring state, obfuscation, and decoys in every task.

Table 7: Interaction parameters by level. Counts in parentheses are tasks. Detail-linked records are a subset of the records absent from the default listing, and nominal requests are the budgets b of Equation (2).

Level	Authentication	Pagination	Data delivery	Detail-linked records	Nominal requests
L1	Public (20)	Numbered (20)	Server-rendered (20)	0	68–104
L2	Optional (4), deferred (8), required (8)	Numbered (10), next-link (10)	Server-rendered (10), asynchronous (10)	0	262–397
L3	Optional (3), deferred (7), required (8)	Numbered (6), next-link (6), hybrid (6)	Asynchronous (9), embedded data (9)	6	322–484
L4	Optional (3), deferred (3), required (6), expiring (3)	Next-link (5), cursor (5), hybrid (5)	Asynchronous (8), browser-rendered (7)	8	472–714
L5	Expiring (12)	Cursor (12)	Obfuscated, with image or streamed data (12)	10	522–773

The higher levels add time pressure and noise on top of these parameters. Cursor lifetimes are 45 seconds at L4 and 20 seconds at L5, and every L4–L5 task includes a recoverable disruption, drawn from truncated pagination, empty pages under rapid paging, an initially stale detail response, and incomplete field fragments. Six L2 tasks and all L3–L5 tasks use opaque identifiers, so agents need to discover links rather than enumerate sequential identifiers, and each L5 task additionally plants 285–414 decoy identifiers.

The mechanisms behind these requirements fall into three classes. Access mechanisms include request-header checks, browser and interaction verification, and simulated CAPTCHA challenges. State mechanisms include authentication, cookie sequences, signed requests, session-dependent routes, and request limits. Representation mechanisms include style and font obfuscation, image-encoded values, streamed data, and alternative document structures. Because all verification pages are self-hosted and handled in the browser, these tasks evaluate tool and protocol use on the simulated sites, not success against commercial verification services.

B.4 Horizon Length

Section 3.1 states that the tasks are long by construction. To support this claim, Table 8 measures task length twice, once as a property of each task and once in the runs. The *sequential dependency depth* of a task counts the requests on its longest dependency chain, in which each request needs a link, identifier, cursor, credential, or session state that only an earlier response provides, while all other requests are issued in parallel along the default listing. The chain starts at the entry page and passes through the login where authentication gates the listing, the challenge steps of the task's gates, the page shell and browser check that precede the data at L3–L5, the listing pages, and a detail view, plus a repeated request where the first detail response is stale. The gates are passed in order, because the protection chain stops at the first failing gate and one cookie holds all session state.

Pagination decides whether this chain is short or long. With numbered or next-link pagination, every listing page can be requested once the first page reveals the page count, so the 68 such tasks are wide rather than deep, with depths of 4–10. Cursor pagination, in contrast, makes the other 17 tasks deep. Because each single-use cursor is issued only in the response to the preceding page, the five cursor tasks at L4 and all twelve L5 tasks require chains of 99–146 dependent requests. Splitting the listing by its status and letter filters shortens these chains but leaves at least 30 requests, and in 28 of the 29 cursor-task runs with state records, the agent consumed at least as many cursors as the default listing has pages. The depth also leaves out session renewals and token rotations, which recur throughout L4–L5 runs.

The run columns show how long the runs actually take. They cover the 525 archived diagnostic runs of the eight locally run configurations, including time-outs, whose counts are therefore lower bounds. LLM turns are model calls, which Claude Code, Pi, and Antigravity record directly. Because Codex does not record them, its runs are left out of that column. Tool calls come from the event streams of Codex, Pi, and Antigravity, whose runs in this diagnostic cohort cover L1–L3, which reflects the coverage of the available traces rather than the full-suite scope of Table 2. Where both counts are logged, a run issues a median of 0.99 tool calls per turn, and 61 Claude Code transcripts from remote runs give 1.00, so the L4–L5 turn counts also approximate tool calls. A median run thus takes 38 LLM turns at L1 and 175–214 at L3–L5, and 97% of L3 runs and about 80% of L4–L5 runs exceed 100 turns. Site requests exceed turns by one to two orders of magnitude, because agents write collection code that issues many requests per tool call.

Table 8: Horizon length by level. Dependency depth is a property of each task, given as its range over tasks. The other columns give medians and interquartile ranges over the runs of the eight locally run configurations.

Level	Dependency depth	LLM turns	Tool calls	Site requests	Minutes
L1	4	38 [31, 46]	39 [32, 49]	842 [722, 961]	9 [5, 15]
L2	4–6	52 [44, 72]	57 [44, 76]	2,622 [2,150, 3,376]	7 [5, 11]
L3	5–7	175 [145, 197]	161 [137, 196]	1,910 [988, 3,410]	30 [18, 30]
L4	7–10 and 111–146*	214 [184, 258]	–	5,953 [1,593, 6,841]	59 [37, 77]
L5	99–140*	178 [109, 198]	–	4,995 [4,474, 5,673]	72 [49, 102]
All	4–146	64 [41, 161]	60 [41, 112]	1,919 [882, 3,263]	15 [7, 30]

Note. * Cursor-paginated tasks. LLM turns cover 327 runs, of which the L4–L5 runs come from the three Claude Code configurations. Tool calls cover 198 runs.

B.5 Run Conditions and Reproducibility

Runs are comparable only when every agent works under the same conditions. Each task therefore starts in a fresh workspace with one server-issued session, and every request, including browser sub-resources associated through the session cookie, is attributed to that session. Within the workspace and the benchmark host, the agent may write collection code, use HTTP, or drive a browser, and it submits through the harness or directly to the site. The evaluator computes the diagnostics even when no submission exists. The single-submission instruction, however, does not prevent the submission endpoint from accepting a replacement, as Appendix I notes.

Request budget. To give efficiency a reference point, each task has a nominal request budget

$$b = \left\lfloor \frac{N}{p} \right\rfloor T + \min\Big(10, \max\big(5, \lfloor \tfrac{N}{10} \rfloor\big)\Big) + o_{\text{auth}} + o_{\text{nav}}, \tag{2}$$

which counts every page of each listing tab, a small sample of detail pages, and the overhead of authentication and navigation. Here N is the number of records, p the page size, and T the number of listing tabs, which is one at L1, two from L2, and three when a protected tab exists. The authentication overhead o_{auth} is 4 on non-public tasks plus 2 when sessions are renewable, and the navigation overhead o_{nav} is 8 plus 26 when the site offers letter navigation. Because the budget covers only a sample of the detail pages, it serves as an efficiency reference rather than a bound on complete collection. Requests beyond soft multiples of the budget receive retry responses, and requests beyond hard multiples are refused. The soft multiples are 10, 10, 10, 12, and 8 at L1–L5 and the hard multiples 40, 40, 40, 30, and 20, and the server records these refusals separately from interface and state failures.

Reproducibility and reachability. Reproducing a result requires both the task specification and the evaluation rules. All reported runs thus use task specification `vnext-2026-09` and the scoring rules of Appendix D. To verify both, rendering tests compare fixed surfaces across independent processes after normalizing security tokens, and scoring tests compare the scores of archived submissions against frozen expectations. Dynamic responses, however, may differ between agents with different interaction histories. To check reachability, a reference crawler uses HTTP requests and parsing at lower levels and a real browser at higher levels, and integration tests confirm that it retrieves every record reachable only through detail links. Complete reference solutions under the evaluation budgets have not been established.

Scope. Some properties lie outside what the benchmark measures. Credential assembly from site fragments is a controlled proxy for information discovery rather than a test of account security. Likewise, the verification mechanisms simulate common web mechanisms and do not measure success against commercial anti-bot services, and prompt-injection resistance is not evaluated.

C Conducting a Benchmark Run

Because the results depend on how a run is executed, this appendix documents the execution end to end. Appendix C.1 describes the pipeline that connects the harness, the agent, the benchmark host, and the evaluator. Appendices C.2 and C.3 then show what the agent is given and how each harness is wired in, and Appendix C.4 shows what the generated sites and their mechanisms look like to an agent. Finally, Appendix C.5 follows one real run from its first request to its submission.

C.1 Execution Pipeline

A run separates acting from recording and scoring, so that the agent cannot influence the evidence against which it is scored. Figure 8 shows these components. The harness and the agent act, the benchmark host serves and records, and the evaluator scores against data the agent never sees. One task runs in six steps.

1. *Server and session.* The harness starts a fresh benchmark server with ground-truth exposure and debugging disabled and with a per-run trace journal. Before the first task, it checks that no review page or reviewer manifest is reachable without the administrator key, and it refuses to run otherwise. For each task, it then starts one session through `POST /api/v1/benchmark/start`, whose response carries the session identifier and the task packet.
2. *Workspace and prompt.* The harness creates a fresh workspace for the task, deleting any earlier one at that path, and writes the task packet, a workspace manifest, and an empty `answer.json` into it. It then builds the prompt of Appendix C.2, whose text is identical for every harness.
3. *Agent execution.* The agent runs in its harness with a shell, Python, and a preinstalled headless Chromium, and it may issue HTTP requests, drive the browser, write and run code, or combine them. So that the server can attribute this traffic, the prompt asks the agent to send the header `X-Benchmark-Session` with every request, and browser sub-resource requests without the header are attributed through the session cookie. The harness enforces the time limit of the level, which is 15, 20, 30, 60, and 120 minutes for L1–L5, by stopping the agent process or container at the deadline.
4. *Recording.* The protection chain of the task runs before every request to the task's pages, except the authentication and verification endpoints. The server then records every attributed request in the session's trace, including data deliveries with record identifiers and field states, state events, and refusals with their class, as Section 3.3 describes. Because the trace is also journaled to disk, a server restart during the run loses nothing.
5. *Submission.* The agent keeps its answer in `answer.json`, which the prompt asks it to rewrite after each batch of records, and the harness submits the file after the agent exits, including after a time-out. Alternatively, the agent may submit directly through the API. If nothing is submitted, the harness calls an administrator-only finalization endpoint with the true completion category, which is a time-out, a budget cap, an agent failure, or an infrastructure failure, so that the stage profile is still computed.
6. *Scoring and logging.* The evaluator scores the submission against the private ground truth and the server trace, which yields Overall with its dimensions and the four-stage profile. The harness then appends one row per task to the run ledger, with the score, dimensions, stage statuses, token usage, elapsed time, and request count.

C.2 What the Agent Receives

The instructions give the agent enough to start the task but none of its answers, so that a good score reflects collection rather than guessing. The agent receives a task packet, shown in Figure 9, and a prompt built from it, whose sections Table 9 lists. The packet states the objective, the entry URL, the session header, where to look for credentials, general hints, and the output schema, including which fields sit on listing, detail, and protected views. It gives neither target values, credentials, nor the number of target records, and with budget enforcement on, it also states the reference request budget. The prompt adds the working-directory rules, the submission contract, and general guidance on common web mechanisms, such as returning verification tokens, keeping URL-embedded session

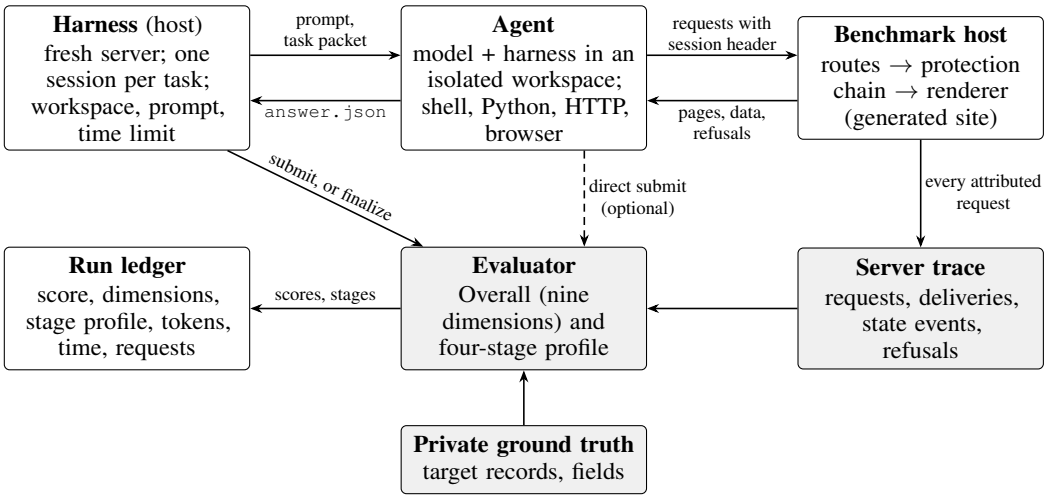


Figure 8: Execution pipeline of one run. The harness starts the server and one session per task, prepares the workspace and prompt, runs the agent under the level's time limit, and submits the answer file. Every request the agent sends with the session header passes the task's protection chain and is recorded in the server trace. Shaded components are private, so the agent never sees the trace or the ground truth and sees scores only in the response to a submission, as Appendix I notes.

identifiers, and following opaque detail links. Because this guidance is identical for all tasks and all harnesses, differences between systems do not stem from the instructions.


```
{"task_id": "L3_community_finance_004", "level": 3,
 "render_mode": "json_embed", "auth_mode": "optional",
 "objective": "Export the full payment operations workflow from Cedar Capital Desk
              as normalized structured data, combining listing, transaction-detail,
              and gated operations fields for every real row.",
 "start_urls": ["/level3/community/finance/4"],
 "tracking": {"header_name": "X-Benchmark-Session", "header_value": "sess_aea11bd4..."},
 "credential_discovery": [
   {"type": "login_page", "url": "/level3/community/finance/4/auth/login"},
   {"type": "help_page",  "url": "/level3/community/finance/4/auth/help",
    "note": "The login page links to a help page that discloses the demo credentials."}],
 "hints": ["Data is not in the raw HTML; look inside <script> tags or XHR responses.",
           "A simple verification gate must be passed before accessing data.",
           "Record IDs are opaque tokens; discover them from listing pages ...",
           "Not every in-scope record appears in the default listing or via sort/filter.
            Some are reachable only by following a related-record reference ..."],
 "request_budget": 394,
 "response_template": {
   "submit_url": "/api/v1/benchmark/submit", "record_id_field": "id",
   "item_schema": {"id": "string", "transaction_id": "string", "amount": "number",
                   "currency": "string", "status": "string", "date": "string",
                   "account": "string", "region": "string", "method": "string",
                   "sender": "string", "risk_score": "number",
                   "internal_comment": "string"},
   "notes": ["Listing fields (...): id, transaction_id, amount, currency, status.",
             "Detail fields (...): date, account, region, method, sender.",
             "Protected fields (require authenticated access): risk_score,
              internal_comment.", ...]}}
```


Figure 9: Task packet of the run in Appendix C.5, abridged. Ellipses mark omitted text, and the session identifier is shortened. The packet names the field sources but not the number of target records, and it points to the credential help page without disclosing the credentials.

C.3 Harnesses and Isolation

The harnesses differ in how they launch an agent and in what they log, so we document each adapter and take all evidence from the harness side. Table 10 lists how each harness is launched and what it records. The locally run configurations executed in containers, with the workspace mounted at `/workspace`, the benchmark host reachable over the network, and writes outside the

Table 9: Sections of the agent prompt, in order. The harness builds the prompt from the task packet, and its text is identical for every harness.

Section	Content
Working directory	Work only in the workspace and keep the full answer in `answer.json`.
Task packet	Task identifier, level, objective, authentication and rendering mode.
Request budget	Reference budget. With budget enforcement, data routes answer with retry responses past a soft multiple and refuse past a hard multiple.
Start URLs, credential discovery, deliverables, hints	Copied from the packet.
Submission contract	Submit URL and session and task identifiers, with a warning not to start another session, because requests under another session are not credited.
Evaluation criteria	Record recall, which sets every dimension to 0 below 10%, field accuracy, precision, completeness of detail and protected fields, workflow coverage, efficiency, and authentication handling.
Data scale and pagination	Sites hold hundreds to over a thousand records, so paginate through all pages and use the site's own totals as the target.
Execution guidance	Crawl only the given site, use a browser-like user agent, return verification tokens, keep URL session identifiers, look for fragment endpoints and required headers in inline scripts, follow opaque detail links, and normalize values to the schema.
Hard requirements	Send the session header on every request, including from generated code, handle protections, never self-report request counts, and submit partial results rather than nothing.
Submission	The answer file, which the harness submits by default, or a single direct submission, followed by a pre-submission checklist.

workspace blocked. To keep the evidence independent of the agent's own claims, the agent's final message is logged but never parsed or scored, token usage comes from the harness's event stream rather than from anything the agent writes, and a usage field in `answer.json` is discarded. The remotely run systems used Claude Agent SDK 0.2.144 (Anthropic, n.d.a) for Opus 5 and Codex CLI 0.149.0–0.149.1 for Sol, 0.150.1 for Terra and Luna, and 0.151.0–0.153.1 for Sol in the multi-agent mode.

Table 10: How each agent harness is launched and what it records.

Harness	Launch	Recorded evidence
Claude Code	Claude Code through its agent SDK, in a container, with tool permissions granted inside the workspace and writes outside it blocked.	Result envelope with token usage. After each model step, the answer file can be scored without being stored, which gives a per-step score curve.
Codex	Codex CLI with a workspace-write sandbox, approvals off, and network access on.	Token usage once per task and no step timestamps.
Pi	Pi CLI in JSON mode, without extensions or skills.	Event stream with per-message usage and timestamps and every tool call.
Antigravity	Antigravity CLI with stream-JSON output and permission prompts disabled.	Event stream with per-step durations and usage.

Note. Codex, Pi, and Antigravity run through a thin wrapper that passes the prompt on standard input and reads the harness's event stream. Runs stopped at the time limit can lack aggregate usage in the analysis records, so the token analyses impute missing totals, as Appendix H describes.

C.4 What the Agent Sees

Screenshots show what the agent works with more directly than a description. Figure 10 shows five generated sites, one per content domain and site family, at the three views that every task shares, an entry page, a paginated listing, and a record's detail page. The sites differ in brand, palette, layout skeleton, field labels, and page chrome such as promotions, cookie banners, chat widgets, and

sponsored blocks, while the task structure stays the same. Figure 11 then shows the mechanisms that an agent meets along the collection path at higher levels. These include an authentication wall and the help page that discloses the credentials at L2, a timing check that waits for human-like interaction at L3, and a browser verification page and an image challenge at L4. At L5, the rendered listing card looks ordinary, but its markup uses per-client class names, opaque detail links, characters reordered by CSS, and a decoy block.

C.5 Anatomy of One Run

A single run makes the stages of the evaluation concrete. Figure 12 follows an archived GLM-5.3-Flash run in Pi on `L3_community_finance_004`, whose packet Figure 9 shows, from its first request to its submission. The task has 1,240 target records and optional authentication. We reconstruct the run from the server's trace journal and the Pi event stream, aligned on the server clock. The run falls into three phases.

- *Exploration, 0–12 minutes.* The first data response arrives after 18 seconds, but the agent then spends the first 12 minutes fetching pages one at a time and analyzing saved copies. This phase takes 111 of its 178 tool calls and 168 requests and yields 5.7% of the target records. A provider error at 2.2 minutes is retried automatically, and the XHR-header check refuses eleven requests to the listing's JSON endpoint between 4.4 and 5.4 minutes, after which no refusal occurs.
- *Collection, 12–20 minutes.* The first crawler run delivers records in a single burst of 294 requests in 48 seconds and reaches 79% of the target records, and a second crawler run at 19.6 minutes delivers the rest, reaching 100% by 19.9 minutes. The agent logs in five times, including at the start of each crawler run, instead of reusing one authenticated session.
- *Assembly, 21–23 minutes.* The agent writes the answer file from 21.2 minutes on, and the run ends at 22.6 minutes, well inside the 30-minute limit.

The delivery records then show where this run lost information. All target records were delivered, and so were the listing fields and 99.5% of the protected fields, which the agent read from an operations tab after logging in. Only 0.3% of the record-level detail fields were served, however. Of the delivered values, 85.5% were returned correctly, and most errors were format-only, at 14.3%, so the run scores 78.6. The run thus shows the pattern of Section 4.3 in miniature, with complete record coverage but an unfetched detail view, and with value losses in formatting rather than in content.

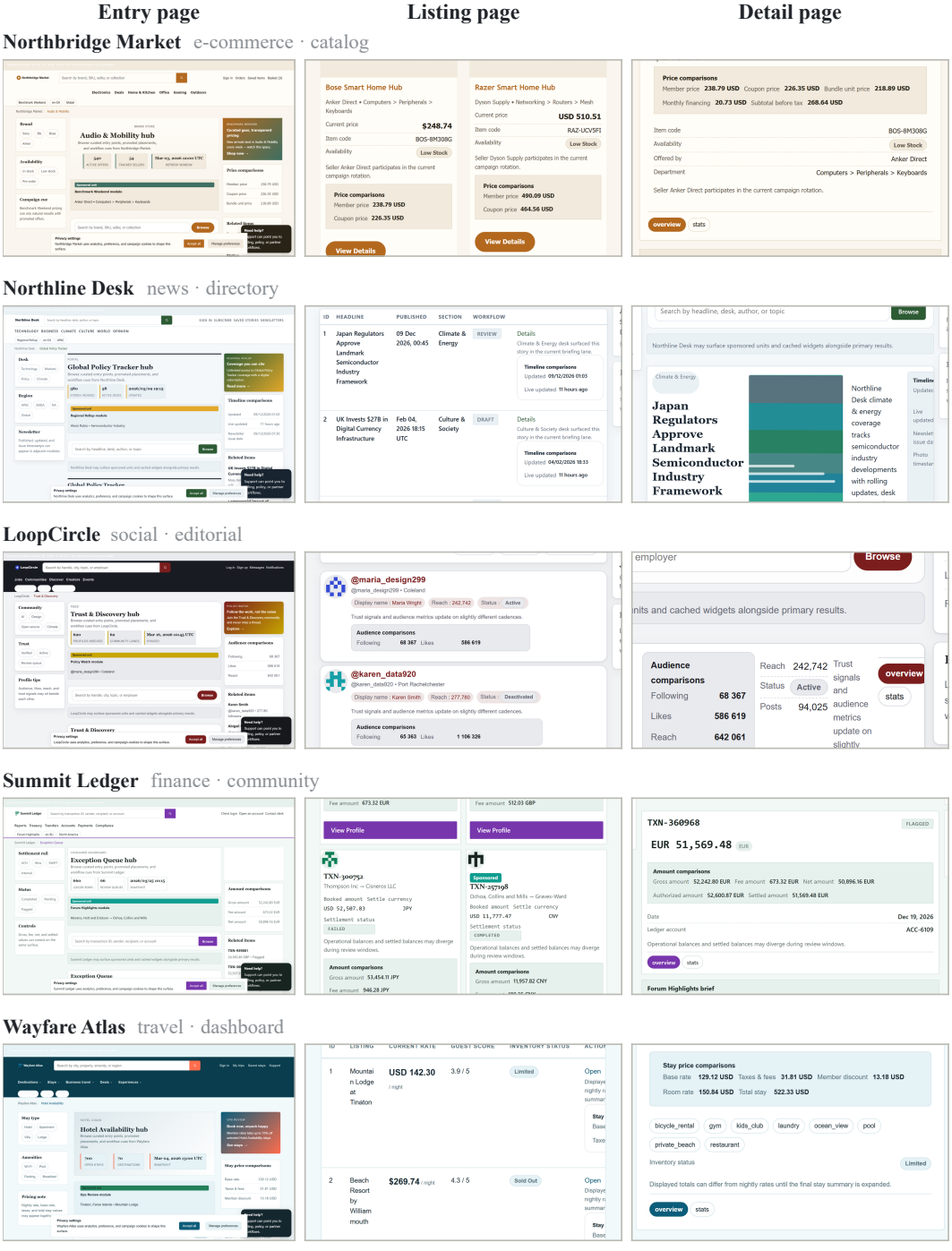
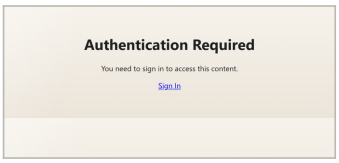
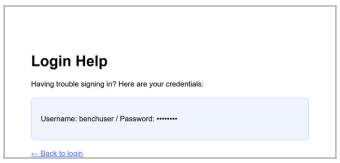


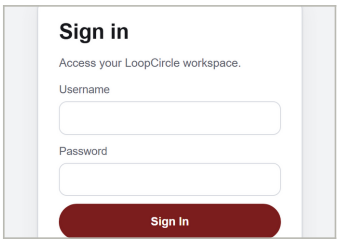
Figure 10: Five generated L1 sites from tasks 1–5, one per row, covering every content domain and site family, each at its entry page, a listing page, and a detail page. The sites differ in brand, palette, layout, field labels, and page chrome, while the task structure stays the same. Entry pages are shown as first loaded, including the cookie banner and chat widget, whereas on listing and detail pages these overlays are hidden and the view is scrolled to the data. Screenshots are embedded at high resolution and can be enlarged.



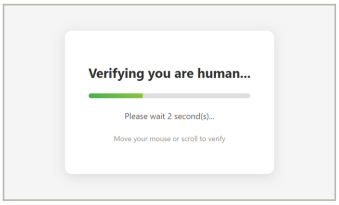
(a) L2: listing requested before login



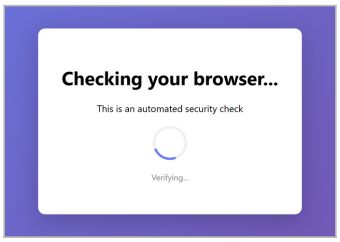
(b) L2: credential help page (password masked here)



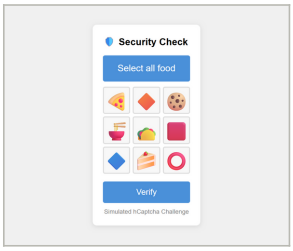
(c) L2: login form



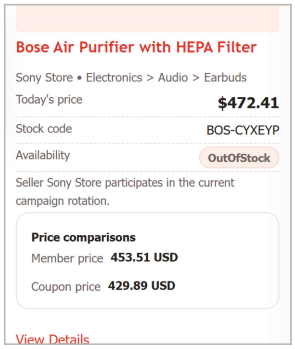
(d) L3: timing check



(e) L4: browser verification



(f) L4: image challenge



(g) L5: a listing card as rendered

```
<article class="qyepqi" data-record-id="2">
  <a href=".../detail/r/AQAAAAJPlSvhNVqXUw">
  <div class="jiwvel"><dt class="mj3t2r">
      Today's price</dt>
    <dd><span class="mnyl26" style="display:inline-flex">
      <span style="order:6">1</span><span style="order:5">4</span>
      <span style="order:0">$</span><span style="order:3">2</span>
      <span style="order:1">4</span><span style="order:2">7</span>
      <span style="order:4">.</span>
    </span></dd></div>
  ...
  <div class="inline-decoy-block vg0a3j">
    <strong>Price comparisons</strong> ...
</article>
```

(h) L5: HTML of the same card (abridged); the price is stored as reordered characters that the browser displays as $472.41

Figure 11: Mechanisms as an agent meets them, captured through a started session with a desktop-browser user agent. (a)–(c) A protected L2 listing before login, the credential help page, and the login form. The password is masked here but not on the site. (d) An L3 timing check. (e) L4 browser verification, which clears itself in a real browser. (f) An L4 image challenge. (g)–(h) An L5 listing card as rendered and its abridged HTML. Class names are reassigned per client, detail links carry opaque tokens, the price is stored as characters reordered by CSS, and the block of price comparisons is a decoy.

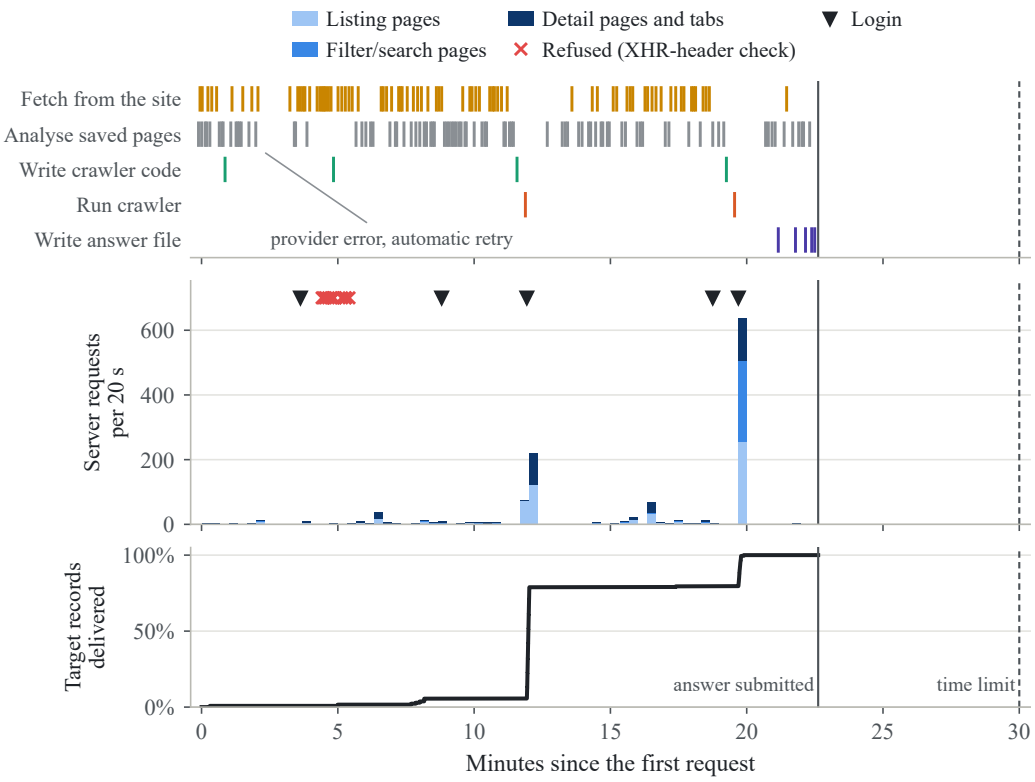


Figure 12: One run of GLM-5.3-Flash in Pi on `L3_community_finance_004`, which scores 78.6. The top panel shows the agent's tool calls by activity, taken from its event stream, and the middle panel shows the requests that the server attributed to the run's session per 20 seconds, by page type, with refusals and logins. The bottom panel shows the share of the 1,240 target records delivered so far. The solid line marks the end of the run and the submission of the answer file, and the dashed line marks the time limit.

D Evaluation Metrics

Section 3.3 introduces the Overall score and the stage diagnostics only briefly. This appendix defines them in full, so that every reported number can be recomputed. After fixing notation in Appendix D.1, we motivate and define each dimension of Overall in Appendix D.2 and measure how much each of its rules changes the scores in Appendix D.3. We then define the stage diagnostics in Appendix D.4 and the accounting that links delivery to output in Appendix D.5.

D.1 Notation and Value Matching

Records and discovery strata. Because records differ in how hard they are to find, we split the target set by discovery route. For a task t, let G be the set of target record identifiers, which falls into three strata. The *default-listing* records G_{list} are reached by paging through the default listing. The *revealed* records G_{rev} are absent from the default listing but returned by search, filter, sort, or letter queries over the full index. The *detail-only* records G_{det} appear in no listing or query result and can be reached only through a link on the detail page of another record. For a run τ, let E be the set of distinct submitted identifiers, F_τ the identifiers whose detail page the server recorded a request for, and Z the identifiers of planted decoy records.

Fields and delivery. The evaluation works at the level of record–field pairs. Let Φ_t be the graded fields of the task, which are its listing and detail fields and, on non-public tasks, its protected fields, and let $U_t = \{(r, f) : r \in G,\ f \in \Phi_t\}$ be the required record–field pairs with gold values $v^*_{r,f}$. As in Section 3.3, D_τ is the set of identifiers in the server's delivery summary, $A_\tau = G \cap D_\tau$ the delivered target records, and $\sigma_\tau(r, f)$ the best recorded delivery state of field f of record r. When a record is served repeatedly, the deliveries are merged by identifier and each field keeps its best state, and only intact fields count as delivered.

Value matching. Submissions rarely use exactly the format of the ground truth, so the comparison tolerates harmless differences. Before submissions, ground truth, and delivery records are compared, identifiers are normalized by trimming strings, casting integer-valued numbers to integers, and decoding opaque display tokens. A submitted value v then matches the gold value v^*, written $\text{match}(v, v^*)$, if the two are equal after whitespace trimming or after removing a leading currency symbol and thousands separators, if they are numerically closer than 0.051, or if they are equivalent booleans, such as true, 1, or yes against false, 0, or no. When v^* is an ISO timestamp, the two also match if they give the same date.

D.2 The Overall Score

Overall summarizes how complete and correct the returned dataset is and whether the agent obtained it within the rules of the site. To cover both aspects, it combines the six dimensions of Table 11. Three *output* dimensions compare the submission with the ground truth, and three *process* dimensions read the server trace, which the agent cannot edit. Below, we motivate each dimension by the shortcut or failure it is designed to catch and then define it. Every dimension lies in $[0, 1]$, and we report scores on a 0–100 scale.

Stratified record recall. A complete dataset includes the records that are hard to find, yet flat recall would reward a crawl that ignores them. Under flat recall, a crawl that pages through the default listing and stops would already earn 55–90% at L2–L5, because only 10–45% of the records lie beyond that listing, as Table 6 shows. We therefore compute recall per discovery stratum,

$$r_{\text{list}} = \frac{|E \cap G_{\text{list}}|}{|G_{\text{list}}|}, \qquad r_{\text{rev}} = \frac{|E \cap G_{\text{rev}}|}{|G_{\text{rev}}|}, \qquad r_{\text{det}} = \frac{|E \cap G_{\text{det}} \cap F_\tau|}{|G_{\text{det}}|}, \tag{3}$$

and combine the non-empty strata with the level-specific weights w_s of Table 12,

$$\text{Recall} = \frac{\sum_{s \in \mathcal{S}_t} w_s\, r_s}{\sum_{s \in \mathcal{S}_t} w_s}, \qquad \mathcal{S}_t = \{s \in \{\text{list}, \text{rev}, \text{det}\} : G_s \neq \varnothing\}. \tag{4}$$

These weights give the records beyond the default listing more weight than their share of the target set. A listing-only crawl is thus capped at w_{list}, which is 0.35 at L5, a crawl that also queries the

Table 11: Dimensions of the Overall score. The effective weights renormalize the nominal weights over the dimensions that apply to the 85 mechanism tasks, which request no files. Output dimensions thus carry 69% of the weight on L1 tasks and 63% on L2–L5 tasks.

			Effective weight	
Dimension	Evidence	Weight	L1	L2–L5
Output dimensions				
Stratified record recall	Submission, truth, detail receipts	0.18	0.257	0.237
Field quality	Submission, truth	0.23	0.329	0.303
Precision with decoy penalty	Submission, truth	0.07	0.100	0.092
Process dimensions				
Workflow coverage	Server trace	0.14	0.200	0.184
Authentication decisions	Server trace	0.06	N/A	0.079
Request efficiency	Server trace, quality	0.08	0.114	0.105
Extension dimensions, research-workflow tasks only				
File integrity	Claims, download receipts	0.14	–	–
Provenance	Claims, source receipts	0.06	–	–
Failure reporting	Reports, truth	0.04	–	–
Total		1.00	1.000	1.000

index at $w_{\text{list}} + w_{\text{rev}}$, which is 0.80 at L5, and only a crawl that also follows detail links reaches 1. Moreover, a detail-only record counts only if the server recorded a request for its detail page. This stratum tests whether the agent followed the link, which a submitted identifier alone does not show.

Table 12: Stratum weights of the record recall by level. Dashes mark strata that are empty at that level. From L2 on, the records beyond the default listing receive more weight than their share of the target set.

		L1	L2	L3	L4	L5
w_{list}	default listing	1.00	0.80	0.65	0.50	0.35
w_{rev}	revealed by search, filter, or sort	–	0.20	0.25	0.40	0.45
w_{det}	detail-only	–	–	0.10	0.10	0.20
Weight beyond the default listing		0.00	0.20	0.35	0.50	0.65
Share of records beyond the default listing		0%	10%	22%	35%	45%

Field quality. The deliverable is a table of values, and most of these values lie outside the listing, since four or five fields per record sit only on detail pages and two more behind authentication. Because an identifier without its fields is of little use, field quality, the dimension with the largest weight, counts every required record–field pair, including those of records that were never submitted,

$$\text{FieldQuality} = \frac{1}{|U_t|} \sum_{(r,f)\in U_t} \mathbf{1}\big[\, f \in \rho_r \;\wedge\; \text{match}(\rho_r[f], v^*_{r,f}) \,\big], \tag{5}$$

where ρ_r is the submitted row for record r, taking the last one if several rows share its identifier. A missing field and a wrong value both score zero.

Precision with a decoy penalty. A collector that submits everything it finds would lose nothing on recall, so precision penalizes rows outside the target set. The L5 sites also mix planted decoys among the real records to test whether retrieved content is validated, and each submitted decoy costs an additional 0.1, up to 0.4,

$$\text{Precision} = \max\Big(0,\; \frac{|E \cap G|}{|E|} - \min\big(0.4,\; 0.1\, n_Z\big)\Big), \tag{6}$$

where n_Z is the number of submitted rows whose identifier lies in Z.

Workflow coverage. The output dimensions show what was returned but not whether the collection path reached every view that holds required information. For example, a run that never opened the authenticated view and a run that opened it but dropped its fields look the same in the output. Workflow coverage therefore credits each kind of view that a complete collection visits, read from the server trace. For a mechanism task, the expected steps Ω_t are the entry page, the paginated listing, and the detail view. From L2, the second listing tab joins them, on non-public tasks the login and the protected listing tab, and on sites with letter navigation that navigation. With Ω_τ the step types that the server recorded for the run,

$$\text{Workflow} = \frac{|\Omega_\tau \cap \Omega_t|}{|\Omega_t|}. \tag{7}$$

Authentication decisions. On non-public tasks, protected fields are served only after login, and on expiring tasks access lapses unless the session is renewed. Field quality penalizes missing protected values but cannot tell whether the agent never authenticated or authenticated and then failed to collect. To separate the two, this dimension credits the access decision itself. It reads the login, renewal, and expiry events in the trace, and without a login it gives partial credit that grows with how much of the task remains collectable,

$$\text{Auth} = \begin{cases} 1 & \text{optional, deferred, or required policy, login recorded,} \\ 0.7 & \text{optional policy, no login, only protected fields withheld,} \\ 0.3 & \text{deferred or required policy, no login, most data routes blocked,} \\ 1 \,/\, 0.8 \,/\, 0.7 \,/\, 0 & \text{expiring policy: renewal / login and expiry / login only / no login.} \end{cases} \tag{8}$$

The dimension does not apply to public tasks, which comprise all of L1.

Request efficiency. The same dataset can be collected with few or many requests, and a collector that re-fetches pages or loops on refusals consumes the site's capacity. Efficiency thus compares the number n of requests that the server records for the session with the nominal budget b of Equation (2). Above the budget, credit decays logarithmically, so a run that uses twice the budget keeps 59% of the credit and a run that uses ten times the budget keeps 30%. Within the budget, credit is capped at the run's collection quality $q = \frac{1}{2}(\text{Recall} + \text{FieldQuality})$, since staying within the budget is easy for a run that collects little,

$$\text{Efficiency} = \begin{cases} \min(1,\, q) & 0 < n \le b, \\ \big(1 + \ln(n/b)\big)^{-1} & n > b, \end{cases} \tag{9}$$

and Efficiency is 0 when no request is recorded.

Aggregation and the recall gate. Two further rules keep Overall from rewarding runs that did little. First, a dimension that a task does not exercise, such as authentication on a public task, carries no information about the run. Scoring it as full credit would add a constant floor to every such task, so we drop inapplicable dimensions and renormalize the remaining weights. Second, a run that returns almost nothing but visited the right views would still score on workflow coverage and authentication, and a recall gate prevents such a near-empty submission from collecting process credit. With $\mathcal{A}_t$ the applicable dimensions of task t, s_d the score of dimension d, and w_d its weight in Table 11,

$$\text{Overall}_\tau = \begin{cases} 0 & \text{if } |E \cap G|/|G| < 0.10, \\ \dfrac{\sum_{d \in \mathcal{A}_t} w_d \, \min\big(1, \max(0, s_d)\big)}{\sum_{d \in \mathcal{A}_t} w_d} & \text{otherwise.} \end{cases} \tag{10}$$

The gate uses flat recall, so it depends only on how many target records were submitted. A task without a submission scores 0, and the Overall of a system is the unweighted mean over its tasks.

Extension dimensions. The research-workflow tasks of Appendix B.2 ask for outputs beyond records, so they add three dimensions that apply only when a task requests the corresponding output. File integrity credits a claimed file only if its record, type, and path match an expected file and the server recorded its download. Provenance credits a source claim only if its label matches and the server served that record from the claimed source. Failure reporting is the recall of the expected failure reports, matched by record and type. None of these dimensions applies to the 85 mechanism tasks.

D.3 Sensitivity of Overall to Its Scoring Rules

The rules above add complexity to the score, so we test whether each of them changes the scores in practice. To this end, we re-score the same 378 archived submissions with one rule relaxed to the weaker form that earlier scoring versions or outcome-only benchmarks use, as Table 13 reports. Because the submissions are fixed, the changes reflect the scoring rules alone. No relaxed protocol lowers the mean score. The largest inflation comes from scoring inapplicable dimensions as full credit, which raises 98% of the tasks by 5.9 points on average, followed by dropping the trace-based process dimensions, which adds 4.6 points. The receipt requirement and the recall gate change the mean little but matter where they apply. For example, crediting detail-only identifiers without a receipt raises detail-only recall from 60.4 to 63.7 at L3, from 32.7 to 36.5 at L4, and from 20.9 to 30.0 at L5. Likewise, a shortcut solver that submits template-shaped rows without collecting data scores 0 on the three research-workflow tasks under the canonical rules but up to 14 under the legacy ones, as Appendix E shows. Rankings, in contrast, are stable, since under any protocol at most one of the 28 pairs of configurations changes order. The rules therefore make the score harder to inflate without reordering systems.

Table 13: Re-scoring the same 378 archived submissions with one scoring rule relaxed. Δ is the mean per-task change from the canonical protocol in points on the 0–100 scale, with a 10,000-resample bootstrap 95% interval. *Inflated* is the share of tasks whose score rises, and *rank flips* counts the pairs of configurations, out of 28, whose order reverses.

Relaxed rule	Δ [95% CI]	Inflated	Rank flips
One rule relaxed			
Detail-only records credited without a detail receipt	+0.0 [+0.0, +0.1]	1%	0
Flat instead of stratified recall	+0.7 [+0.6, +0.8]	31%	0
Inapplicable dimensions scored as full credit	+5.9 [+5.6, +6.1]	98%	1
No quality cap on in-budget efficiency	+0.0 [+0.0, +0.0]	0%	0
Process dimensions dropped	+4.6 [+4.0, +5.1]	85%	1
No recall gate	+0.8 [+0.3, +1.3]	2%	0
Combined			
Legacy, all above relaxed but process dimensions kept	+7.6 [+6.9, +8.3]	100%	1
Outcome-only, receipts, strata, and process dimensions dropped	+5.7 [+5.1, +6.2]	89%	1

Note. The submissions come from DeepSeek V4.1 Flash and GLM-5.3-Flash in Claude Code, Codex, and Pi and from DeepSeek V4 Pro and GLM-5.3 in Claude Code.

D.4 Stage Diagnostics

Whereas Overall scores the final outcome, the stage diagnostics locate where a run lost progress. Each of them is computed per task and then aggregated across tasks, as Table 14 summarizes. With $B_\tau = \{(r, f) \in U_t : \sigma_\tau(r, f) \text{ intact}\}$ the delivered pairs and $\phi_\tau(r) = \{f : (r, f) \in B_\tau\}$ the delivered fields of record r, the two continuous diagnostics are

$$\textsc{Acquire}_\tau = \text{mean}\left\{\frac{|A_\tau|}{|G|},\ \frac{|B_\tau|}{|U_t|},\ \text{file coverage if files are requested}\right\}, \tag{11}$$

$$\textsc{Assemble}_\tau = \frac{1}{|A_\tau|}\sum_{r\in A_\tau}\ \max_{\rho\in\text{rows}(r)}\ \frac{|\{f \in \phi_\tau(r) : \text{match}(\rho[f], v^*_{r,f})\}|}{|\phi_\tau(r)|}, \tag{12}$$

where $\text{rows}(r)$ are the submitted rows with identifier r. An omitted record contributes 0, a submitted record without delivered fields contributes 1, and Assemble is undefined when $A_\tau = \varnothing$. Two stricter measures complement Assemble. Strict row fidelity is the share of delivered target records submitted exactly once with every delivered field correct, and the duplicate rate is the share of submitted identifiers with more than one row. For traces recorded before field states were logged, the full graded field set replaces $\phi_\tau(r)$.

Statuses. Besides the continuous scores, each stage receives a status of pass, partial, fail, N/A, or Unknown, and the Access and Control columns of Table 2 aggregate these statuses into pass rates. Access passes when a data-bearing surface is reached, or when every refusal is state- or

Table 14: Stage diagnostics and their aggregation across tasks. Each score is computed per task before aggregation, and tasks without the required evidence are excluded from the denominator of that diagnostic.

Diagnostic	Within-task measurement	Across-task aggregation
ACCESS	Status of whether a data interface was reached.	Pass rate over tasks with a known, applicable status.
CONTROL	Protocol compliance, whether the state requirements were met with matching recovery.	Pass rate over applicable tasks with a known status that passed Access.
ACQUIRE	Mean of record, field, and file coverage, Eq. (11).	Mean over tasks with a defined score.
ASSEMBLE	Graded fidelity over delivered records, Eq. (12).	Mean over tasks with a defined score, where delivery without a submission counts as 0.

budget-related, since such a refusal shows that the interface was reached. It fails when no surface is reached and the refusals are tool-related or absent. CONTROL passes when every state requirement of the task and every ordered transition is covered, no terminal state before the last data access remains unrecovered, and no control-class refusal is unresolved. It is N/A when the task has no state requirement and no state event or state-related refusal occurs. ACQUIRE passes when every coverage component is complete and is partial when the smallest component is at least 0.5. ASSEMBLE passes when strict fidelity, precision, and every applicable claim recall are complete and no identifier is duplicated.

Matched recovery. A later success does not always repair an earlier failure, so a recovery counts only if it follows the failure and matches the failed condition. An authentication refusal requires a successful login followed by access to the refused kind of surface, and a rejected cursor requires a later successful cursor consumption. Likewise, session expiry requires renewal or revalidation followed by data access, and a rate-limit refusal requires a later successful request to the same surface. Reaching an unrelated public page after an authentication refusal therefore does not restore access to the protected view.

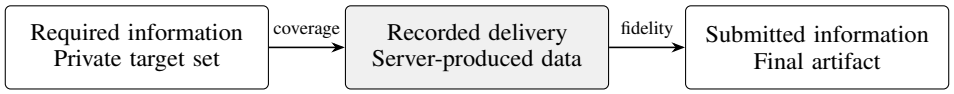
Missing evidence. Some runs lack the evidence that a diagnostic needs, and scoring them as failures would blame the agent for gaps in the logs. Evidence that was not recorded therefore yields Unknown rather than a failure. An infrastructure failure or an incomplete trace makes every stage Unknown. ACQUIRE is Unknown when no exact delivery summary exists and the capped receipt lists, which hold at most 400 entries, cannot establish full coverage. ASSEMBLE fails when data were delivered but nothing was submitted and is N/A when neither happened. Delivered identifiers outside G are reported as non-target deliveries and excluded from coverage.

D.5 DELIVERY–OUTPUT ACCOUNTING

The stage diagnostics rest on the observation that the same output can arise from very different runs. Figure 13 illustrates this with two runs that reach the same output coverage, one through partial delivery with a faithful output and the other through full delivery followed by losses in the submission. Only the delivery records tell the two apart, which is why the evaluation separates coverage from fidelity.

The same comparison applies to individual record–field pairs. Let K_τ be the pairs returned correctly under the value-matching rule. With complete delivery evidence, Table 15 then distinguishes the four combinations of delivery and output correctness, and Figure 4 reports their empirical distribution, refined by cause as Appendix G.1 describes.

One of these combinations needs care. A correct output without delivery support could come from inference, predictable content, another information path, or a recording limitation, so we record such values as correct without delivery support rather than as acquired. Overall requires receipts for the detail-only stratum, files, and source claims, but not for every ordinary field value, so output correctness and delivery-supported correctness remain distinct quantities. For $|B_\tau| > 0$, delivery-



Illustrative catalog run 1,060 required records	Served	Correct output	ACQUIRE	ASSEMBLE
Run A with partial delivery and faithful output	530	530	50%	100%
Run B with full delivery and partial output	1,060	530	100%	50%

Figure 13: Two hypothetical catalog runs with the same output coverage but different acquisition and assembly scores, assuming intact fields and correct, unique rows. Arrows denote comparisons between data sources.

Table 15: Field-level combinations of recorded delivery and output correctness, assuming sufficient delivery evidence. Without such evidence, delivery is Unknown rather than recorded as absent.

Required field pair	Correct in output	Missing or incorrect in output
Intact delivery recorded	Preserved after delivery, $B_\tau \cap K_\tau$.	Lost after delivery, $B_\tau \setminus K_\tau$.
No intact delivery recorded	Correct without delivery support, $K_\tau \setminus B_\tau$.	Neither served nor returned, $U_t \setminus (B_\tau \cup K_\tau)$.

supported correct coverage factorizes as

$$\frac{|B_\tau \cap K_\tau|}{|U_t|} = \frac{|B_\tau|}{|U_t|} \cdot \frac{|B_\tau \cap K_\tau|}{|B_\tau|}. \tag{13}$$

Both factors use record–field pairs as their unit. ACQUIRE, in contrast, averages record, field, and file coverage, and ASSEMBLE averages within-record fidelity, so the product ACQUIRE $\times$ ASSEMBLE is not an output score.

E EVALUATION INTEGRITY TESTS

The diagnostics are useful only if the evaluator applies its rules as intended. We therefore test the evaluator of Section 3.3 on constructed inputs whose correct outcome is known in advance. The tests cover answer leakage, retrieval evidence, scoring boundaries, and session isolation.

Answer leakage and reachability. A task would be trivial if its answers leaked into what the agent sees. We therefore check task instructions and templates for answer values, credentials, and target counts, and we render each site with an empty result set to detect target values that leak through site decoration. Conversely, a task would be unfair if its hidden records could not be found, so we check that hidden records form the intended partition and remain reachable through discoverable routes. For query-conditioned tasks, we also check that the published search or filter interface selects the same target set as the private evaluation data.

Credentials and protected information. Similarly, access control tests the agent only if it behaves as specified. We check that credentials obtained from discoverable pages unlock the intended content and that protected fields stay absent from unauthenticated and unrelated views. We also check that renewal tokens are session-specific and rotated and that public search results do not expose fields reserved for detail views. Together, these checks preserve the intended dependence between access state and information retrieval.

Retrieval claims require evidence. Because the evaluator credits some items only with a receipt, we test that claims without evidence earn nothing. A submitted detail-only identifier receives no recall credit without a detail receipt. Likewise, file and provenance claims receive no credit when the delivery is absent, incorrect, or mismatched, whereas an exact claim with matching evidence receives credit, so a plausible file URL alone cannot establish a download. We also check that live and archived scoring agree and that malformed or unsupported submissions are rejected.

Representation and scoring boundaries. The generated content and the scoring thresholds also need to behave consistently. We check that target identifiers and layouts are reproducible across interpreter settings and that adaptive interference neither corrupts detail pages, generates a decoy equal to a target value, nor escalates under static conditions. Scoring tests additionally check the decoy penalty, the stratum weights, the numeric matching tolerance, and the recall gate, for which nine of 100 target identifiers yield an Overall of zero whereas ten pass the gate.

Distinguishing acquisition from assembly. Since the stage diagnostics are meant to separate failure modes, we construct runs whose correct stage labels are known. A run with no site interaction should fail Access. A full crawl followed by a half submission should pass Acquisition but fail Assembly, and a listing-only crawl with a faithful submission should show an acquisition gap. Deferred authentication without login should produce a Control failure that a successful login resolves, while access to an unrelated public page should leave an authentication refusal unresolved. Finally, a finalized run without a submission should keep its observable diagnostics, whereas an infrastructure failure with an incomplete trace should yield Unknown.

Isolation and shortcut checks. Lastly, the diagnostics should neither reach the agent nor be open to gaming. We check that diagnostic metrics neither affect Overall nor appear in agent-facing results. Because only server-issued sessions identify tracked runs, we also check that a fabricated identity cannot reset access or budget state. In addition, we test a shortcut solver that constructs template-shaped rows and unsupported workflow claims. Its score is required to stay below 40 on the 0–100 scale, and it obtains 0 under the canonical rules and at most 14 under the legacy rules of Table 13.

F Additional Leaderboard Results

This appendix supplements Section 4.2 with details that the main text leaves out for space. Appendix F.1 first collects notes on the main-text tables and figures, Appendix F.2 then gives level-wise scores for more systems, and Appendix F.3 finally compares single- and multi-agent execution.

F.1 Notes on the Main-Text Tables and Figures

Several conventions affect how the numbers in the main text read, so we collect them here. Table 2 reports ten systems on all 85 tasks, including Gemini 3.8 Flash and DeepSeek V4 Pro, whereas GLM-5.3-Flash appears only in the 58-task harness comparison of Table 3. In Table 2, Access and Control are pass rates over applicable tasks, Acquire and Assemble are mean continuous scores, and Overall counts unsubmitted tasks as zero. The Qwen3.8-Flash stage profiles cover 62 of the 85 tasks, and missing diagnostics are not scored as zero. Opus 5 is reported from aggregates. In Table 3, brackets after Overall give 95% archetype-bootstrap intervals of the mean, and Δ is the paired difference from Claude Code. All of its rows use the standard 15/20/30-minute limits, and Access is 100 in every row. Control uses only applicable tasks, which number 36 for DeepSeek V4.1 Flash.

The figures follow similar conventions. In Figure 3, panel headers give the harness and the change from L1 to L5, and unsubmitted tasks count as zero. In Figure 4, shares are averaged over runs, grey marks unsubmitted runs whose field-level delivery is unknown, and Figure 15 adds the L4–L5 configurations. Figure 5 shows the eight configurations on L1–L3 and all configurations pooled on L4–L5, where dashes mark fewer than three applicable runs and the Control rows use journal-corrected verdicts. Finally, in Figure 6, a run counts with its final score only if it finished within the budget, and the left panel compares DeepSeek V4.1 Flash in three harnesses, while the right panel compares five configurations.

F.2 Results by Difficulty Level

To complement Figure 3, Figure 14 shows four more systems from Table 2 whose level-wise results are available. Updated level-wise results for Gemini 3.8 Flash and DeepSeek V4 Pro are not included, and Table 2 reports their full-suite results.

F.3 Single- versus Multi-Agent Execution

Splitting a long collection task across several agents is a natural idea, so we test whether it helps. Table 16 compares GPT-5.6 Sol run as a single agent at max effort with its multi-agent Ultra mode at ultra effort on the same 27 L4–L5 tasks. The multi-agent mode lowers Overall from 53.8 to 41.4, with losses in both Acquire and Assemble. Because the Ultra mode also changes the reasoning effort, the CLI version, and the run date, however, this contrast does not isolate the effect of multi-agent execution.

Table 16: GPT-5.6 Sol as a single agent and in the multi-agent Ultra mode on the same 27 L4–L5 tasks. Δ is the paired Overall difference from the single agent.

Execution mode	Subm.	Control ↑	Acquire ↑	Assemble ↑	Overall ↑	Δ [95% CI]
Single agent, max effort	27	11.1	54.1	65.3	53.8	–
Ultra multi-agent, ultra effort	26	15.4	46.3	58.0	41.4	-12.4 [-24.0, -1.3]

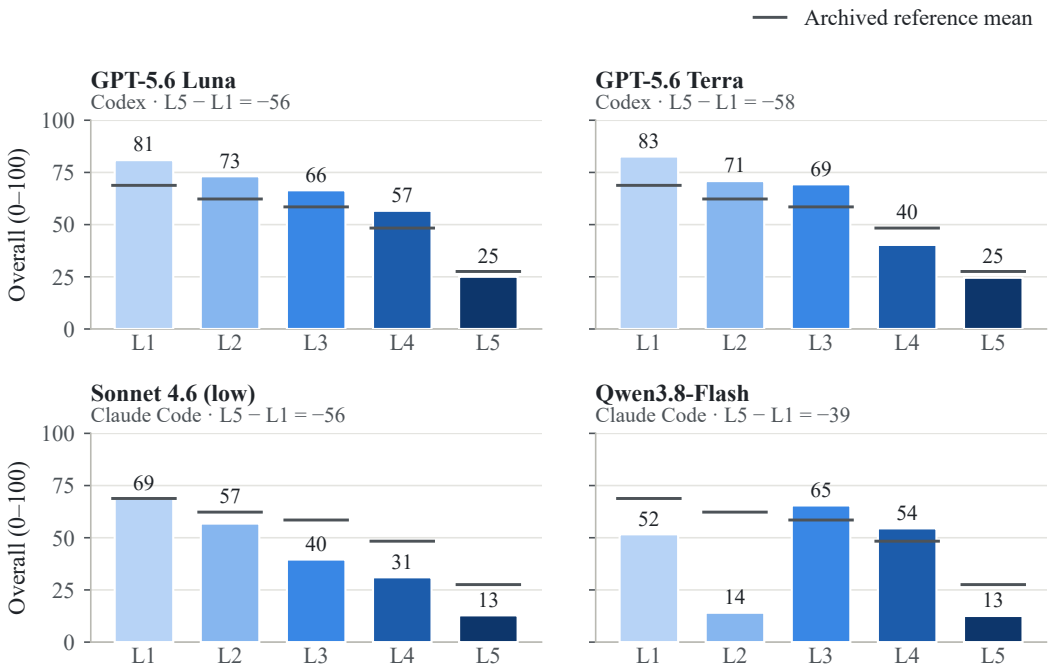


Figure 14: Overall by difficulty level for four additional leaderboard systems not shown in Figure 3. Panels share the scale and level colors of that figure. Grey ticks reproduce its archived reference mean, rather than the mean of the ten systems in the updated leaderboard. Unsubmitted tasks count as zero.

G Additional Fine-Grained Diagnostics

This appendix documents the analyses behind Section 4.3. Because these analyses need server traces and delivery summaries, they use the diagnostic cohort of 525 runs by the eight locally run configurations of Section 4.1. Its records cover 85 tasks each for DeepSeek V4.1 Flash and GLM-5.3 in Claude Code, 65 tasks for DeepSeek V4 Pro in Claude Code, which are the 58 L1–L3 tasks and seven L4 tasks, and the 58 L1–L3 tasks for each of the other five configurations. These counts describe the archived cohort rather than the tasks evaluated in the current leaderboard, where Gemini 3.8 Flash and DeepSeek V4 Pro each have 85-task results. Moreover, the archived results have not been reconciled with the updated GLM-5.3-Flash results in Table 3, so we do not use them to explain its harness differences. The analyses read server traces, delivery summaries, and submissions, and the Control-family analyses use the corrected verdicts of Appendix G.3. Unless stated otherwise, values are means over the runs in which a quantity applies, on a 0–100 scale, and brackets give 95% bootstrap intervals over the 20 task archetypes.

G.1 Sub-Components and the Loss Partition

The four stage scores are too coarse to show which part of a stage fails, so we split each stage into sub-components. Table 17 defines the sub-components used in Section 4.3. Each is computed per run from the evidence named in the last column, and all definitions were fixed before any outcome was analyzed. The Access and Control sub-components refine the stage statuses of Appendix D.4, whereas the Acquire and Assemble sub-components split the continuous stage scores by record stratum, field source, and error type.

Loss partition. To see where the required information goes, Figure 4 partitions the required record–field pairs U_t of each run in the notation of Appendix D.5. Pairs in $B_\tau \cap K_\tau$ were delivered and returned correctly. Every pair outside B_τ receives exactly one pre-delivery cause, which is that its record was never delivered, split by stratum, that its field was never fetched, split by source, or that its protected value was withheld or degraded. Every pair in $B_\tau \setminus K_\tau$ receives exactly one post-delivery cause, which is a missing submission, an omitted record, an absent field, a format-only mismatch, an approximate value, or a substantively wrong value. A pair in $K_\tau \setminus B_\tau$, which is correct in the output without recorded delivery and makes up 3.5% of all pairs, keeps its pre-delivery cause, so the partition counts delivery-supported correctness only. As a consistency check, $|U_t|$, $|B_\tau|$, and $|B_\tau \cap K_\tau|$ agree with the delivery accounting on every submitted run. For an unsubmitted run, however, only stage-level rates exist, so we attribute its missing records by stratum, count its delivered pairs as never submitted, and leave the remainder unattributed, which Figure 4 shows in grey. Shares are computed per run and then averaged over runs, and Figure 15 shows the full partition, including the L4–L5 configurations.

The partition singles out a few dominant causes. Across the 524 profiled runs, the largest pre-delivery causes are unopened detail pages, which account for 14.5% of all pairs, and records that were never reached, which account for 11.2%. The largest post-delivery causes are runs without a submission, at 6.2%, and correct values in the wrong surface form, at 7.6%. At L1, unopened detail pages and formatting dominate, at 22.3% and 12.9%. At L5, by contrast, records never delivered make up 63% of all pairs, split between the default listing, at 29.6%, and records behind filters and search, at 32.9%. The same shift from fields to records holds within DeepSeek V4.1 Flash and GLM-5.3 in Claude Code, the two configurations with records at every level, whose never-delivered records grow from at most 5% of the pairs on L1–L3 to 39–47% on L4–L5 while their unfetched detail fields stay at 12–17%. The configurations also differ. DeepSeek V4 Pro and the GLM-5.3-Flash arms lose 11–21% of the pairs to runs that collected data but never submitted, and Gemini 3.8 Flash loses a quarter of all pairs to detail fields it never fetched.

G.2 Verification Gates in Access

A pass rate alone hides how much a gate costs the agent, so we also count the refusals behind each gate. Table 4 in Section 4.3 judges each gate family on data-bearing pages, which are pages that delivered records for the task in at least one run of any configuration. A gate that blocked only other pages, such as the entry page or, from L3 on, the HTML listing shell whose data arrive through the JSON endpoint, is counted separately, because clearing it is not needed to collect data. Runs whose

Table 17: Stage sub-components. Each is computed per run and averaged over the runs in which it applies.

Stage	Sub-component	Definition	Evidence
ACCESS	Reaching data	Any data-bearing surface reached, with requests and seconds until the first data response.	Stage profile
	Listing and detail channels	A listing surface, such as an HTML listing or JSON endpoint, and a detail surface, such as a detail page or field tab, reached.	Delivery summary
	Five verification gate families	XHR header on the data API, browser proof, client fingerprint, timing check, and CAPTCHA-style challenge. Each gate met is cleared, bypassed, or not cleared on data-bearing pages, as Appendix G.2 details.	Refused requests
CONTROL	Eight protocol families	Authentication, token renewal, URL session identifier, cookie chain, API token, request signing, cursor, and rate limit. A family applies when the task requires the protocol or a refusal or terminal state event of that family occurs, and it passes when it is not among the reasons for a failed verdict.	Stage profile, state journal
	No failure loop	No repeated identical terminal failure.	Stage profile
ACQUIRE	Record strata	Delivered share of the target records in the default listing, behind filters or search, and reachable only by cross-reference.	Stage profile
	Field sources	Delivered share of the required record–field pairs whose value sits on the listing, on the detail page, or behind authentication.	Delivery summary
	Protected-field states	Protected pairs withheld for lack of authentication, degraded, or never fetched.	Delivery summary
ASSEMBLE	Hand-off	A submission exists, given that at least one target record was delivered.	Submission
	Identity	Delivered records kept, precision, and no duplicate identifiers.	Submission
	Fidelity by error type	Delivered pairs returned correctly, with errors split into record omitted, field absent, format-only mismatch, approximate value, and substantively wrong value.	Submission

gate blocked only such pages are therefore excluded from the clearance columns, which removes 21 fingerprint runs and 14 challenge runs. Clearance is eventual rather than ordered, so a blocked page counts as cleared if it delivered records at any point of the run. The column for gates met before first data identifies gates that fired before the agent's first data response, and the last column divides the median number of refused requests by the nominal request budget of Appendix B.5.

Attributing refusals to gate families requires the name of the protection that fired. The archived traces record this name only when the challenge body is JSON, so for the other 77,668 refusals we recover it from the server's error log, matched by run and time window, or from the set of active protections when only one was active. All but 14 of these refusals are resolved this way, with 75,881 coming from access-class gates and 1,773 from cookie-chain checks, a control-class protection.

G.3 PROTOCOL FAMILIES IN CONTROL

Control is a single verdict per run, so it does not show which protocol failed. Table 18 therefore breaks the verdict down by protocol family and level. A family applies in a run when the task requires the protocol or when a refusal or terminal state event of that family occurs, and it passes when it is not among the reasons for a failed verdict. The corrected rates replay the full state journal for 72 runs and approximate it from the archived trace elsewhere. Because the correction changes only authentication at L2–L3, cookie chains, and API tokens, the last column of the table gives the archived verdicts for comparison.

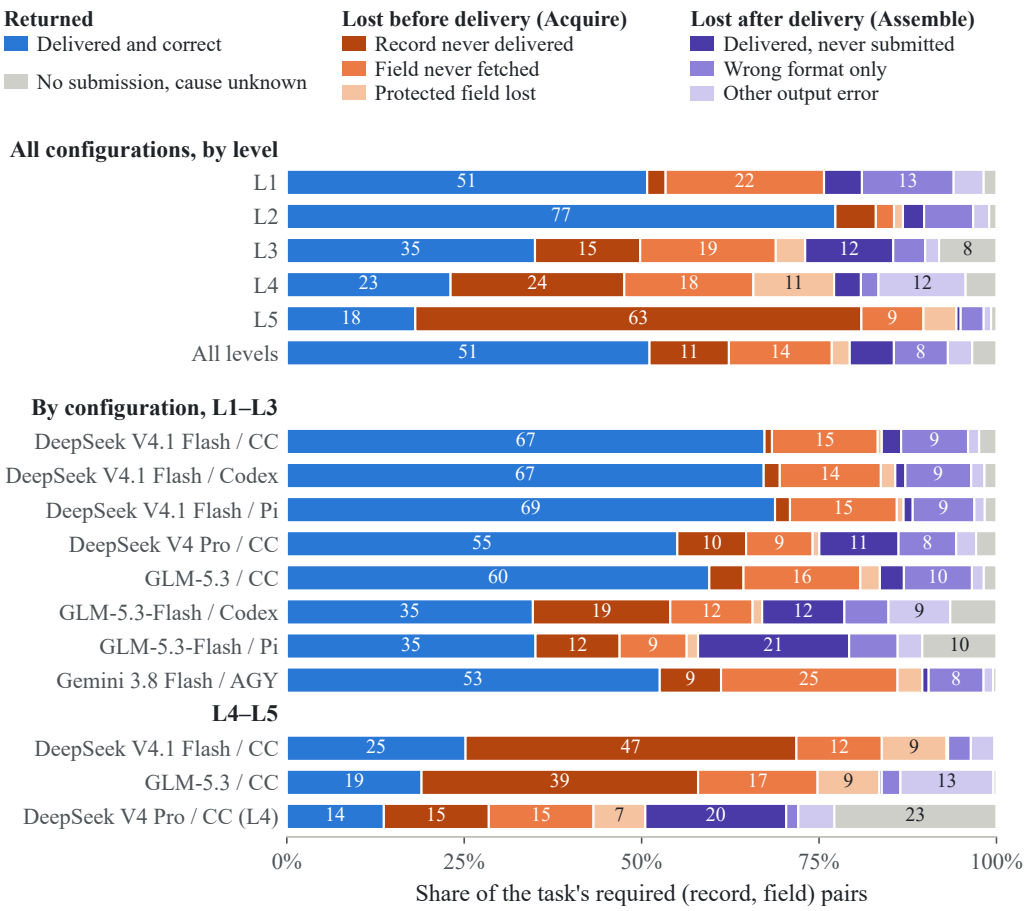


Figure 15: Full loss partition of the required record–field pairs, averaged over runs. The top block pools all eight configurations by level. The bottom block compares configurations on the 58 L1–L3 tasks and, for the three configurations with diagnostic records there, on L4–L5, where the DeepSeek V4 Pro archive covers L4 only. Labels give percentages of at least 7.

The families differ widely in difficulty. URL session identifiers are the hardest, because the agent has to carry a server-issued identifier through every URL, which it does in 8% of applicable runs. On L1–L3, only DeepSeek V4 Pro in Claude Code and Gemini 3.8 Flash, each at 30%, and GLM-5.3-Flash in Pi, at 10%, ever pass it, whereas cookie chains pass in 44–89% of applicable L1–L3 runs. Authentication falls from 71% at L2 to 4% at L5. Most failed authentication verdicts are refusals that were never followed, after login, by a successful request to the same kind of surface, so the matched-recovery rule of Appendix D.4 does not credit them. In contrast, values recorded as withheld for lack of authentication account for only 0.8% of protected fields, whereas 33.9% sit on pages that were never fetched, as Appendix G.4 shows.

G.4 Acquire and Assemble by Level

To see how these losses change with difficulty, Figure 16 follows the Acquire and Assemble subcomponents across levels, pooled over configurations. Three patterns underlie the loss partition of Figure 4.

- *The per-record detail page is the main acquisition gap.* Overall, 93% of listing fields are served but only 54% of detail fields, and at L3 the figures are 95% and 24%. Protected fields are lost the same way, since at L4–L5, 91% of protected pairs sit on pages that were never fetched and none were withheld for lack of authentication.

Table 18: Control pass rate by protocol family and level, all configurations pooled, in percent with applicable runs in parentheses. Rates use the journal-corrected verdicts, and the last column gives the archived verdict over all levels. No L1 task requires a stateful protocol.

Family	L2	L3	L4	L5	All levels	Archived
Authentication	71 (128)	30 (119)	23 (30)	4 (24)	45 (301)	36
Token renewal	–	–	50 (4)	75 (20)	71 (24)	71
URL session identifier	–	9 (79)	0 (8)	–	8 (87)	8
Cookie chain	69 (72)	–	–	67 (6)	69 (78)	65
API token	37 (79)	–	–	–	37 (79)	33
Request signing	–	–	36 (28)	17 (12)	30 (40)	30
Cursor	–	–	67 (12)	79 (24)	75 (36)	75
Rate limit	–	0 (1)	0 (1)	0 (4)	0 (6)	0

- *Records behind filters, search, and cross-references are the second gap.* Filter and search recall is 48% at L2, and cross-referenced recall falls from 59% at L3 to 11% at L5.
- *Assembly preserves identity and values but not form or hand-off.* Precision and identifier uniqueness stay at 96–100%, and substantively wrong values stay at or below 2.4% of delivered pairs, whereas format-only mismatches affect 13% of them and 19% at L1. Hand-off is lowest at L3, at 68%, where runs collect and then run out of time. At L4, 26% of delivered pairs are absent from the submitted rows, meaning that detail and protected values were served but left out of the export.

These comparisons across levels carry one caveat. Only three configurations have diagnostic records at L4–L5 in this cohort, so changes between L3 and L4 mix the level with the set of configurations.

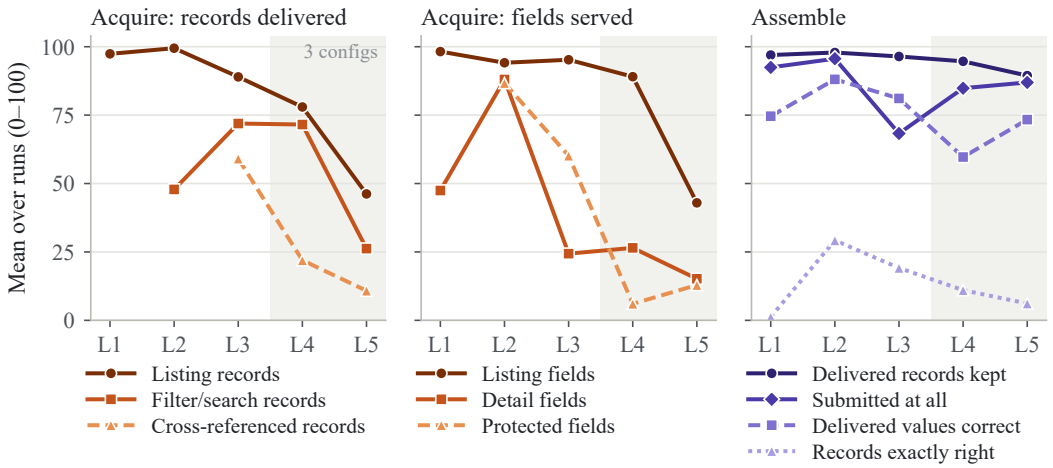


Figure 16: Acquire and Assemble sub-components by level, pooled over the configurations with diagnostic records at each level, which are eight at L1–L3 and three at the shaded L4–L5. Values are means over the runs in which the sub-component applies. Filter and search records and protected fields first occur at L2, and cross-referenced records at L3. Colors follow Figure 4, with orange for Acquire and purple for Assemble.

H Additional Efficiency Results

This appendix supplements Section 4.4 with two analyses that the main text only points to, namely how the score builds up within a run and which resources go with better outcomes. Because the harnesses log different evidence, we first state what each harness provides. We then report how the score arrives within a run in Appendix H.1 and which resources go with better outcomes in Appendix H.2.

Evidence by harness. The resource analyses use the archived diagnostic cohort of Appendix G, but not every harness logs every quantity. Per-step scores exist only for the Claude Code runs of the two DeepSeek models, whose harness scored the answer file after every step. Timestamped steps, which the phase split below needs, exist for Claude Code, for Pi through message timestamps, and for Antigravity through step durations, which sum to the wall-clock time within 1%. Codex, in contrast, reports usage once per task and no step times. Moreover, runs stopped at the time limit report no token usage on the Codex, Pi, and Antigravity paths and for GLM-5.3 in Claude Code. We therefore impute their prompt tokens per configuration from a regression of log tokens on log elapsed time and level. Prompt tokens count input, cache reads, and cache writes.

H.1 Progress within a Run

A final score does not show when a run earned it, although this timing decides what a run returns if it stops early. Figure 17 and Table 19 therefore follow the score of the answer held during a run for the two configurations with per-step scoring, and the table adds the L4–L5 runs. For runs with a positive final score, the first answer comes after 60–66% of the run's tokens, and 90% of the final score follows only 4–10 percentage points of tokens later, so the score arrives in one late step, and a run stopped before that step scores zero even when data have already been delivered to it. Because runs place this step at different points, the mean curve in the left panel of Figure 17 rises only gradually. Once written, answers rarely get worse, with a mean largest drop within a run of about 1 point. DeepSeek V4 Pro writes its L4 answers only at the very end of its runs. After the answer stops changing, DeepSeek V4.1 Flash still spends 18% of its tokens, and a longer tail goes with lower field coverage. Even so, it leads V4 Pro at every evaluated budget.

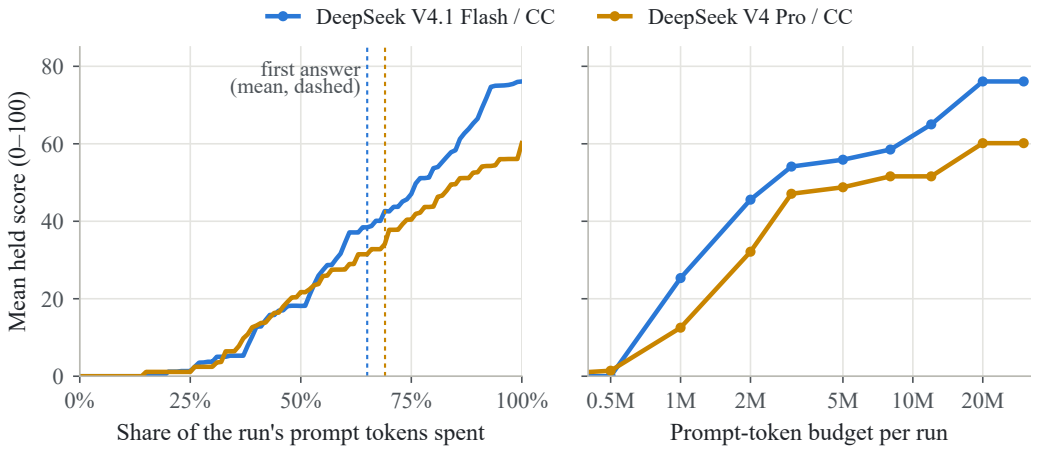


Figure 17: Mean score of the answer held during a run on the 58 L1–L3 tasks, for the two configurations with per-step scoring. The left panel plots it against the share of the run's own prompt tokens, with dashed lines at the mean point of the first answer, and the right panel against an absolute prompt-token budget on a log scale. A run holds 0 before its first answer and keeps its final score after it ends.

Where do the tokens go? To see which activities consume the tokens, Figure 18 splits the prompt tokens of each run into four phases aligned with the stages. Orientation runs from the start to the first data response and corresponds to Access. Development runs from the first data response to

Table 19: Within-run score curves for the two configurations with per-step scoring. Token shares are means over runs with a positive final score, as a share of each run's own prompt tokens, and the mean held score averages the held score over the run's tokens. Final scores refer to the stated level subsets of the archived diagnostic cohort, not the 85-task leaderboard.

Configuration	Levels	Runs (final > 0)	First answer	90% of final	After last change	Largest drop	Mean held score	Final
DeepSeek V4.1 Flash	L1–L3	58 (55)	63%	68%	18%	1.1	27.2	76.1
	L4–L5	27 (18)	66%	76%	14%	1.4	13.3	40.6
DeepSeek V4 Pro	L1–L3	58 (44)	60%	65%	23%	1.0	23.0	60.2
	L4	7 (3)	89%	93%	5%	0.0	3.1	29.1

the first bulk collection loop and covers probing the site and writing the crawler. Collection runs from the first loop to the last site request and corresponds to Acquire, including re-runs and work on verification gates. Closing runs from the last site request to the end and covers assembling, checking, and submitting.

Orientation costs 1–7% of the tokens everywhere, so the phases differ mainly in what follows it. At L1, development and closing take 46–69% of the tokens, whereas from L3 on, collection takes 67–88% and closing only 3–11%, because runs submit right after their last request or run out of time. At L1, DeepSeek V4 Pro, GLM-5.3-Flash, and Gemini 3.8 Flash spend 29–34% of their tokens on development, against 17% for DeepSeek V4.1 Flash in Claude Code or Pi, which suggests that efficient agents turn deliberation into collection code early.

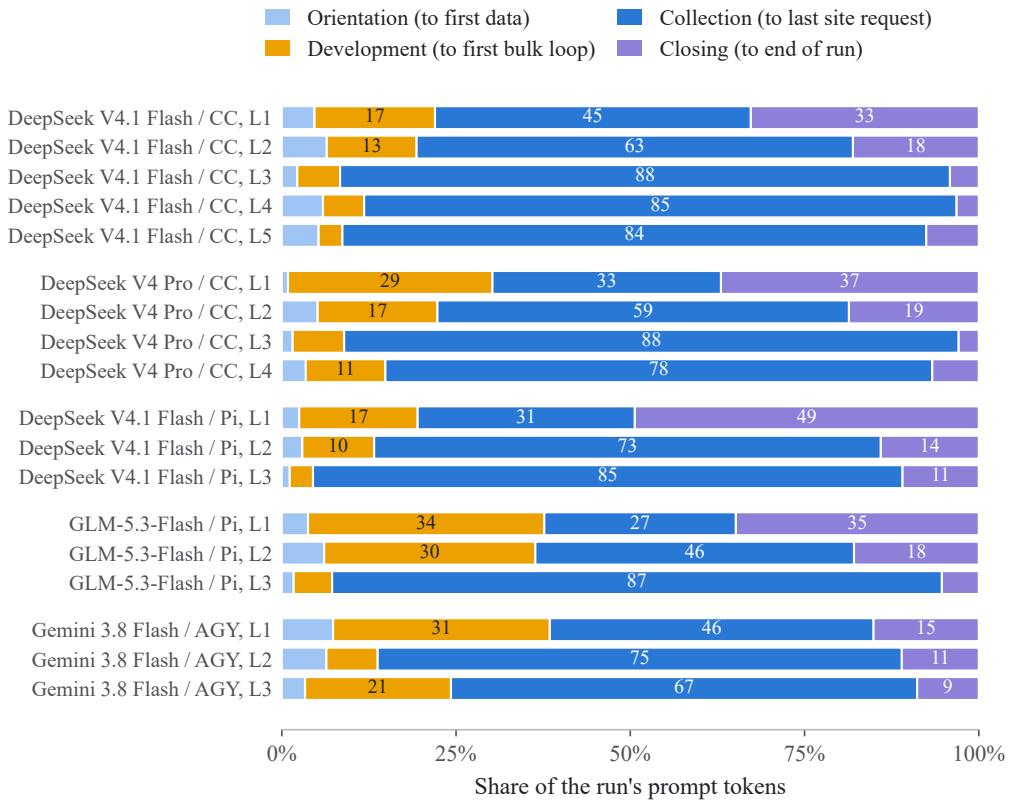


Figure 18: Share of each run's prompt tokens by stage-aligned phase, for the five configurations whose logs allow the split. Server timestamps placed on the model timeline divide each run into orientation, development, collection, and closing, which end at the first data response, the first bulk collection loop, the last site request, and the end of the run. Labels give shares of at least 8%.

H.2 Resource Use and Outcomes

The within-run curves cover only two configurations, so we also ask across all runs which resources go with better outcomes. Figure 19 reports observational correlations between log resource use and seven outcomes, controlling for task and configuration. More site requests go with higher record coverage, field coverage, and detail-field coverage, with correlations of $+0.42$, $+0.49$, and $+0.38$. Prompt tokens, in contrast, go with more records behind filters and search, at $+0.38$, but with fewer detail fields, at -0.18, and longer wall-clock time goes with a lower Overall, at -0.12. The token correlations use the 397 runs that report usage and exclude the other 128 runs. These associations do not identify causal effects, but together with the token phases of Appendix H.1, they suggest that coverage comes from issuing requests rather than from deliberation.

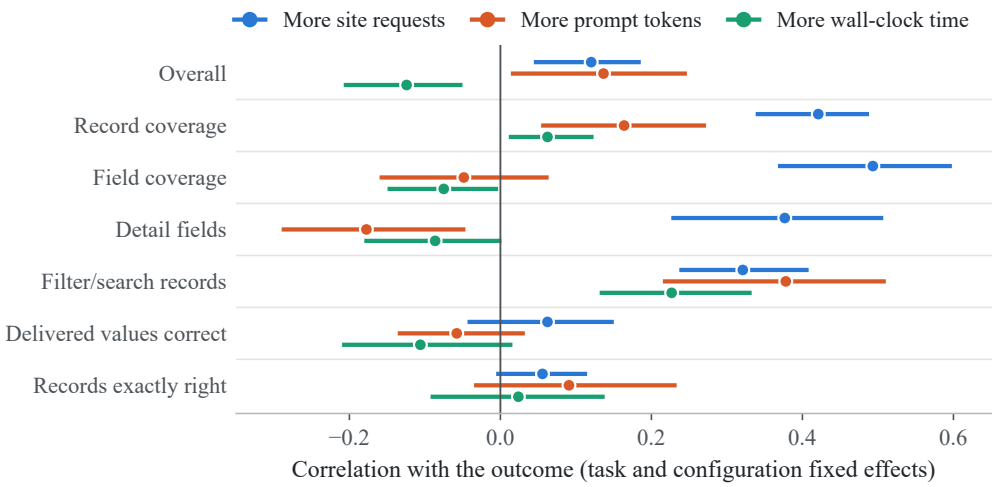


Figure 19: Correlation between log resource use and seven outcomes, controlling for task and configuration fixed effects and shown with 95% archetype-bootstrap intervals. Token correlations use the runs that report usage.

I LIMITATIONS

We close with the main limitations of the benchmark and of our analyses.

- *Realism.* The generated, self-hosted sites give control over content, mechanisms, and delivery records but do not reproduce the live web, and their verification mechanisms do not stand in for commercial anti-bot services.
- *Confounded difficulty.* Scale, mechanisms, and time limits change together across levels, so the level trends do not isolate a single factor. Ablations that toggle one mechanism family at a time are left to future work.
- *Scope of the diagnostics.* The trace-based analyses cover only the eight locally run configurations in an archived cohort that differs from the updated leaderboard. Their associations are observational, and parts of their evidence are reconstructed or imputed, such as gate names, some Control verdicts, and the token usage of time-outs.
- *Validation.* The evaluator tests use constructed inputs, so neither the agreement of the stage labels with human review of real runs nor complete reference solutions under the evaluation budgets have been established.
- *Submission feedback.* A submission response reports Overall and its dimensions, and a later submission replaces an earlier one, so assembly scores may mix self-correction with the use of this feedback.