CLAWFORGE: GENERATING EXECUTABLE INTERACTIVE BENCHMARKS FOR COMMAND-LINE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Interactive agent benchmarks face a tension between scalable construction and realistic workflow evaluation. Hand-authored tasks are expensive to extend and revise, while static prompts and clean-state environments miss failures that arise only when agents operate over persistent, partially completed workflows. Existing interactive benchmarks have advanced agent evaluation significantly, but do not systematically test whether agents preserve valid artifacts, repair stale state, replace incorrect objects, or reconcile conflicting evidence before acting. We present **ClawForge**, a generator-backed benchmark framework for executable command-line workflows under state conflict. The framework compiles scenario templates, grounded slots, initialized state, reference trajectories, and validators into reproducible task specifications, then evaluates agents step by step using normalized end state and observable side effects rather than exact trajectory matching. We instantiate the framework as ClawForge-Bench, a 362-task snapshot spanning 17 scenarios and 6 ability categories. Results across 11 model endpoints show that strict full-pass accuracy ranges from 35.9% to 61.6%, Wrong-State Replacement ranges from 0.0% to 52.2%, and Interrupted Workflow Resume ranges from 16.7% to 100.0%, revealing strongly non-nested capability profiles. Partial-credit and step-efficiency analyses further show that many failures are near-miss closures rather than early breakdowns, and that models exhibit qualitatively different failure styles under state conflict. Code and Data: `https://anonymous.4open.science/r/ClawForge`.

1 INTRODUCTION

As Large Language Model (LLM) agents (Yao et al., 2022; Shinn et al., 2023) move from one-shot prompting to persistent software workflows, benchmark construction becomes a bottleneck in its own right. A realistic workflow task is not just an instruction paired with an answer: it requires initialized state, valid and invalid pre-existing artifacts, expected side effects, and executable validation logic. These ingredients must also be reproducible and auditable, because small changes in state can turn an apparently equivalent task into a different decision problem. Hand-authored interactive tasks are therefore expensive to scale and difficult to revise once coverage or fairness issues emerge (Zheng et al., 2023; Chang et al., 2024).

This problem is especially acute for command-line agents (OpenClaw, 2026; Merrill et al., 2026; Research, 2026). Real workflows require agents to inspect task boards, inbox threads, calendars, files, runtime configuration, weather state, and messaging surfaces before deciding whether to add, preserve, repair, or replace state. Many failures only appear after execution: an agent may create a duplicate task, leave stale state unresolved, choose the wrong branch under conflicting evidence, or stop before the final side effect is produced. Existing interactive benchmarks have advanced agent evaluation significantly (Qin et al., 2023; Liu et al., 2023; Zhou et al., 2023), but most initialize tasks from clean state and do not systematically test how agents handle pre-existing partial, stale, or conflicting artifacts (Ye et al., 2026; Ji et al., 2026; Li et al., 2026a; Meng et al., 2026a;b). This means a central part of workflow competence remains unmeasured: correctness that depends on the final state of a live environment, not only on a plausible textual answer or command sequence.

To address these challenges, we present **ClawForge**, a generator-backed benchmark framework for executable state-conflict workflows. The generator compiles scenario templates, grounded slots,

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

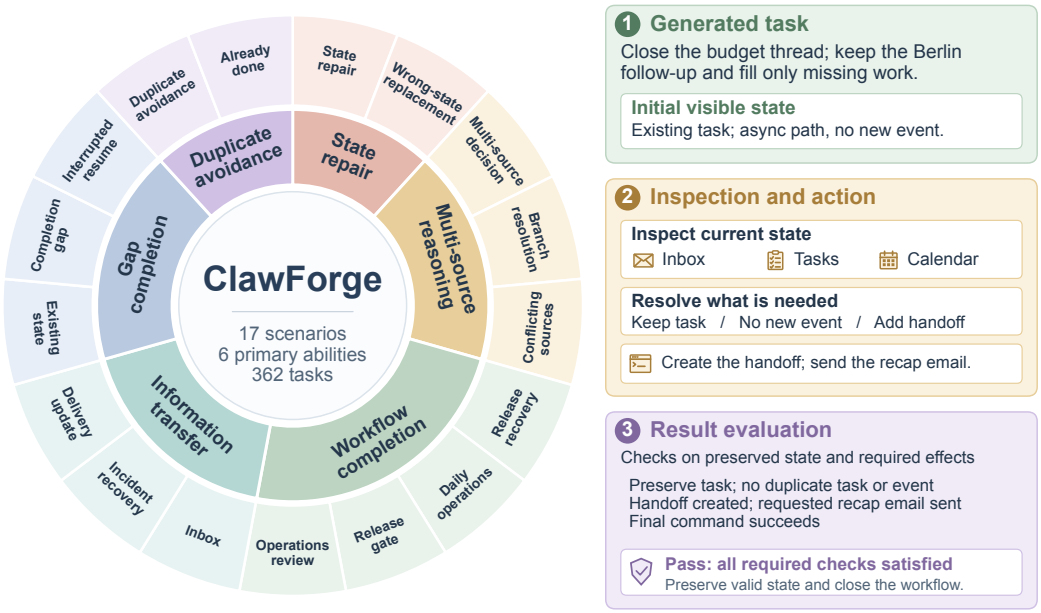


Figure 1: ClawForge-Bench overview. Left: six primary ability categories (inner ring) and 17 scenario families (outer ring). Right: a representative episode requiring preservation of valid state and completion of missing workflow artifacts.

seeded environment state, reference trajectories, validators, and metadata into executable task specifications. The execution environment then runs those tasks step by step through a CLI-style interface and scores the resulting workflow state and observable side effects rather than exact command imitation. Automatic generation is therefore not merely a way to produce more prompts; it keeps task construction, execution, and evaluation grounded in one auditable specification.

Our contributions are threefold. First, ClawForge provides an extensible pipeline for generating and validating stateful command-line tasks. Second, we instantiate it as *ClawForge-Bench*: **17 scenarios** across six ability categories (Figure 1) that target duplicate-aware completion, stale-state repair, wrong-state replacement, multi-source branch selection, and workflow closure. Third, we characterize 11 strong model endpoints on the same 362-task snapshot. Strict accuracy ranges from 35.9% to 61.6%, Wrong-State Replacement ranges from 0.0% to 52.2%, and scenario-level profiles remain strongly non-nested. Partial credit and step usage further distinguish near-miss closure from unproductive exploration.

2 Task Generation and Execution

ClawForge is a generator-backed benchmark system rather than a prompt collection. The central design principle is that each generated task is an executable specification: the instruction, initialized state, reference trajectory, validators, and metadata are produced together and then evaluated through the same stateful runtime. This section describes how tasks are generated (§2.1) and executed (§2.2); Section 3 defines the evaluation protocol.

2.1 Automated Benchmark Generation

Each task is generated from a scenario template together with grounded slots such as city, timezone, recipient, topic, due date, or calendar start time. This yields surface diversity while ensuring that every generated task maps back to a known scenario and can be evaluated by the same validators. As shown in Figure 2, the generation pipeline proceeds in stages: the scenario template is first grounded with concrete slot values, then a state-mode (mock or real) is selected and the initial environment state is instantiated accordingly, followed by instruction rendering. The pipeline then synthesizes

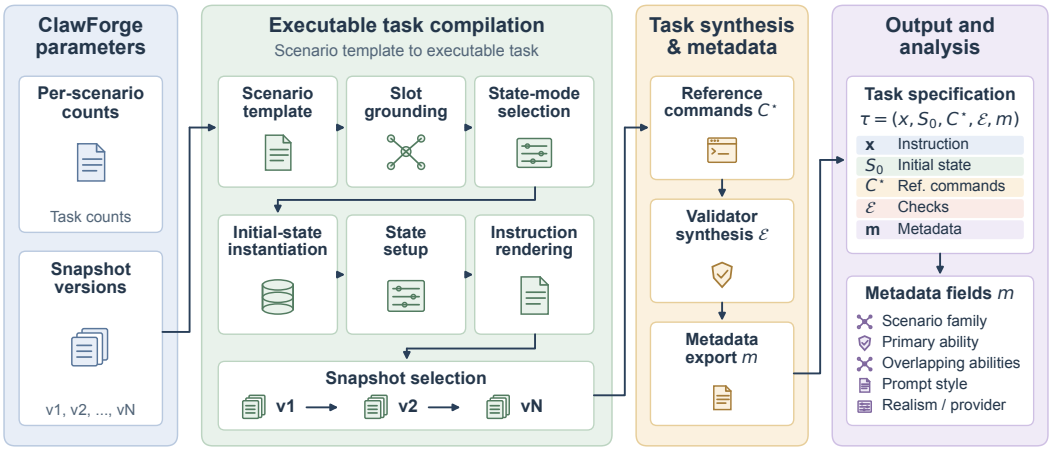


Figure 2: Automated benchmark generation pipeline. Scenario templates are compiled into executable task specifications $\tau = (x, S_0, C^\star, \mathcal{E}, m)$ through slot grounding, state initialization, instruction rendering, reference command synthesis, and validator generation.

reference commands $C^\star$ by executing the intended workflow against the initialized state, generates validators that check required state transitions and side effects, and exports structured metadata (scenario family, primary ability, prompt style). The final output is a complete executable task object.

A generated task is an executable specification with five coupled components:

$$\tau = (x, S_0, C^\star, \mathcal{E}, m), \tag{1}$$

where x is the instruction, S_0 the initialized state, $C^\star$ a reference command trajectory, $\mathcal{E}$ the executable checks, and m the structured metadata. Each τ is instantiated from a scenario family σ, a grounded slot assignment z, and a prompt policy p. The key design choice is that all five components are generated together from the same scenario specification, so the benchmark object is an executable workflow task rather than a prompt paired with an offline answer key.

Because the generator is parameterized by scenario family, different scenarios can encode fundamentally different decision structures rather than only surface wording variations. Some are gap-completion tasks, some require explicit repair or replacement of stale state, and others require branch selection across several information sources. This structural diversity lets the benchmark separate state conflict, closure efficiency, and multi-source decision failures rather than measuring only surface command following (Section 4). Once generated, each task is executed through the interactive environment described next.

2.2 Interactive Environment

Generated tasks are executed in a stateful environment rather than graded as static prompts. ClawForge exposes a CLI-style interface over workflow surfaces such as tasks, calendar, email, messaging, files, runtime configuration, weather, and recurring checks. An episode begins with a natural-language instruction. At each step, the agent emits one command, the environment executes it, and the next observation is built from the resulting outputs and updated state. This execution-and-evaluation loop is summarized in Figure 3, which makes explicit that execution and result-first grading happen inside one coupled episode rather than as separate offline stages.

This loop is intentionally stateful: many failures only become visible after execution. An agent may recreate an existing object, leave stale state untouched, update the wrong entity, or stop before the final side effect is produced. ClawForge therefore treats execution state as a first-class object in both rollout and evaluation (see Appendix B.1 for implementation details).

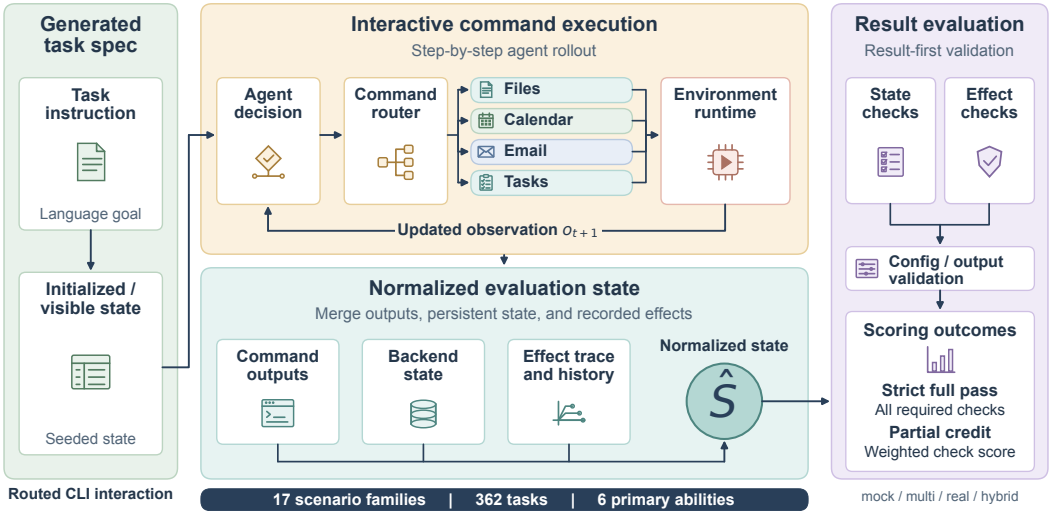


Figure 3: Interactive execution and evaluation loop. Agents emit commands step by step; the environment executes them, records state changes and effect traces, and merges everything into a normalized evaluation state $\hat{S}$ for result-first scoring.

3 Evaluation Protocol

ClawForge evaluates tasks functionally rather than by comparing surface-form trajectories. The evaluator operates over normalized execution state and explicit effect traces, so multiple trajectories can pass as long as they produce the required state transitions and observable side effects. Because tasks are generated together with their validators, evaluation stays tied to the same scenario semantics that produced the task. This result-first design lets us distinguish between early breakdowns and structured near misses, such as preserving the wrong object, omitting one final side effect, or stopping after a partially correct repair.

3.1 Episode Rollout

Each episode is instantiated from a generated task containing an instruction, initialized state, reference trajectory, executable checks, and structured metadata. On reset, the environment materializes task state, applies any task-specific state overrides, initializes the selected backend, and builds the evaluator. The agent receives the instruction once and then emits exactly one command at each subsequent step. The environment executes that command, records state-changing effects, and returns the next observation. Algorithm 1 summarizes this interaction. When a rollout ends, the environment builds a normalized state $\hat{S}$ from the command history, accumulated effect traces, the latest process outputs, and the merged backend state. The evaluator then applies task-defined checks over $\hat{S}$ rather than comparing against one exact command sequence. The default protocol uses `multi` mode, which routes commands to their corresponding workflow surfaces, with a configurable maximum step limit (see Appendix B.1 for additional runtime details).

3.2 Scoring

We report two complementary metrics. Let $\mathcal{D}$ denote the evaluation set. *Strict full-pass accuracy* counts the fraction of tasks where every required check passes:

$$\mathrm{Acc}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \mathbf{1}[\tau \text{ passes all required checks}]. \tag{2}$$

Partial-credit score measures how much of each workflow was completed correctly. If a rollout has checks $\{c_i\}_{i=1}^{n}$, each with score $s_i \in [0, 1]$ and weight $w_i > 0$, the per-task score is

$$\mathrm{Score}(\hat{S}) = \frac{\sum_{i=1}^{n} w_i s_i}{\sum_{i=1}^{n} w_i}, \tag{3}$$

Algorithm 1 ClawForge episode rollout

Load task T and copy base state S_0 into an isolated state directory
Apply task-specific state overrides
Initialize routed backend B and evaluator E
Initialize observation $o_0 \leftarrow$ instruction, config, gateway status, command hints
for $t = 1, \ldots, H$ **do**
  if agent requests stop or emits `DONE` **then**
    break
  end if
  Agent emits one command a_t
  Execute a_t through routed backend B
  Record stdout/stderr, exit code, command metadata, and inferred effects
  Construct next observation o_t
  if $a_t \in \{\texttt{done}, \texttt{exit}, \texttt{quit}\}$ or $t = H$ or a rollout stopping rule triggers **then**
    break
  end if
end for
Build normalized evaluation state $\hat{S}$ from command history, effects, latest outputs, config, and merged backend state
Return final result $E(\hat{S}, T)$

and we report the average over $\mathcal{D}$. The two metrics use the same evaluator but summarize it differently: strict accuracy rewards complete closure, while partial credit distinguishes near-miss failures from early breakdowns. When agents are served by external providers, ClawForge also records provider-side failures and provider-impacted tasks as supplementary records.

4 Experiments

We evaluate ClawForge-Bench (17 scenarios, 362 tasks, two prompt styles) across 11 model endpoints. We ask four questions: (1) Does the benchmark separate strong models without saturating? (2) Which abilities and scenarios expose the largest capability gaps? (3) How efficiently do agents convert interaction steps into workflow closure? (4) Can partial credit distinguish near-miss closure from early breakdown?

4.1 Experimental Setup

All evaluations follow the protocol from Section 3 in `multi` mode with full interaction history. We evaluate Kimi-K2.5 (Team et al., 2026); GPT-5.2 (OpenAI, 2025a), OpenAI o3 (OpenAI, 2025c), GPT-5.3-Codex (OpenAI, 2025b), GPT-5.4 (OpenAI, 2026a), GPT-5.5 (OpenAI, 2026b), GPT-5.6 Sol (OpenAI, 2026d), GPT-5.6 Luna (OpenAI, 2026c), and GPT-6 Astra (OpenAI, 2026e); and Claude Sonnet 4.6 (Anthropic, 2026b) and Claude Opus 4.6 (Anthropic, 2026a). We report strict full-pass accuracy and partial-credit score as defined in Section 3.2. All results use the complete 362-task snapshot; Appendix D.1 documents run provenance, serving details, a reference-consistency audit, and the corresponding sensitivity analysis. Claude Sonnet 4.6 step counts in Table 4 are not fully symmetric with the other trace sets, as detailed in the appendix.

4.2 Benchmark Results

The results answer all four questions. Aggregate accuracy shows model separation without saturation; ability and scenario slices localize the hardest state conflicts; step counts expose inefficient exploration; and partial credit separates incomplete closure from earlier failures.

4.2.1 Overall Benchmark Separation

Table 1 shows that the benchmark is neither saturated nor dominated by one model family. GPT-6 Astra leads with 61.6% strict accuracy and 0.9106 partial credit, followed by GPT-5.6 Sol at 57.7%

Table 1: **Main results** over all 362 tasks. Models are grouped by family.

Model	Strict Acc.	Partial Credit
Kimi-K2.5 (Team et al., 2026)	35.9%	0.7404
GPT-5.2 (OpenAI, 2025a)	38.7%	0.7883
GPT-o3 (OpenAI, 2025c)	39.5%	0.8071
GPT-5.3-Codex (OpenAI, 2025b)	40.6%	0.8238
GPT-5.4 (OpenAI, 2026a)	40.9%	0.8078
GPT-5.5 (OpenAI, 2026b)	51.1%	0.8292
GPT-5.6 Luna (OpenAI, 2026c)	40.9%	0.8281
GPT-5.6 Sol (OpenAI, 2026d)	57.7%	0.8707
GPT-6 Astra (OpenAI, 2026e)	**61.6%**	**0.9106**
Claude Sonnet 4.6 (Anthropic, 2026b)	42.3%	0.7473
Claude Opus 4.6 (Anthropic, 2026a)	45.3%	0.7868

Table 2: **Strict accuracy (%) by primary ability.** Each row is a disjoint partition of the 362-task set. Models are grouped by family.

Primary Ability	N	Kimi	GPT								Claude	
		K2.5	5.2	o3	5.3-C	5.4	5.5	5.6 Sol	5.6 Luna	6 Astra	Son. 4.6	Opus 4.6
duplicate avoidance	20	95.0	95.0	95.0	95.0	**100.0**	**100.0**	90.0	95.0	90.0	**100.0**	95.0
gap completion	52	26.9	42.3	50.0	51.9	53.8	88.5	**90.4**	51.9	65.4	78.8	78.8
information transfer	32	71.9	62.5	50.0	71.9	81.2	**93.8**	87.5	81.2	84.4	59.4	59.4
multi-source reasoning	86	31.4	31.4	25.6	37.2	40.7	39.5	**45.3**	36.0	**45.3**	39.5	37.2
state repair	92	18.5	26.1	33.7	13.0	12.0	35.9	46.7	20.7	**63.0**	17.4	29.3
workflow completion	80	37.5	35.0	36.2	42.5	35.0	27.5	42.5	32.5	**58.8**	28.8	32.5

strict accuracy, while Kimi-K2.5 is lowest at 35.9%. Strict accuracy and partial credit also expose different behavior: GPT-5.6 Luna ties GPT-5.4 at 40.9% strict accuracy but obtains higher partial credit (0.8281 versus 0.8078), and GPT-5.3-Codex reaches 0.8238 partial credit despite 40.6% strict accuracy. This mismatch indicates that the suite distinguishes models that accumulate correct intermediate state from models that convert a smaller subset of trajectories into complete workflow closure.

Among the four newer GPT endpoints with paired task records, Astra passes 223/362 tasks, followed by GPT-5.6 Sol at 209/362, GPT-5.5 at 185/362, and GPT-5.6 Luna at 148/362. The paired task-bootstrap difference between Astra and Sol is 3.9 percentage points (95% CI: -0.8 to 8.6), so the observed aggregate lead is not statistically resolved on this task set. By contrast, Sol exceeds GPT-5.5 by 6.6 points (2.2 to 10.8), GPT-5.5 exceeds Luna by 10.2 points (5.5 to 14.9), and Astra exceeds Luna by 20.7 points (15.5 to 26.0). These intervals quantify task-set sampling uncertainty, not repeated-run variance.

Task-level overlap reveals non-nested success sets: Astra and Sol jointly pass 179 tasks; 44 are passed only by Astra and 30 only by Sol, while 109 defeat both. The net 14-task lead masks 74 tasks with discordant outcomes. Paired coverage therefore complements aggregate accuracy.

4.2.2 Ability-Level Analysis

The ability table shows that higher aggregate performance does not remove the targeted bottlenecks (Table 2). Multi-source reasoning ranges from 25.6% to 45.3%, while state repair ranges from 12.0% to 63.0%. Astra leads state repair and workflow completion, Sol leads gap completion, and GPT-5.5 leads duplicate avoidance and information transfer. The profiles are not nested: Astra passes 24/46 Wrong-State Replacement tasks but only 15/30 Interrupted Workflow Resume tasks, whereas Sol and GPT-5.5 pass 29/30 and 30/30 resume tasks, respectively.

Table 3 adds a partial-credit perspective. Many rollouts that fail the strict criterion still make substantial progress: o3 reaches 0.7207 partial credit on state repair despite only 33.7% strict accuracy, and Astra reaches 0.8913 partial credit while passing 63.0%. Models often identify the correct repair direction but miss one final side effect or replacement step, or stop as if the workflow were complete

Table 3: **Partial-credit score (0–1) by primary ability.** Higher scores indicate more complete workflow execution even when strict evaluation fails. Models are grouped by family.

Primary Ability	N	Kimi	GPT								Claude	
		K2.5	5.2	o3	5.3-C	5.4	5.5	5.6 Sol	5.6 Luna	6 Astra	Son. 4.6	Opus 4.6
duplicate avoidance	20	0.9930	0.9930	0.9930	0.9930	**1.0000**	**1.0000**	0.9857	0.9929	0.9857	**1.0000**	0.9930
gap completion	52	0.7681	0.8813	0.8192	0.8731	0.8735	**0.9423**	0.9368	0.8558	0.9077	0.8940	0.8810
information transfer	32	0.9300	0.9059	0.8862	0.9259	0.9572	**0.9893**	0.9807	0.9548	0.9777	0.9060	0.9140
multi-source reasoning	86	0.8303	0.8020	0.7486	0.8277	0.8206	0.8421	0.8589	0.8124	**0.8766**	0.7910	0.8670
state repair	92	0.4297	0.5886	0.7207	0.6755	0.6171	0.6522	0.7627	0.7274	**0.8913**	0.4130	0.4980
workflow completion	80	0.8435	0.8431	0.8800	0.8719	0.8605	0.8385	0.8917	0.8510	**0.9254**	0.8630	0.8690

Table 4: **Accuracy (%) and average executed steps by reference-trajectory length.** Format: Acc./Steps. Shorter reference trajectories indicate structurally simpler workflows; models are grouped by family.

GT	N	Kimi	GPT								Claude	
		K2.5	5.2	o3	5.3-C	5.4	5.5	5.6 Sol	5.6 Luna	6 Astra	Son. 4.6	Opus 4.6
5 steps	77	36.4/7.8	37.7/7.3	40.3/8.4	48.1/6.2	44.2/6.3	40.3/8.2	41.6/9.3	45.5/8.1	40.3/9.6	40.3/8.1	40.3/9.6
6 steps	115	41.7/10.3	40.0/10.7	49.6/10.3	42.6/6.6	39.1/8.1	47.8/12.6	62.6/14.0	35.7/12.8	87.0/12.9	38.3/14.5	46.1/16.6
7 steps	45	53.3/11.7	53.3/13.7	51.1/12.4	28.9/8.2	33.3/9.6	68.9/14.4	73.3/15.4	57.8/15.8	73.3/13.6	31.1/20.7	37.8/19.9
8 steps	53	22.6/10.8	41.5/10.7	39.6/11.9	43.4/8.5	50.9/8.8	75.5/11.2	77.4/13.2	45.3/12.6	49.1/15.1	69.8/17.7	67.9/15.8
9 steps	72	25.0/10.3	26.4/12.0	15.3/9.4	34.7/8.1	37.5/8.5	38.9/11.3	43.1/14.3	30.6/12.0	45.8/12.7	37.5/14.8	37.5/14.9

after only a partial repair. The score view supports a useful failure taxonomy: early breakdown, correct direction but incomplete closure, stale-state mishandling, and hallucinated completion.

Among failed rollouts, mean partial credit is 0.7671 across Astra's 139 failures and 0.6940 across Sol's 153 failures. Substantial satisfaction of weighted checks thus persists after excluding successful tasks, supporting joint reporting of strict completion and partial progress. Since the failure sets differ, these means characterize residual errors rather than establish a matched-task ranking.

4.2.3 Step Efficiency

Table 4 pairs reference-trajectory length with average executed steps, revealing two distinct failure styles across the evaluated models. GPT-5.3-Codex and GPT-5.4 are step-efficient, staying near eight executed steps on the longest reference buckets, while Claude Opus 4.6 uses 16–20 average steps on GT-6 through GT-8 tasks. The strongest bucket also varies sharply by model: Astra reaches 87.0% on six-step tasks, while Sol and GPT-5.5 peak on eight-step tasks. Reference length is therefore not a monotonic proxy for difficulty. *Step-efficient but imperfect* models reach compact, partially correct solutions but miss one final operation, whereas *high-step low-conversion* models keep exploring without improving closure proportionally. The combined accuracy-plus-steps view shows not just whether a model passes, but how efficiently it converts interaction budget into closure.

4.2.4 Scenario-Level Discrimination

Figure 4 compares all 17 scenarios for Kimi-K2.5 and five recent GPT endpoints. Already-Done Skip is saturated in this view, while Duplicate Avoidance ranges from 80.0% to 100.0%. Wrong-State Replacement ranges from 0.0% to 52.2%, and Release Gate remains difficult at 3.3–16.7%. Interrupted Workflow Resume produces the widest separation in this comparison (16.7–100.0%): GPT-5.5 passes all 30 tasks but none of the 46 replacement tasks, whereas Astra passes 24/46 replacement tasks but only 15/30 resume tasks. Appendix Tables 7 and 8 provide selected cross-family results and the complete eight-GPT scenario breakdown, respectively.

Shared failures are concentrated rather than evenly distributed: Branch Resolution and Release Gate account for 51 of the 109 tasks failed by both Astra and Sol (46.8%), although these scenarios comprise only 70/362 tasks (19.3%). Release Gate is particularly consistent across GPT-5.5, Sol, and Astra: each passes the same 5/30 tasks and obtains 0.8611 mean partial credit. Thus, substantial satisfaction of weighted checks coexists with a shared strict-completion bottleneck. This concentration

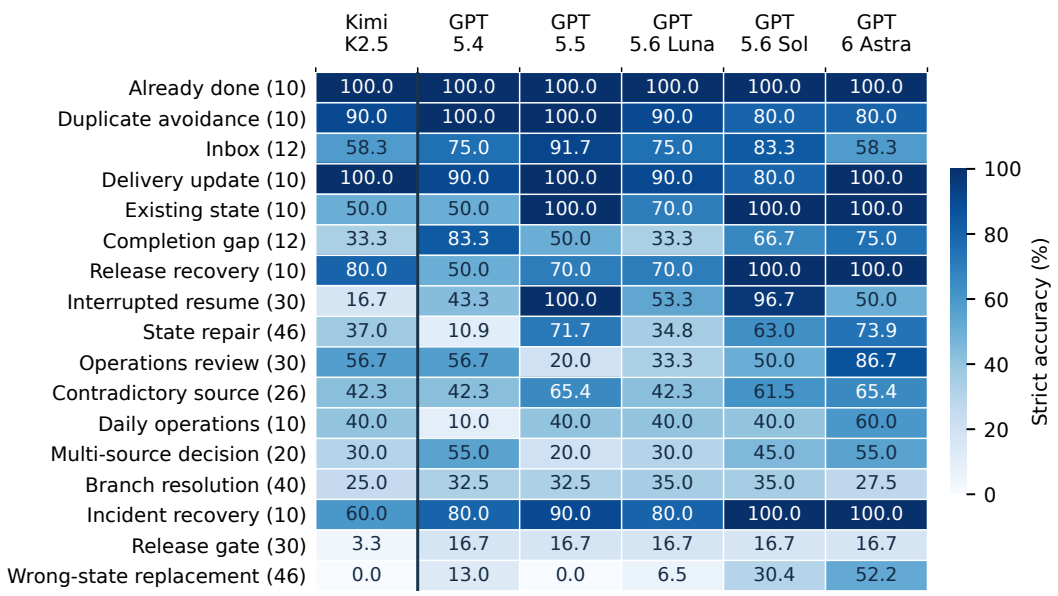


Figure 4: Per-scenario strict accuracy (%) for Kimi-K2.5 and five recent GPT endpoints; parentheses give task counts. Older GPT endpoints are omitted for readability. Full GPT results appear in Appendix Table 8.

identifies specific scenarios for further trace analysis; the aggregate scores alone do not establish which checks or decision steps cause the failures.

4.3 Case Studies

The aggregate results show *where* models fail; two representative tasks illustrate *how*. One requires replacing incorrect state, whereas the other requires recognizing and preserving valid state.

Case 1: Wrong-State Replacement. Correct closure requires both retiring stale state and creating its replacement. Astra passes 24/46 tasks, Sol passes 14/46, and GPT-5.5 passes none (Figure 4). Neither retirement alone nor creation alongside a stale artifact satisfies the task: the evaluator checks both obligations, distinguishing replacement from ordinary task creation.

Task. `hard_decision_workflow_261`: "Something in the Seattle release setup was staged wrong. Check whether the next step is stale, replace the stale next step with the correct version, and leave the next step and sync correctly set."

Initial state. A stale, low-priority task exists; the replacement task and sync do not.

Required actions. Retire the stale task, create `Seattle release replacement next step`, and add the matching replacement sync.

Evaluation. Both stale-task retirement and replacement-state creation are required.

Paired outcomes distinguish replacing wrong state from preserving valid state. On replacement tasks, 11 are passed only by Astra and one only by Sol, with 21 failed by both. On resume tasks, 14 are passed only by Sol and none only by Astra. These reversals localize the aggregate difference to distinct state-handling demands rather than a uniform advantage across workflows.

Case 2: Interrupted Workflow Resume. GPT-5.5 completes all 30 tasks, Sol completes 29/30, and Kimi-K2.5 completes 5/30. Here, recreating a valid task or calendar event violates the duplicate guard. Success instead requires inspecting existing artifacts and completing only the missing handoff and recap.

Task. `hard_decision_workflow_307`: "The New York release work already has pieces in place. Check the current model, the next step on the board, and the sync on the calendar, finish the missing handoff file, send manager@example.com a recap, and leave the existing setup alone."

Initial state. The task, sync, and review notes exist; only the handoff and recap email are missing.

Required actions. Read the notes, create the handoff, preserve existing state without duplicates, and send the recap.

Evaluation. Existing state must remain, with no duplicates, and both missing side effects must be completed.

These cases isolate the benchmark's central distinction: Case 1 requires replacing what is *wrong*, whereas Case 2 requires preserving what is *right*. A clean-state benchmark would reduce both to ordinary completion. Appendix F provides the full scenario catalog.

5 Related Work

Interactive agent benchmarks. Recent work increasingly evaluates LLM agents as sequential decision-makers that interleave reasoning, tool use, and action rather than produce a single final answer, with ReAct serving as a representative formulation (Yao et al., 2022). This shift has motivated executable or semi-executable benchmarks for software repair, web interaction, planning, and tool use, including SWE-bench (Jimenez et al., 2023), AgentBench (Liu et al., 2023), WebArena (Zhou et al., 2023), OSWorld (Xie et al., 2024), GAIA (Mialon et al., 2023), Claw-Eval (Ye et al., 2026), and ClawsBench (Li et al., 2026a). However, these benchmarks mainly evaluate task completion or trajectory correctness under clean or weakly constrained initial state, making it difficult to measure failures caused by stale state, conflicting artifacts, or incomplete workflows.

Persistent memory and long-horizon agents. A related line of work studies persistent memory and long-horizon behavior in LLM agents. Virtual-context approaches, including MEMGPT (Packer et al., 2023), MEMORYOS (Kang et al., 2025), and stream-based controllers (Wang et al., 2024), extend interaction length through memory paging or context management. Structured and graph-based systems, such as MEMORYBANK (Zhong et al., 2024), MEM0 (Chhikara et al., 2025), ZEP (Rasmussen et al., 2025), A-MEM (Xu et al., 2025), and O-MEM (Wang et al., 2025), instead organize memory into persistent representations for long-horizon retrieval and planning. These approaches primarily improve memory mechanisms, but provide limited evaluation of whether agents correctly preserve, repair, or update workflow state during extended execution.

Stateful workflow evaluation. ClawForge focuses on evaluating these workflow-state behaviors directly. Unlike existing benchmarks that mainly evaluate final task completion or exact trajectories (Li et al., 2026b; Ye et al., 2026), ClawForge evaluates agents under persistent workflow state, where tasks may already be partially completed, inconsistent, stale, or duplicate-sensitive before execution begins. Agents must preserve valid state, detect outdated artifacts, resolve conflicting information, avoid redundant mutations, and complete workflows end to end. Correctness is defined over normalized end state and observable side effects rather than exact action sequences, allowing multiple valid trajectories.

6 Conclusion

We presented ClawForge, a generator-backed framework for evaluating command-line agents under state-conflict workflows through automated task construction, stateful execution, and result-first evaluation. ClawForge-Bench contains 362 tasks from 17 scenarios and 6 ability categories, testing whether agents inspect existing state, preserve valid artifacts, repair or replace stale ones, reconcile conflicting evidence, and complete downstream effects. Across 11 model endpoints, strict full-pass accuracy ranges from 35.9% to 61.6%. Wrong-State Replacement ranges from 0.0% to 52.2%, while Interrupted Workflow Resume ranges from 16.7% to 100.0%. These contrasting profiles show that higher aggregate accuracy does not imply uniformly stronger state inspection, repair, and workflow closure. Partial-credit and step-efficiency analyses distinguish compact near-miss closures that omit a final side effect from high-step exploration that fails to produce a valid end state. Joint outcome and trace analysis goes beyond pass/fail scoring and exact command matching. Future work will broaden coverage and streamline template construction and auditing.

AI USE STATEMENT

Generative AI tools were used to aid and polish the writing of this paper, including improvements to grammar, phrasing, clarity, and organization. All AI-assisted text was reviewed, verified, and revised by the authors. The authors take full responsibility for the final content of the paper, including its claims, analyses, and presentation.

ETHICS STATEMENT

The authors have read and commit to adhering to the ICLR Code of Ethics. This work evaluates language-model agents in synthetic or seeded command-line workflow environments and does not involve human subjects or private user data. The benchmark nevertheless studies agents that can modify tasks, calendars, messages, files, and system state; deploying such agents without appropriate oversight could cause real-world operational, privacy, or communication harms. We mitigate these risks by evaluating in controlled environments, reporting state-changing effects and failure modes explicitly, and presenting benchmark performance as diagnostic evidence rather than as proof of readiness for autonomous deployment. Additional discussion appears in Appendix E.1.

REPRODUCIBILITY STATEMENT

We support reproducibility by specifying the task-generation process, initialized state, rollout protocol, result-first evaluator, and scoring metrics in Sections 2 and 3. Appendix A documents the scenario inventory and executable task structure; Appendix B provides runtime details and representative evaluation checks; and Appendix D reports data provenance, artifact fingerprints, execution configurations, limitations, statistical analyses, and compute resources. The full scenario catalog in Appendix F provides concrete task instances and expected workflow outcomes for all benchmark scenario families.

REFERENCES

Anthropic. The claude 3 model family: Opus, sonnet, haiku, 2024. https://www.anthropic.com/news/claude-3-family.

Anthropic. Introducing opus 4.6, 2026a. https://www.anthropic.com/news/claude-opus-4-6.

Anthropic. Introducing sonnet 4.6, 2026b. https://www.anthropic.com/news/claude-sonnet-4-6.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 15(3):1–45, 2024.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Haonian Ji, Kaiwen Xiong, Siwei Han, Peng Xia, Shi Qiu, Yiyang Zhou, Jiaqi Liu, Jinlong Li, Bingzhou Li, Zeyu Zheng, et al. Clawarena: Benchmarking ai agents in evolving information environments. *arXiv preprint arXiv:2604.04202*, 2026.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25972–25981, 2025.

Xiangyi Li, Kyoung Whan Choe, Yimin Liu, Xiaokun Chen, Chujun Tao, Bingran You, Wenbo Chen, Zonglin Di, Jiankai Sun, Shenghan Zheng, et al. Clawsbench: Evaluating capability and safety of llm productivity agents in simulated workspaces. *arXiv preprint arXiv:2604.05172*, 2026a.

Xirui Li, Ming Li, Derry Xu, Wei-Lin Chiang, Ion Stoica, Cho-Jui Hsieh, and Tianyi Zhou. Clawenvkit: Automatic environment generation for claw-like agents. *arXiv preprint arXiv:2604.18543*, 2026b.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Fanqing Meng, Lingxiao Du, Zijian Wu, Guanzheng Chen, Xiangyan Liu, Jiaqi Liao, Chonghe Jiang, Zhenglin Wan, Jiawei Gu, Pengfei Zhou, et al. Clawmark: A living-world benchmark for multi-turn, multi-day, multimodal coworker agents. *arXiv preprint arXiv:2604.23781*, 2026a.

Fanzhe Meng, Guoxin Chen, Jiale Zhao, Shuang Sun, Zhiyu Lin, Wayne Xin Zhao, Ruihua Song, Ji-Rong Wen, and Kai Jia. Calibforge: Adversarial solver calibration for scaling learnable terminal tasks. *arXiv preprint arXiv:2608.06352*, 2026b.

Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

OpenAI. Introducing gpt-5.2, 2025a. https://openai.com/index/introducing-gpt-5-2.

OpenAI. Introducing gpt-5.3-codex, 2025b. https://openai.com/index/introducing-gpt-5-3-codex/.

OpenAI. Introducing o3 and o4-mini, 2025c. https://openai.com/index/introducing-o3-and-o4-mini/.

OpenAI. Introducing gpt-5.4, 2026a. https://openai.com/index/introducing-gpt-5-4/.

OpenAI. GPT-5.5 Model Documentation, 2026b. https://developers.openai.com/api/docs/models/gpt-5.5.

OpenAI. GPT-5.6 Luna Model Documentation, 2026c. https://developers.openai.com/api/docs/models/gpt-5.6-luna.

OpenAI. GPT-5.6 Sol Model Documentation, 2026d. https://developers.openai.com/api/docs/models/gpt-5.6-sol.

OpenAI. GPT-6 Astra Model Documentation, 2026e. https://developers.openai.com/api/docs/models/gpt-6-astra.

OpenClaw. Openclaw — personal ai assistant, 2026. https://github.com/openclaw/openclaw.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Nous Research. Hermes agent — the agent that grows with you, 2026. `https://github.com/nousresearch/hermes-agent`.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-{\alpha}: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

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

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, et al. Claw-eval: Toward trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

APPENDIX CONTENTS

The appendix documents benchmark construction, evaluation, supplementary results, and reproducibility. Section F.1 provides a page index for the 17 individual scenario examples.

A Benchmark Construction . . . 13
A.1 Scenario Inventory . . . 13
A.2 Example Task Instruction Card . . . 15
A.3 Implementation Detail . . . 15

B Evaluation Details . . . 16
B.1 Execution Mode and Evaluator Details . . . 16
B.2 Evaluation Check Examples . . . 17
B.3 Execution Examples . . . 17

C Supplementary Results . . . 17
C.1 Scenario Slice Table . . . 18
C.2 Failure Pattern Notes . . . 18
C.3 Length and Efficiency Notes . . . 19

D Data and Reproducibility . . . 20
D.1 Data Provenance and Provider Impact . . . 20
D.2 Limitations and Scope . . . 21
D.3 Statistical Significance . . . 21
D.4 Compute Resources . . . 21

E Broader Impact and Release . . . 22
E.1 Broader Impact . . . 22
E.2 Asset Credits and Terms . . . 22
E.3 Open Access Release Status . . . 22

F Full Scenario Catalog . . . 23
F.1 Scenario Example Directory . . . 23
F.2 Full Scenario Example Catalog . . . 23

A BENCHMARK CONSTRUCTION

This section documents the scenario taxonomy, generated task objects, and implementation details that support reproducibility. We first present the full scenario inventory (§A.1), then show a representative task instruction card (§A.2), and finally describe the implementation contract (§A.3).

A.1 SCENARIO INVENTORY

Table 5 lists all 17 scenario families in the ClawForge-Bench, together with their primary ability classification, overlapping ability tags, and a brief description of the required workflow.

Table 5: Full scenario inventory for `hard_decision_workflow`. Scenario names are rendered in paper form rather than as generator slugs; release counts are omitted because the generator allows per-scenario counts to be reconfigured.

Scenario	Primary Ability	Ability Tags	Description
Inbox	Information transfer	information transfer, workflow completion	Close an inbox thread, inspect board and calendar state, and add only the missing follow-up.
Release Recovery Runbook	Workflow completion	workflow completion, schedule inference	Restore a partially staged release review by switching models and filling the missing review artifact.
Channel Incident Recovery	Information transfer	information transfer, workflow completion	Send a channel update, verify tracking on the board, and add only the missing incident next step.
Daily Operations Commitment Loop	Workflow completion	workflow completion, schedule inference	Use today's schedule plus board and cron state to fill the single missing daily operations commitment.
Release Gate	Workflow completion	workflow completion, state inspection	Inspect model, board, and calendar state before closing the release gate cleanly.
Delivery Update	Information transfer	information transfer, workflow completion	Repair a delivery path, avoid duplicate board work, and choose the right live or async closure.
Operations Review	Workflow completion	workflow completion, schedule inference	Use forecast and schedule context to finish the missing operations setup before placing the review.
Existing State	Gap completion	gap completion, state inspection, duplicate avoidance, workflow completion	Audit a partially completed recurring setup and fill exactly one missing piece without rebuilding the rest.
Duplicate Avoidance	Duplicate avoidance	duplicate avoidance, completion recognition	Preserve existing operations setup and choose the correct primary or backup review block.
Multi-Source Decision	Multi-source reasoning	multi-source reasoning, branch resolution, workflow completion	Combine email, forecast, and calendar evidence to decide between live and async closure.
State Repair	State repair	state repair, state inspection, workflow completion	Detect one stale release artifact and repair it while preserving the rest of the setup.
Completion Gap	Gap completion	gap completion, state inspection, workflow completion	Inspect a partial release setup and add only the missing task and/or sync.
Branch Resolution	Multi-source reasoning	multi-source reasoning, branch resolution, duplicate avoidance	Resolve a live-vs-async branch from forecast and calendar context without duplicating setup.
Already Done Skip	Duplicate avoidance	duplicate avoidance, completion recognition, information transfer	Verify that a release setup is already correct and send a recap without rebuilding anything.
Wrong-State Replacement	State repair	state repair, replacement planning, workflow completion	Replace one wrongly staged artifact with the correct release state and preserve the valid pieces.
Interrupted Workflow Resume	Gap completion	gap completion, state inspection, duplicate avoidance, workflow completion	Resume partially completed release work, finish the missing artifact, and avoid recreating existing state.
Contradictory Source Resolution	Multi-source reasoning	multi-source reasoning, branch resolution, workflow completion, state inspection	Reconcile conflicting signals across sources before choosing the correct closure path.

A.2 Example Task Instruction Card

To make the generated task format concrete, we show one complete instruction card from a gap-completion scenario. This illustrates how the five task components (instruction, initial state, required actions, checks, and difficulty) are coupled in a single specification.

Task ID. `hard_decision_workflow_204` (`completion_gap_followthrough`)
Primary ability. Gap completion
Ability tags. `gap_completion`, `state_inspection`, `workflow_completion`

Instruction. "I already have part of the Berlin release work in motion, and the review sync is the missing piece. Review what's there, read the release notes, refresh the handoff file, leave the next step on the board in place and add only the missing sync on the calendar, and leave the rest alone."

Initial visible state. The initial state already contains a pending board item for the Berlin release next step and a drafted release-review note. The calendar sync is still missing, and the handoff artifact needs to be created or refreshed. This means the task does not begin from an empty workspace: the agent must preserve valid state while identifying the one missing closure path.

Required correction and closure. A semantically correct completion must keep the existing Berlin next-step task in place, create or refresh a handoff artifact, and add the missing Berlin release sync on the calendar. It must not create a duplicate next-step task. The reference trajectory uses the benchmark date anchor for Europe/Berlin, but the evaluator accepts alternative command traces as long as the final state closes the workflow without introducing duplicate board work.

Representative required checks. This task mixes state-preservation and side-effect checks: the existing Berlin board item must still be present; a handoff-style file creation or refresh effect must be visible; no duplicate next-step task may be created; and the missing calendar sync must exist by the end of the rollout.

Why this task is hard. The task is easy to understand semantically but easy to fail procedurally. Models can recognize that a review sync is missing and still fail by recreating the board item, omitting the handoff refresh, or stopping after scheduling the sync and incorrectly assuming the workflow is complete.

A.3 Implementation Detail

This subsection describes how the task specification above is realized at runtime. At task level, each `hard_decision_workflow` episode is an executable specification rather than a bare prompt. The specification bundles five pieces of information: a natural-language instruction, scenario metadata, task-specific `initial_state_overrides`, a reference command trajectory, and typed evaluation checks. In practice, the task object also stores instruction variants, prompt style, realism and scenario tags, public or hidden task notes, and provider-related metadata used for analysis. At reset time, the environment loads a base state, applies the overrides declared by the task, initializes the backend for that state directory, and then exposes the instruction plus compact execution hints to the agent. For `hard_decision_workflow_204`, the override layer inserts an already-existing Berlin next-step task and a drafted release-review note, so the evaluator can later distinguish correct gap completion from unnecessary re-creation.

During rollout, the environment records each command, its stdout/stderr, exit code, and inferred effects such as created files, completed tasks, or created calendar events. The rollout record therefore preserves not only what the agent typed, but what externally visible state transitions were actually produced. Scoring is then performed over a normalized result state rather than over exact trajectory imitation. This design matters for hard workflows because many tasks require a sequence of decisions before any mutation is safe: the agent often has to determine whether the relevant object already exists, whether it is stale, whether it should be preserved, and only then whether the correct action is to add, update, or replace state. In other words, the benchmark object is not just "what command should be typed next," but "what workflow state should exist when the rollout is over."

The default benchmark runtime is `multi` mode, which uses explicit routing rather than one monolithic executor. Command families such as `openclaw`, `calendar`, `email`, `tasks`, `weather`, `file`, and `curl` are dispatched to their corresponding skills or adapters. This routing layer is important for reproducibility because it preserves cross-surface interaction while keeping the execution contract stable across benchmark runs.

One useful way to think about the implementation is as a four-stage contract: (1) construct initialized workflow state, (2) execute one command at a time under routed backends, (3) infer explicit state-changing effects from those executions, and (4) merge the resulting traces into evaluator-facing state. The appendix carries these details so that the main paper can focus on what the benchmark reveals about model behavior rather than on runtime mechanics alone.

B Evaluation Details

This section shows how result-first scoring works in practice. We first describe the execution modes and evaluator internals (§B.1), then present representative evaluation checks (§B.2), and finally explain how to read the full scenario examples (§B.3).

B.1 Execution Mode and Evaluator Details

Algorithm 2 gives the episode-level rollout contract used by the main experiments.

Algorithm 2 ClawForge episode rollout (detailed)

```
Load task T, base state S_0, and backend B
Apply task-specific state overrides and initialize evaluator E
Initialize observation o_0 ← instruction, config, gateway status, command hints
for t = 1, ..., H do
    Agent emits one command a_t
    Execute a_t through backend B
    Update backend state and append inferred effects
    Construct next observation o_t
    if a_t ∈ {done, exit, quit} or t = H then
        break
    end if
end for
Build normalized evaluation state Ŝ
Return final result E(Ŝ, T)
```

ClawForge supports four execution modes with different fidelity and stability trade-offs:

- **`mock`** uses a single in-process mock backend for unit tests and deterministic debugging.
- **`multi`** is the default benchmark setting, with explicit command-family routing through registered skills and adapters.
- **`real`** replaces the `openclaw` branch with a subprocess-backed real backend while keeping the same routing architecture.
- **`hybrid`** extends `real` with gateway lifecycle management, including auto-start, port injection, and health polling.

Internally, the evaluator builds a normalized state $\hat{S}$ that merges configuration state, gateway state, bounded command history, the latest stdout/stderr/exit code, and explicit effect traces (created calendar events, sent emails, created files, completed tasks, etc.). This normalization is particularly important for the hard benchmark because many tasks are judged by the relationship between pre-existing and newly created state: the evaluator must determine not only that an artifact exists, but whether an existing object was preserved, a stale object was retired, and the required downstream side effects were produced.

B.2 Evaluation Check Examples

Table 6 shows representative result-first checks from two hard tasks: `hard_decision_workflow_204` (gap completion) and `hard_decision_workflow_263` (wrong-state replacement). The first task rewards preserving correct existing state while filling a single missing gap. The second additionally requires retiring stale state and putting the replacement plan in place. This is why replacement-style tasks are materially harder: they require preserving the valid part of the workflow, modifying the invalid part, and still achieving end-to-end closure.

Table 6: Representative task-level evaluation checks. These checks illustrate the benchmark's result-first philosophy: trajectories may differ, but the rollout must leave the required state and effects behind.

Check family	Concrete condition	Why it matters
State (`204`)	Existing Berlin follow-through task remains present	Verifies that the rollout preserves already-correct board state rather than deleting or replacing it unnecessarily.
Effect (`204`)	`files_created count_gte 1` and created path contains `handoff`	Accepts semantically correct handoff refreshes without forcing one exact filename.
Effect (`204`)	`tasks_created not_exists`	Implements the no-duplicate guard for the existing release next step.
Effect (`204`)	`calendar_events_created count_gte 1`	Checks that the missing review sync was actually scheduled rather than merely mentioned.
Effect (`263`)	`tasks_completed count_gte 1`	Requires the stale Paris follow-up to be retired rather than simply ignored.
Effect (`263`)	Replacement task created and replacement sync created	Enforces replacement-and-closure: the corrected next step and the replacement sync must both exist.
Output (both)	Final command exits successfully	Prevents a rollout from being counted as complete when the last required mutation failed.

Different checks can also carry different weights in the partial-credit score. In practice, this means the benchmark can distinguish between a rollout that gets the main task direction right but misses one final side effect, and a rollout that never reaches the correct repair or branch decision in the first place. The score layer is therefore not a soft alternative to evaluation, but an additional view on how close a failed rollout came to the intended state transition.

B.3 Execution Examples

The appendix now separates two example roles. The later full scenario catalog gives one long-form example for every released scenario family, while the present section explains how to read those examples. Each long example follows the same schema: a concrete instance, the initial visible state, the key repair or branch decision, a short representative action trace, and the evaluator-facing success condition. This keeps the examples comparable even when one scenario is mostly about duplicate avoidance and another is mostly about state replacement or multi-source resolution.

A useful reading strategy is to compare three things across the catalog: first, whether the task begins from empty or partially populated state; second, whether the required operation is additive, repair-oriented, or replacement-oriented; and third, whether final closure depends on one surface or on coordination across board, calendar, email, file, and cron state. That contrast is exactly what makes the ClawForge-Bench analytically useful.

C Supplementary Results

This section extends the main-text analysis with selected 11-model scenario slices and a complete eight-GPT scenario table (§C.1), recurring failure patterns (§C.2), and step-efficiency interpretation

(§C.3). The main-text ability, partial-credit, and reference-length tables likewise include all 11 endpoints.

C.1 Scenario Slice Table

Table 7 highlights six scenario families that are particularly useful for interpreting the benchmark. The selected slices include near-saturated checks (`already_done_skip_followthrough`, `duplicate_avoidance_followthrough`), a uniformly hard shared slice (`release_gate_followthrough`), the hardest replacement slice (`wrong_state_replacement_followthrough`), a strong separator (`interrupted_workflow_resume`), and a multi-source branch-selection slice (`multi_source_decision_followthrough`).

Table 7: Selected scenario-level full-pass accuracy (%) for all 11 endpoints over the same 362 tasks. Models are grouped by family.

Scenario	N	Kimi	GPT								Claude	
		K2.5	5.2	o3	5.3-C	5.4	5.5	5.6 Sol	5.6 Luna	6 Astra	Son. 4.6	Opus 4.6
already_done_skip_followthrough	10	100	100	100	90	100	100	100	100	100	100	100
duplicate_avoidance_followthrough	10	90	90	90	100	100	100	80	90	80	100	90
release_gate_followthrough	30	3	17	7	10	17	17	17	17	17	17	17
wrong_state_replacement_followthrough	46	0	0	15	17	13	0	30	7	52	0	2
interrupted_workflow_resume	30	17	33	43	33	43	100	97	53	50	87	90
multi_source_decision_followthrough	20	30	15	45	40	55	20	45	30	55	50	55

The unified scenario view confirms that aggregate ordering hides qualitatively different profiles. Astra leads Wrong-State Replacement at 24/46 and ties the best Multi-Source Decision result at 11/20, but passes only 15/30 Interrupted Workflow Resume tasks. GPT-5.5 is perfect on duplicate avoidance and interrupted resume while passing none of the 46 replacement tasks; Sol passes 14/46 replacement tasks and 29/30 resume tasks. Release Gate remains uniformly difficult, with every model passing at most 5/30 tasks.

Table 8 completes the GPT comparison across all 17 scenario families. Scenario counts sum to 362 for every endpoint. The complete breakdown further distinguishes related workflows: GPT-5.5 reaches 100.0% on Interrupted Workflow Resume but 20.0% on Operations Review, whereas Astra reaches 50.0% and 86.7%, respectively. The scenario called State Repair is distinct from the primary-ability category of the same name: the latter pools State Repair and Wrong-State Replacement (46 tasks each).

C.2 Failure Pattern Notes

Beyond per-scenario accuracy, the benchmark reveals recurring failure patterns that cut across scenario families. The main text argues that many failures are structured rather than arbitrary; Table 9 expands that claim with concrete pattern-level categories that recur across scenario families.

Table 8: Full scenario-level strict accuracy (%) for all eight GPT endpoints on the complete 362-task snapshot. N is the number of tasks in each scenario.

Scenario	N	GPT							
		5.2	o3	5.3-C	5.4	5.5	5.6 Luna	5.6 Sol	6 Astra
Already done	10	**100.0**	**100.0**	90.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Duplicate avoidance	10	90.0	90.0	**100.0**	**100.0**	**100.0**	90.0	80.0	80.0
Inbox	12	75.0	50.0	75.0	75.0	**91.7**	75.0	83.3	58.3
Delivery update	10	80.0	**100.0**	90.0	90.0	**100.0**	90.0	80.0	**100.0**
Existing state	10	40.0	70.0	70.0	50.0	**100.0**	70.0	**100.0**	**100.0**
Completion gap	12	66.7	50.0	**83.3**	**83.3**	50.0	33.3	66.7	75.0
Release recovery	10	70.0	40.0	50.0	50.0	70.0	70.0	**100.0**	**100.0**
Interrupted resume	30	33.3	43.3	33.3	43.3	**100.0**	53.3	96.7	50.0
State repair	46	52.2	52.2	8.7	10.9	71.7	34.8	63.0	**73.9**
Operations review	30	46.7	63.3	70.0	56.7	20.0	33.3	50.0	**86.7**
Contradictory source	26	42.3	0.0	50.0	42.3	**65.4**	42.3	61.5	**65.4**
Daily operations	10	20.0	40.0	50.0	10.0	40.0	40.0	40.0	**60.0**
Multi-source decision	20	15.0	45.0	40.0	**55.0**	20.0	30.0	45.0	**55.0**
Branch resolution	40	32.5	32.5	27.5	32.5	32.5	**35.0**	**35.0**	27.5
Incident recovery	10	30.0	0.0	50.0	80.0	90.0	80.0	**100.0**	**100.0**
Release gate	30	**16.7**	6.7	10.0	**16.7**	**16.7**	**16.7**	**16.7**	**16.7**
Wrong-state replacement	46	0.0	15.2	17.4	13.0	0.0	6.5	30.4	**52.2**

Table 9: Representative failure patterns that recur across the hard benchmark.

Failure pattern	Typical benchmark symptom	Why it matters analytically
Early breakdown	Wrong branch or wrong object selected before meaningful state change	Indicates misunderstanding of the task logic rather than near-miss closure.
Incomplete closure	Main repair direction is correct, but one final side effect is absent	Distinguishes almost-finished rollouts from fully incorrect ones.
Stale-state mishandling	Valid state is overwritten, or stale state is left in place while new state is added	Captures failures in state conflict resolution rather than in simple completion.
Hallucinated completion	Agent stops after partial repair and behaves as if the workflow were done	Explains why some runs earn high score but still fail strict evaluation.
Excess-step inefficiency	Rollout consumes many extra steps without proportionate progress toward final closure	Exposes weak workflow efficiency rather than just low pass rate.

C.3 Length and Efficiency Notes

The previous subsections examine what models get wrong; this subsection examines how efficiently they use their interaction budget. The length-stratified analysis in the main paper pairs reference-trajectory length with average executed steps. A useful interpretation is that GT length measures the structural size of the intended workflow, while executed steps measure how much interaction budget a model actually spends on that workflow. These quantities need not match closely. A model can be step-efficient but still miss one final operation, or it can consume many additional steps without improving final closure.

This distinction is particularly important for stateful tasks. In many ClawForge-Bench episodes, extra steps are not neutral: they often correspond to repeated inspection, redundant list queries, or additional state mutations that do not improve the final evaluator-visible state. The benchmark therefore treats step usage as a behavioral signal about workflow efficiency, not just as an implementation detail of the rollout trace.

D DATA AND REPRODUCIBILITY

D.1 DATA PROVENANCE AND PROVIDER IMPACT

The main experimental tables report outcomes over one shared ClawForge-Bench snapshot. For Kimi-K2.5, GPT-5.2, GPT-o3, GPT-5.3-Codex, GPT-5.4, and Claude Opus 4.6, strict accuracy, partial credit, and length-bucket step averages are computed from task-level traces over the evaluated run set. Claude Sonnet 4.6 aggregate accuracy uses the same task denominator, but its length-bucket step values come from the available task-level traces in a merged run. Consequently, we treat those values as descriptive efficiency evidence rather than a clean latency comparison.

For GPT-5.2, o3, GPT-5.3-Codex, and GPT-5.4, overall partial credit is taken from the original run summary; ability-level partial credit is recomputed from the available task scores, which are logged to two decimal places. Small rounding differences between the weighted ability means and overall scores are therefore expected. All GPT scenario pass counts and reference-length statistics are recomputed from task-level records.

The newer GPT endpoints use the same frozen 362 task specifications in `multi` mode with full history, medium reasoning effort, the Responses API, `store=false`, a 20-step interaction cap, and a 1,024-token output cap. GPT-6 Astra, GPT-5.6 Sol, GPT-5.5, and GPT-5.6 Luna each have one latest clean record per task and no provider-impacted final records; task, specification, and split fingerprints match across all four runs. Requests were routed through a local Codex2API compatibility gateway backed by ChatGPT OAuth, not the official OpenAI API. The GPT-5.x runs used gateway commit `fcc1b1c43eb2b8f929844b7f676bc00aeb5ffb5d`, while the later Astra run used `b1cd0f7fa56d98e9712ccae2942fc7b443e0929a`; response metadata was audited to require the requested model ID on every clean record. Delays of 3 seconds between tasks and 1.5 seconds between agent steps were used for quota stability, so we do not compare wall-clock latency.

For artifact identity, the newer-GPT manifests record the following SHA-256 fingerprints:

- ordered task IDs: `6bf820fb9e365beb287987a23d324a9d619613d68e896adabd3e84bc0884c4b7`,
- task specifications: `5d63cb7f50bd9fd6d327b204952f78a6f0451afdce81f02f69f074162ffb6b5a`,
- total split file: `58f744a9120717420f9748ccca5231a36418936e4915276067a11d14502c4b86`.

A deterministic reference-consistency audit found seven tasks whose provided solution commands omit a required transition from the default model to the evaluator-required model: `hard_decision_workflow_44`, `48`, `49`, `57`, `64`, `68`, and `69`. The expert replay and all four newer-GPT endpoints fail exactly these seven audited tasks. We nevertheless retain the complete 362-task snapshot as the primary denominator, matching the main study and avoiding a post hoc change of evaluation population. As a sensitivity check, removing those tasks yields 62.8% for Astra, 58.9% for Sol, 52.1% for GPT-5.5, and 41.7% for Luna; pass counts and model ordering are unchanged.

Provider-aware reporting is included because benchmark runs over external model endpoints can introduce retries, filtered outputs, or fallback behavior that affect an otherwise valid interactive rollout. In the reporting scheme, `provider_failures` counts tasks that fail outright because a provider-side issue prevents a normal completion. `provider_impacted_tasks` counts tasks whose trajectories complete but are materially disturbed by provider-side effects such as compact fallbacks or filtered responses. These statistics are explanatory rather than primary: they do not replace full-pass accuracy or partial-credit score, but help readers judge whether serving noise is large enough to affect a comparison. The intended reading order is therefore to examine accuracy and score first, use the length table to interpret closure efficiency, and then use provider-aware statistics to determine whether a model's execution trace was unusually noisy at the serving layer.

D.2 Limitations and Scope

ClawForge-Bench focuses on state-conflict workflows in controlled command-line environments. This scope supports reproducible analysis of workflow behavior, while extending coverage to additional domains and longer-horizon settings remains valuable future work. The reported results use a shared task snapshot; repeated evaluations would further characterize run-to-run variability. Serving paths and interaction budgets vary across endpoints, so small performance differences should be interpreted in light of these configurations. The following paragraphs provide additional context on benchmark scope and evaluation conditions.

The reported model comparisons are benchmark evaluations over one shared ClawForge-Bench snapshot rather than variance estimates over many independently repeated runs. This matters especially for externally served agents, where provider-side retries, filtered outputs, and compact-fallback behavior can perturb a rollout even when the task specification is fixed. The paper surfaces those effects through provider-aware accounting, but it does not eliminate them.

The hard suite emphasizes result-first closure under persistent state, so some conclusions are benchmark-specific by construction. The strongest separations in this paper come from duplicate-sensitive completion, stale-state repair, replacement, and branch resolution. Other capabilities that matter in agent deployment, such as open-ended research, web navigation, or multimodal perception, are outside the scope of this benchmark family.

The Sonnet 4.6 reporting line is not perfectly symmetric with the other entries. Its strict-accuracy value uses the shared task denominator, but some diagnostic step statistics rely on the available merged best-available traces rather than one fully uniform trace set. The main text and Section D.1 therefore treat those step counts as efficiency evidence, not as a clean latency benchmark.

Execution configurations differ across endpoints: the newer GPT models use a 20-step rather than 25-step budget and a distinct compatibility serving path. The Astra run also uses a later revision of that gateway than the GPT-5.x runs. These differences are documented in §D.1 and should be considered when interpreting small cross-configuration gaps. Each newer-GPT result contains one completed rollout per model-task pair, so paired intervals measure task-set sampling uncertainty, not decoding or repeated-run variance.

D.3 Statistical Significance

For the earlier endpoints, we report observed outcomes on a fixed task snapshot without attaching confidence intervals to adjacent aggregate rankings. Large gaps tied to stable scenario families, such as the persistent difficulty of replacement-style tasks or the near-saturation of already-done and duplicate-avoidance slices, are therefore more important to the benchmark argument than tiny differences between neighboring models.

For GPT-5.5, GPT-5.6 Luna, GPT-5.6 Sol, and GPT-6 Astra, we additionally report paired nonparametric bootstrap intervals over all 362 task IDs using 10,000 resamples and a fixed seed of 42. The strict-accuracy differences are 3.9 percentage points for GPT-6 Astra minus GPT-5.6 Sol (95% CI: -0.8 to 8.6), 6.6 points for Sol minus GPT-5.5 (2.2 to 10.8), 10.2 points for GPT-5.5 minus Luna (5.5 to 14.9), and 20.7 points for Astra minus Luna (15.5 to 26.0). Because there is one rollout per model-task pair, these intervals quantify task-set sampling uncertainty only; they do not capture provider, decoding, or repeated-run stochasticity.

D.4 Compute Resources

The reported experiments do not train new models. The local benchmark workload is therefore orchestration-heavy rather than training-heavy: task reset, routed command execution, evaluator-state construction, and result aggregation all run through the ClawForge runtime in `multi` mode with full interaction history. Seven endpoints use a 25-step budget and four use 20 steps. In both configurations, the benchmark runner itself is CPU-oriented and does not require local GPU execution; model inference is supplied by externally served endpoints.

E Broader Impact and Release

E.1 Broader Impact

Positive impact. This work improves measurement quality for interactive agents. Many real-world failures arise from state conflicts rather than one-shot misunderstanding. By explicitly capturing failure modes such as duplication, stale state, incorrect branching, and incomplete workflow closure, the benchmark enables more precise diagnosis of where agents remain unreliable.

Risks. Improved benchmark coverage may indirectly accelerate deployment of agents in coordination, operations, or messaging workflows, where incorrect execution can incur real-world cost. Benchmark progress may also be misinterpreted as readiness for broader automation beyond the evaluated scenarios.

Misuse and mitigation. Workflow benchmarks can be used to optimize agents that manipulate calendars, messages, files, or system state, including in settings with limited oversight. To mitigate this, the benchmark is built on synthetic or seeded environments, emphasizes transparent reporting of failure modes, and frames evaluation as a diagnostic tool rather than evidence of safe autonomous deployment.

E.2 Asset Credits and Terms

The paper uses two main classes of existing assets: prior benchmark literature and externally served foundation-model endpoints. Prior interactive-benchmark and agent-evaluation papers are cited in the introduction, method, experiment, and related-work sections. The evaluated models are likewise credited by provider and citation: Kimi-K2.5 (Team et al., 2026) to the Kimi team, GPT models (OpenAI, 2026a;b;d;c;e) to OpenAI, and Claude models (Anthropic, 2024) to Anthropic.

These evaluated models are not treated as open-weight assets in this paper. They are proprietary provider endpoints accessed under the corresponding provider's service terms rather than redistributed model weights. The paper therefore makes a sharper distinction than many open-model evaluations: for these systems, the relevant usage condition is provider-controlled API or hosted-access terms, not an open-source model license embedded in the benchmark package itself.

The new benchmark artifacts described in this paper are ClawForge task specifications, seeded state, evaluator contracts, and reporting views. They are documented as new assets of the study rather than borrowed benchmark files relabeled as original contributions. The paper does not claim to redistribute third-party model weights, proprietary datasets, or third-party application data snapshots.

E.3 Open Access Release Status

This submission documents the benchmark protocol, task structure, evaluator contract, and reporting methodology in enough detail to support scientific interpretation, but it does not attach an anonymized open-access release bundle for the benchmark snapshot and code at submission time. That is a release-status fact, not a hidden omission.

The practical reason is that a faithful release package for this benchmark is more than a PDF appendix: it must include task specifications, state initialization assets, command-routing runtime code, evaluator definitions, and clear instructions for running the shared hard snapshot against supported model providers. The paper already documents those components conceptually, but the anonymized distribution bundle is not part of the present submission package.

F Full Scenario Catalog

F.1 Scenario Example Directory

Table 10: Directory of the full scenario example catalog. Page references point to the start of each long example.

Scenario	Page	Scenario	Page
Inbox	23	Multi-Source Decision	28
Release Recovery Runbook	24	State Repair	29
Channel Incident Recovery	24	Completion Gap	29
Daily Operations Commitment Loop	25	Branch Resolution	30
Release Gate	25	Already Done Skip	30
Delivery Update	26	Wrong-State Replacement	31
Operations Review	26	Interrupted Workflow Resume	31
Existing State	27	Contradictory Source Resolution	32
Duplicate Avoidance	27		

F.2 Full Scenario Example Catalog

The following catalog now uses one concrete representative task instance for every released scenario family. Each box is grounded in one actual `hard_decision_workflow_xxx` specification rather than in a scenario-level summary, so the instruction wording, seeded state, representative commands, and evaluator-visible closure conditions all come from a real released task.

Example A: Inbox (`hard_decision_workflow_1`)

Representative instance. `hard_decision_workflow_1` asks: "The budget email thread in the inbox needs attention today. Then check the Berlin board and calendar; right now the next step is already on the board, and if it only needs async follow-up, leave the calendar alone, so fill only what is missing, note it in a handoff file, and send alice@example.com a quick update."

Initial state. The seeded board already contains the pending task `Berlin budget follow-up`. The task instruction also implies that the follow-through should stay async if the existing calendar state already supports that branch, so the missing work is the handoff note plus the external update rather than a new calendar block.

Required correction. Read the budget thread, verify that the staged Berlin task can stay in place, avoid adding a duplicate task or review event, create the handoff note under `/handoff`, and send the requested email update to `alice@example.com`.

Representative actions.

1. `email search --query 'budget'`
2. `email read --id email_seed_3`
3. `tasks list --status pending`
4. `calendar list`
5. `calendar today --timezone Europe/Berlin`
6. `file create --path '/handoff/berlin-budget-followup.txt' --content 'Berlin budget follow-up handoff note.'`
7. `email send --to alice@example.com --subject 'Budget follow-up for Berlin' --body 'Tracking the next step for Berlin.'`

Evaluation outcome. The evaluator accepts this rollout only if the handoff file is created, the update email is sent, the pre-existing Berlin follow-up task remains sufficient, and no duplicate task or calendar event is introduced. This makes the task a real inbox-grounded completion problem rather than a generic email summary.

Example B: Release Recovery Runbook (`hard_decision_workflow_7`)

Representative instance. `hard_decision_workflow_7` asks: "Please tighten up the Berlin release review. Check what's already on the board and calendar, use anthropic/claude-opus-4-6, refresh the handoff file, and add only the missing review slot or decision-log task."

Initial state. The override layer already stages a pending Berlin release next-step task, so the workflow is not empty. What is missing is the corrected review configuration under the required model path together with a refreshed handoff artifact.

Required correction. Confirm the current model, switch to `anthropic/claude-opus-4-6`, preserve the existing Berlin release next step, refresh the handoff file, and add only the missing Berlin review slot rather than recreating the release workflow from scratch.

Representative actions.

1. `openclaw config get agent.model`
2. `tasks list --status pending`
3. `calendar list`
4. `calendar today --timezone Europe/Berlin`
5. `openclaw models set anthropic/claude-opus-4-6`
6. `file create --path '/ops/release-handoff.txt' --content 'Berlin release handoff notes.'`
7. `calendar add-event --title 'Berlin release review' --start 2026-03-10T09:00`

Evaluation outcome. A passing rollout must visibly set the model to the required Opus path, create the handoff file, keep the existing decision-log reminder and board task intact, and place the Berlin review event on the calendar. The task fails if the model switch is skipped or if valid staged state is rebuilt unnecessarily.

Example C: Channel Incident Recovery (`hard_decision_workflow_13`)

Representative instance. `hard_decision_workflow_13` asks: "Please get the incident update to #general on discord, check the incident next step on the board first to see whether it is already there, add only the missing piece, and send alice@example.com a recap when it's handled."

Initial state. The board already contains the pending task `Incident escalation follow-up`. The missing work is therefore the externally visible incident post plus the recap email, not another board task.

Required correction. Audit the task board first, post the incident escalation update to Discord `#general`, avoid recreating the follow-up task that already exists, and send the requested recap to `alice@example.com`.

Representative actions.

1. `openclaw security audit`
2. `tasks list --status pending`
3. `openclaw channels login --channel discord`
4. `openclaw message send --channel discord --target #general --message 'Incident escalation started. Please acknowledge.'`
5. `openclaw channels list --json`
6. `email send --to alice@example.com --subject 'Incident escalation recap' --body 'The escalation is posted and being tracked.'`

Evaluation outcome. Strict evaluation requires both external side effects: the Discord message to `#general` and the recap email. It also checks that the existing follow-up task remains the only board artifact. This makes the task a clean example of information transfer plus duplicate-aware closure.

Example D: Daily Operations Commitment Loop (`hard_decision_workflow_19`)

Representative instance. `hard_decision_workflow_19` asks: "Can you check today's Europe/Berlin schedule for Berlin, look at the next step on the board and the recurring daily check, and only fill the missing piece?"

Initial state. The seeded state already contains `Berlin existing ops next-step task`. No recurring daily-check cron is present yet, so only the recurring commitment is missing.

Required correction. Inspect the Berlin schedule, verify the existing next-step task, check the current cron state, and add only the missing recurring daily-check job without touching the valid board state.

Representative actions.

1. `weather forecast --location 'Berlin' --days 3`
2. `calendar today --timezone Europe/Berlin`
3. `tasks list --status pending`
4. `openclaw cron list`
5. `openclaw cron add --name hard-ops-01 --cron '0 9 * * *' --message 'Run Berlin daily ops check'`

Evaluation outcome. This rollout passes only if the Berlin next-step task remains in place and exactly one new cron job is created for the daily ops check. It fails when the model duplicates board work, adds the wrong recurring artifact, or stops after inspection without committing the missing scheduler state.

Example E: Release Gate (`hard_decision_workflow_43`)

Representative instance. `hard_decision_workflow_43` asks: "Please get the Singapore release gate sorted out. Confirm the current model, review the board and calendar first, then update the handoff file, the next step on the board, and the sync on the calendar."

Initial state. This instance does not preload the release artifacts through overrides, so the full gate closure must be assembled from inspection plus creation. The hidden constraints require a specific model path, a handoff file, a release-gate follow-through task, and a matching sync.

Required correction. Confirm the current model, inspect board and calendar state, switch to `anthropic/claude-3-7-sonnet-latest`, create the release handoff file, create the Singapore release-gate task, and place the release-gate sync on the calendar.

Representative actions.

1. `openclaw config get agent.model`
2. `tasks list --status pending`
3. `calendar list`
4. `calendar today --timezone Asia/Singapore`
5. `openclaw models set anthropic/claude-3-7-sonnet-latest`
6. `file create --path '/ops/release-handoff.txt' --content 'Singapore release gate handoff notes.'`
7. `tasks add --title 'Singapore release gate follow-through' --priority high --due 2026-03-08`
8. `calendar add-event --title 'Singapore release gate sync' --start 2026-03-10T09:00`
9. `calendar today --timezone Asia/Singapore`

Evaluation outcome. Evaluation checks all four closure outputs together: model set, handoff file created, task created, and calendar event created. The task therefore exposes whether the agent can carry a release-gate workflow through to a fully consistent final state rather than stopping after one local fix.

Example F: Delivery Update (`hard_decision_workflow_73`)

Representative instance. `hard_decision_workflow_73` asks: "Can you get the outage update out to #launch on discord? Fix the delivery path if needed, check what's already on the board, the next step may already be on the board, so leave it alone if it is there, if the target is shared, leave a short live block on the calendar, and send bob@example.com a quick recap."

Initial state. The board already contains `Outage delivery next step`. The required shared-target branch also expects a short live calendar block, but neither the Discord post nor the recap email has been sent yet.

Required correction. Leave the staged outage next step alone, post the outage update to Discord `#launch`, add the live follow-up calendar block, and send the recap to `bob@example.com`.

Representative actions.

1. `tasks list --status pending`
2. `openclaw channels list --json`
3. `openclaw channels login --channel discord`
4. `calendar add-event --title 'Outage delivery follow-up' --start 2026-03-12T14:00`
5. `openclaw message send --channel discord --target #launch --message 'Outage update posted and being tracked.'`
6. `email send --to bob@example.com --subject 'Outage delivery recap' --body 'The delivery path is recovered and the update is posted.'`

Evaluation outcome. The evaluator requires all three visible side effects while also enforcing that no duplicate board task is created. This task is representative because it mixes messaging, scheduling, and duplicate-aware state preservation inside one delivery workflow.

Example G: Operations Review (`hard_decision_workflow_83`)

Representative instance. `hard_decision_workflow_83` asks: "Can you look at the forecast and today's Europe/London calendar for London, review the next step on the board and the recurring daily check, and only add what's missing before you put the review on the calendar?"

Initial state. The seeded state already contains `London existing ops next-step task`. The missing pieces are the recurring daily-check cron and the review event itself, both conditioned on current forecast and London calendar context.

Required correction. Read the forecast, inspect the London calendar, confirm the existing ops task, create the missing daily-check cron, and place the `London backup ops review` event on the calendar.

Representative actions.

1. `weather forecast --location 'London' --days 1`
2. `calendar today --timezone Europe/London`
3. `openclaw cron list`
4. `tasks list --status pending`
5. `openclaw cron add --name hard-ops-07-01 --cron '15 8 * * 1-5' --message 'Run London daily ops check'`
6. `calendar add-event --title 'London backup ops review' --start 2026-03-10T13:00`

Evaluation outcome. A passing rollout must keep the existing task intact while creating both the cron job and the review event. The task fails if the model schedules a review without completing the missing recurring setup or if it adds duplicate board state while trying to be safe.

Example H: Existing State (`hard_decision_workflow_113`)

Representative instance. `hard_decision_workflow_113` asks: "The Boston daily ops setup is only partway done. Look over the next step on the board, the review slot on the calendar, and the recurring daily check, then finish just the missing piece and send alice@example.com a short recap."

Initial state. The override layer preloads both `Boston existing ops next-step task` and `Boston existing ops review block`. The only missing piece is the recurring daily-check cron, after which a recap email must be sent.

Required correction. Audit the partially completed Boston ops setup, preserve the existing task and review block, create only the missing cron job, and send the recap to `alice@example.com`.

Representative actions.

1. `openclaw cron list`
2. `tasks list --status pending`
3. `calendar list`
4. `calendar today --timezone America/New_York`
5. `openclaw cron add --name existing-hard-ops-01 --cron '30 9 * * 1-5' --message 'Run Boston daily ops check'`
6. Send the Boston ops recap email to `alice@example.com`.

Evaluation outcome. Strict evaluation checks that the existing task and review block remain present, no duplicate task or calendar event appears, the cron job is created, and the recap email is sent. This makes the example a concrete existing-state completion task rather than a vague recurring-workflow summary.

Example I: Duplicate Avoidance (`hard_decision_workflow_123`)

Representative instance. `hard_decision_workflow_123` asks: "Some of the London ops setup is already in place. Check the board, forecast, and calendar, then add a backup review block on the calendar if the forecast looks risky; otherwise add a primary review block on the calendar, and avoid recreating the existing setup."

Initial state. The workflow already contains `London existing ops next-step task` and an active recurring cron `london-existing-daily-ops-check`. The only missing artifact is the review block, whose title depends on the forecast branch.

Required correction. Inspect the existing board and cron state, read the London forecast and calendar, choose the backup review branch supported by the task instance, and add only the `London backup ops review block` without recreating existing setup.

Representative actions.

1. `openclaw cron list`
2. `tasks list --status pending`
3. `calendar list`
4. `weather forecast --location 'London' --days 1`
5. `calendar today --timezone Europe/London`
6. `calendar add-event --title 'London backup ops review block' --start 2026-03-20T08:30`

Evaluation outcome. The evaluator rewards recognition that most of the workflow already exists. It checks that the board task and daily check stay in place, no duplicate task or cron job is added, and the review event is created with a backup-oriented title.

Example J: Multi-Source Decision (`hard_decision_workflow_133`)

Representative instance. `hard_decision_workflow_133` asks: "For Austin, check the review email note, the forecast, and the America/Chicago calendar, decide whether this stays live or shifts async, leave the next step on the board, keep the daily check scheduled, put the review on the calendar if live, and send leadership@example.com an update only if async."

Initial state. This Austin instance begins without task-specific overrides. The closure branch must be inferred from the review email note, the forecast, and the America/Chicago calendar, and the live branch requires creating the task, cron job, and calendar review event.

Required correction. Read the review email note, inspect the forecast and Chicago calendar, decide that the workflow stays live, create the Austin next-step task, keep the daily check scheduled, and place the `Austin primary ops review` event on the calendar.

Representative actions.

1. `weather forecast --location 'Austin' --days 1`
2. `calendar today --timezone America/Chicago`
3. `email search --query 'review'`
4. `email read --id email_seed_5`
5. `openclaw cron list`
6. `tasks list --status pending`
7. `tasks add --title 'Austin ops next step' --priority medium --due 2026-03-08`
8. `openclaw cron add --name multi-source-hard-ops-01 --cron '0 9 * * *' --message 'Run Austin daily ops check'`
9. `calendar add-event --title 'Austin primary ops review' --start 2026-03-10T09:00`

Evaluation outcome. The evaluator requires task creation, cron creation, and review-event creation together. A rollout that sends an async email instead of committing to the live branch, or that keeps inspecting without deciding, still fails even if the intermediate reasoning looks plausible.

Example K: State Repair (`hard_decision_workflow_153`)

Representative instance. `hard_decision_workflow_153` asks: "The Boston release setup has something stale in it. Review the next-step task on the board and the sync on the calendar, repair the stale next step, refresh the handoff file, and leave the next step and sync correctly set."

Initial state. The seeded state contains a stale low-priority task, `Existing Boston release follow-up`, under the `hard_wrong_task` setup. The correct release follow-through, handoff refresh, and Boston release sync do not yet exist.

Required correction. Detect that the seeded Boston follow-up is stale, refresh the handoff file, complete the outdated task, create the corrected high-priority task `Boston release follow-through`, and add the `Boston release sync` calendar event.

Representative actions.

1. `tasks list --status pending`
2. `calendar list`
3. `calendar today --timezone America/New_York`
4. `file create --path '/ops/release-handoff.txt' --content 'Boston refreshed release handoff notes.'`
5. `tasks complete --title 'Existing Boston release follow-up'`
6. `tasks add --title 'Boston release follow-through' --priority high --due 2026-03-16`
7. `calendar add-event --title 'Boston release sync' --start 2026-03-19T15:00`

Evaluation outcome. This rollout passes only if the stale object is actually retired and both replacement artifacts are created. It fails when the model notices the conflict but leaves the outdated task in place, adds the new task without cleanup, or stops after a partial repair and behaves as if the workflow were complete.

Example L: Completion Gap (`hard_decision_workflow_199`)

Representative instance. `hard_decision_workflow_199` asks: "Part of the Seattle release work is already handled, but the next step is the missing piece. Check what's there, read the release notes, refresh the handoff file, and leave the sync on the calendar in place and add only the missing next step on the board."

Initial state. The override layer preloads the review note `/ops/release-review.txt` and an existing Seattle release sync on the calendar. What is missing is the board next step plus the refreshed handoff file.

Required correction. Read the drafted release notes, preserve the existing Seattle sync, create the handoff file, and add only the missing high-priority task `Seattle release next step` on the board.

Representative actions.

1. `openclaw config get agent.model`
2. `tasks list --status pending`
3. `calendar list`
4. `calendar today --timezone America/Los_Angeles`
5. `file read --path '/ops/release-review.txt'`
6. `file create --path '/ops/release-handoff.txt' --content 'Seattle completion-gap release handoff notes.'`
7. `calendar list --from 2026-03-10 --to 2026-03-10`
8. `tasks add --title 'Seattle release next step' --priority high --due 2026-03-08`

Evaluation outcome. Evaluation checks exactly that structure: the handoff file must be created, the release sync must remain an already-existing artifact, no duplicate calendar event may be created, and the new task must appear. This is a real gap-completion example where the missing piece is the board task rather than the calendar sync.

Example M: Branch Resolution (`hard_decision_workflow_211`)

Representative instance. `hard_decision_workflow_211` asks: "The Sydney ops setup already has part of the work in place. Please review the forecast, today's Australia/Sydney calendar, the task already on the board, and the recurring daily check schedule before working out whether this stays live or moves async, and send bob@example.com the update if it moves async."

Initial state. The seeded state already contains `Sydney existing ops next-step task` and an active daily-check cron. The task instance resolves to the async branch, so the live review path should not be scheduled.

Required correction. Inspect the Sydney forecast, calendar, board task, and cron state; decide that the workflow moves async; preserve the existing task and daily check; and send the backup-plan email to `bob@example.com` instead of creating a calendar review block.

Representative actions.

1. `weather forecast --location 'Sydney' --days 1`
2. `calendar today --timezone Australia/Sydney`
3. `openclaw cron list`
4. `tasks list --status pending`
5. Send the Sydney backup-plan email to `bob@example.com`.

Evaluation outcome. The evaluator checks the preserved task and cron state, the absence of duplicate board or cron artifacts, the email side effect, and the absence of a duplicate calendar event. This makes the task a concrete branch-resolution example instead of a generic live-vs-async summary.

Example N: Already-Done Skip (`hard_decision_workflow_251`)

Representative instance. `hard_decision_workflow_251` asks: "The New York release setup should already be in place. Please verify the current model, the next-step task on the board, and the sync on the calendar before you send finance@example.com a short recap and leave it alone if it is already right."

Initial state. Both key artifacts are already present through overrides: `New York existing release next step` on the board and `New York existing release sync` on the calendar. The only remaining obligation is to verify the state and send the finance recap.

Required correction. Confirm the current model, inspect the board and calendar, recognize that no repair is needed, and send the requested recap to `finance@example.com` without creating any new task or event.

Representative actions.

1. `openclaw config get agent.model`
2. `tasks list --status pending`
3. `calendar list`
4. `calendar today --timezone America/New_York`
5. Send the New York release recap email to `finance@example.com`.

Evaluation outcome. This task passes only when the model resists unnecessary mutation. The evaluator checks the model setting, the existing task and sync, the recap email, and the absence of duplicate task or calendar creations. It is therefore a real recognition-and-restraint task, not an empty no-op.

Example O: Wrong-State Replacement (`hard_decision_workflow_261`)

Representative instance. `hard_decision_workflow_261` asks: "Something in the Seattle release setup was staged wrong. Check whether the next step is stale, replace the stale next step with the correct version, and leave the next step and sync correctly set."

Initial state. Under `hard_wrong_task`, the workflow starts with `Existing Seattle release follow-up`, a low-priority stale task that should not survive in the final state. The replacement task and replacement sync are both still missing.

Required correction. Inspect the seeded Seattle task, determine that it is stale, complete it, create the corrected task `Seattle release replacement next step`, and add the matching `Seattle replacement release sync` event.

Representative actions.

1. `tasks list --status pending`
2. `calendar list`
3. `calendar today --timezone America/Los_Angeles`
4. `tasks complete --title 'Existing Seattle release follow-up'`
5. `tasks add --title 'Seattle release replacement next step' --priority high --due 2026-03-09`
6. `calendar add-event --title 'Seattle replacement release sync' --start 2026-03-10T13:00`

Evaluation outcome. Strict evaluation requires both retirement of the stale task and creation of the corrected replacement state. Adding the replacement without cleanup, or cleaning up without recreating the follow-through, still fails. That is why this slice remains one of the hardest state-conflict settings in the benchmark.

Example P: Interrupted Workflow Resume (`hard_decision_workflow_307`)

Representative instance. `hard_decision_workflow_307` asks: "The New York release work already has pieces in place. Check the current model, the next step on the board, and the sync on the calendar, finish the missing handoff file, send manager@example.com a recap, and leave the existing setup alone."

Initial state. The seeded release state already contains `New York existing release next step`, `New York existing release sync`, and drafted notes in `/ops/release-review.txt`. The only missing artifact is the handoff file, after which a recap email is required.

Required correction. Resume from the partially completed New York release setup, read the existing review notes, create the missing handoff file, preserve the staged task and sync, and send the recap to `manager@example.com`.

Representative actions.

1. `openclaw config get agent.model`
2. `tasks list --status pending`
3. `tasks search --query 'New York release'`
4. `calendar list`
5. `calendar today --timezone America/New_York`
6. `file read --path '/ops/release-review.txt'`
7. `file create --path '/ops/release-handoff.txt' --content 'New York resumed release handoff notes.'`
8. Send the New York resume recap email to `manager@example.com`.

Evaluation outcome. The evaluator accepts only resume-style closure: the task and sync must remain existing artifacts, no duplicates may be created, the handoff file must appear, and the recap email must be sent. Restarting the workflow from scratch or mutating already-correct state still fails.

Example Q: Contradictory Source Resolution (`hard_decision_workflow_337`)

Representative instance. `hard_decision_workflow_337` asks: "Please reconcile the latest review email note, the forecast, and today's Europe/London calendar for London. Leave the next step on the board, keep the recurring daily check scheduled, and either put the review on the calendar or send bob@example.com the async update."

Initial state. This London instance begins without seeded overrides, so all action depends on reconciling the latest review email, the forecast, and the current London calendar. The correct branch here is async: the task and recurring check should be created, but no review event should be added.

Required correction. Read the latest review email note, inspect forecast and calendar evidence, create `London contradictory-source next step`, add the recurring contradiction-check cron, and send the async update to `bob@example.com` while avoiding a calendar review block.

Representative actions.

1. `email search --query 'review'`
2. `email read --id email_seed_5`
3. `weather forecast --location 'London' --days 1`
4. `calendar today --timezone Europe/London`
5. `calendar list --from 2026-03-01 --to 2026-03-01`
6. `tasks list --status pending`
7. `tasks add --title 'London contradictory-source next step' --priority high --due 2026-03-11`
8. `openclaw cron add --name contradictory-source-01 --cron '0 18 * * 1-5' --message 'Run London daily contradiction check'`
9. Send the London async update email to `bob@example.com`.

Evaluation outcome. The evaluator checks exactly those branch-specific side effects: task created, cron created, email sent, and no duplicate calendar event. This makes the example a genuine contradictory-evidence resolution task rather than a generic multi-source lookup problem.