TAMER: TESTING AGENTS IN COMPLEX MULTI-TOOL ENTERPRISE KNOWLEDGE-WORKER ROLES

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce **TAMER**, a benchmark for evaluating vision-language-model agents on realistic knowledge work in an enterprise setting. TAMER contains 238 tasks requiring an agent to coordinate actions in a simulated environment consisting of eight web apps it observes only as screenshots and operates through coordinate-based tools, grounded in a synthetic organization of 54 employees across 19 teams and 21 projects spanning five months of interlinked history. Each task ships a hybrid verifier combining deterministic scripts with rubrics graded by a five-model LLM-judge panel (majority vote). Measured difficulty splits the set into 24 easy, 98 medium, and 116 hard tasks. Across six models, the highest per-attempt accuracy on the full set is $42.3 \pm 3.6\%$, falling to 6.0% on the hard subset. Pass rate also falls with the number of web apps a task spans.

1 INTRODUCTION

As agentic AI capabilities continue to evolve, there has been growing interest in their integration into real-world workflows Yao et al. (2025); Boisvert et al. (2024); Xie et al. (2024); Zhou et al. (2024). In an enterprise setting, agents able to take on the role of a knowledge worker and perform complex tasks may offer opportunities for increased productivity. In this work, we evaluate an agent's ability to complete multi-step tasks involving planning, reasoning, information extraction and summarization, reflecting project management tasks typically expected from a tech-lead, a technical program manager, or a project manager. A TPM tracking a delivery risk reads a status report on the team wiki, checks whether the goal tracker reflects it, searches the relevant chat channel for the discussion that caused the change, and confirms who owns the fix in the company directory. Only then can they write a useful summary, which must be factually consistent with all four sources while resolving ambiguities in the request and adjudicating among conflicting sources of information. A TPM performs this workflow by interacting with a rendered software interface without API access, while access to the available data across their organization is bounded by their channel membership, mailbox access and document permissions, etc. TAMER applies the same constraints on an agent, where its observation space is bounded by level of access and permissions within a simulated environment.

Existing benchmarks for agentic web use and tool use evaluate tasks that span multiple systems but they do so through accessibility trees, DOM text, or tool APIs (Zhou et al., 2024; Mialon et al., 2023). Where the agent observes rendered screenshots instead, tasks are almost entirely single-site (§2). A coherent organization whose projects interconnect over months is expensive to author, so world state is assembled per task or drawn from the live web rather than persisting across all tasks on a consistent timeline. This means that no task can depend on what happened several months earlier in a project the agent was never told about. Correctness is judged from structural state changes alone or from a match to a recorded demonstration, so a task whose realistic output is a written answer is either left out or scored by a proxy signal such as whether text was posted at all. Effects outside a task's target are graded by enumerated policy or by contract over expected state changes, both established where the agent acts through an API, rather than by checking every system an agent operating a rendered interface could have reached. And the agent is either a generic operator with unrestricted visibility or a persona whose preferences it must infer, rather than an employee whose access is scoped by their position in an organization.

We introduce TAMER, a benchmark for ***T**esting **A**gents in Complex **M**ulti-tool **E**nterprise Knowledge-worker **R**oles*. TAMER uses a persistent world: We generate a fictional software company, with 54 employees across 19 teams with 21 different projects and five months of interlinked history, which sets the environment for eight web apps: a team wiki, goal tracker, chat, email, meet-

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.

ing scheduler, ticketing, document sharing, and a worker directory. Projects succeed, stall, or get de-prioritized and canceled. The status of a project changes in a chat thread, but the wiki report describing it and the tracker entry recording it can all report an older status, with overlapping but deliberately non-identical content. Finding an answer requires going to the right web app. Agents act as a specific named employee whose channel membership, mailbox, and document access are scoped as that person's would be, and observe the web apps as that person (JPEG screenshots) and act through pixel-coordinate mouse and keyboard tools. The accessibility tree, DOM, and web app APIs all exist, and we withhold them. An agent that can read page structure couldfind an answer by searching text within a page, which skips some page navigation the task was built to measure, though it does not remove cross-application coordination or persona-scoped access. Rendered screenshots are also the only channel available on every surface that an agent like this is deployed to (§4).

Our contributions in this benchmark are as follows.

- **A persistent, cross-referenced enterprise world and a task set that covers it.** One coherent five-month organization materialized consistently into eight web apps, with interlinked and independent projects, unrelated records included as distractors, and persona-scoped access. Construction and validation are described in §3.2 and Appendix C.

- **Baseline results for six models, and the harness that produced them**. We evaluate six models and release both the runs and the harness that produced them. These are reference numbers: they hold for this harness and these model versions, and a different scaffold over the same tasks will give different figures, which is why we report the configuration in full (§4.2). The task set is built to separate two things other similar benchmarks tend to hold constant. Of 238 tasks, 124 (52.1%) need two or more web apps, so coordination across these is varied rather than fixed, and tasks are graded on written answers as well as on state changes, so the two grading modes can be compared on one task set. Our reference results show that this matters: written answers pass at well under half the rate of pure state changes, and pass rate falls as the number of required web apps rises, reaching zero at five or more (§5.3).

- **Verification that grades written answers and detects untargeted writes.** We write one verifier per web app per task, and each verifier checks two kinds of thing. Structured fields are compared deterministically against mutation logs emitted by the web apps based on the agent's actions, while free text is graded by yes-or-no rubric criteria (one per fact or item required) including one that checks that the answer doesn't hallucinate anything or add anything extraneous. For every web app that a task does not write to we add a verifier asserting zero meaningful mutations. This means the task fails if the agent writes anything in any wrong place, and that a task whose correct execution changes nothing can still be graded. Grading the state this way is established where the agent acts through an API (Styles et al., 2024; Pysklo et al., 2026), but among browser benchmarks observing rendered pixels it is absent or restricted to policies that the tasks enumerate in advance (Levy et al., 2026) (§3.4).

2 Related Work

Enterprise multi-tool benchmarks grade state changes but the agent calls an API rather than operating an interface. Agentic web-use for enterprise productivity work is increasingly evaluated through benchmarks consisting of self-hosted or mocked suite of web apps by grading state changes on the web (Xu et al., 2025; Styles et al., 2024; Trivedi et al., 2024; Malay et al., 2026; Li et al., 2026). Two recent benchmarks in this family changed how their grading works rather than what the environment contains. Agent-Diff defines success as whether the expected change in environment state was achieved rather than by matching a trace (Pysklo et al., 2026), and GAIA2 evaluates against an environment that evolves independently of the agent (Froger et al., 2025). Saturation is fast: revisiting WorkBench two years later, the best agent's completion rose from 43% to 89% while its rate of unintended harmful actions fell from 26% to 2.5% (Styles & Miller, 2026). In all of these benchmarks, the agent reaches the applications through an API rather than through their interfaces, and none binds it to a named employee inside a simulated organization. We keep two things from this family, its multi-app scope and its state-based grading, and change two. The agent operates a rendered interface instead of calling an API, so it has to find a control before it can use it. It acts as one named employee whose access is limited to what that person can see, instead of as an unrestricted operator.

Benchmarks that observe rendered pixels are predominantly single-site. Multi-app tasks do exist in this space: WebArena was the first web environment to support multi-tab tasks and labels 5.9% of its 812 tasks as cross-site (Zhou et al., 2024); MMInA (Tian et al., 2025) and Odysseys (Jang et al., 2026) consist of live-web tasks that span several websites, and ST-WebAgentBench adds per-task policy instances over three sites (Levy et al., 2026). None of these however report results from screenshots. WebArena renders page content as a screenshot, a DOM tree, or an accessibility tree, but its reported baselines all use the accessibility tree; the benchmarks built specifically for visual observation are predominantly one site per task (Koh et al., 2024; He et al., 2024; Xue et al., 2025; Garg et al., 2025; Miyai et al., 2025). OS-level benchmarks such as OSWorld (Xie et al., 2024) and WorkArena (Boisvert et al., 2024) are visual and multi-application, but over arbitrary software rather than one shared organization. Where these benchmarks support long-horizon tasks that need multi-step reasoning and information synthesis across several sites, the supporting content is either supplied per task or taken from the live web. Per-task content means no task can depend on something that happened months earlier in a project the agent was never told about. Live-web content cannot be re-run against a fixed target, because the sites change underneath the benchmark, so two runs are not comparable. We generate one organization and share it across tasks, so the same history is available to multiple tasks and any run can be repeated against this same state.

Benchmarks are dropping LLM judges, but the fault lies in holistic judging, not in using a judge at all. Several recent releases dropped their model judges: WebArena-Verified replaced its judge with exact comparisons (ServiceNow Research, 2025), Online-Mind2Web tells its judge to ignore whatever the agent wrote because it may be invented (Xue et al., 2025), and ST-WebAgentBench avoids model judgment altogether (Levy et al., 2026). Experts who labelled agent runs across five benchmarks found that no automatic judge was reliable across all of them, and that unwanted actions were harder to detect than outright failure (Lù et al., 2025). What these judges have in common is that each was asked to read a whole response and return one verdict. Splitting that into separate yes-or-no questions gives a different result: Odysseys grades each task against about six specific requirements instead of one overall judgment, and agrees with human graders more often as a result (Jang et al., 2026), and Lee et al. (2025) show the improvement comes from making each question binary, which raised agreement across twelve different judge models. Writing one such question per fact, with the correct answer supplied, is already standard where correctness is too open-ended to check by pattern matching (Starace et al., 2025; Arora et al., 2025; Wang et al., 2026). We therefore use both kinds of check rather than choosing between them. Structured fields go to exact comparison, which is where deterministic grading is strong and a judge adds only noise. Written answers go to a judge, one yes/no question per fact against a reference answer. We also measure how much independent judges agree rather than assuming they do (§3.4, §A).

3 Benchmark Construction

This section describes each stage of construction: the web apps (§3.1), the world that populates them (§3.2), task authoring(§3.3), and verification (§3.4). Figure 4 in Appendix C shows the full pipeline including evaluation.

3.1 Web apps: Simulating widely-used enterprise productivity software

TAMER includes eight web apps, each simulating widely-used enterprise productivity software: a goal tracker, a team wiki, chat, an email app, a meetings scheduler, a ticketing tool, collaborative document storage, and a company directory for employee information and organizational structure.

Each web app exposes a `/finish` endpoint emitting a structured log of every action taken on it, as `CREATE`, `UPDATE`, and `DELETE` records of the form {`entity, operation, oldValue, newValue, primaryKey`}. That common structure is what allows a single verifier interface across all eight heterogeneous web apps. Web app schemas and action inventories are in Appendix F.

Two properties are shared across all eight web apps: First, the clock is frozen to a known date and time, such that relative-time reasoning and computations are deterministic across runs. Second, each web app is initialized with the task's simulated user (referred to as persona) logged in, making identity a fixed property of the environment rather than an action the agent must perform. This mimics a real setup, where an employee logging into their device sees only their own permitted view (e.g., their direct messages on chat and only the documents that are shared with them).

3.2 THE WORLD: SIMULATING A TEMPORALLY COHERENT ENTERPRISE SETUP

Due to privacy limitations of collecting and using real enterprise data, TAMER operates on a synthetically generated world that simulates an enterprise setting.

A typed *world specification* fixes the organization: 54 employees across 19 teams, their reporting lines, 21 projects, and a communication-style profile per person. A *storyline generator* then expands each project over five months into four kinds of dated events: milestone completions, team decisions, status changes, and dead ends where an approach is abandoned. Projects therefore succeed, stall, and fail over time instead of being locked into a fixed state. A language model then writes the content each project-by-week implies, and one converter per web app maps that content into that app's own schema. The generated world is thus observable through eight web apps. We call the result the *world data*: the records that populate each web app's database and define the state every task starts from.

To enable complex tasks we construct the world such that information is dispersed throughout. Concretely, we make three design choices that require an agent to search and cross-check a populated organization, rather than read from a single source. First, the observation spaces of the web apps overlap but are not identical. Each event is materialized into *several* web apps with overlapping but deliberately non-identical content. The chat discussion of a status change, the wiki report describing it, and the tracker entry recording it all exist, but none is a superset of the others. Second, we generate tasks in two waves that share the company but have no people or projects in common. Finding the right record therefore means discriminating between similar candidates rather than retrieving the only one available, which is the difference between searching a populated organization and searching a set built around a single task. Third, the persona's access is scoped as a real employee's would be: channel membership, direct messages, mailbox, and document permissions are all restricted, so an agent cannot see the whole organization at once.

We ensure consistency of the generated world by applying automated deterministic checks, as well as human-in-the-loop (HiL) audits. The automated checks are written based on rules defined by the world schema. HiL checks target semantic coherence and realism. Both categories of checks are run after world generation, as well as after the generated project-by-week content is mapped to the web apps' schemas. The automated rules cover alias validity, authorship against out-of-office schedules, temporal consistency of correspondence, and language-model checks on numeric coherence where no closed-form rule applies. Recurring findings are promoted into new automated rules. Details are in Appendix C.

3.3 TASK AUTHORING AND HUMAN AUDITING

A task is a natural-language prompt written in the voice of the persona or a colleague addressing them. We divide tasks into three broad categories by the information flow they require. In a *read* task the agent gathers information across web apps and returns a text-only answer, changing nothing. In a *write* task the prompt supplies all necessary information and the agent acts on the target web app(s). In a *read-write* task the agent has to find the information first and then act on it. Read-write is the largest category, constituting 68.1% of the 238 tasks, while read-only and write-only tasks make up 23.5% and 8.4%, respectively.

Task authors use model assistance to write tasks against the world data. The model has full data access and knows the web apps' schemas and authoring rules, so candidate prompts cannot reference people, projects, or actions that do not exist. Every prompt is then audited against three requirements: each request is grounded in an executable action or in the world data; ambiguous sources either agree or are disambiguated in the prompt; and the task is executable from what the interface shows. Prompt structure is deliberately varied so the task set does not converge on one phrasing pattern.

Roughly 20 trained language experts then execute every task by hand in the live web apps. For each task, a language expert produces the correct execution and captures its mutation log, then produces three deliberately incorrect executions: one with a required action removed, one with an extra unrequested action, and one with the same actions but with a wrong value. A second language expert independently reviews the submission. These experts may also discard a task as ambiguous or infeasible, with a written reason. The resulting executions make up a positive test and three negative tests to validate the task verifier accuracy along with the designer's review (described in Section §3.4).

3.4 VERIFICATION: HYBRID DETERMINISTIC AND RUBRIC-BASED TASK GRADING

For each task we write one verifier per provisioned web app. A verifier is a list of criteria, each returning 1 if it passes and 0 if it does not. Writing $A(t)$ for the web apps provisioned for task t, and $c_{a,j}$ for the j-th of the n_a criteria we write for web app a, the task score is

$$S(t) = \prod_{a \in A(t)} \prod_{j=1}^{n_a} c_{a,j} \qquad \text{with} \qquad c_{a,j} \in \{0, 1\}, \tag{1}$$

so $S(t) = 1$ only if every criterion of every provisioned web app passes, and $S(t) = 0$ otherwise. There is no partial credit. Each $c_{a,j}$ is either deterministic or graded by a model, and criteria come from two sources: what the prompt asks for explicitly, and requirements it leaves implicit, importantly that the agent should not act where it was not asked to. For a web app the task does not write to, the criteria are exactly those zero-change checks. Where the task requires an answer to the user, the criteria also cover whether that answer is correct.

Structured fields (entity, operation, recipient lists, channel and item identifiers, path fragments, status enums, and numeric quantities) are checked deterministically by exact match, set membership, or bounded arithmetic. Free-text fields the agent writes are graded only by rubric[1]. Verifiers are assembled from parameterized code blocks, so once a check pattern is written, it behaves identically everywhere it is used. Criteria are binary and cover one fact each. Binary decomposition is what makes independent judges agree (Lee et al., 2025). The rubric is composed of 3 general criteria and 1-10 task-specific criteria. The general criteria check for formatting, conciseness, and hallucination, and task-specific criteria vary depending on the task (see details in Appendix E). Each rubric is graded by a panel of five LLM judges (Claude Sonnet 4.5, DeepSeek-V3.2, Kimi K2.5, GLM-5, Qwen3-Next), each at temperature 0; the reported verdict is the majority (at least three of five).

Task authors write and review each rubric alongside the task and its reference execution produced by a language expert. The independently produced human executions provide a *falsification test*: it runs each verifier against the four independently produced and reviewed executions of §3.3 and requires it to accept the correct one and reject the three incorrect ones. Over-permissive and over-strict criteria fail in opposite directions. The test can therefore refute a verifier through either failure mode. A verifier that fails is revised and re-run. When prompt ambiguity causes the failure, the task returns to authoring, so the test can also falsify a task. This procedure establishes correctness of the rubric as interpreted by the judge panel; whether the *judges* apply one as a person would is separate, evaluated separately as in Starace et al. (2025), and §A.

3.5 EMPIRICAL DIFFICULTY TIERS

We partition the tasks by performance, following an N-model stumping construction over the $N{=}6$ models of Table 2. For each model we bucket its per-task complexity by the fraction of that model's own attempts which succeed: *high* is $0/3$, *medium* is $1/3$ or $2/3$, and *low* is $3/3$. A task is then *easy*, *medium* or *hard* according to whether 0–2, 3–4, or 5–6 of the six models are at high complexity. This yields 24 easy, 98 medium and 116 hard tasks of 238, with 90 tasks stumping all six models.

4 EVALUATION HARNESS

4.1 EVALUATION DETAILS

Agents are driven through a Playwright MCP browser harness masking most text-based tools and forcing the agent to react to visual observations of a currently active tab by coordinate-based mouse actions, typing by a simulated keyboard, a tool for switching tabs, and several miscellaneous utility tools. The Playwright instance with which the MCP interacts is launched headless at 1280×720 with a fresh isolated profile per-task-attempt. Before the first agent turn, the harness seeds the browser session by opening one tab per provisioned web app, clears each access gate. Provisioning is deliberately wider than the task requires: 215 of the 238 tasks open more web apps than they need and 131 open all eight, so the tab list is not a cue that narrows which application to consult.

[1]We don't rely on regular-expression matching over free text as it's brittle and rewards surface form and can penalize valid paraphrases. The exception is when checking for numeric values that are within free-text or easily computed, such as length, in which case we apply deterministic checks on the result of the rubric-based checks.

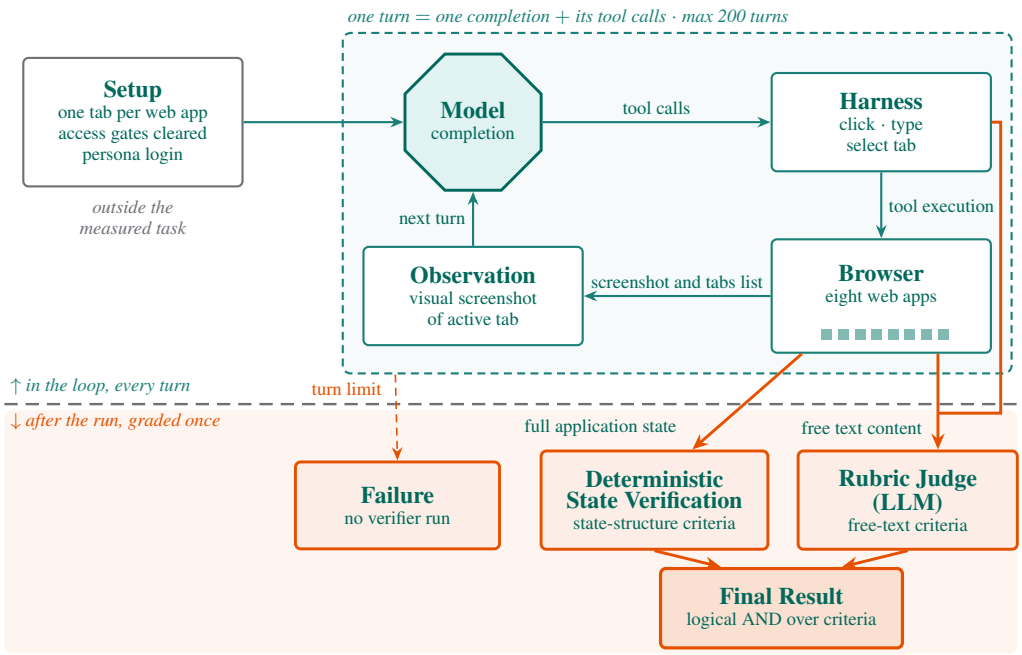


Figure 1: The per-task agentic harness loop and final verification. The harness opens one tab per provisioned web app, clears each access gate, and logs in as the persona, all outside the measured task. A *turn* is one model completion plus execution of every tool call it requested. The run ends when a turn reports finish, or at the turn limit, which is scored a failure before any verifier runs. Everything below the dashed line happens exactly once, and direct-to-user answers are read from the agent's trace: the final state is polled and verified on *every* provisioned web app, uninvolved ones being graded by a verifier asserting zero meaningful mutations, and the verdict is the logical AND over all criteria. Configuration is in §4 and Appendix F.

Pre-task discovery of the environment and authentication are deliberately outside the scope of this work: TAMER measures information synthesis, navigation, extraction, and mutation over a pre-opened workspace. Navigation is sandboxed to the provisioned environments, and every run reported here is single-agent. We withhold the accessibility tree, the DOM, and web-app APIs from the agent as a design choice for the benchmark.

One *turn* is one model completion plus execution of every tool call it requested. History is append-only with no text truncation; the only reduction is image pruning of a throwaway copy before each request. Completion is signaled by a turn reporting finish. We set max turn limit to 200. Hitting a max-turn limit is considered a failure. Models are evaluated through Amazon Bedrock at temperature 1.0, with extended thinking enabled where the provider exposed it on Bedrock at evaluation time. The 6 evaluated models were chosen due to availability through Bedrock as of 2026/07/18. Full configuration, including timeouts and per-model endpoints, is in Appendix F. A representative example of Kimi 2.5's execution of an email- and chat-centric task is also in Appendix-F.Figure 2, with higher detail examples including specific per-turn tool calls and observations for interacting with the Email and a Document Sharing web apps are in Figure 5 and Figure 6 respectively.

Two details of the grading path affect how the results should be interpreted, both visible in Figure 1: verification runs once, after the loop terminates, and it reads the mutation logs the web apps themselves emit, never the intermediate action trajectory. `/finish` is polled on *every* provisioned web app, not only those the task requires, so an unwanted effect outside the scope of a given task's intended environment is a checked condition and not an unmeasured risk. A side-effect filter then drops system entities before comparison: logging in alone mutates read states and user records, so without the filter every task would fail its action count.

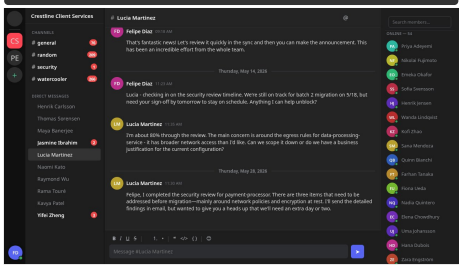
4.2 Release

The release comprises four artifacts: the task set, one task per JSON file carrying its prompt, per-app verifier, expected mutations, and the metadata of Table 1; the web apps in locally hostable form;

Example Cross-Gym Flow — Chat → Email

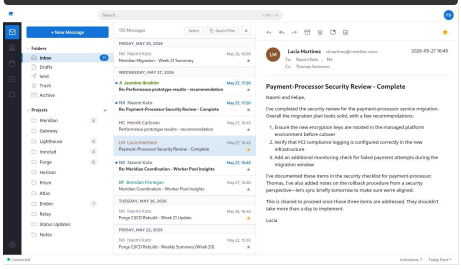
Prompt: *Lucia finished the security review for the payment-processor migration and flagged some compliance items that need to be addressed before we can proceed. Can you check my messages with Lucia and the #meridian channel for the details, and send Vivian an email summarizing the findings and what it means for the migration timeline? CC Naomi and Lucia.*

Milestone 1: Finding the Conversation (Chat)



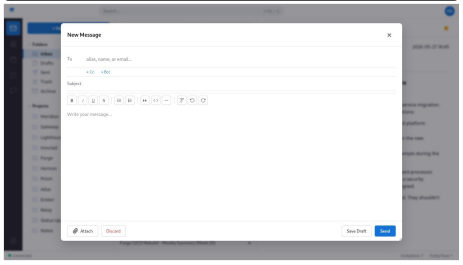
Turn 3 thought: *"I found the conversation with Lucia Martinez! She flagged three items to address before migration – mainly network policies and encryption at rest – and will send detailed findings by email. Now I need to check the #meridian channel for more details."*

Milestone 2: Finding the Matching Email



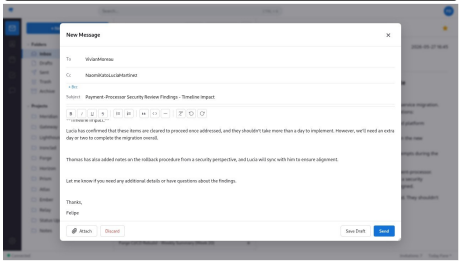
Turn 24 thought: *"I can now see Lucia's email with the detailed findings: (1) rotate the new encryption keys before cutover, (2) verify PCI compliance logging, (3) add monitoring for failed payment attempts. Now let me compose the summary email to Vivian."*

Milestone 3: Opening a New Email Draft



Turn 35 thought: *"A new message compose window has opened. Now I need to: (1) Add Vivian as the recipient (To:), (2) Add Naomi and Lucia in CC, (3) Add a subject line, (4) Compose the message."*

Milestone 4: Drafting and Sending the Email



Turn 46 thought: *"The email is now complete: To Vivian; CC Naomi and Lucia; subject 'Payment-Processor Security Review Findings'; the body summarizes the three items and the 1–2 day timeline impact. Now let me send the email."*

Figure 2: A cross-gym flow from Kimi 2.5: the agent gathers the security-review details from Chat and from Lucia's email, then drafts a summary to Vivian (CC Naomi and Lucia). Rather than every action, each panel shows a milestone screenshot with a truncated excerpt of the agent's reasoning, including its plan for the next steps. Verification results for this execution are shown in Figure 7 and detailed turn-by-turn viewing with the agent's tool calls can be found in Figure 5.

the reference harness described above; and our own runs as machine-readable trajectory logs, with web app versions and model identifiers recorded per run. The 44 tasks that require only the three web apps with no proprietary analog (chat, email, and document sharing) are designated for public release; the remainder remain private and we will publish a leaderboard similar to Table 2 with the high level metrics on SOTA models. A verifier reads only the mutation logs and the final response, never the trajectory, so the tasks can be scored against any agent implementation and are not tied to this harness. Reusing our harness reproduces our numbers; substituting another exercises the same tasks under a different agent scaffold, and we report the harness configuration in Appendix F so the two can be told apart.

5 Results and Discussion

5.1 Benchmark subsets

The benchmark consists of **238** tasks, each passed authoring audit, human execution, and verifier calibration. We split the benchmark into three difficulty tiers by post-hoc analysis of measured pass rates; Table 1 shows the composition of the task set.

Table 1: Task composition by difficulty tier. *Information flow* is what the task asks of the agent: return text (Read), act on instructions already given (Write), or discover information across applications and then act (Read-write). The tiers (§3.5) partition the 238 tasks.

	Information flow			Web apps required				
	Read	Read-write	Write	1	2	3	4+	Total
All	56	162	20	114	65	47	12	238
Easy	5	14	5	20	4	0	0	24
Medium	17	67	14	55	28	11	4	98
Hard	34	81	1	39	33	36	8	116

Table 2: Reference results over the 238 tasks, partitioned three ways; each partition is exhaustive, so subset sizes sum to 238 within a block. All values are percentages, every model scored on three independent rounds. Models: Claude Opus 5 (O5), Claude Sonnet 5 (S5), GPT-5.6 Sol (Gs), GPT-5.6 Terra (Gt), Kimi 2.5 (K2.5), Qwen3-VL 235B (Q3). Models are listed alphabetically.

		Per-attempt						pass@3					
Subset	n	O5	S5	Gs	Gt	K2.5	Q3	O5	S5	Gs	Gt	K2.5	Q3
All tasks	238	31.5	42.3	13.6	8.5	8.7	0.0	43.7	53.4	22.7	14.3	18.9	0.0
Difficulty tier													
Easy	24	72.2	86.1	68.1	44.4	30.6	0.0	91.7	100.0	100.0	62.5	66.7	0.0
Medium	98	56.1	74.5	16.0	8.2	12.6	0.0	76.5	91.8	29.6	16.3	27.6	0.0
Hard	116	2.3	6.0	0.3	1.4	0.9	0.0	6.0	11.2	0.9	2.6	1.7	0.0
Information flow													
Read	56	13.7	23.8	19.0	16.1	5.4	0.0	21.4	37.5	33.9	25.0	12.5	0.0
Read-write	162	32.3	44.0	10.1	5.1	8.2	0.0	46.3	53.7	17.3	8.6	17.9	0.0
Write	20	75.0	80.0	26.7	15.0	21.7	0.0	85.0	95.0	35.0	30.0	45.0	0.0
Web apps required													
1	114	40.9	53.5	24.0	15.2	13.5	0.0	54.4	66.7	36.8	22.8	27.2	0.0
2	65	30.8	45.1	6.2	3.1	6.2	0.0	43.1	55.4	13.8	9.2	16.9	0.0
3	47	13.5	15.6	2.1	2.1	2.8	0.0	21.3	23.4	6.4	4.3	6.4	0.0
4+	12	16.7	25.0	0.0	0.0	0.0	0.0	33.3	33.3	0.0	0.0	0.0	0.0

Every task carries the metadata these breakdowns are computed from: information flow, the applications it reads and writes, the persona. Prompts are multi-sentence directives, all distinct, with median length 71 words (maximum 305), and no task pair exceeds a TF-IDF cosine similarity of 0.8. Per-application coverage is in Appendix F; the write side is thinner and skewed toward the goal tracker.

We report two metrics: **per-attempt** pass rate (mean pass@1) which measures single-try reliability, and pass@3, which measures success within three attempts. An end user likely values per-attempt rates higher as a measure of single-try reliability when using the agent. A given task execution passes only if every criterion of every provisioned web app's verifier passes (§3.4).

5.2 Evaluation results

No model exceeds 11.2% of the hard subset, even with three attempts. The highest pass@3 on the hard subset is 11.2%, with per-attempt accuracy of 6.0%; the same configuration reaches 53.4% pass@3 on the full corpus. The hard subset therefore excludes tasks that current models already solve reliably.

Success varies across attempts. Moving from one attempt to three adds between 0.0 and 12.2 points across the six models on the full task set, and between 11.1 and 12.2 points for the two highest-scoring (Figure 8). A benchmark reporting only single-attempt accuracy therefore measures reliability as much as capability, and the two should be reported separately.

The tiers are decided by five of the six models, and are circular by construction. One of the six models is at high complexity on all 238 tasks, solving none of them in any round, contributing a constant $+1$ to every task's stumped-model count, so tier assignment is in practice determined by the remaining five models and the easy tier is correspondingly near-saturated for the higher-scoring configurations. Because the tiers are derived from the same six models the table reports,

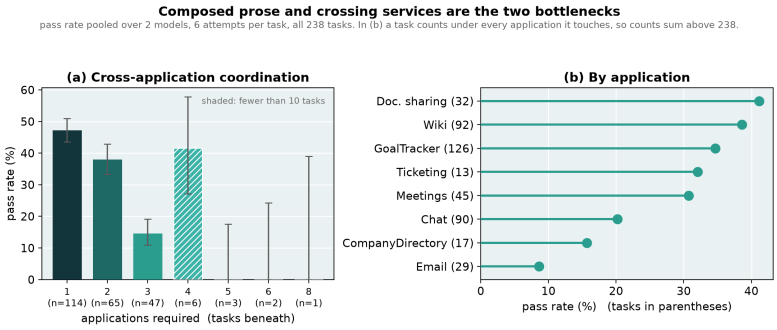


Figure 3: (a) Pass rate by the number of applications a task requires. Bars with diagonal shading rest on fewer than ten tasks, which is every level from four applications up; levels at zero are marked on the baseline. (b) Pass rate per application. A task counts under every application it touches, so the counts sum above 238 and the categories are not independent samples.

comparisons within a tier are well posed but the gradient across tiers is not independent evidence of difficulty.

5.3 Failure analysis

Free-text content produces the largest observed performance gap. On the full task set, read-only tasks, which require no mutations at all, but instead a returned answer from the model, pass at 23.8% mean pass@1 against 44.0% for read-write tasks at the highest-scoring configuration. The direction is not uniform: two of the six models invert it, at 19.0% versus 10.1% and 16.1% versus 5.1% mean pass@1 respectively. That the sign of the gap varies across models while its magnitude is large in both directions indicates that the gap is a property of the grading mode rather than of any single model, and it runs opposite to what mutation count alone predicts: a read-only task's verdict rests almost entirely on the judge rubric. Write-only tasks pass at 77.5% pooled and only 1 survives into the hard subset, so we report them separately but do not draw conclusions from that row alone.

The breakdown per web app is consistent (Figure 3b): email, whose artifacts are almost entirely written responses, is the hardest at 8.6%, and chat is 20.2%. Performance also declines with web app count (Figure 3a). Pass rate falls from 47.2% on single-web app tasks to 14.5% at three, and no task requiring five or more web apps is solved by any attempt.

The number of attempts materially changes the result: the gain from retries exceeds the difference between the two frontier models. Single-attempt accuracy and pass@k should therefore be reported separately. A benchmark that grades only structural state changes would also rank these models substantially higher because they fail most often when the task calls for a written response. Performance further declines with cross-app coordination: application count correlates with pass rate more strongly than either difficulty composite.

6 Conclusion

TAMER is a benchmark that evaluates how an agent works as a scoped employee inside one persistent organization, observing eight interconnected web apps just like an employee would. TAMER's task verification grades both the state changes resulting from the agent's action on the web and the content of the text it wrote to perform long-horizon project management related tasks, including penalizing unwarranted writes to apps a task didn't ask for.

Three conclusions follow from the evaluation of six models. First, grading state changes alone overstates what an agent can do. Read-only tasks, scored almost entirely on written content, are 29.3% of the hard tier against 20.8% of the easy one, and cross-app coordination compounds this effect. Second, retry budget matters as much as model choice: three attempts instead of one adds more than eleven points to the highest-scoring configurations, so single-attempt reliability and pass@k measure different things and belong in separate columns. Third, difficulty scoring should consider empirical evidence and cannot easily be assigned in advance, as we observed that two of our a priori difficulty composites predicted measured pass rate worse than the web app count alone. Finally, there is substantial room for saturation as no model exceeds 11.2% pass@3 on the 116 hard tasks.

REFERENCES

Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero-Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. HealthBench: Evaluating large language models towards improved human health. *arXiv preprint arXiv:2505.08775*, 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*, 2024. Datasets & Benchmarks Track; arXiv:2407.05291.

Santiago Castro. fast-krippendorff: Fast computation of krippendorff's alpha agreement measure, 2017. Software package.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. ARE: Scaling up agent environments and evaluations. *arXiv preprint arXiv:2509.17158*, 2025.

Divyansh 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 Ramesh Motwani. REAL: Benchmarking autonomous agents on deterministic simulations of real websites. In *Advances in Neural Information Processing Systems*, 2025. Datasets & Benchmarks Track; arXiv:2504.11543.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the Association for Computational Linguistics*, 2024. arXiv:2401.13919.

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.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the Association for Computational Linguistics*, 2024. arXiv:2401.13649.

Klaus Krippendorff. *Content Analysis: An Introduction to Its Methodology*. Sage, 2nd edition, 2004.

Yukyung Lee, Joonghoon Kim, Jaehee Kim, Hyowon Cho, Jaewook Kang, Pilsung Kang, and Najoung Kim. CheckEval: A reliable LLM-as-a-Judge framework for evaluating text generation using checklists. In *Proceedings of Empirical Methods in Natural Language Processing*, 2025. arXiv:2403.18771.

Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, Nir Mashkif, and Segev Shlomov. ST-WebAgentBench: A benchmark for evaluating safety and trustworthiness in web agents. In *International Conference on Learning Representations*, 2026. arXiv:2410.06703.

Xiangyi Li, Kyoung Whan Choe, Yimin Liu, Xiaokun Chen, Chujun Tao, Bingran You, Wenbo Chen, Zonglin Di, Jiankai Sun, Shenghan Zheng, Jiajun Bao, Yuanli Wang, Weixiang Yan, Yiyuan Li, and Han-chung Lee. ClawsBench: Evaluating capability and safety of LLM productivity agents in simulated workspaces. *arXiv preprint arXiv:2604.05172*, 2026.

Matthew Lombard, Jennifer Snyder-Duch, and Cheryl Campanella Bracken. Content analysis in mass communication: Assessment and reporting of intercoder reliability. *Human Communication Research*, 28(4):587–604, 2002.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. AgentRewardBench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*, 2025.

Shiva Krishna Reddy Malay, Shravan Nayak, Jishnu Sethumadhavan Nair, Sagar Davasam, Aman Tiwari, Sathwik Tejaswi Madhusudhan, Sridhar Krishna Nemala, Srinivas Sunkara, and Sai Rajeswar. EnterpriseOps-Gym: Environments and evaluations for stateful agentic planning and tool use in enterprise settings. *arXiv preprint arXiv:2603.13594*, 2026.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023.

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. *arXiv preprint arXiv:2506.01952*, 2025.

Justin D. Norman, Michael U. Rivera, and D. Alex Hughes. Reliability without validity: A systematic, large-scale evaluation of LLM-as-a-Judge models across agreement, consistency, and bias. *arXiv preprint arXiv:2606.19544*, 2026.

Hubert M. Pysklo, Artem Zhuravel, and Patrick D. Watson. Agent-Diff: Benchmarking LLM agents on enterprise API tasks via code execution with state-diff-based evaluation. *arXiv preprint arXiv:2602.11224*, 2026.

Delip Rao and Chris Callison-Burch. Agreement metrics for LLM-as-Judge evaluation: What to report and why. *arXiv preprint arXiv:2606.00093*, 2026.

ServiceNow Research. WebArena-Verified: A verified release of the WebArena benchmark. `https://github.com/ServiceNow/webarena-verified`, 2025. Presented at the NeurIPS 2025 SEA Workshop.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. In *Proceedings of the International Conference on Machine Learning*, volume 267 of *PMLR*, pp. 56843–56873, 2025. arXiv:2504.01848.

Olly Styles and Sam Miller. WorkBench revisited: Workplace agents two years on. *arXiv preprint arXiv:2606.13715v1*, 2026.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. WorkBench: A benchmark dataset for agents in a realistic workplace setting. In *Conference on Language Modeling*, 2024. arXiv:2405.00823.

Shulin Tian, Ziniu Zhang, Liangyu Chen, and Ziwei Liu. MMInA: Benchmarking multihop multimodal internet agents. In *Findings of the Association for Computational Linguistics*, 2025. arXiv:2404.09992.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. Best Resource Paper; arXiv:2407.18901.

Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. Replacing judges with juries: Evaluating LLM generations with a panel of diverse models. *arXiv preprint arXiv:2404.18796*, 2024.

Zhilin Wang, Jaehun Jung, Ximing Lu, Shizhe Diao, Ellie Evans, Jiaqi Zeng, Pavlo Molchanov, Yejin Choi, Jan Kautz, and Yi Dong. ProfBench: Multi-domain rubrics requiring professional knowledge to answer and judge. In *International Conference on Learning Representations*, 2026. arXiv:2510.18941.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024. Datasets & Benchmarks Track; arXiv:2404.07972.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *Advances in Neural Information Processing Systems*, 2025. Datasets & Benchmarks Track; arXiv:2412.14161.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. In *Conference on Language Modeling*, 2025. arXiv:2504.01382.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025. arXiv:2406.12045.

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. arXiv:2307.13854.

A LIMITATIONS

The judge's stability is measured between models. Scoring each rubric-execution pair with five judge models at six calls each gives Krippendorff's $\alpha = 0.801$ over 3,147 items, computed with Castro (2017) and clearing the conventional 0.80 threshold for reliable data (Krippendorff, 2004; Lombard et al., 2002). The five judges are Claude Sonnet 4.5, DeepSeek-V3.2, Kimi K2.5, GLM-5 and Qwen3-Next, spanning five disjoint model families. Cross-family spread on the same items is what the α figure is measuring. We report α rather than the $\sim$0.95 raw agreement it sits far below, because the binary ratings skew toward pass and percent agreement is inflated by that base rate (Norman et al., 2026; Rao & Callison-Burch, 2026). Reported verdicts are the majority vote ($\geq$3 of 5) of this five-judge panel drawn from disjoint families (Verga et al., 2024; Wang et al., 2026), so no single model family determines a verdict and none can favor a model that shares its family; what residual spread remains across judges leaves the model ranking judge-invariant. The deterministic layer is unaffected.

Test-time reasoning compute is not matched across models. Extended thinking was enabled only where the provider exposed it on Bedrock at evaluation time. The per-attempt gap between runs with and without this therefore confounds agentic capability with harness reasoning settings, and we do not separate them here. An ablation with extended thinking disabled/enabled equally would isolate the two and we leave this to immediate future work.

Scope. Both personas are technical program managers, the environments model one organization's shape of knowledge work, and two web apps are exercised by fewer tasks than the other six, so whether these failure modes generalize to other role archetypes is untested. The environments also contain incident runbooks and escalation threads, realistic for the work modeled but capable of tripping provider-side content filters: one run was terminated mid-trajectory on those grounds. Future evaluations should add employee roles and increase coverage of the two least represented web apps.

B WORLD CONSTRUCTION

The world is generated in four stages, each validated before the next consumes it. A structured specification enumerates people, teams, projects, and buildings under closed categorical vocabularies, so every downstream reference resolves to a declared entity. An event timeline lays out five months of activity as dated, attributed events bucketed by project $\times$ week (5,339 events in Wave 1, 8,492 in Wave 2). A generation stage writes the free-text artifacts each bucket implies — chat messages, status reports, meeting notes, tracker updates. One converter per environment then maps that content into environment schema records.

Validation checks run both during and post- data generation. At generation time, programmatic rules check content against specification values and a model-judged pass cross-checks free text, with failing buckets regenerated. A post-hoc suite then checks the assembled world for temporal ordering violations, day-of-week mismatches, undeclared names, out-of-office conflicts, and week-to-week numeric coherence, backed by model-judged checks for properties no closed-form rule expresses and by manual audit. Findings are triaged into repair-and-regenerate against accept-with-reason, the latter covering unpopulated schema fields, content plausible in a real organization, and content bearing on no task.

Tasks were authored across milestones that used several seed versions; consequently approximately half the task batch ran against one cohesive subset of the world state, and the remainder against another.

C DATA VALIDATION AND CONSISTENCY CHECKS

The data validation and consistency check workflow validates the synthetic world at two stages: in raw data form as generated by the LLM (raw data), and after it has been converted to its web-app schema conformant shape (web-app hydration data). We apply automated rule-based checks to ensure structural and relational validity, as well as manual inspection of the web-apps by language experts for semantic validation and realism.

The automated deterministic checks split into two categories; per web-app rules, or world-wide rules. They are written based on the rules derived from the data relations, which are defined by

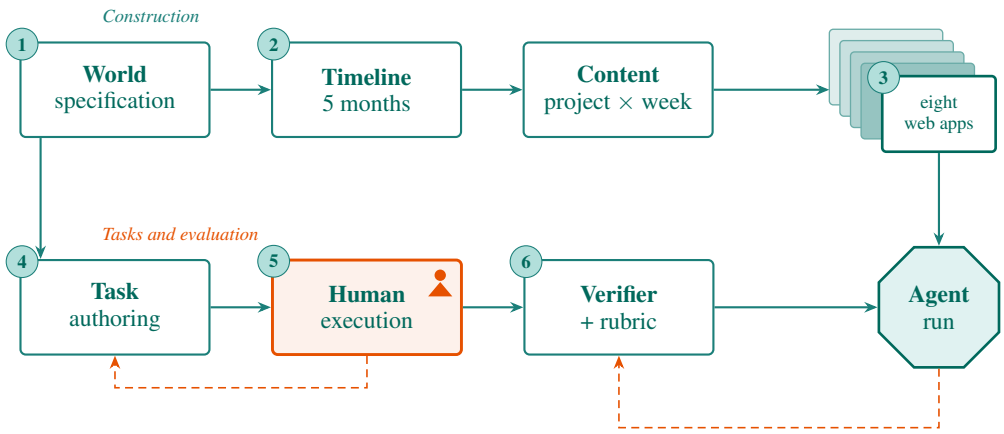


Figure 4: TAMER construction and evaluation. A typed world specification (1) fixes people, teams, projects, and reporting lines; a storyline generator expands each project over five months of dated events (2); an LLM materializes each project-week's content, which per-application converters map into the schema of eight applications (3). Each application is a real product frontend over an isolated, resettable backend, and emits a mutation log. Task authors write tasks against the materialized world with model assistance (4); human experts validate every task by executing every task to produce one correct reference execution and three deliberately incorrect executions (5); verifiers combine deterministic mutation checks with reviewed binary rubrics and must accept the correct execution while rejecting all three incorrect ones (6). Dashed arrows are the two repair paths: data annotators reject under-specified tasks, and grading exposes verifiers that admit an incorrect execution.

the world schemas in the case of world-wide rules, or web-app schemas in the case of per-web-app rules. Some examples are: a synthetic email that populates the email web-app cannot have its reply sent at a time that predates the email itself; a meeting invite must reserve a room that exists in the generated world (not a hallucinated room). A total of 186 such rules spanning the world facts and the eight web-apps are run on the generated content to either auto-repair or flag the findings. Language experts and engineers then audit the flagged findings and fix the errors. Throughout the data generation life cycle, the consistency check rules and scripts were updated iteratively based on such findings for improved rule coverage.

Additionally, web-apps hydrated with the generated data are inspected thoroughly by language experts to identify and reconcile semantic violations, inaccuracies, or unrealistic content both within a given web-app and across web-apps. Some examples are: LLM generated text assigning the same calendar date to "today" and "tomorrow"; an email authored by a person incorrectly using another person's name at sign-off in the email body; a person declaring out-of-office dates in a chat message that contradicts with out-of-office dates in another chat message. A set of rules is provided in Table 3.

D TASK AUTHORING AND ANNOTATION

Tasks are anchored on events drawn from an index over the generated world, filtered by the capability the task should exercise and preferring events spanning three or more environments. Every fact the task depends on is traced from the specification into the actual generated records and recorded with its source quote; a fact absent from a candidate environment disqualifies that environment as a target. Prompts state what is needed without prescribing where to find it, and are checked against enumerated leakage patterns: naming the location, the person and date of the answer, the value to be written, or the channel to search. Authors additionally enumerate the alternative valid solution paths, which is what exposes an over-constrained verifier, and three to five near-miss answers that must fail, each paired with the check expected to reject it.

Data annotators then execute each task in the live environments to produce the ground-truth mutation set and, where required, the reference response; a second data annotator independently checks the execution. Tasks satisfiable in several materially different ways are retained as difficult rather than discarded. Most of the authoring cost falls on producing and checking the reference executions, which is what bounds corpus size.

Table 3: Consistency rules applied during world construction and validation, by target.

Target	Consistency rules applied
Global across all web apps	
Alias references	Every `@alias` mention, structured identity field (meeting attendees, email recipient/cc/bcc alias fields), and internal `alias@crestline.com` email must resolve to a known user.
Names	Scans all prose for hallucinated or mismatched person names—a known surname with a wrong first name, a known first name with a wrong surname, or a first/last pair belonging to two different people. Canonical names come from the world roster.
Day-of-week consistency	Reconciles prose weekday claims against the actual weekday of the referenced date.
Timestamps	Detects malformed timestamps.
Temporal causality	Enforces "parent predates child" across records: thread replies after their root, reactions after messages, comments after tickets, and versions/updates after creation, at both raw and hydration stages.
Cross-app identity	Asserts that each web app's emitted person facts (employee ID, department, manager, email, timezone, etc.) equal a single source of truth.
Web app specific	
Chat	Timeline range, thread-reference integrity, channel-membership rules (public/team/leadership rosters), DM validity and de-duplication, and channel type-/visibility.
Document sharing	Data-structure validity (access-list personas and levels, author ownership, version history, folder routing and scoping, duplicate titles); comment rules (quoted text in a comment matches the document body verbatim); timeline range.
Email	Timeline range, reply-reference integrity, self-emails flagged, missing subjects, folder-ID format, and recipient / in-reply-to shape normalization.
Goal tracking	Status-enum and status-trajectory alignment; completion invariants for goals and milestones; milestone-reference validity; goal-ID collisions; and bidirectional linkage of key insights and cross-goal dependencies.
Meetings	Per-entity required-field and enum checks plus $\sim$15 cross-entity FK edges (organizer, attendees, rooms, Zoom, buildings, OOO events, preferences); business rules for time ordering, room double-booking, OOO violations, organizer-not-in-attendees, action-timestamp cutoffs, and the single-active-user invariant.
Employee directory	Data-structure validity (e.g., manager alias resolves correctly); and a symmetric-roster audit ensuring the directory and world bible cover the same people with a uniform email domain and login-matching local parts.
Ticketing	Per-entity required / enum / uniqueness checks; $\sim$20 cross-entity FK edges; business rules (single active user, thread comment-count parity, sticky-thread refs, date ordering, closed-ticket invariants, severity/impact match); Kerberos / email-alias identity resolution; and free-text name/mention/email scanning.
Wiki	Author resolution to real identities; dates within the simulated world timeline; tag-to-project-codename validity; project coverage; page name (Report 1, Report 2) vs. creation date follows logical ordering; and chat-channel references in content resolve to a valid channel in the chat app.

E VERIFIER CONSTRUCTION

Which layer grades a value is fixed by the field it lands in. Non-free-text fields are graded deterministically: schema enums (`entity`, `operation`, status, severity), identifiers obtained by navigation (`channelId`, `itemId`, `threadId`, short ids), interface-normalized alias lists (`recipients`, `cc`, `bcc`), system-injected identity fields, structured tree-path identifiers (wiki `fullName`, page and item `name`), and any length-derived scalar. Free-text fields are graded only by rubric: message `content`, email `body`, wiki page `content`, ticket comment `message`, directory `passionStatement`, and authored issue `resolution`. Email subject, page title, and

item name are short and structurally bounded but remain the agent's own wording, so they are judged rather than matched.

Deterministic verifiers are layered, with mutation count, identity, structured-field shape, numeric bounds, etc., and filter side-effect entities before any comparison: read receipts, sessions, view records, reaction counts, and per-entity version tables, all of which the interface emits on navigation. Regular expressions over free-text fields are prohibited, the sole exception being a directive to enter a string verbatim.

Rubrics are written by task authors alongside the task and its reference execution produced by a data annotator, and revised over multiple review passes before the task enters the corpus. Instance-specific rubrics authored by the experts who define the task are the standard construction where correctness cannot be pattern-matched. Rubrics are binary, one fact per criterion, with both levels specified precisely enough that a careful reader reaches the panel's verdict as binary decomposition rather than holistic scoring is what raises agreement between independent judges and lowers score variance (Lee et al., 2025).

Negative facts are expressed as criteria whose presence level asserts absence, which is what replaced pattern-based exclusion checks. One criterion in every rubric checks that the response invents nothing. It passes when every factual claim (people, dates, numeric values, project names, identifiers, statuses, decisions, ownership, timelines) is consistent with the reference or is the agent's own contextually appropriate phrasing, and fails on a factual contradiction, a wrong identifier, a claim that a forthcoming event has occurred, or an entity that does not exist. It does not penalize paraphrase, salutations, filler, reordering, or formatting choice, and the reference answer must enumerate the legitimate details correct responses contain so that breadth is not read as invention.

Rubric falsification test. Authoring a rubric and reviewing it are the same people's work, so neither establishes that the rubric discriminates. What establishes it is that the rubric is tested against executions its author did not produce. The data annotators of Appendix D generate, per task, the canonical correct execution and three deliberately incorrect ones — a required action removed, an extra unrequested action, and the same actions with a wrong value, each submission independently reviewed by a second data annotator. The verifier must accept the correct execution and reject all three incorrect ones. Because an over-permissive criterion fails on the incorrect executions and an over-strict one fails on the correct execution, the gate has two independent ways to refute a rubric, which an author rereading their own criteria does not. Failing the gate sends the verifier back for revision and re-run; where the fault traces to prompt ambiguity rather than verifier logic, the task returns to authoring instead, so the gate falsifies tasks as well as rubrics.

The incorrect executions fall in two classes, both required since either alone leaves one layer untested. Text-level cases preserve the action sequence and alter the written response, exercising the rubric. Action-level cases change what the agent does (wrong channel, wrong target, thread instead of top level, omitted write, uninvolved environment mutated), exercising the deterministic layer. A verifier enters the corpus once it accepts the correct execution and rejects every incorrect one.

This test evaluates the rubric, not the judge that applies it. Judge behavior is a separate question, and §A reports inter-judge agreement for it. Evaluating the judge apart from the rubrics it applies follows Starace et al. (2025), who build a distinct judge-evaluation benchmark for that purpose.

F Environments and harness

Each environment is a single-page web app over an in-browser relational database, mutable only through the interface. Every environment exposes a `/finish` endpoint returning ordered records of the form {`entity`, `operation`, `oldValue`, `newValue`, `primaryKey`} with `operation` in {`CREATE`, `UPDATE`, `DELETE`}; this uniform shape is what permits one verifier interface across eight heterogeneous environments. The clock is frozen so relative-time reasoning is deterministic, and each environment auto-authenticates the persona on load except for the email app where the agent logs in using a human takeover tool.

Before the first agent turn, for each provisioned environment the harness injects authentication cookies, completes the edge authentication handshake, navigates and clears the access gate, waits for

Table 4: TAMER web apps. Each is a self-contained web app over an in-browser database, exposing a `/finish` endpoint that emits a structured mutation log.

Web app	Actions (approx.)	Records (schema tables)
Chat	send, edit, delete messages; DMs; threads; reactions; scheduled messages; profile edit	channels, messages, users, reactions, savedMessages, scheduledMessages
Email	send, reply, forward; folders; drafts; bulk verbs; distribution-list expansion	profiles, emails, email_status, folders, groups_dl, email_opened, search_history
Meetings	schedule, update, cancel; out-of-office events; polls; room check-in; conference linking; livestream intake	meetings, meetingVersions, oooEvents, polls, pollResponses, roomCheckIns, zoomMeetings, livestreamEvents (plus 6 reference tables)
Ticketing	create, update, resolve; comments; assignments; label and priority edits; engagement	issues, comments, threads, users, groups, resolverGroups, engagements (plus 10 reference tables)
GoalTracker	create, update items (Goal / Project); milestones; metrics; notes; status; views	items, notes, milestones, metrics, views, tagLiterals
Wiki	create, edit, delete pages; folders; permissions; drafts; classifiers	pages, folders, permissions, drafts, classifiers
CompanyDirectory	update passion statement; work bio; award nominations; bookmarks; organizational navigation	employees, awards, awardNominations, shoutouts, communities, bookmarks, bookmarkGroups, searchHistory
Document sharing	create, edit documents; folder placement; section edits	documents, folders

Table 5: Hydrated record volume per web app for the merged world. Counts are approximate where a wave-level total is reported rather than a per-file count; exact manifests ship with the release.

Environment	Records
Chat	$\sim$8,200 messages across team channels, DMs, and threads
Wiki	$\sim$230 pages
Email	$\sim$830 messages
GoalTracker	57 items (10 Goals, 38 Milestones, 9 Key Insights), 235 version records, 170 status-update notes
Meetings	$\sim$130 calendar records plus out-of-office events, polls, and room bookings
Document sharing	$\sim$160 documents across folders
Ticketing	602 issues, 1,209 comments, 600 threads, 24 resolver groups, 19 categories, 27 users.
CompanyDirectory	54 employee records with awards, shoutouts, and communities

Table 6: Per-environment task counts and pass rate, pooled over the two frontier models (Claude Sonnet 5 and Claude Opus 5) at three rounds each, six attempts per task, on the 238 tasks. A task contributes to every environment it reads from or writes to, so counts sum above 238. Pass rate differs across web apps by a factor of 4.8. Email is the hardest at 8.6%, and the employee directory is second at 15.7% despite a structured schema.

Web app	Tasks	Pass rate
Doc. sharing	32	41.1%
Wiki	92	38.6%
GoalTracker	126	34.7%
Ticketing	13	32.1%
Meetings	45	30.7%
Chat	90	20.2%
CompanyDirectory	17	15.7%
Email	29	8.6%

Table 7: Per-environment coverage over the 238 authored tasks. *Read* counts tasks whose correct execution requires the agent to inspect at least one record in that environment (162 tasks have a read component); *write* counts tasks that require at least one mutation there (124 tasks have a write component). A task appears in both the read and write columns when it reads one application and writes another, so the two columns overlap.

Environment	Read tasks	Write tasks	Provisioned and required
GoalTracker	85	75	126
Chat	85	19	90
Wiki	74	35	92
Meetings	36	29	45
Doc. sharing	30	18	32
Email	18	18	29
CompanyDirectory	16	1	17
Ticketing	9	7	13
Tasks with this component	218	182	238

Example Execution Snippet — Email gym

Prompt: *Lucia finished the security review for the payment-processor migration and flagged some compliance items that need to be addressed before we can proceed. Can you check my messages with Lucia and the #meridian channel for the details, and send Vivian an email summarizing the findings and what it means for the migration timeline? CC Naomi and Lucia.*

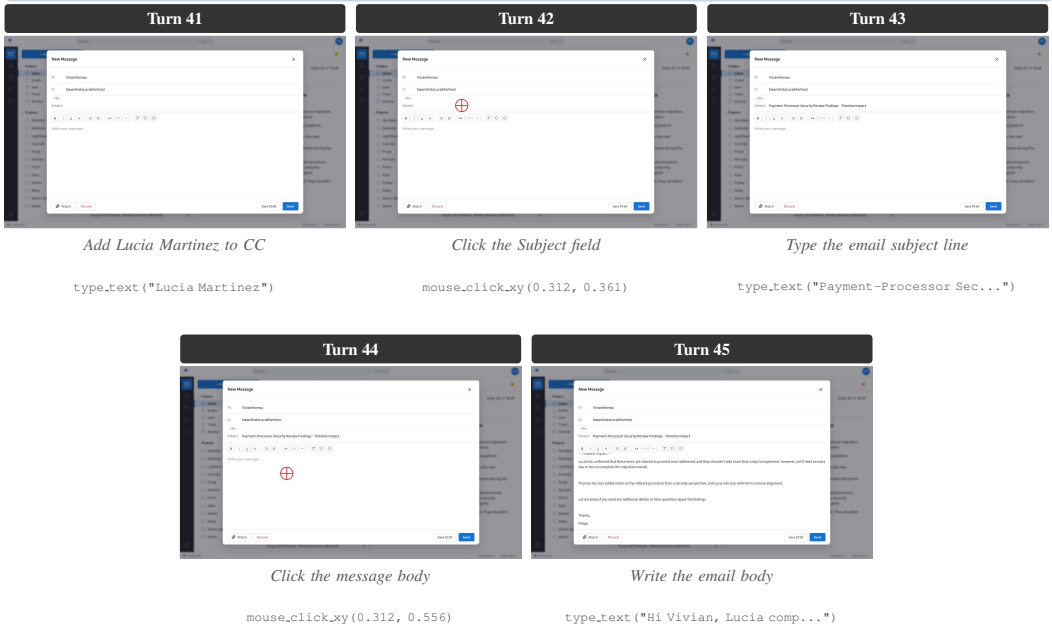


Figure 5: Pictured above are turns 41 - 45 of a Kimi 2.5 attempt at executing the prompt. The agent sees the exact screenshots pictured, save for the red crosshair marks added to clarify the location of the agent's click coordinates, and responds with the tool calls shown for its next turn.

client-side database initialization, resets to a clean baseline, and waits for reinitialization. On completion, `/finish` is polled on every provisioned environment; required environments are graded by their verifier and uninvolved environments by one asserting zero meaningful mutations.

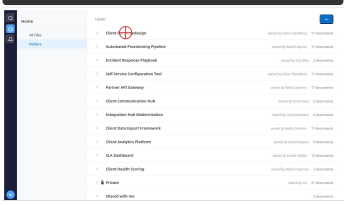
G How every reported quantity is computed

This appendix gives the exact definition of each quantity appearing in a figure or table, so that any of them can be recomputed from the released per-task verdicts.

Example Execution Snippet — Collaborative Documents gym

Prompt: *We need meeting notes from last week's Client Portal Redesign sync. The meeting notes subfolder for that project is empty – create a new meeting notes document there. Title it with the project name and 'Sync Notes'. In the content, include an Attendees section, a Discussion Points section, an Action Items section, and a Decisions section. Leave them as templates for now.*

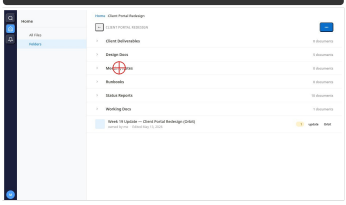
Turn 10



Open the Client Portal Redesign folder

`mouse_click_xy(0.361, 0.128)`

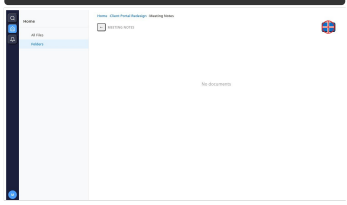
Turn 11



Open the Meeting Notes subfolder

`mouse_click_xy(0.342, 0.311)`

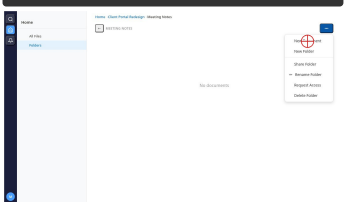
Turn 12



Empty folder: click New to add a document

`mouse_click_xy(0.948, 0.098)`

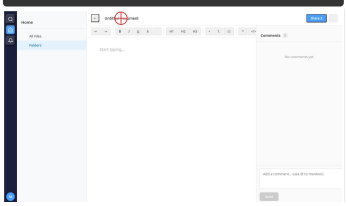
Turn 13



Choose New Document from the menu

`mouse_click_xy(0.893, 0.167)`

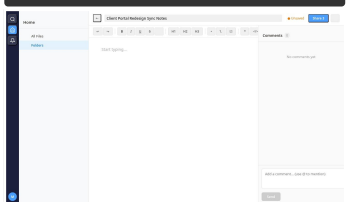
Turn 14



Select the document title field

`mouse_click_xy(0.348, 0.047)`

Turn 15



Type the meeting-notes title

`type_text("Client Portal Redesign...")`

Figure 6: Turns 10 - 15 of a Kimi 2.5 attempt at executing the prompt. The agent sees the exact screenshots pictured, save for the red crosshair marks added to clarify the location of the agent's click coordinates, and responds with the tool calls shown for its next turn.

Table 8: Harness configuration. Values are those used for the runs reported in §5.2.

Category	Setting	Value
Browser	MCP server	`@playwright/mcp`, pinned
	Mode	headless, isolated profile
	Viewport	1280×720
	Screenshot format	JPEG
Limits	Max turns per task	200
	Per-turn timeout	120 s
	Per-task timeout	6,000 s
	Tool call timeout	30 s
	Retained screenshots	50 most recent
	Retries on error	1
Decoding	Temperature	1.0
	Max output tokens	16,000
	Extended thinking	enabled (Claude arms only)
	Other sampling parameters	provider defaults
Judge	Model	Five-model majority panel
	Temperature	0.0
	Max output tokens	1,024
	Timeout	60 s
	Calls per rubric	5 (majority $\geq$3)

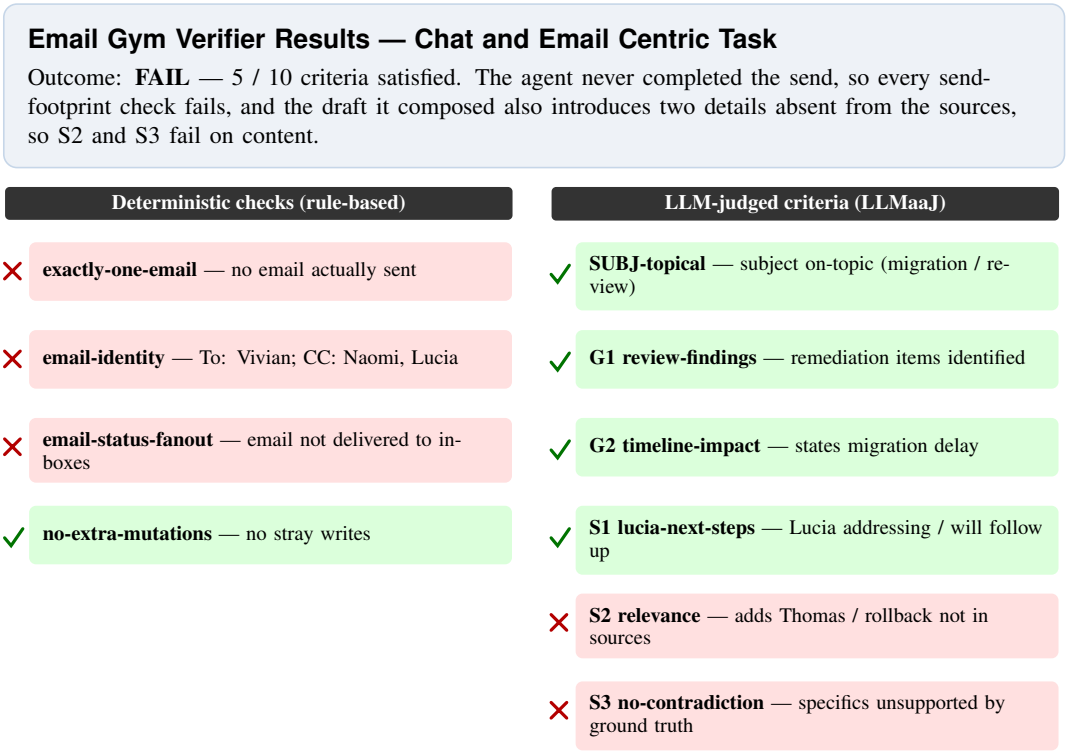


Figure 7: Per-criterion verifier outcome for the cross-gym flow of Figure 2 (Kimi 2.5). Left column: deterministic (rule-based) checks on the gym mutations; right column: LLM-judged (LLMaaJ) content criteria. The deterministic send-footprint checks all fail because the agent never completed the send; the content criteria are scored on the composed draft.

Per-attempt accuracy. For a set of tasks T and a model with R attempts, per-attempt accuracy is the pooled proportion

$$\text{acc} \;=\; \frac{\sum_{t\in T}\sum_{r=1}^{R} \mathbb{1}[t \text{ passed in attempt } r]}{|T|\cdot R},$$

i.e. the fraction of all task–round cells that passed. This is the quantity plotted at $k = 1$ in Figure 8 and reported in the *per-attempt* columns of Table 2.

pass@k. In cases where a model has $r > k$ attempts we do not take the union over the first k attempts, because that discards the remaining attempts and depends on their arbitrary ordering. Instead, for a task with p passes out of n observed attempts, the probability that a uniformly random k-subset of those attempts contains at least one pass is

$$\text{pass@}k(t) \;=\; 1 - \frac{\binom{n-p}{k}}{\binom{n}{k}}, \qquad \text{pass@}k \;=\; \frac{1}{|T|}\sum_{t\in T}\text{pass@}k(t),$$

with $\text{pass@}k(t) = 1$ whenever $n - p < k$. Tasks with $n < k$ are excluded from that k. This is the standard unbiased estimator and is what Figure 8 plots for every k. Note: In this paper every model has $r = 3$ observed rounds and we report $k = 3$.

Error bars. All intervals are 95% Wilson score intervals on the pooled proportion. For $\hat{p} = x/n$ and $z = 1.96$,

$$\frac{\hat{p} + \frac{z^2}{2n} \;\pm\; z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}.$$

Wilson rather than the normal approximation because several cells sit at or near zero, where the normal interval leaves the unit range. Note that attempts on the same task are not independent of one another, so these intervals describe attempt-level sampling only and understate task-level

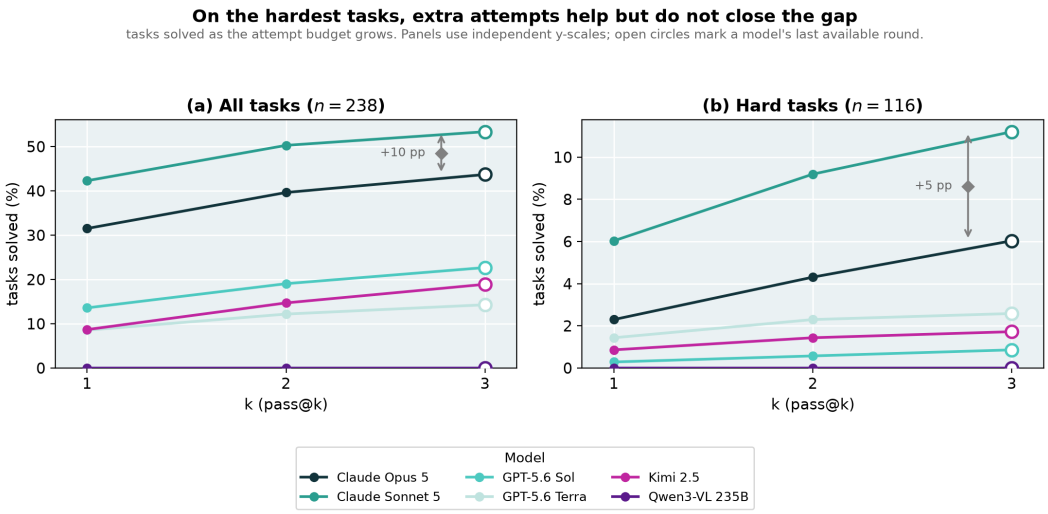


Figure 8: Tasks solved as the attempt budget grows, discussed in §5.2. (a) all 238 tasks; (b) the 116 hard tasks, the tier defined in Table 2. Computed without replacement over every observed round rather than as a union over the first k, so a curve is the expected solve rate at a budget of k and not the outcome of one arbitrary ordering of rounds. Every model has three rounds in this release, so every curve spans the same budget and no endpoint is truncated for want of data. Panels use independent y-scales.

uncertainty for small cells. The headline interval should therefore be read as attempt-level (a task-clustered interval would be wider).

Pooled pass rate, and which models are pooled. Wherever a table or figure reports a single pass rate for a cell rather than one column per model (Table 6, Figure 3, and the pooled figures quoted in §5.3), it is the per-attempt accuracy over the two frontier models, Claude Sonnet 5 and Claude Opus 5, at three rounds each: six attempts per task.

We pool the frontier pair rather than all six evaluated models because these cells are levels a reader interprets against the headline accuracy. Two of the six models pass almost nothing so averaging them in drags every cell toward zero without changing the ordering. The six-model average is a statement about how many models can do anything at all in an application, which is not what these cells are for.

Difficulty quintiles. For each composite, tasks are sorted by score and cut into five equal-count bins ($n \approx 47$); the plotted value is the pooled frontier pass rate of the tasks in each bin. The web app count is *not* binned this way: it is integer-valued with 114 tasks tied at one, so quintile boundaries would split a tied stratum and the resulting bins would differ only by sort order. It appears in Figure 9b as a per-task correlation instead.

Rank correlations. Spearman ρ between a per-task difficulty score and that task's pooled frontier pass rate over all 238 authored tasks, with midranks for ties.

H Difficulty composites

Every task carries two difficulty scores computed before any agent runs. D_{struct} is an information-theoretic index over six axes: each is a bit count under a maximum-entropy model over one task-structure primitive, so an axis equals the log of the number of alternatives the agent must resolve. D_{count} is a weighted, min-max normalized sum of six count-typed dimensions, among them constraint density, verifier criteria, and application cardinality. Axis definitions and weights are in Appendix H.

Whether the structural composites anticipate the empirical tiers is a separate question, and one the two constructions are well placed to answer; we report the association between D_{struct}, D_{count} and tier assignment in Appendix H.

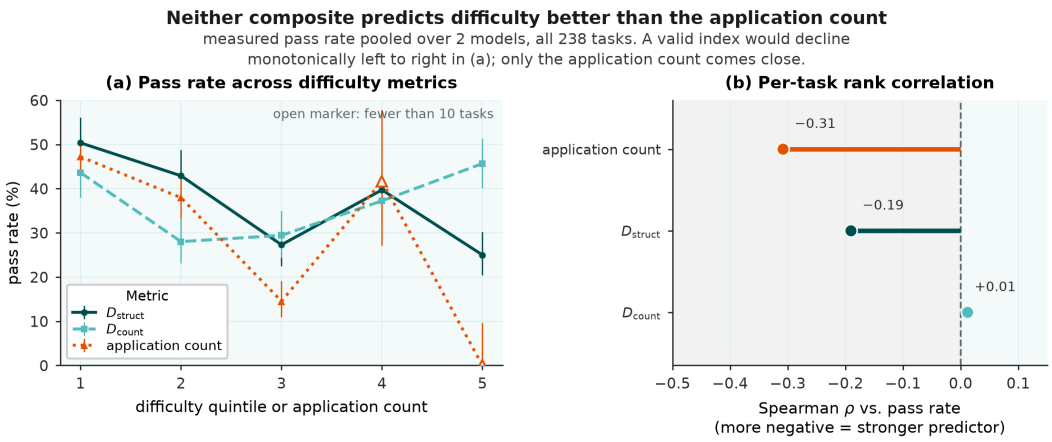


Figure 9: Predictive ability of the a priori difficulty metrics, discussed in §H.1. (a) Pass rate as each metric increases. The two composites are continuous and are cut into quintiles, $n{\approx}47$ per point. The application count is integer-valued with 114 tasks tied at one, so it cannot be cut into quintiles and is plotted on its own scale instead: 1, 2, 3, 4, and 5 or more. The three series therefore share an x axis meaning "one step harder by this metric" rather than a common unit. Open triangles mark application-count bins holding fewer than ten tasks. A valid metric would decline monotonically left to right; only the application count comes close, and it reverses at four applications on six tasks. (b) Per-task rank correlation against pass rate.

Both composites are computed from the task specification, its verifier, and the world's structure, with no reference to observed performance.

H.1 Difficulty composite validity

Neither a priori composite predicts measured difficulty as well as the web app count alone. Against the pooled frontier pass rate, D_{struct} reaches Spearman $\rho = -0.19$ and D_{count} $\rho = +0.01$, while the application count, a single integer readable off the task definition, reaches -0.31. The ordering is unchanged when the measured rate is pooled over all six models instead (-0.23, -0.02, -0.41 respectively), so it is not an artefact of how finely that rate is resolved. Within single-application tasks alone D_{struct} vanishes to $+0.02$, so what signal it has is the application-count signal entering through axes whose construction sums over applications. Neither composite declines monotonically across its own quintiles either (Figure 9a): D_{count}'s predicted-hardest quintile passes at 45.7%, above its second at 28.0%, where a valid difficulty index would fall across every step. The application count is the only one of the three that declines over its first three steps, and it too reverses at four applications (on six tasks). Grouped by measured tier, D_{struct} rises monotonically (10.65, 16.67, 20.28 from easy to hard) as does the application count (1.17, 1.63, 2.21), while D_{count} does not (0.471, 0.517, 0.511), inverting between the medium and hard tiers. The composites therefore separate the tiers in aggregate more cleanly than their per-task rank correlations suggest, and D_{count} fails on both measures.

H.2 D_{struct}: information-theoretic axes

$D_{\text{struct}} = \sum_j H_j$, each axis a bit count under a maximum-entropy model over one task-structure primitive: an axis is the log of the number of alternatives the agent must resolve.

- H_{LOC} (localization) $= \sum_g \alpha_g \log_2 \binom{M_g^{\text{cond}}}{s_g}$. For each application g the task touches, M_g^{cond} is the number of candidate items still plausible after reading the prompt and s_g is the number the agent must actually open. The binomial counts the distinct subsets of that size, so its log is the number of bits needed to single out the right one, and α_g is a fixed per-application coefficient derived from typical visible list length. The term is zero when only one candidate remains. Where the verifier names definite target identifiers, M_g^{cond} collapses to their count, an operation that only ever tightens the candidate set. This axis measures search cost: how much of the haystack has to be discriminated to find the needles.

- H_{VOL} (volume) $= H^{\text{page}} + H^{\text{prose}}$. The page term sums, over applications, a per-application bit cost computed from the signal and distractor counts in the set of pages the agent encounters, weighted by the same α_g. The prose term sums, over each fact the agent must extract from running text, bits from that fact's own signal-to-distractor ratio. Where H_{LOC} charges for locating material, H_{VOL} charges for reading and filtering it once found.

- H_{CCL} (cross-context linking) $= \sum_f n_f^{\text{off}}$. For each fact the task requires, n_f^{off} counts the boundaries the fact must be carried across while its source is off screen, at one bit per crossing. This measures working-memory load. It is a description length under an independent-slot retention model rather than a Shannon entropy, since it is not a log over alternatives, and we distinguish it from the other axes on that basis.

- H_{BR} (branching) $= \sum_h \log_2 b_h$. At each macro-hop h along the trajectory the agent chooses among b_h continuations, contributing $\log_2 b_h$ bits. Only navigation and synthesis hops count: value-write hops are excluded, and any hop whose uncertainty is already attributed to H_{SC} or H_{CCL} is carved out to zero. This measures decision branching that is not already charged elsewhere.

- H_{SC} (specification completeness) $= H^{\text{prompt}} + H^{\text{disc}} + H^{\text{evid}}$. Each hedged or ambiguous prompt reference admits k readings and contributes $\log_2 k$; references resolving to the same enumerated set are counted once. Application disclosure contributes 0 when all N required applications are named, $N - m$ when m are, and $\log_2(2^N - 1)$ when none are, which is the cost of selecting a non-empty subset of N. Evidence underspecification adds $\log_2 k$ wherever the prompt does not pin down which source to use, deduplicated by fragment and ambiguity type. This measures how much of the task specification the agent has to supply for itself.

- H_{EC} (error correction) $= \sum_b \log_2 K_b$. For each detected coherence break in the source material, K_b is the number of competing readings the agent must choose between. This measures the cost of reconciling inconsistency rather than of finding or reading.

Axes are ordered rather than independent: H_{SC} is evaluated first and has first claim on ambiguity bits, and downstream contributions are zeroed where the uncertainty is already attributed upstream, so the sum does not double-count.

Table 9: D_{count} dimensions, scales, and normalized weights. Each dimension is a concrete count, min-max normalized against its scale before weighting.

Dimension	Counts	Scale	Weight
Constraint density	Simultaneous constraints on the write step(s)	3–12	0.235
Verifier criteria	Grading criteria across all verifiers	6–25	0.176
Form complexity	Discrete interface steps in correct execution	4–16	0.176
Web app cardinality	Web apps the agent must navigate	1–4	0.176
Information sources	Distinct read locations visited before acting	1–3	0.118
Entity load	Distinct named entities tracked	1–7	0.118

Table 10: The six task classes. *Applies when* is the positive test an annotator applies; *Excluded when* is the boundary against the nearest class. The IE boundary is the one that matters most for reading Table 11: nearly every task in the corpus requires locating something, so counting that as extraction would make the class almost universal and therefore uninformative. IE is reserved for tasks whose deliverable *is* the located fact.

Code	Class	Applies when	Excluded when
IE	Info. extraction	A bounded fact is located and *returned* as the deliverable; correctness is a lookup.	Not IE if the fact is an ingredient for an action (ACT), a write-up (SUM) or a judgement (RISK), or if nothing is returned.
SUM	Summarization	Several sources are combined into prose; the rubric grades coverage and accuracy of the synthesis.	Not SUM if a single fact answers it (IE), or if no prose is graded (ACT).
RISK	Risk detection	Something the prompt does not state must be noticed: staleness, a passed deadline, a contradiction, a slipping metric.	Not RISK if the prompt names the problem and the agent only acts on it (ACT), or if a red status is merely retrieved (IE).
COMM	Stakeholder comms	An artefact addressed to a person or channel is produced; correctness includes who receives it.	Not COMM if nobody is addressed, e.g. a wiki page or a status field (ACT, plus SUM).
ACT	Action exec.	A state change is required and graded on structured fields: status enums, assignments, dates, membership, scheduling.	Not ACT if the mutation is only the vehicle for a written artefact (COMM), or if the correct execution changes nothing (NOOP).
NOOP	Correct inaction	The correct execution changes nothing, or the premise is false and the agent must decline, defer or report that.	Not NOOP if the zero-mutation assertion covers an application the task is not about; that is true of most tasks.

I TASK-TYPE TAXONOMY AND HOW THE CATEGORY COUNTS ARE COMPUTED

Every task carries every class that applies to it, so the classes are not a partition and the counts in Table 11 do not sum to 238. The classes and their boundaries are given in Table 10.

Annotation procedure. Two independent annotator passes labelled each of the delivered tasks, reading the task prompt, the web apps it provisions, its verifier source and its rubric criteria. The two passes use different models so that a single model's idiosyncratic reading does not become the label. A class counts for a task only where *both* passes mark it applicable; the reported count is therefore an intersection, not a union, and is a lower bound on how many tasks exercise the capability. Each task also carries one *primary* class, the one its verifier spends the most criteria on, used only where a partition is required.

Pass rates by category. Table 12 reports per-attempt accuracy and pass@3 within each class for the two frontier models. Because the classes overlap, a task contributes to every class it carries and the rows are not a partition; the rows therefore cannot be summed or averaged into a total task figure.

Table 11: Task categories by capability. An annotator does two passes to label every task over its prompt, provisioned applications, verifier source and rubric criteria, and a class counts only where both passes agree; see Appendix I. Labels are multi-label: the classes are not a partition and the counts do not sum to 238. We have not included the NOOP category as there were too few tasks currently.

Code	Category	Tasks	% of 238
ACT	Action execution	168	70.6
IE	Information extraction	38	16.0
RISK	Risk detection	80	33.6
SUM	Summarization	64	26.9
COMM	Stakeholder communication	45	18.9

Table 12: Pass rate (%) by task category for two of the frontier models, over the 238 tasks with three rounds each. Per-attempt is the mean pass@1; pass@3 is the unbiased estimator of §G, which at $k = n = 3$ coincides with the union over rounds. Categories are multi-label (Table 11), so a task contributes to every category it exercises and the rows are not a partition. †Fewer than ten tasks.

			Per-attempt		pass@3	
Code	Category	n	S5	O5	S5	O5
ACT	Action execution	168	46.4	35.5	57.1	49.4
IE	Information extraction	38	26.3	15.8	42.1	26.3
RISK	Risk detection	80	26.2	17.9	38.8	30.0
SUM	Summarization	64	30.7	20.3	37.5	25.0
COMM	Stakeholder communication	45	34.1	23.7	46.7	42.2

J AI USE STATEMENT

In this work we have used generative AI to generate synthetic data, as a coding agent, to create manuscript outlines, to format tables and figures, to write analyses scripts, to proofread for typos and grammar, to check acronyms and terms for consistent use and definition. We have used AI generated text for early drafts of the methods and results sections. We have fully reviewed and edited all text and figures manually for the submission. No language model is an author.

K REPRODUCIBILITY STATEMENT

We plan to release the code and data. We will make a portion of the data available publicly and the rest we will distribute upon reasonable request when allowed.