CARET: EVIDENCE-GROUNDED COMPILATION OF VERIFIABLE AGENT TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Engineering workflow traces capture task requirements, tool interactions, and observed workspace changes. Turning these traces into reusable training tasks requires recovering missing context and reconstructing an unresolved state while preserving the source objective. We introduce CARET, an evidence-grounded framework for compiling these traces into executable agent tasks. A shared evidence contract links source requirements, workspace facts, unresolved dependencies, and verifier properties. It guides the joint construction of the initial workspace, instruction, verifier, and reference solution. Admission checks evaluate the initial state, reference repair, and a designated surface-only repair in separate fresh sandboxes. Execution feedback guides bounded revisions, and revised candidates must pass renewed checks before admission. We construct a corpus of 10,139 tasks and collect 24,187 trajectories. The task format supports demonstration collection for supervised fine-tuning (SFT) and resettable interaction for reinforcement learning (RL), with disjoint source-group partitions. Experiments compare CARET with OpenThoughts-Agent and ClawGym at 4B-Instruct, 8B, and 14B scales after SFT and subsequent RL. CARET achieves the highest PinchBench and WildClawBench pure-text scores in all six scale–stage comparisons. After RL, its margins over the strongest baseline at each scale are 2.00–2.40 and 2.40–2.70 percentage points, respectively. Component ablations support the training utility of the evidence contract, joint construction, and execution-guided revision.

1 INTRODUCTION

Language-model agents carry out workflows that combine file inspection, data analysis, tool use, and changes to persistent workspace state. Recent benchmarks evaluate these capabilities across software maintenance, command-line tasks, and everyday assistant work (Merrill et al., 2026; Ye et al., 2026; Kilo AI Team, 2026a; Ding et al., 2026). Training for such workflows requires executable tasks: an agent needs a meaningful starting state, opportunities to act, and feedback that distinguishes successful completion from a plausible-looking output.

Engineering workflow traces record task requests, tool interactions, and observed workspace changes, providing evidence for reusable tasks. However, a historical trace does not directly provide a reusable training environment. Its files or dependencies may be missing, private services may be unavailable, and the recorded workspace may already contain the solution. Reconstructing the components independently can also change the problem: the instruction may request an outcome that the workspace cannot support, or the verifier may reward an artifact without checking the dependency that makes it correct. The resulting bottleneck is preserving the source objective while jointly constructing an unresolved workspace and an evidence-supported success criterion.

Existing approaches obtain executable training data from several sources. SWE-Gym (Pan et al., 2025) packages repository issues with runtime environments and tests. Terminal-Task-Gen (Pi et al., 2026) synthesizes tasks from skills, while RST (Li et al., 2026) evolves verified task bundles by extending their solutions. TerminalWorld (Chu et al., 2026) and Terminal-Universe (Wu et al., 2026a) recover tasks or environments from recorded behavior. These approaches establish the value of executable synthesis and reconstruction. Our focus is how source traces constrain which changes preserve the recorded objective and which reconstructed facts may determine reward.

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.

Our insight is that the same source trace should constrain the task components *jointly*. Each scored property needs a traceable connection to an observed requirement, a recoverable workspace fact, and a dependency the agent must resolve. This motivates *evidence-constrained task synthesis*: preserve the workflow objective while constructing an unresolved, executable problem whose instruction and verifier follow the same evidence. A shared contract makes these relations explicit, so a missing prerequisite or an unsupported scoring requirement can be identified during construction.

We propose CARET (*Compiling Agent Records into Executable Tasks*), a framework that implements this idea. CARET extracts an evidence contract from source traces, recovers the necessary workspace, and jointly constructs an unresolved state, instruction, verifier, and reference repair. Fresh-sandbox executions check the unchanged workspace, the complete repair, and a surface repair that leaves the central dependency unresolved. Execution feedback supports bounded revision before admission. Accepted tasks supply both verified demonstrations for supervised fine-tuning (SFT) and resettable environments for reinforcement learning (RL).

We evaluate the resulting corpus through coverage and fixed-solver analyses, development checks, and downstream training on three agent benchmarks. Table 1 compares OpenThoughts-Agent, ClawGym, and CARET using Qwen models at 4B-Instruct, 8B, and 14B scales after SFT and SFT followed by RL. CARET obtains the highest PinchBench and WildClawBench pure-text scores within each scale and stage. Its 4B-Instruct model reaches 59.34 on ClawEval-General, 62.40 on PinchBench, and 26.30 on WildClawBench pure text after RL. The comparison also reveals metric-specific trade-offs: CARET does not lead every ClawEval dimension or every aggregate comparison. Construction analyses and component ablations in Section 5 examine the contributions of the evidence contract, joint construction, and execution-guided revision.

Our contributions are threefold:

- **Evidence-constrained joint task compilation.** CARET compiles source traces into executable agent tasks. A shared evidence contract links source requirements, workspace facts, unresolved dependencies, and verifier properties, constraining the joint construction of the initial unresolved workspace, instruction, verifier, and reference solution.
- **Execution-guided validation and revision.** We check the initial state, a reference repair, and a designated surface-only repair in separate fresh sandboxes. Execution feedback guides bounded revisions to task components; revised candidates undergo renewed static and execution checks before admission.
- **A reusable task corpus and its training utility.** We construct 10,139 executable tasks and collect 24,187 trajectories. The task format supports demonstration collection for SFT and resettable interaction for RL, with source-group-disjoint partitions. Experiments at 4B-Instruct, 8B, and 14B scales evaluate downstream training utility; component ablations assess the evidence contract, joint construction, and execution-guided revision.

2 PRELIMINARIES

We introduce the source traces, the Harbor task interface, and the notation used to describe CARET's construction and validation procedure in Section 3.

Typed traces. An input trace is a de-identified sequence $\tau = (e_1, \ldots, e_n)$ of messages, tool actions, and observations in temporal order. It records the source workflow; policy rollouts are new interactions with a constructed task.

Harbor task format. Each output follows the Harbor (Harbor Framework Team, 2026) task interface:[1] `task.toml` specifies runtime configuration; `instruction.md` states the public objective; `environment/` defines the environment build; `tests/test.sh` runs the verifier; and `solution/solve.sh` supplies a reference solution. Harbor makes the reference solution optional; CARET includes it for construction-time validation. Policy rollouts expose the instruction and initialized workspace to the agent while withholding the verifier and reference solution. Verification follows the rollout.

[1] Official documentation: task format, reference solutions, and verification.

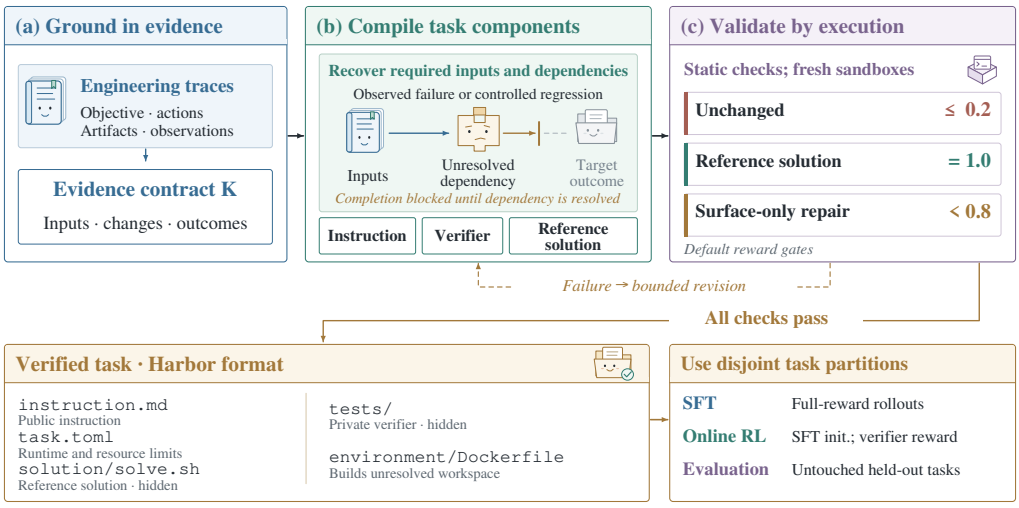


Figure 1: CARET compiles source traces into verified Harbor tasks. (a) Governed logs provide the evidence for a shared contract. (b) The contract guides workspace recovery and joint construction of the instruction, verifier, and reference solution; an unresolved dependency blocks the target outcome at initialization. (c) Static checks and three fresh-sandbox reward gates determine admission, with bounded revision for repairable failures. The lower row shows the exported package and its downstream uses.

Workspace and reward notation. For the following analysis, I denotes the instruction, W^- the initial unresolved workspace, and V the verifier. For an action sequence A, $W^- \oplus A$ denotes the workspace resulting from executing A under the task's runtime configuration. We use $A^\star$ for the actions of the reference solution and $\varnothing$ for taking no repair actions.

Harbor leaves the numerical scoring rule to the task author. We write $R_V(W^-, A)$ for the emitted reward, including any task-specific output rounding. Our default deterministic verifier uses $J \geq 1$ equally weighted binary checks, with $v_j(W) = 1$ if workspace W passes check j and 0 otherwise. The unrounded fraction of passed checks is

$$R_V^{\text{raw}}(W^-, A) = \frac{1}{J} \sum_{j=1}^{J} v_j(W^- \oplus A), \qquad v_j(W) \in \{0, 1\}. \tag{1}$$

3 CARET

CARET compiles source traces into unresolved, verifiable tasks in four steps (Figure 1). **(1) Governed logs:** select authorized, de-identified traces whose objectives, inputs, and dependencies can be reconstructed. **(2) Evidence contract:** extract a shared specification linking the recorded workflow to admissible changes and observable success criteria. **(3) Joint construction:** recover a self-contained workspace, establish an unresolved dependency, and jointly derive the instruction, verifier, and reference solution. **(4) Execution validation:** apply static checks and three sandboxed execution controls, revise repairable failures, and export accepted tasks in Harbor format.

3.1 Governed Logs

Our source traces are collected from internal engineering workflows and processed under the authorization and de-identification procedures in Appendix C. We retain traces with an observable objective, a recoverable entry point, and enough execution evidence to reconstruct task-critical inputs and dependencies. Each selected trace preserves the order of user requests, tool actions, artifacts, and observations, so construction can recover which state each action operated on. Traces requiring unavailable private state, live credentials, or subjective judgment are excluded. The task starts after its prerequisites are established and before the requested objective is solved.

Traces sharing a user, session, project, workspace, or near-duplicate request form a source group. Each group and its descendants belong to one SFT, RL, or evaluation partition, assigned before generation and policy rollouts. Project and defect-family caps limit concentration within each partition.

3.2 Evidence contract

The extractor identifies the objective, relevant artifacts, commands, observations, and state transitions, then converts them into a shared contract K. The contract specifies the workflow to preserve, required inputs, allowed changes, observable outcomes, and information available to the agent. It distinguishes observed facts and supported reconstructions from unresolved assumptions; an unsupported assumption cannot determine required behavior or reward.

An evidence spine links each scored property to its construction basis:

$$\Gamma = \{\langle a_j, w_j, d_j, v_j \rangle\}_{j=1}^{J}, \tag{2}$$

where trace anchor a_j supports workspace fact w_j, unresolved dependency d_j, and verifier property v_j. A requirement without this path is removed or causes rejection. The same contract therefore constrains what the workspace contains, what the agent must resolve, and what the verifier may score.

3.3 Joint construction

The rehydrator follows imports, build files, local links, and data dependencies to restore the smallest self-contained workspace that preserves the contracted workflow. Required public resources are pinned during image construction, and an offline build and entry-point check confirm that the workspace can run independently. CARET then restores an observed unresolved state or introduces a controlled regression at a supported file, schema, or dependency. A constructed regression is recorded as a transformation of the recovered workspace. The objective and critical input–output relations remain fixed, while the unresolved dependency prevents the initial workspace from satisfying the requested outcome.

From this state and K, the compiler jointly derives the public instruction, private verifier, and reference repair. The instruction states the objective and exposes sufficient starting evidence; the verifier checks outcomes and invariants while accepting valid alternative repairs; and the reference repair supplies an executable witness of solvability. A second, surface-only repair fixes obvious symptoms while leaving the core dependency unresolved. Changes to any component trigger consistency checks against the others and the contract. Construction parameters and examples appear in Appendix E.

3.4 Execution validation

Static checks validate the task layout, required files, runtime configuration, dependency closure, and instruction–verifier consistency. They also remove duplicates and prevent solutions, private tests, or repair-revealing artifacts from being exposed to the agent. Candidates with missing requirements or unsupported scored behavior are rejected before sandbox execution.

Dynamic checks execute the unchanged workspace, complete repair, and declared partial repair in separate fresh sandboxes:

$$r_0 = R_V(W^-, \varnothing), \quad r_\star = R_V(W^-, A^\star), \quad r_{\text{surf}} = R_V(W^-, A^{\text{surf}}). \tag{3}$$

The default gates require $r_0 \leq 0.2$, $r_\star = 1.0$, and $r_{\text{surf}} < 0.8$: the task must require work, admit a complete solution, and reject the declared surface-only shortcut. Repeated clean executions check reward determinism. Repairable failures receive bounded revision (Appendix E.3) using execution logs, followed by renewed static and dynamic checks; persistent failures are rejected. Accepted candidates are exported as Harbor packages containing the instruction, runtime configuration, unresolved environment, private verifier, and reference solution.

4 Experiments

We evaluate the downstream utility of tasks produced by three construction pipelines: OpenThoughts-Agent, ClawGym, and CARET. The experiments ask whether CARET's evidence-constrained construction improves agent performance across model scales, and whether its tasks remain useful for

reinforcement learning after supervised fine-tuning. Component ablations in Section 5.1 examine the contribution of individual construction constraints.

4.1 Experimental setup

Benchmarks and metrics. We select benchmarks that test practical assistant workflows involving tool use, artifact creation, and environment changes. ClawEval-General covers service orchestration and analysis while assessing completion, robustness, and safety (Ye et al., 2026). PinchBench covers everyday tasks such as scheduling, research, and file management (Kilo AI Team, 2026a). WildClawBench tests long-horizon execution in native agent environments (Ding et al., 2026). Together, they assess transfer from CARET's training tasks to external assistant workflows.

We evaluate ClawEval-General (161 tasks, three trials per task), PinchBench (147 tasks, five complete runs), and the pure-text subset of WildClawBench (34 tasks) (Ye et al., 2026; Kilo AI Team, 2026a; Ding et al., 2026). For ClawEval-General, we report Pass^3 and Pass@3, measuring success in all three trials and at least one trial, respectively, together with Avg. Score and the completion, robustness, and safety scores. For PinchBench, we report the mean Overall score and Min, Max, Median, and Std across complete runs. For WildClawBench, we report the mean pure-text score. Scores use a percentage scale; aggregation and timing conventions are provided in Appendix A.

Task-construction baselines. OpenThoughts-Agent adapts existing task collections, including NL2Bash and InferredBugs, and filters generated RL tasks through sandbox execution (OpenThoughts-Agent Team, 2025). ClawGym constructs tasks, workspaces, and verifiers from user scenarios and reusable skills (Bai et al., 2026). CARET jointly constructs these components from source-trace evidence. We compare the resulting tasks' SFT and RL utility; a broader pipeline comparison appears in Table 11.

Models and training stages. We compare Qwen3-4B-Instruct-2507, Qwen3-8B, and Qwen3-14B (Yang et al., 2025a), with Vanilla, SFT, and SFT→RL results. Each pipeline uses approximately 16K SFT trajectories; CARET, ClawGym, and OpenThoughts-Agent use 764, 796, and 728 RL tasks, respectively. SFT uses ms-swift for three epochs; RL uses PaddleRL with GRPO for two epochs, starting from the same method's SFT model at the same scale. For CARET, SFT retains full-reward rollouts after filtering infrastructure failures, leakage, tool-call loops, and near duplicates, with per-task and per-generator caps. RL tasks pass the construction checks in Section 3.4. Exact counts, loss masking, and hyperparameters are reported in Appendix B.

4.2 Main results

Downstream utility of task construction. In Table 1, CARET leads PinchBench and WildClawBench pure text in all six scale–stage groups. After RL, its margins over the strongest baseline at each scale range from 2.00–2.40 percentage points on PinchBench and 2.40–2.70 points on WildClawBench. The advantage is present after SFT and persists after RL, supporting the downstream utility of tasks constructed from source-trace evidence. On ClawEval-General, CARET leads both pass metrics across the six groups, while Avg. Score is slightly lower than ClawGym at 8B SFT (57.22 versus 57.54) and OpenThoughts-Agent at 14B RL (61.59 versus 61.62). The component ablations examine which construction constraints contribute to these downstream results.

Gains from SFT to RL. CARET's SFT→RL models improve over its SFT models at all three scales: PinchBench gains 3.00–3.10 percentage points, WildClawBench gains 3.00–3.70 points, and ClawEval-General Avg. Score gains 0.99–3.22 points. These gains address a second use of the constructed tasks: beyond supplying supervised demonstrations, they provide executable feedback for further policy improvement. Together with the SFT results, the comparison supports using the task corpus at both training stages.

5 Analysis

We analyze the contribution of CARET's construction components, characterize the resulting data, and examine how execution checks preserve task requirements.

Table 1: **Measured downstream performance on ClawEval-General, PinchBench, and WildClaw-Bench pure text.** Scores are percentages. Reference rows are excluded from ranking; yellow/bold and blue/underlined cells mark the best and second-best values within each model scale and training stage (lower is better for Std). RL denotes SFT followed by RL; runtime is not ranked.

		ClawEval-General						PinchBench					WCB (PT)	
Type	**Method**	Pass^3	Pass@3	Compl.	Robust.	Safety	Avg.	Overall	Min	Max	Median	Std↓	Score	Elapsed (min)
Qwen3-4B-Instruct-2507														
Reference	Vanilla	10.56	24.84	40.96	95.46	94.85	49.66	52.80	51.30	54.33	52.80	1.14	19.50	82
SFT	OpenThoughts-Agent-v1	11.18	32.92	47.52	**95.60**	94.00	54.11	57.90	56.57	59.25	57.50	1.09	21.80	74
SFT	ClawGym	13.04	36.02	49.71	93.55	**95.10**	55.88	57.20	56.07	58.48	57.31	0.97	21.00	108
SFT	**CARET (Ours)**	**15.53**	**38.51**	**50.66**	94.40	94.60	**56.52**	**59.30**	**58.33**	**59.91**	**59.68**	**0.60**	**23.10**	108
RL	OpenThoughts-Agent-v1	13.04	36.65	49.91	**95.28**	94.12	55.88	59.10	57.93	59.92	58.94	0.72	23.90	97
RL	ClawGym	14.91	39.75	52.01	93.79	**94.92**	57.52	60.00	59.05	60.96	59.79	**0.65**	23.20	79
RL	**CARET (Ours)**	**18.63**	**43.48**	**54.48**	94.25	94.68	**59.34**	**62.40**	**60.84**	**63.16**	**62.63**	0.84	**26.30**	95
Qwen3-8B														
Reference	Vanilla	11.18	27.33	49.50	88.36	87.50	50.49	54.60	53.69	55.18	54.96	0.62	21.00	154
SFT	OpenThoughts-Agent-v1	13.04	36.65	50.64	94.90	**94.80**	56.73	60.80	59.87	61.99	60.88	0.72	24.80	189
SFT	ClawGym	14.91	39.75	**52.31**	**95.10**	93.80	**57.54**	60.20	59.21	61.28	60.09	**0.68**	25.20	143
SFT	**CARET (Ours)**	**17.39**	**42.86**	51.77	94.20	94.60	57.22	**62.20**	**60.66**	**63.32**	**62.57**	1.08	**27.00**	168
RL	OpenThoughts-Agent-v1	16.15	40.99	53.39	94.58	**94.92**	58.87	62.40	61.63	63.61	62.08	0.69	27.40	164
RL	ClawGym	15.53	40.37	53.89	**95.34**	93.62	58.45	63.10	62.54	63.95	63.03	**0.47**	26.80	184
RL	**CARET (Ours)**	**20.50**	**47.20**	**55.83**	94.05	94.68	**60.44**	**65.20**	**64.07**	**65.94**	**65.18**	0.69	**30.00**	191
Qwen3-14B														
Reference	Vanilla	12.42	29.19	53.49	86.61	87.01	52.69	56.50	55.85	57.56	56.26	0.64	24.00	225
SFT	OpenThoughts-Agent-v1	15.53	40.37	55.12	**95.50**	93.70	59.53	63.50	63.03	64.64	63.18	**0.60**	28.40	297
SFT	ClawGym	17.39	44.10	55.17	94.60	**95.00**	60.16	64.00	62.55	65.55	64.16	1.02	28.00	298
SFT	**CARET (Ours)**	**19.88**	**47.20**	**56.27**	94.80	94.40	**60.60**	**65.60**	**64.35**	**66.84**	**65.81**	1.02	**30.10**	247
RL	OpenThoughts-Agent-v1	19.25	45.34	**58.02**	**95.18**	93.82	**61.62**	65.90	64.42	67.46	65.79	1.03	30.50	227
RL	ClawGym	18.63	44.72	56.82	94.84	**94.82**	61.37	66.70	65.72	67.48	66.67	**0.59**	31.10	208
RL	**CARET (Ours)**	**22.98**	**51.55**	57.36	94.65	94.48	61.59	**68.70**	**67.58**	**70.12**	**68.62**	0.84	**33.80**	223

Coverage: ClawEval-General: 161 tasks × 3 trials; PinchBench: 147 tasks × 5 complete runs; WCB (PT): 34 pure-text tasks. PinchBench Overall is the mean of the five run scores; Min/Max/Median/Std summarize those same runs, with Std using denominator $R = 5$ and reported in percentage points.

5.1 Ablation Study

Table 2 ablates three construction components, evaluating task validity, fixed-solver success, and downstream PinchBench performance with Qwen3-4B-Instruct-2507 trained by SFT and RL. Full CARET achieves 89.47% validity and 62.40% on PinchBench. Removing the evidence contract produces the largest downstream drop, to 56.74% (↓ 5.66 percentage points), even as fixed-solver Pass@1 rises from 26.32% to 66.16%. This pattern is consistent with a shift toward easier tasks that provide less useful training supervision. Constructing components independently reduces validity to 21.24% and PinchBench to 58.36% (↓ 4.04 points), supporting joint alignment of the instruction, workspace, verifier, and reference repair. Finally, removing execution-guided revision lowers validity to 38.53% and PinchBench to 60.19% (↓ 2.21 points), supporting the use of execution feedback to recover repairable candidates. Component interventions are detailed in Appendix F.

5.2 Task-Construction Cost

Table 3 reports measured construction costs per accepted task. Full CARET costs $8.01 per task, compared with $15.31 without the evidence contract, $33.74 without joint construction, and $18.60 without execution-guided revision. These correspond to cost reductions of 47.68%, 76.26%, and 56.94%, respectively. The pipeline uses Opus 4.8 for planning and review and GLM-5.2 for task generation. Teacher trajectories generated by GLM-5.2 and Opus 4.8 are accounted for separately and excluded from these costs.

Table 2: Component ablation of CARET.

Variant	Validity	Pass@1	Pass@4	PinchBench	Δ
Full CARET	**89.47**	26.32	41.75	**62.40**	—
w/o evidence contract	46.81	66.16	80.83	56.74	↓ 5.66
w/o joint construction	21.24	49.47	63.39	58.36	↓ 4.04
w/o execution-guided revision	38.53	35.18	48.27	60.19	↓ 2.21

Metrics are percentages; Δ is the decrease in PinchBench relative to Full CARET, in percentage points. DeepSeek-V4-Flash evaluates each variant's valid task pool. PinchBench evaluates Qwen3-4B-Instruct-2507 after SFT→RL. Pass@1/@4 characterize task challenge; lower values are not automatically preferable.

Table 3: Construction cost per accepted task. Teacher rollout costs are excluded.

Variant	Cost (USD/task)	Full CARET reduction (%)
Full CARET	8.01	—
w/o evidence contract	15.31	47.68
w/o joint construction	33.74	76.26
w/o execution-guided revision	18.60	56.94

5.3 Task and trajectory characteristics

Figure 2 summarizes 10,139 tasks and 24,187 trajectories. Five primary workflow domains each account for 15.6%–25.2% of tasks; outer labels show categories or overlapping application facets (Appendix D). Median tasks have 244-word instructions, 12 workspace files, four contract constraints, and 32 verifier checks.

OpenClaw, Claude Code, and Hermes contribute 9,214, 8,052, and 6,921 trajectories. Median trajectories contain 18 assistant messages, 20 tool calls, and six distinct shell commands. Assistant steps exclude system, user, and tool messages.

5.4 Task difficulty and construction revision

Task difficulty. We probe difficulty on a validated CARET construction batch, separately from the component ablation in Table 2. GPT-5-mini, GLM-5.1, and DeepSeek-V4-Flash obtain Pass@1 of 50%, 32%, and 25%, and Pass@4 of 66%, 54%, and 39%, respectively.

Validity after revision. In the construction batch, reported validity increases from 70.48% on the first pass to 89.17% after execution-guided revision, a gain of 18.69 percentage points. Failed executions guide candidate corrections, followed by renewed admission checks before acceptance.

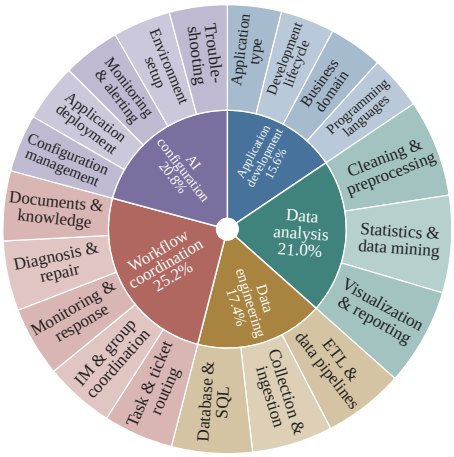
5.5 Execution validation diagnostics

History-to-state binding. A deterministic nanocomposite-hydrogel fixture tests whether scoring follows the reconstructed workspace state. Its 19-file workspace starts at reward 0.19 and reaches 1.0 under the reference repair. History replay reproduces the checkpoint digest; changing the active sample, undoing the import, or changing only the reported claim is rejected. This result shows that the fixture distinguishes a valid state repair from the tested state and claim manipulations.

5.6 Case study: preserving the source task

Figure 3 illustrates three task constructions for a restoration request: repair the existing program, import 21 backup records, and preserve three existing records. The left column changes the repair objective to a new importer; the middle column retains the request but provides no old records to preserve. The right column links the original restore entry, populated initial state, and preservation

(a) Workflow taxonomy



(b) Task and trajectory statistics

Attribute	Mean	Median	P90
Per task			$n = 10{,}139$
Instruction length (words)	261	244	378
Workspace files	18.8	12	35
Contract constraints	4.5	4	10
Verifier checks	33.6	32	48
Per trajectory			$n = 24{,}187$
Assistant steps	20.4	18	34
Tool calls	22.2	20	36
Distinct shell commands	6.7	6	11

Training trajectories by scaffold

OpenClaw	Claude Code	Hermes
9,214 (38.1%)	8,052 (33.3%)	6,921 (28.6%)

Figure 2: Task coverage and instance statistics for CARET. (a) Inner sectors show task-domain proportions; outer sectors identify workflow categories or overlapping application-development facets, rather than subcategory frequencies. (b) Mean, median, and 90th percentile (P90) across all 10,139 tasks and 24,187 collected trajectories, grouped by unit of observation. Instruction length is measured in words; assistant steps count assistant messages only. The lower table reports each scaffold's trajectory count and share of the collected corpus.

checks through the same requirements. This illustrates the roles of the evidence contract and joint construction. A controlled diagnostic on the packaged restoration task is reported in Appendix H.

6 Related Work

Executable environments and task synthesis. SWE-bench (Jimenez et al., 2024), Terminal-Bench (Merrill et al., 2026), and Harbor (Harbor Framework Team, 2026) provide executable evaluation settings; ClawEval, WildClawBench, and PinchBench cover practical assistant workflows (Ye et al., 2026; Ding et al., 2026; Kilo AI Team, 2026a). Training-task synthesis draws on repositories, skills, and verified seeds. SWE-smith injects bugs, R2E-Gym back-translates commits and synthesizes tests, and SWE-Flow derives test-verifiable development steps (Yang et al., 2025b; Jain et al., 2025; Zhang et al., 2025). Terminal-Task-Gen conditions generation on skills, BenchEvolver modifies solutions, and RST recursively extends validated bundles (Pi et al., 2026; Wu et al., 2026b; Li et al., 2026). CARET starts from partially observed source traces without an existing test harness, using source evidence to constrain the objective, workspace, unresolved state, and verifier.

Task recovery from recorded behavior. OS-Genesis derives GUI tasks retrospectively from exploratory interactions (Sun et al., 2025). TerminalWorld recovers instructions, reference scripts, and Docker environments from public terminal recordings, then checks complete, no-op, and partial solutions (Chu et al., 2026). Terminal-Universe combines file replay and workspace completion to recover intent or generate tasks within and across workspaces (Wu et al., 2026a). CARET makes trace anchors, workspace facts, unresolved dependencies, and verifier properties an explicit shared evidence contract. Its surface repair leaves a declared dependency unresolved; TerminalWorld instead truncates or removes reference steps. Three-state validation is shared with prior work. Our ablations evaluate the evidence contract, joint construction, and execution-guided revision; they do not isolate the incremental value of dependency-targeted controls over alternative partial-solution checks.

Task construction for agent post-training. OpenThoughts-Agent v1 adapts NL2Bash and InferredBugs for SFT; its RL pipeline expands instruction–command pairs, generates tests, and filters candidates through sandbox execution and agent rollouts (OpenThoughts-Agent Team, 2025). ClawGym synthesizes tasks from personas and skills, with prepared workspaces and code-based or rubric-based

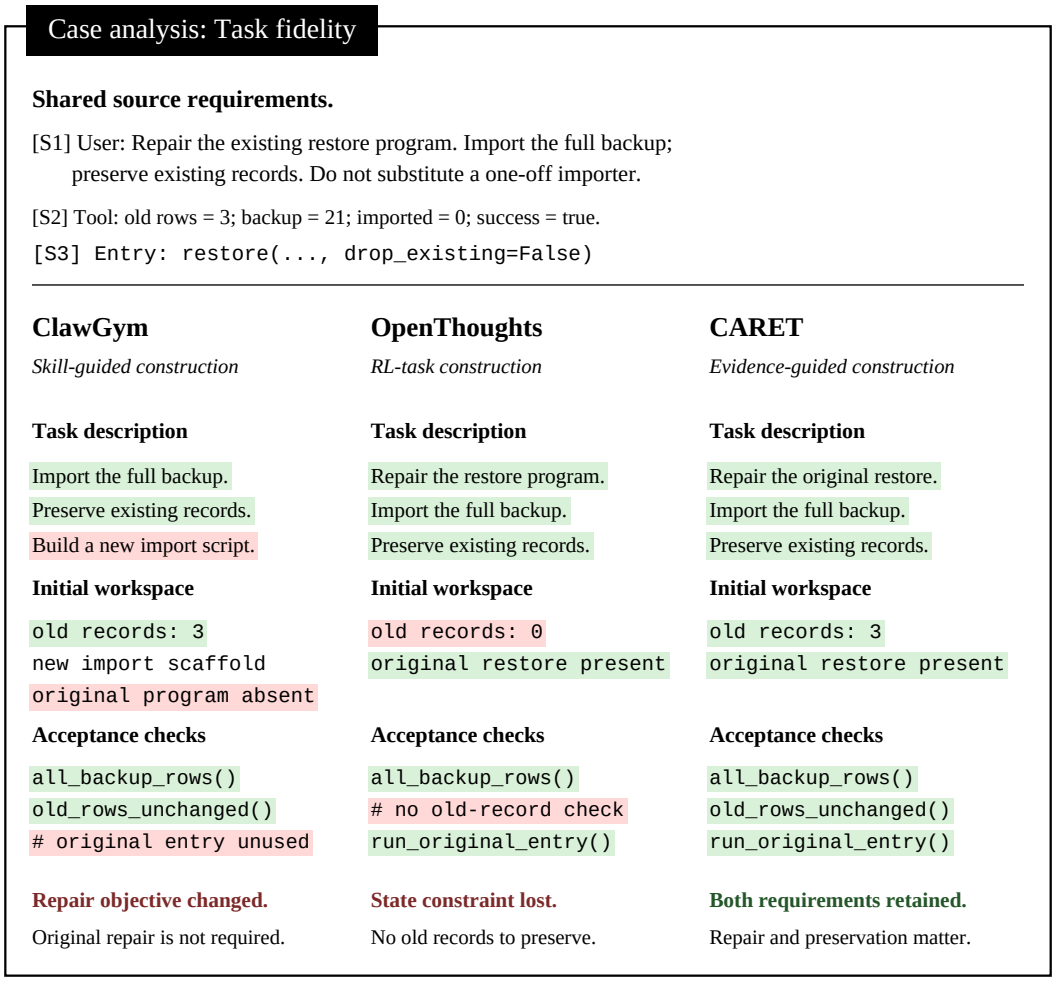


Figure 3: **Preserving task requirements across instruction, workspace, and verification.** Green marks preservation; red marks changes or omissions.

verifiers (Bai et al., 2026). SWE-Gym learns from successful repository trajectories, while RST uses synthesized tasks for both SFT and online RL (Pan et al., 2025; Li et al., 2026). CARET supplies both training stages with tasks compiled from recorded engineering evidence: a shared contract aligns task components, and execution checks guide revision and admission. Appendix G compares resource scale, source grounding, training use, and admission procedures.

7 Conclusion

CARET turns engineering workflow traces into reusable agent tasks. A shared evidence contract grounds joint construction, while execution feedback guides admission and bounded revision. The resulting task format supports demonstration collection for SFT and resettable interaction for RL. Across 4B-Instruct, 8B, and 14B models, CARET achieves the highest PinchBench and WildClaw-Bench pure-text scores among the compared methods after both training stages, while ClawEval reveals metric-specific trade-offs. Component ablations support the utility of the construction constraints and revision procedure. The execution checks establish reference solvability and rejection of a designated surface repair; broader verifier coverage remains to be assessed.

Ethics Statement

CARET compiles tasks from source traces, so the main risks are privacy leakage, accidental credential disclosure, reconstruction of proprietary content, and reward functions that accept unsafe shortcuts. Controls include purpose limitation, redaction, provenance tracking, offline execution, least-privilege containers, leakage scans, counterfactual execution, and stratified human review. Purpose authorization and de-identification occur before construction, and traces requiring live credentials, unreleasable semantics, or identifiable user content are rejected. Automated checks do not replace source authorization or release review. The governance and release boundary is detailed in Appendix C.

Reproducibility Statement

The source package includes aggregate results and scripts for regenerating the reported tables. Construction and evaluation protocols are detailed in Section 3 and Section 4, with additional details in the appendices. We will release governance-approved task bundles, ordered manifests, and aggregate audit reports to support re-execution of the admission checks and verification of task-level reward behavior.

AI Use Statement

Generative AI tools assisted with method discussion, English drafting, figure layout, LaTeX editing, and consistency checking of cross-references and reported numbers. All experiments, evaluation runs, and measurements were designed and executed by the authors, and every reported value was checked by the authors against the underlying evaluation records. The authors remain responsible for the final claims, citations, figures, and release boundary.

References

Fei Bai, Huatong Song, Shuang Sun, Daixuan Cheng, Yike Yang, Chuan Hao, Renyuan Li, Feng Chang, Yuan Wei, Ran Tao, Bryan Dai, Jian Yang, Wayne Xin Zhao, and Ji-Rong Wen. ClawGym: A scalable framework for building effective claw agents. *arXiv preprint arXiv:2604.26904*, 2026. URL https://arxiv.org/abs/2604.26904v3. Version 3, May 16, 2026.

Zhaoyang Chu, Jiarui Hu, Xingyu Jiang, Pengyu Zou, Han Li, Chao Peng, Peter O'Hearn, Earl T. Barr, Mark Harman, Federica Sarro, and He Ye. Terminalworld: Benchmarking agents on real-world terminal tasks. *arXiv preprint arXiv:2605.22535*, 2026. URL https://arxiv.org/abs/2605.22535.

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, JingYi Yang, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, Yubo Ma, Haodong Duan, Jing Shao, Jiaqi Wang, Dahua Lin, Kai Chen, and Yuhang Zang. Wildclawbench: A benchmark for real-world, long-horizon agent evaluation. *arXiv preprint arXiv:2605.10912*, 2026. URL https://arxiv.org/abs/2605.10912.

Kanishk Gandhi, Shivam Garg, Noah D. Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling RL environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2026. URL https://arxiv.org/abs/2601.16443.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. Zenodo, 2026. URL https://doi.org/10.5281/zenodo.20953922.

Hamish Ivison, Junjie Oscar Yin, Rulin Shao, Teng Xiao, Nathan Lambert, and Hannaneh Hajishirzi. TMax: A simple recipe for terminal agents. *arXiv preprint arXiv:2606.23321*, 2026. URL https://arxiv.org/abs/2606.23321.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2504.07164.

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? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

Kilo AI Team. Pinchbench: Benchmarking system for evaluating LLM models as openclaw coding agents. GitHub repository, 2026a. URL https://github.com/pinchbench/skill.

Kilo AI Team. Pinchbench leaderboard: Run variance statistics. GitHub repository, 2026b. URL https://github.com/pinchbench/leaderboard. Scoring implementation, commit d3d88668756b8c588023feca5696ee7af0f312bd.

Zhongzhi Li, Yucheng Shi, Zongxia Li, Ruhan Wang, Anhao Li, Zixun Huang, Junyao Yang, Lei Ke, Ninghao Liu, Haitao Mi, and Leowei Liang. Recursive synthesis for long-horizon terminal tasks. *arXiv preprint arXiv:2608.05466*, 2026. URL https://arxiv.org/abs/2608.05466.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2601.11868.

OpenThoughts-Agent Team. Launching the OpenThoughts-Agent project. Official release documentation, December 2025. URL https://www.openthoughts.ai/blog/agent.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737, 2025. URL https://arxiv.org/abs/2412.21139.

Renjie Pi, Grace Lam, Mohammad Shoeybi, Pooya Jannaty, Bryan Catanzaro, and Wei Ping. On data engineering for scaling LLM terminal capabilities. *arXiv preprint arXiv:2602.21193*, 2026. URL https://arxiv.org/abs/2602.21193.

Qijia Shen, Zhiqi Huang, Vamsidhar Kamanuru, et al. SETA: Scaling environments for terminal agents. *arXiv preprint arXiv:2607.10891*, 2026. URL https://arxiv.org/abs/2607.10891.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. OS-Genesis: Automating GUI agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5555–5579, 2025. URL https://aclanthology.org/2025.acl-long.277/.

Jie Wu, Zhenru Zhang, Beichen Zhang, Xuwu Wang, Yuhui Su, Mouxiang Chen, Peng Wang, Zhihai Wang, Que Shen, Hao Zhou, An Yang, Fei Huang, Yujiu Yang, and Dayiheng Liu. Terminal-Universe: Turning agent trajectories into scalable terminal environments. *arXiv preprint arXiv:2609.04148*, 2026a. URL https://arxiv.org/abs/2609.04148v1.

Yangzhen Wu, Aaron J. Li, Wenjie Ma, Li Cao, Ziheng Zhou, Mert Cemri, Shu Liu, Yuran Xiu, Chenxiao Yan, Haikun Zhao, Bin Yu, Ion Stoica, and Dawn Song. Benchevolver: Frontier task synthesis via solution-centric evolution. *arXiv preprint arXiv:2606.01286*, 2026b. URL https://arxiv.org/abs/2606.01286.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a. URL https://arxiv.org/abs/2505.09388.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents, 2025b. URL https://arxiv.org/abs/2504.21798.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. Claw-eval: Towards trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*, 2026. URL https://arxiv.org/abs/2604.06132.

Lei Zhang, Jiaxi Yang, Min Yang, Jian Yang, Mouxiang Chen, Jiajun Zhang, Zeyu Cui, Binyuan Hui, and Junyang Lin. SWE-Flow: Synthesizing software engineering data in a test-driven manner. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025. URL https://arxiv.org/abs/2506.09003.

Kaijie Zhu, Yuzhou Nie, Yijiang Li, Yiming Huang, Jialian Wu, Jiang Liu, Ximeng Sun, Zhenfei Yin, Lun Wang, Zicheng Liu, Emad Barsoum, William Yang Wang, and Wenbo Guo. TermiGen: High-fidelity environment and robust trajectory synthesis for terminal agents. *arXiv preprint arXiv:2602.07274*, 2026. URL https://arxiv.org/abs/2602.07274.

A Evaluation Metrics and Aggregation

Our evaluated Qwen models are text-only, so evaluation excludes multimodal tasks. We use ClawEval-General (161 tasks) and the WildClawBench pure-text subset (34 tasks). ClawEval-General also excludes multi-turn tasks; this is a separate restriction of our evaluation scope.

For ClawEval-General, let $c_{it}, r_{it}, s_{it} \in [0, 1]$ denote completion, robustness, and safety for task $i \in \{1, \ldots, N\}$ and trial $t \in \{1, 2, 3\}$, with $N = 161$. Following the official scoring rule, $q_{it} = \mathrm{round}(s_{it}(0.8c_{it} + 0.2r_{it}), 4)$, and the average score is $\frac{100}{3N} \sum_{i=1}^{N} \sum_{t=1}^{3} q_{it}$. The dimension columns report the corresponding means on a percentage scale. With $b_{it} = \mathbf{1}[q_{it} \geq 0.75]$, we report the strict three-trial pass rate $\mathrm{Pass}^3 = \frac{100}{N} \sum_i \prod_{t=1}^{3} b_{it}$ and $\mathrm{Pass@3} = \frac{100}{N} \sum_i \mathbf{1}[\sum_t b_{it} > 0]$. These measure passing all three trials and at least one trial, respectively.

For PinchBench, each complete run yields an aggregate score P_j from the normalized task scores. Overall denotes $\bar{P} = \frac{1}{R} \sum_{j=1}^{R} P_j$, where $R = 5$. Min, Max, and Median summarize the same five run scores. We report $\mathrm{Std} = \sqrt{\frac{1}{R} \sum_{j=1}^{R} (P_j - \bar{P})^2}$ in percentage points (Kilo AI Team, 2026b). These statistics describe evaluation-run variation, rather than variation across tasks or training seeds. For WildClawBench, we report the mean task-level overall score on its 34 pure-text tasks, multiplied by 100. Its cumulative time is the sum of task execution durations in minutes, rather than parallel wall-clock time or model-training time.

B Training Data and Hyperparameters

Training data and initialization. Table 4 reports the actual training-set sizes for each construction pipeline. The same counts and settings apply at all three model scales. These counts describe the training sets, rather than the full constructed corpus or the number of examples processed across epochs. SFT starts from Qwen3-4B-Instruct-2507, Qwen3-8B, or Qwen3-14B. For each method and scale, RL starts from that method's SFT checkpoint at the same scale. The results therefore compare complete method-specific SFT-to-RL pipelines. The training sets differ in size; equal epoch counts do not imply equal sampling or optimization budgets.

Table 4: Actual downstream training-set sizes, used at each model scale.

Stage	CARET	ClawGym	OpenThoughts-Agent
SFT trajectories	15,715	16,073	15,087
RL tasks	764	796	728

Supervised fine-tuning. SFT minimizes token-level negative log-likelihood only on model-generated outputs: reasoning, assistant text, and serialized tool calls. System and user messages and tool observations remain in the context but are masked out of the loss. Table 5 lists the SFT settings. The maximum sequence length is an upper bound on training examples, not a claim that every trajectory contains 128K tokens.

Table 5: SFT hyperparameters shared across methods and model scales.

Parameter	Setting
Framework	ms-swift
Epochs	3
Global batch size	16 sequences
Per-device batch size	1
Optimizer	`adamw_torch_fused`
Learning rate	10^{-5}
Learning-rate schedule	Cosine
Warmup ratio	0.05
Maximum training sequence length	131,072 tokens
Precision	BF16
Distributed training	DeepSpeed ZeRO-3

Reinforcement learning. RL uses GRPO in PaddleRL with tasks packaged in Harbor format. Each sampling batch contains eight task prompts, with eight rollouts per prompt, for a nominal total of 64 trajectories. This sampling count is distinct from the optimization mini-batch size or the number of parameter updates. The generation cap is 16K tokens per model generation call, rather than a single 16K budget for the entire multi-turn rollout. The task's code verifier provides the terminal reward. No explicit KL penalty is applied; the policy objective uses the lower and upper clipping parameters in Table 6.

Table 6: RL hyperparameters shared across methods and model scales.

Parameter	Setting
Framework / algorithm	PaddleRL / GRPO
Task format	Harbor
Epochs	2
Global prompt batch size	8
Rollouts per task prompt	8
Nominal trajectories per sampling batch	64
Learning rate	10^{-6}
Weight decay	0.1
Optimizer betas	$(0.9, 0.95)$
Maximum tokens per generation call	16K
KL coefficient	0
Clipping ϵ (low / high)	0.2 / 0.28
Reward	Terminal code-verifier reward

C Governance and Release Boundary

Purpose authorization and de-identification occur before construction. The public task bundle excludes raw source traces. A private artifact may be replaced by a fixture only when the trace constrains every behavior scored by the task. Traces requiring live credentials, unreleasable semantics, or identifiable user content are rejected. Public dependencies may be fetched during image construction and pinned by digest; validation and policy rollouts remain offline and least-privileged.

D Corpus Composition

The task inventory contains 10,139 tasks across five workflow domains, with 24,187 collected trajectories. These corpus-wide totals are distinct from the downstream training sets reported in Table 4. Application facets are overlapping marginals: each facet sums to the application count, but the four facets must not be summed together. Figure 2 illustrates this taxonomy.

Application development. We characterize these tasks along four complementary axes: application type (web front ends, mobile and desktop applications, and backend services), development lifecycle

Table 7: Task distribution across the five workflow domains.

Domain	Tasks	Share
Application development	1,579	15.6%
Data analysis	2,130	21.0%
Data engineering	1,767	17.4%
Workflow coordination	2,554	25.2%
AI configuration	2,109	20.8%
Total	10,139	100.0%

(requirements analysis, architecture design, feature implementation, and enhancement), business domain (enterprise SaaS, content and social applications, education, and finance), and programming language. These axes are annotations of the same task and can overlap; they are not mutually exclusive subsets whose counts can be added together.

Data analysis. Three scenarios distinguish the requested analytical output. Cleaning and preprocessing handle missing values, type conversion, deduplication, format normalization, and outliers. Statistical analysis and mining produce aggregates, trends, metrics, or inferred patterns. Visualization and reporting turn these results into charts and structured Markdown or spreadsheet reports.

Data engineering. ETL and pipeline tasks cover staged data transformations, dependencies between data stages, scheduling, incremental processing, and partition backfills. Collection and ingestion concern acquiring data from APIs, crawlers, files, or databases and loading it into a destination. Database and SQL development concern queries, data manipulation, table definitions, schema maintenance, and dialect migration. This distinction separates movement between data stages from acquisition at the source and operations within a database.

Workflow coordination. Five scenarios cover task and ticket routing, IM and group-message coordination, monitoring and condition-triggered responses, diagnosis and multi-point repair, and document and knowledge curation. Their outputs include consistent task states, coordinated actions, resolved incidents, and organized documents. For offline reconstruction, task-critical interactions and state changes must be represented by trace-supported local artifacts or fixtures.

AI configuration. Five scenarios cover configuration management, application deployment, monitoring and alerting, environment setup, and troubleshooting. These tasks configure models and service endpoints, deploy AI applications, set up monitoring and runtime dependencies, and diagnose failures. Their primary deliverable is a functioning AI system; workflow coordination instead concerns task progression and consistent workflow states.

E CARET Construction and Training Details

E.1 Reconstructability and evidence provenance

A record is reconstructable when it identifies an entry point, constrains every task-critical input, permits dependencies to be pinned, and supports an offline build and outcome check. Repositories, notebooks, scripts, and spreadsheets may meet these conditions. For a private workbook, a de-identified fixture is permitted only when the trace constrains the required sheets, schema, formulas, value ranges, and behavior. Unrecoverable cells or a live private service cause rejection.

The extractor distinguishes trace observations, authorized public resources, generated fixtures, and unresolved assumptions. The contract records allowed mutation anchors, symptoms, discoverable clues, verifier properties, forbidden shortcuts, disclosure boundaries, network policy, and resource limits. An unresolved assumption cannot define a scored requirement.

Construction reports record evidence provenance, defect labels, masking relations, and validation rewards. This metadata supports task authoring and auditing and is kept outside the agent-visible workspace; it is not an additional required component of the Harbor interface.

Table 8: Task counts by primary scenario. Scenarios sum to their parent domain.

Domain	Primary subscenario	Tasks
Data analysis	Cleaning & preprocessing	605
	Statistics & data mining	717
	Visualization & reporting	808
Data engineering	ETL & data pipelines	413
	Collection & ingestion	661
	Database & SQL	693
Workflow coordination	Task & ticket routing	428
	IM & group coordination	587
	Monitoring & response	634
	Diagnosis & repair	464
	Documents & knowledge	441
AI configuration	Configuration management	518
	Application deployment	329
	Monitoring & alerting	493
	Environment setup	415
	Troubleshooting	354

Table 9: Application-development counts by annotation facet. Each facet assigns one primary value to every application task and sums to the application total.

Facet	Primary value	Tasks
Application type	Web front end	258
	Mobile apps	373
	Desktop apps	302
	Backend services	331
	Other / unspecified	315
Development lifecycle	Requirements	338
	Architecture	332
	Implementation	267
	Enhancement	309
	Other / unspecified	333
Business domain	Enterprise SaaS	285
	Content & social	276
	Education	277
	Finance	360
	Other / unspecified	381
Programming languages	Python	331
	JavaScript / TS	193
	Java	339
	C / C++	346
	Other / unspecified	370

Workspace reconstruction follows imports, build files, local links, documentation references, and data dependencies until all critical components are included. Public resources may be retrieved during image construction and are pinned by version and digest. Validation and rollouts run offline. A replayable trace prefix fixes the starting checkpoint. When only a completed state is recoverable, the new unresolved state is recorded as a controlled transformation rather than a historical state.

E.2 Structural difficulty and diagnostic repairs

The hard-task construction contract requires at least five interacting defects across at least three failure traits, including at least three traps, one or two obvious decoys, and a masking chain of depth at least two. A trap makes the immediate repair plausible but insufficient: resolving the task requires runtime

evidence or a cross-component contract. The verifier contains at least 25 separately scored checks, with trap-controlled behavior accounting for at least half. These are hard-task construction priors; frozen-policy rollouts determine empirical difficulty. Full-corpus check counts are summarized separately in Figure 2.

The complete repair $A^\star$ restores all scored properties. `solution/solve_obvious.sh` implements A^{surf} by fixing the obvious decoys while leaving every trap intact. For example, repairing report formatting should not resolve an upstream schema or calibration error that still invalidates the output. Both the deeper dependency and its scored consequence must be supported by the contract. The verifier checks recomputed outputs, invariants, cross-field relations, and forbidden shortcuts. It must also accept valid alternative repairs.

E.3 Admission Gates and Bounded Recompilation

The admission thresholds are defined in Section 3.4. Repeated clean executions check reward determinism. Package checks cover schema validity, dependency closure, component references, and leakage. Caches, backups, version-control residue, precomputed answers, and repair-revealing comments are removed. Verifier review tests requirement coverage, unsupported constraints, invalid-output rejection, and valid-alternative acceptance.

Each failed baseline provides targeted feedback. High r_0 calls for better-covered defects; $r_\star < 1$ requires correcting the reference path or verifier contract; and high r_{surf} requires revisiting unresolved dependencies and their checks. Corrections undergo renewed evidence and fairness review before all baselines are rerun. Revision and leakage revalidation stop at their configured retry limits. Scrubbing must preserve all three rewards. Persistent build failure, hidden requirements, leakage, or budget exhaustion causes rejection. A downgrade is allowed only when the candidate meets the complete lower-difficulty contract.

F Component Ablation Details

Metrics. Table 2 reports one construction-validity percentage per variant. Validity denotes the proportion of candidate tasks accepted as usable; it is distinct from a solver's success on those tasks. DeepSeek-V4-Flash provides the fixed-solver probe, with Pass@1 and Pass@4 measured on each variant's own valid task pool. Higher solver success indicates easier tasks for this probe, and is not itself evidence of more useful training data. PinchBench reports the Overall score of Qwen3-4B-Instruct-2507 after SFT and RL. The final column gives the decrease relative to Full CARET in percentage points, displayed with a downward arrow.

Table 10: Component interventions for the ablation in Table 2.

Variant	Intervention
Full CARET	Evidence contract, joint construction, and execution-guided revision.
w/o evidence contract	Retain the trace and reconstruction, but remove the explicit trace-to-property contract shared across components.
w/o joint construction	Generate instruction, verifier, and reference repair independently from the same contract and recovered state, without cross-component revision.
w/o execution-guided revision	Allow a single compilation attempt; remove failure-log-driven recompilation.

Table 11: Comparison of agent-task construction pipelines and training resources. Resource scales describe the full resources, not the training subsets in Table 4. Resource definitions and source notes appear in Appendix G.

		Resource scale		Task properties		Training use		Admission
Resource / pipeline	Domain	Tasks	Trajectories	Grounded	Executable	SFT	RL algorithm	Three-state execution
Terminal-Corpus (2026)	Terminal	N/R	254,000+[b]	✓	✓	✓	—	×
OpenThoughts-Agent v1-SFT (2025)	Terminal	N/R	15,209	△	✓	✓	—	—
OpenThoughts-Agent v1-RL (2025)	Terminal	∼700	N/R	△	✓	—	✓ (N/R)	—
SWE-Gym (2025)	SWE	2,438	N/R	✓	✓	✓	—	—
Endless Terminals (2026)	Terminal	3,255	N/R	×	✓	×	✓ PPO	—
TermiGen (2026)	Terminal	3,500+	3,291	×	✓	✓	—	—
TMax (2026)	Terminal	14,600	N/R[a]	×	✓	✓[a]	✓ DPPO	×
RST (2026)	Terminal	37,484	327,189	✓	✓	✓	✓ PPO	—
SETA (2026)	Terminal	4,567	1,112	✓	✓	✓	✓ GRPO	—
ClawGym (2026)	Claw	13.5k	24.5k	△	✓	✓	✓ GRPO	—
TerminalWorld (2026)	Terminal	1,530	N/R	✓	✓	—	—	✓
Terminal-Universe (2026a)	Terminal	N/R	31,977	✓	✓	✓	—	—
CARET (ours)	Terminal	10,139	24,187	✓	✓	✓	✓ GRPO	✓

✓: reported; ×: excluded by the stated procedure; —: not reported; △: mixed source grounding; N/R: not reported. Grounded concerns source constraints; Executable concerns the environment. [a]TMax's SFT uses a separate warm-start set. [b]Terminal-Corpus count follows the comparison in Li et al. (2026).

G Sources and scope of the pipeline comparison

Table 11 compares released resources rather than matched training budgets. OpenThoughts-Agent's SFT and RL releases are separate resources; TerminalWorld is an evaluation benchmark.

Properties and admission procedures. Grounded denotes source constraints from tasks, repositories, skills, or recorded interactions; it does not imply recovery of the original state. Executable denotes an interactive environment, without requiring a verifier for every SFT record. Training columns refer to use of the listed resource, rather than earlier training of its initial model. The table legend defines the symbols. Three-state execution requires unchanged, complete-solution, and declared partial-solution trials; Table 12 gives the procedures and source locations.

Table 12: Reported validation procedures underlying the admission comparison.

Pipeline	Procedure and source location
Terminal-Corpus	Uses domain containers, synthesized tests, and trajectory filtering. It explicitly does not generate oracle solutions (Pi et al., 2026, Sections 4.2.3–4.4).
OpenThoughts-Agent v1-SFT	The release's "SFT Data—Instruction Sourcing" section states that a verifier is not necessary for SFT data generation. The project also has a separate v1-RL dataset; this row refers to the SFT release (OpenThoughts-Agent Team, 2025).
OpenThoughts-Agent v1-RL	The launch post's "RL Data" section describes NL2Bash-seeded instruction and command generation, generated tests executed in a sandbox, and filtering tasks on which GPT-5-Codex receives zero reward. Approximately 700 of 10,000 generated tasks remain. No deliberate partial-solution admission trial is specified (OpenThoughts-Agent Team, 2025).
SWE-Gym	Executes original code and the human gold patch to establish test improvement; no declared partial-patch admission run is specified (Pan et al., 2025, Section 3.1).
Endless Terminals	Requires completion tests to fail initially and at least one successful solution among 16 attempts. No declared partial-solution gate is described (Gandhi et al., 2026, Section 3).
TermiGen	Iteratively repairs Docker builds and uses a judge to refine unit tests. Error injection occurs during trajectory collection, rather than as a three-control task-admission requirement (Zhu et al., 2026, Sections 3.2–3.3).
TMax-15K	Explicitly skips solution-based validation during generation; checks executability and uses soft filtering during RL (Ivison et al., 2026, Section 3.1).
RST	Gives each candidate one oracle execution and retains full-reward tasks; no three-control protocol is described in the oracle-validation procedure (Li et al., 2026, p. 22).
SETA	Requires no-op failure and oracle success, with an additional judge for tasks whose rollouts all fail. Its public generation instructions also require the empty and oracle runs; no third partial-solution admission run is specified (Shen et al., 2026, Section 3.1 and Appendix A.2).
ClawGym	Combines persona-driven and skill-grounded synthesis; reviews task quality, code-verifier execution and coverage, and complementary rubric criteria. No three-state admission protocol is reported (Bai et al., 2026, Sections 3.4 and 4).
TerminalWorld	AllPassing requires all tests to pass under the reference; Nop requires all tests to fail initially; Partial truncates or ablates the reference and requires at least one failed test. Admission requires all three trials (Chu et al., 2026, Section 3.4).
Terminal-Universe	Single-WS and Cross-WS use agent-authored pytest and retain passing teacher rollouts; Multi-Round uses round-level filtering. Intent Recovery is not agentic-verifier-filtered. No full three-state protocol is reported (Wu et al., 2026a, Section 3.3 and Appendix D).

Both TerminalWorld and CARET apply three-state checks. TerminalWorld truncates or ablates reference steps; CARET leaves a contracted dependency unresolved under the thresholds in Section 3.4. This procedural difference does not alone establish superior verifier quality.

Resource-specific accounting. Endless Terminals uses its tasks for PPO; its initialization was distilled on other datasets (Gandhi et al., 2026, Section 5). TMax's 14,600 RL environments exclude its separate 2.2k-environment, 16.5k-trajectory SFT warm-start set; its RL algorithm is DPPO (Ivison et al., 2026, Sections 3.3 and 4.1). SETA retains 1,112 SFT trajectories from 1,488 rollouts on 4,567 environments (Shen et al., 2026, Appendix A.9). TerminalWorld's 80,870 source recordings are not training trajectories; the table reports its 1,530 validated tasks (Chu et al., 2026). Terminal-Universe's 31,977 demonstrations comprise 25,386 Single-WS, 3,512 Cross-WS, and 3,079 Multi-Round records, rather than a unique-task count (Wu et al., 2026a, Section 3.3).

The exact Terminal-Corpus (254,000+) and OpenThoughts-Agent v1-SFT (15,209) trajectory counts follow Li et al. (2026)'s resource comparison; the OpenThoughts launch post rounds its SFT count to 15k and does not specify the RL algorithm (OpenThoughts-Agent Team, 2025). Other reported

counts follow the corresponding releases of SWE-Gym, Endless Terminals, TermiGen, RST, and ClawGym (Pan et al., 2025; Gandhi et al., 2026; Zhu et al., 2026; Li et al., 2026; Bai et al., 2026).

H CONTRACT-GUIDED RECONSTRUCTION DIAGNOSTIC

We use the packaged database-restoration task `as-tjvwrpibtq`. The two contract chains are instantiated from its instruction, initialization routine, restore entry, and backup SQL. This is a controlled fixture diagnostic, not a recovered source-trajectory replay or a naturally occurring generation failure. The gate and field-level audit are implemented for this diagnostic; the original task verifier primarily checks reported record counts.

For C1, the private harness checks that the expected existing record identities are present before restoration, then snapshots all their fields. For C2, each repair is installed in a fresh source workspace and executed through the byte-identical original migration entry. The harness records the restore call and independently compares the resulting database with the backup and the pre-restoration snapshot. It does not prescribe the internal repair algorithm.

We deliberately delete the initial records after initialization. C1 raises `C1_MISSING_INITIAL_RECORDS` before any restore call; the unchanged requirement therefore cannot be paired with an empty target. Scripted reconstruction recreates the trusted source fixture and rebinds the snapshot used for preservation checks. The packaged reference repair and an alternative SQL-parser repair both restore all 21 backup records and preserve all three old records. The unchanged program fails restoration. Changing one old field after the reference run also fails the audit, despite unchanged row counts.

These probes show how the instantiated contract excludes an untestable initial state and checks outcomes after reconstruction. They do not measure an LLM generator's repair rate or isolate the framework's contribution relative to equally careful manual engineering. Aggregate component evidence is reported in Section 5.1.