ClawTrace: Cost-Aware Tracing for LLM Agent Skill Distillation

Boqin Yuan*
University of California San Diego
San Diego, CA
b4yuan@ucsd.edu

Yue Su*†
Carnegie Mellon University
Pittsburgh, PA
Epsilla
Union City, NJ
yuesu@andrew.cmu.edu

Renchu Song
Epsilla
Union City, NJ
richard@epsilla.com

Sen Yang
Epsilla
Union City, NJ
eric@epsilla.com

Jing Qin
Epsilla
Union City, NJ
ricki@epsilla.com

Abstract

Skill-distillation pipelines learn reusable rules from LLM agent trajectories, but they lack a key signal: how much each step costs. Without per-step cost, a pipeline cannot distinguish adding a missing step to fix a bug from removing an expensive step that never affected the outcome. We use the cost-attribution gap to ask whether the rule types inside a distilled skill transfer the same way to new tasks. **ClawTrace** records cost-attributed agent traces and compiles each session into a **TraceCard**; **CostCraft** reads TraceCards and writes three kinds of skill patches: *preserve*, *prune*, and *repair*. We find a pattern aggregate metrics hide. On 30 held-out SpreadsheetBench tasks across two seeds, removing prune patches roughly tripled the quality-regression count without lowering median cost. Across the full 84-task SkillsBench transfer, CostCraft saves no aggregate cost. All three quality regressions trace to the preserve lane, and both quality wins trace to the prune lane: prune patches act as quality guardrails while preserve patches drive regressions. We argue that reusable agent skills should be evaluated at the rule-type level, not as monolithic instruction packages. To support this, we release ClawTrace, the TraceCard schema, and the full set of typed skills.

CCS Concepts: • **Computing methodologies → Artificial intelligence**.

Keywords: Agent skills, Skill distillation, Cost-aware tracing, Agent evaluation

*Both authors contributed equally to this work.
†Work done during an internship at Epsilla.

This work is licensed under a Creative Commons Attribution 4.0 International License.
Agent Skills '26, San José, CA
© 2026 Copyright held by the owner/author(s).

ACM Reference Format:
Boqin Yuan, Yue Su, Renchu Song, Sen Yang, and Jing Qin. 2026. ClawTrace: Cost-Aware Tracing for LLM Agent Skill Distillation. In *Agent Skills '26 Workshop: ACM Conference on AI and Agentic Systems, May 26, 2026, San José, CA.* ACM, New York, NY, USA, 14 pages.

1 Introduction

An agent skill is a structured instruction package that guides an LLM agent without weight updates [19]. Recent systems distill such skills directly from agent execution traces [14, 26, 28]. The aggregate gains are real but uneven: on SkillsBench [12], curated skills raise the mean pass rate by 16.2 percentage points, yet 16 of 84 tasks regress and self-generated skills offer no average benefit. Aggregate metrics conceal offsetting wins and losses.

A central reason for these regressions is that existing distillation pipelines treat a skill as a single unit. They partition trajectories into successes and failures, then extract rules from each partition [14]. This partition conflates two distinct operations. *Repairing* a bug requires adding a missing step to a failed trajectory. *Compressing* waste requires removing an expensive but inessential step from a trajectory that already succeeded. The second operation requires per-step cost: the pipeline has to identify which steps were expensive before it can argue that removing them is safe. Standard observability stacks (LangSmith [9], Langfuse [10], Phoenix [3]) record per-span tokens and cost following OpenTelemetry conventions [16]. They expose this information as dashboard analytics for human operators rather than as a compact summary a distillation pipeline can ingest. A skill-mining analyst needs ranked per-step costs, redundancy flags, and failure markers in a format small enough that many sessions fit a single context window.

This paper contributes on two fronts. First, **ClawTrace** is an open-source tracing platform that registers eight event

Code: https://github.com/epsilla-cloud/clawtrace.
Project page: https://www.clawtrace.ai/.

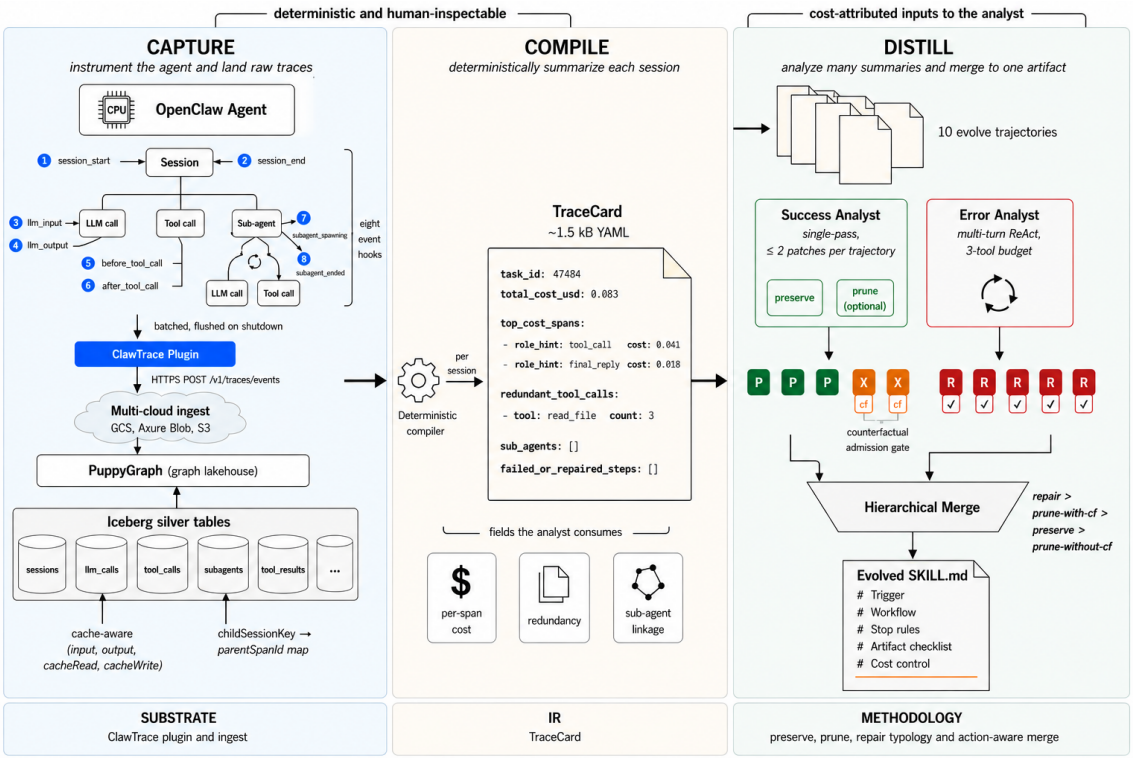

Figure 1. End-to-end architecture. **Capture**: ClawTrace instruments an agent session via eight event hooks. **Compile**: a deterministic compiler produces a TraceCard per session. **Distill**: CostCraft emits preserve, prune, and repair patches that merge into an evolved `SKILL.md`.

hooks covering every LLM call, tool use, and sub-agent spawn. It links child sessions to their parent spans and bills cached tokens at their actual rate. From these events it compiles a **TraceCard** per session: a ~1.5 kB YAML summary listing per-step USD cost, typed token counts, redundant tool-call clusters, and failed-or-repaired steps. The plugin currently ships for OpenClaw, and the schema is intentionally framework-agnostic; in principle any agent harness can produce TraceCards by posting conformant JSON to the ingest API, though we have only validated the OpenClaw path end-to-end.

Second, **CostCraft** reads TraceCards and emits three kinds of skill patches: *preserve* (keep what worked), *prune* (drop unnecessary cost), and *repair* (fix specific failures). The split replaces Trace2Skill's [14] success-versus-error split with a correctness-versus-efficiency split. Section 3.3 gives the patch-admission rules and the conflict-aware merge.

The cross-benchmark evaluation supports a third contribution: *distilled agent skills contain separable rule types that transfer in different ways.* On 30 held-out SpreadsheetBench [13] tasks across two seeds and the full 84-task SkillsBench [12] benchmark, prune rules target recurring waste and transferred to new tasks, while preserve rules encoded SpreadsheetBench-specific conventions and overfit. For the agent-skills community, this suggests that reusable skills should be evaluated not only as whole instruction packages, but also by the transfer behavior of their component rule types.

2 Related Work

Agent skills and benchmarks. Anthropic [19] introduced agent skills as structured packages that augment an LLM agent without weight updates. Xu and Yan [25] distinguish skills from atomic tools and one-off plans, defining them as reusable modules with applicability conditions. SkillsBench [12] provides the first systematic benchmark: curated skills raise mean pass rate by 16.2 percentage points, but self-generated skills offer no average benefit. SWE-SkillsBench [6] evaluates skill injection in software-engineering tasks and reports performance drops under context mismatch. SpreadsheetBench [13] provides 912 real-world tasks with deterministic cell-match grading and is our primary evaluation benchmark.

Trajectory mining and skill distillation. Trace2Skill [14] is the closest prior work. It dispatches parallel analyst sub-agents under a success-versus-error split, with the error analyst using a multi-turn ReAct loop [27] with oracle access. We adopt its analyst structure but replace the two-way split

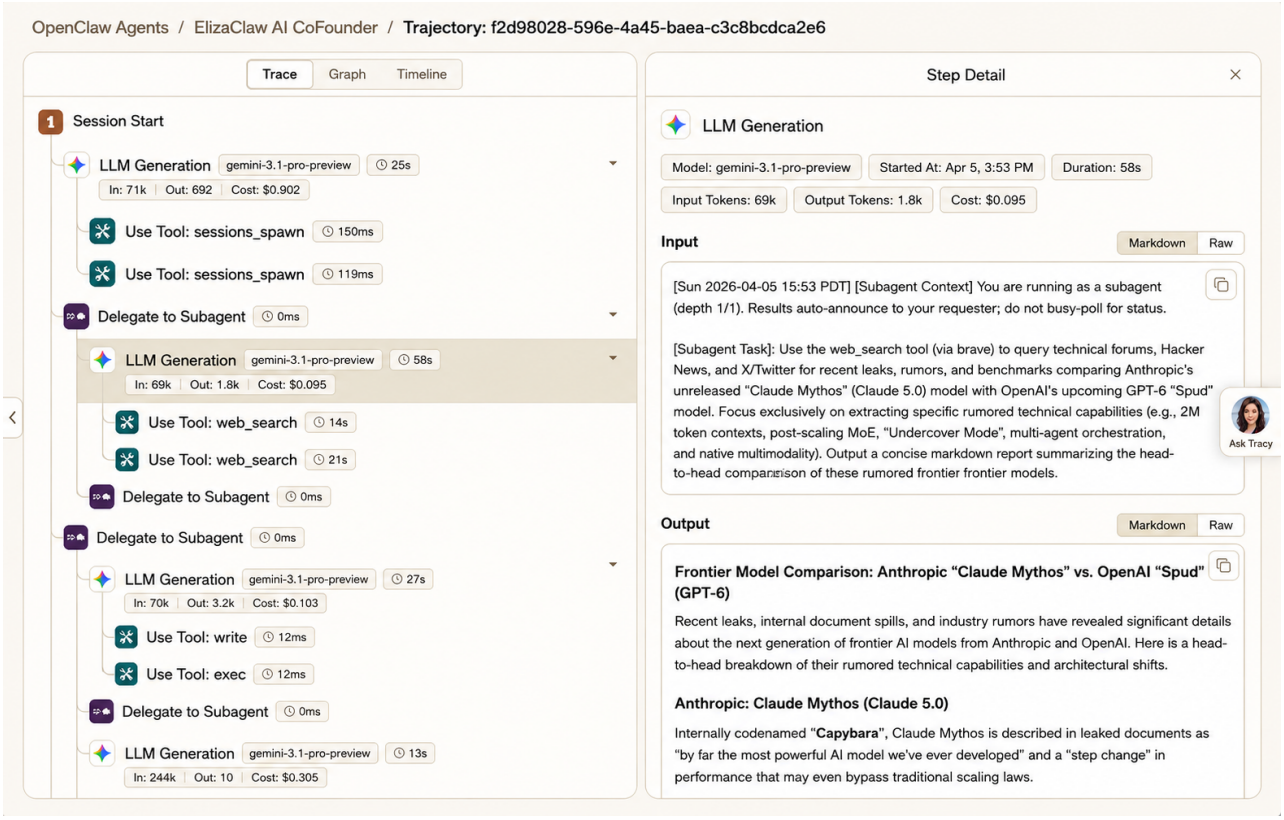

Figure 2. ClawTrace execution-path view showing per-span cost attribution, tool-call payloads, and sub-agent nesting for a single SpreadsheetBench trajectory.

with a three-way split that adds prune patches for removing waste from successful trajectories. CoEvoSkills [28] couples a Skill Generator with a co-evolving Surrogate Verifier, achieving 71.1% pass rate on SkillsBench through per-task iterative optimization rather than cross-task distillation. EvoSkill [1] iteratively diagnoses failures and validates skill updates with ground-truth supervision. AutoSkill [26] maintains a skill lifecycle from user chat trajectories. SkillWeaver [29] generates web API skills through structured exploration. These systems address different settings from our offline cross-task distillation.

Experience memory and observability. Voyager [21] accumulates skills through open-ended interaction. Reflexion [20] refines decisions through verbal self-reflection. ReasoningBank [17] stores trajectory-derived insights in a retrieval bank; Trace2Skill shows that distillation into a compact skill document outperforms retrieval by 13.8 percentage points. SkillRL [24] co-evolves skills and policies via reinforcement learning; we study frozen-model, training-free adaptation only. On the observability side, LangSmith, Langfuse, Phoenix, and Arize build on OpenTelemetry [16] conventions to provide step-level logs and per-span token costs. Dong et al. [5] survey the emerging AgentOps landscape and identify cost tracking and multi-agent tracing as open challenges. These tools serve human operators; none produces a compact intermediate representation that a distillation pipeline can consume directly.

LLM cost optimization. FrugalGPT [4] reduces LLM inference cost through model cascading and caching, and RouteLLM [15] learns to route queries to cheaper models when quality permits. These systems optimize cost at inference time. CostCraft operates at a different point: it mines cost patterns from past trajectories to produce reusable skill rules that reduce waste in future runs.

3 Method

The system has three stages (Figure 1). In the **Capture** stage, the ClawTrace plugin records raw trace events during an agent session and writes them to cloud storage. In the **Compile** stage, a deterministic compiler summarizes each session into a TraceCard. In the **Distill** stage, CostCraft reads TraceCards and produces an evolved skill document through parallel analysis and conflict-aware merging.

3.1 ClawTrace Instrumentation

ClawTrace is an OpenClaw-native plugin that registers eight event hooks covering an agent session's full lifecycle: session start and end, LLM input and output, tool-call entry and exit, and sub-agent spawn and termination. Figure 2 shows the resulting execution-path view with per-span cost attribution. The plugin batches events in memory and flushes them on agent shutdown to a cloud ingest endpoint (`POST /v1/traces/events`). The endpoint accepts plain JSON with no framework-specific dependency, so any agent harness can produce valid downstream artifacts by posting conformant payloads. A graph-lakehouse pipeline built on PuppyGraph [18] materializes eight Iceberg silver tables from the raw events. Two design decisions are load-bearing for the rest of the paper.

Cross-trace linkage for multi-agent runs. Modern multi-agent systems [7, 11, 22, 23] routinely spawn sub-agents that delegate subtasks, producing nested call trees that a flat trace cannot represent. When OpenClaw spawns a sub-agent, ClawTrace links the child session back to the parent's tool-call span through a `childSessionKey` → `parentSpanId` map that persists across flush boundaries. Generic observability stacks support parent-child hierarchies but treat each sub-agent as a separate trace, requiring manual tagging to recover the call graph.

Cache-aware per-step cost. ClawTrace bills four token rates separately: `input`, `output`, `cacheRead`, and `cacheWrite`. Providers price cache reads at roughly one-tenth the fresh input rate. On a typical 50-trajectory SpreadsheetBench run, cache reads constitute 30 to 50 percent of total input volume. Counting them at the fresh-input rate would overstate true cost by 1.6 to 2.0× and would distort the per-step cost ranking that the distillation pipeline depends on.

3.2 TraceCard Compilation

A TraceCard is a compact YAML summary of one agent session, produced deterministically from the session's span tree. A typical TraceCard is 1.2 to 1.8 kB, small enough that dozens fit into an analyst's context window. Table 1 lists the schema fields. Cost and token fields are computed directly from the spans and the provider's pricing table. Three fields use heuristics. `role_hint` classifies each LLM step into one of five categories (e.g., `tool_call`, `final_reply`) based on the step's tool-call ratio and position in the turn sequence. `redundant_tool_calls` groups tool calls with the same name whose arguments are at least 80% similar by normalized Levenshtein distance, flagging any group of two or more as redundant. `sub_agents.output_used_in_final` measures how much of a sub-agent's output appears in the parent's final message using Jaccard overlap.

We audited the redundancy detector on 10 traces (5 with redundant clusters, 5 without). Precision was 100%: all 8 detected clusters were genuine redundant file reads. Recall was roughly 80%; one exact-duplicate pair was missed because heredoc payloads drifted past the similarity threshold. The sub-agent Jaccard heuristic is an instrumentation capability that has not been validated on this dataset, because none of the 50 SpreadsheetBench baselines under `openai-codex/gpt-5.4` spawned sub-agents. Evaluating it requires a multi-agent workload, which we leave to future work.

Table 1. TraceCard schema. Heuristic fields are marked with †.

Field	Description
`total_cost_usd`	Cache-aware USD cost for the session
`total_tokens`	`input`, `output`, `cacheRead`, `cacheWrite`
`top_cost_spans`	Top-5 spans sorted by `cost_usd`, each with `kind`, `role_hint`†, `tokens`, `args_sample`
`redundant_tool_calls`†	Clusters of ≥2 calls with Levenshtein ≥0.8
`sub_agents`	Per-child `total_cost_usd` and `output_used_in_final`† (Jaccard)
`failed_or_repaired`	Tool results matching error patterns

3.3 CostCraft: Three-Action Distillation

CostCraft is a three-stage distillation pipeline (Figure 1, right panel). Stage 1 runs a set of tasks without any skill and records their trajectories. Stage 2 analyzes each trajectory and proposes skill patches. Stage 3 merges all patches into one skill document. The architecture builds on Trace2Skill's [14] parallel-analyst and hierarchical-merge design; our contributions are the three-way patch typology, the cost-attributed analyst input, and the conflict-aware merge.

Three patch types. Each patch is labeled with one of three actions. *Preserve* keeps a behavior that contributed to success. *Prune* removes a step that was expensive but did not affect the outcome; it must name the specific high-cost step it targets and include a counterfactual argument for why removal is safe. *Repair* fixes a failure mode found in a failed trajectory, grounded in oracle evidence. The key distinction from prior work is that prune patches come from *successful* trajectories: the task already passed, so the patch is about efficiency, not correctness.

Success Analyst. This analyst examines each successful trajectory and produces up to two patches: one preserve patch describing the behavior to keep, and optionally one prune patch targeting an expensive step. A prune patch is admitted only when three conditions hold: the analyst names a specific entry from the TraceCard's `top_cost_spans` list, provides a natural-language counterfactual, and phrases the rule as something to avoid rather than a hard cost cap.

Error Analyst. This analyst examines each failed or partially failed trajectory using a multi-turn ReAct loop with three tools: `inspect_mismatches` (read which rubric items failed), `read_gold_snippet` (look up the expected answer), and `final_patch` (emit the repair). Oracle access is available only during offline skill authoring, not at agent inference time; the resulting repair rules are distilled into the skill document and applied without oracle access on future tasks. Preserve and prune patches require no oracle at all, since they are mined from naturally successful trajectories. Only repair patches use supervised labels, and only during the offline evolution phase. The analyst has a budget of 3 lookups. If it cannot diagnose the failure within that budget, it emits a low-confidence patch that the merge step deprioritizes.

Conflict-aware merge. Stage 3 merges all patches into a single skill document via LLM-based hierarchical reduction. The merge enforces a priority order: repair patches with causal diagnosis rank highest, then prune patches with a named cost target and counterfactual, then preserve patches that appear in two or more trajectories. Singleton preserve patches are dropped. When two patches conflict, repair supersedes prune, which supersedes preserve. The final skill has five sections: Trigger, Workflow, Stop rules, Artifact checklist, and Cost control. Post-checks enforce section-heading presence, a 1200-token ceiling, and no task-specific information leakage.

3.4 Failure Taxonomy

To motivate the three-way patch typology, we built an operational taxonomy from the 16 non-perfect baselines in our 50-task SpreadsheetBench sample, categorizing them into seven failure types (Table 2). This taxonomy is specific to our sample and annotator; we use it to guide analysis, not as a general classification of agent failures. Following Husain [8], we coded each baseline by its failure reason, its first three mismatched rubric items, and its TraceCard step-type distribution.

Table 2. Operational failure taxonomy (16 non-perfect baselines). Each category maps to a patch type.

ID	Category	N	%	Patch
T1	No deliverable	3	19	repair
T2	Wrong content type	1	6	repair
T3	Formula not eval.	1	6	repair
T4	Placeholder mismatch	4	25	repair
T5	Case/whitespace	4	25	repair
T6	Logic error	2	13	preserve
T7	Precision rounding	1	6	repair

Six of seven categories map to repair; one maps to preserve; none maps to prune. This is expected: a failure cannot be fixed by removing a behavior that was never there. Prune patches come only from successful trajectories that contain wasteful steps. The taxonomy also predicts the low prune-match rate in Section 4: prune rules can only help held-out tasks whose baselines happen to exhibit the same waste patterns observed during training. At our 10-task training scale, the learned prune rules cover two specific wastes (unnecessary workspace-file reads and redundant file re-reads), so held-out tasks without those patterns do not benefit.

Table 3. Experimental conditions. TC = TraceCard, CF = counterfactual. All conditions share the same merge algorithm and evolve set.

Condition	Cost in TC	Prune	CF gate
Baseline (S_0)	N/A	N/A	N/A
Full CostCraft	✓	merged	enforced
No-prune	✓	discarded	N/A
No-cost-attr.	stripped	merged	enforced
No-CF	✓	merged	disabled

4 Experiments

We ask four questions. (1) Does cost information in TraceCards matter for distillation quality? (2) Do prune patches protect quality, or do they only reduce cost? (3) Do aggregate metrics hide important per-task effects? (4) Does a skill trained on SpreadsheetBench transfer to a different benchmark? All agent runs use `openai-codex/gpt-5.4` via OpenClaw with ClawTrace instrumentation; we report two seeds (`seed=0` and `seed=1`) on the SpreadsheetBench held-out set unless noted otherwise.

4.1 Setup

We evaluate on SpreadsheetBench [13], a 912-task benchmark where the agent must produce a spreadsheet and a deterministic grader checks each cell against the gold answer. Per-task quality $Q \in [0, 1]$ is the fraction of rubric items satisfied. Q=1.0 means every cell matches; Q=0 means no scorable file was produced. Per-task cost is the cache-aware USD total from ClawTrace. We sanitize the agent workspace before each run to prevent earlier runs from leaking learned context into later ones.

The experiment has four phases. First, we run 50 tasks sampled from the 200-task professional subset, stratified by difficulty, with no skill and record each trajectory. Second, we split the 50 tasks into an evolve set of 10 for training, a held-out set of 30 for evaluation, and 10 reserved for pipeline development. The evolve set contains 4 successes, 4 partial successes, and 2 failures. Third, we run CostCraft on the 10 evolve-set TraceCards to produce one skill. We repeat this step under four ablation conditions (Table 3), each time removing one pipeline signal to produce a different skill. Fourth, we run all 30 held-out tasks under each skill at the

same seed as the original baseline and compare quality and cost per task.

The ablation conditions map onto prior work. No-cost-attribution approximates the Trace2Skill [14] information regime: the analyst sees trajectory structure and outcomes but not per-step cost. No-prune approximates a success/failure-only patch policy where successful trajectories yield only preserve rules.

Skill distillation always risks regressions: rules learned from one task can conflict with the requirements of another. SkillsBench [12] reports this pattern even for human-curated skills (16 of 84 tasks get worse). Our evaluation measures whether each pipeline signal reduces the regression rate, not whether distillation eliminates regressions entirely.

4.2 Main Results

Each held-out task is compared to its own baseline under the same seed. A task regresses when $Q_{\text{skill}} < Q_{\text{baseline}} - 0.01$; a win is the reverse. Figure 3 reports the outcomes.

Cost attribution and prune rules each carry independent weight. Stripping cost from TraceCards (the No-cost-attribution condition) more than doubles the median cost uplift on successful tasks, from +22% to +49%, and raises the regression count from 4 to 6. Five of those six are catastrophic, with the agent stopping at Q=0 without producing output; Full CostCraft exhibits this failure mode only once. Independently, removing prune patches (the No-prune condition) triples the regression count at seed 0 from 4 to 13 while median cost stays comparable (+15% vs. +21%), and eight of the 13 No-prune regressions produce no output at all. The two ablations alter different pipeline signals, so the effects can be attributed separately to cost attribution and to prune-rule presence.

The prune-derived Cost-control section thus functions as a quality guardrail in this ablation rather than as a pure cost-compression mechanism. Without it, the remaining skill sections push the agent toward broken behavior on tasks that already worked: the agent continues issuing tool calls but never writes the final answer. This protective role is orthogonal to cost compression and is not captured by Trace2Skill's [14] success-versus-error split.

The regression-count gap is robust across seeds; severity is not. Repeating all 30 held-out tasks at `seed=1` preserves the direction: Full CC has 3 regressions versus No-prune's 5, and pooled per-task means across both seeds show 5 versus 14 regressions (2.8×). The "no deliverable" severity is more seed-sensitive: 11 of 13 No-prune regressions deliver Q=0 at seed 0, but only 1 of 5 at seed 1. Within-seed cost has a coefficient of variation near 43% on this benchmark, so we report cost percentages as approximate; per-seed and pooled tables are in Appendix B.10.

Aggregation hides per-task effects. All three skill conditions show a positive median cost change, yet the same conditions contain two full recoveries (Q=0 → 1.0 and Q=0 → 0.84) alongside catastrophic No-prune failures. Without a per-task breakdown, these wins and losses cancel and the result reads as noise.

4.3 Regime-Partitioned Breakdown

Figure 4 shows the cumulative effect: each ablation removes one pipeline signal, and quality preservation drops monotonically from 83.3% under Full CostCraft to 53.3% under No-prune. Splitting the 30 held-out tasks by their baseline outcome reveals effects that aggregate numbers cancel out (Table 5 in Appendix B.9). On failed baselines, repair patches recover 2 of 3 tasks to near-perfect quality, at a cost premium that failed tasks can tolerate. On successful baselines, 2 of 17 tasks contain the specific waste patterns targeted by the learned prune rules. On those 2 tasks, Full CostCraft still increases cost (median +30%), but the No-prune ablation causes one of them to fail entirely (Q=0).

4.4 Prune Coverage and the Efficiency Lever

The trained skill contains two prune rules, each learned from 3 training trajectories: (1) skip workspace-memory files (`MEMORY.md`, `SOUL.md`) when the task is self-contained, and (2) read each input file once and cache its content rather than re-reading.

A held-out task matches a prune rule when its baseline TraceCard contains the targeted waste pattern. In our 30-task sample, 2 of 17 successful held-out tasks match (11.8%). On those 2 tasks, Full CostCraft still increases cost (+38% and +22%), but the No-prune ablation causes one to fail entirely (Q=0). In this sample, the prune rules are preventing quality collapse rather than compressing cost; whether this protective role generalizes beyond our training scale remains open.

Counterfactual gate (No-CF).. On a 15-task success-heavy subset, disabling the counterfactual admission gate produced 2 additional regressions compared to Full CostCraft, even though the analyst voluntarily included counterfactual text in all prune patches regardless of the gate setting. The gate appears to shape counterfactual *quality* rather than *presence*. The sample is small (N=15); we treat this as suggestive (Appendix B.7).

Three per-task case studies (recovery, protection, and regression) are in Appendix B.5.

4.5 SkillsBench Cross-Benchmark Evaluation

To test whether a SpreadsheetBench-trained skill transfers to other task types, we ran the full SkillsBench [12] benchmark (84 tasks) under two conditions: baseline (`gpt-5.4` with no skill) and baseline plus the SpreadsheetBench-trained CostCraft skill. The tasks cover data processing, scientific computing, document analysis, security, code generation, video processing, and agent planning; none involves spreadsheets. Each task is graded by a Docker-contained pytest

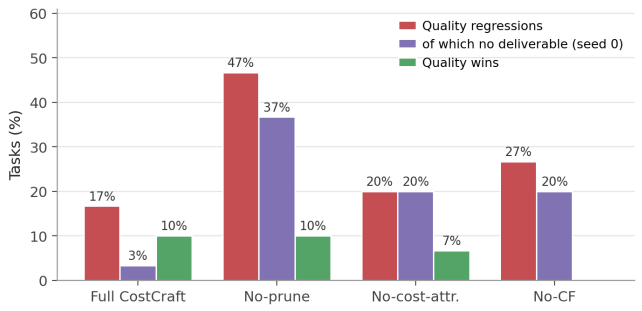

Figure 3. Quality outcome rates across ablation conditions on 30 held-out SpreadsheetBench tasks. Regressions and wins for Full CostCraft and No-prune are pooled per-task means across two seeds; No-cost-attribution and No-CF report a single seed (N=30 and N=15). The middle (purple) bar in each group is the seed-0 subset of regressions that delivered Q=0 from a passing baseline; this severity is seed-sensitive (Appendix B.10). Full CostCraft has the lowest regression rate (17%) and matches No-prune on win rate (10%).

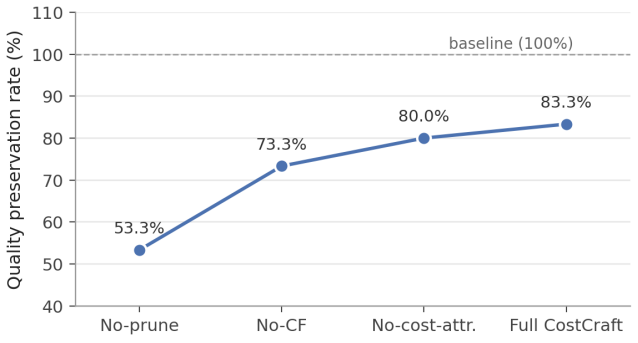

Figure 4. Quality preservation rate across ablation conditions. Full CostCraft and No-prune use pooled per-task means across two seeds; No-cost-attribution and No-CF use a single seed. Each point removes one pipeline signal (left to right adds signal). Full CostCraft preserves quality on 83.3%; No-prune drops to 53.3%.

verifier that emits a binary pass/fail. Three N values appear in what follows. Of the 84 tasks, 4 failed at the Docker-eval container layer on both conditions (image build errors or test timeouts) and contribute no usable signal. The remaining tasks yield **82 cost-paired tasks** from which Figure 5 is drawn, **74 quality-gradeable pairs** on which we report wins, regressions, and ties (8 had build or grading issues on at least one side), and **79 paired-direction tasks** for the cost-decrease direction statistic in the next paragraph (computed only on tasks where the baseline billed nonzero cost). ClawTrace's TraceCard compiler ran on every trace without modification.

CostCraft does not save cost on aggregate over this benchmark: the median baseline cost is \$0.143 and the median CostCraft cost is \$0.144, indistinguishable at the precision the experiment supports. Many SkillsBench tasks fail under both systems, so aggregate cost mixes successful and no-deliverable trajectories and is dominated by domain-hard items such as manufacturing scheduling, PDDL planning, and quantum simulation, on which the baseline returns no deliverable and CostCraft neither helps nor hurts. We therefore treat paired direction and trace-attributed failure mode as the primary transfer signal rather than the aggregate. Cost decreased on 41 of 79 paired tasks (52%), with a median −45% reduction where it decreased and a median +43% increase where it increased. Figure 5 visualizes the full-benchmark cost pattern, with the five quality-changing events highlighted.

Prune rules transferred; preserve rules overfit. Each of the five quality-changing events on the full benchmark is attributable, by inspection of the merged skill, to a single patch

Table 4. Trace-attributed SkillsBench events. The three regressions trace to the preserve lane (SpreadsheetBench-specific formatting rules); the two wins trace to the prune lane (cost-control rules freeing session headroom). "—" marks an undefined ratio (baseline bills no cost).

Task	Outcome	Q	Δ cost
Regressions (preserve lane)			
`energy-market-pricing`	reg.	1.0 → 0.0	−63%
`gravitational-wave-detection`	reg.	1.0 → 0.0	−50%
`invoice-fraud-detection`	reg.	1.0 → 0.0	+21%
Wins (prune lane)			
`citation-check`	win	0.0 → 1.0	—
`glm-lake-mendota`	win	0.0 → 1.0	+74%

lane (Table 4). The three regressions trace to the preserve lane: its SpreadsheetBench-specific formatting rules inject conventions that the SkillsBench verifier does not expect. Two of the three also exhibit large cost drops (−63%, −50%) because the agent follows the bad rule but does so cheaply, never reaching the work the verifier scores. The two wins trace to the prune lane: the cost-control rules free enough session headroom for the agent to reach a final output step that the baseline either fails to attempt (`citation-check` bills no cost on the baseline) or attempts at lower fidelity. Quality recovery comes at a cost premium on `glm-lake-mendota` (+74%); the per-call cost reduction from the "read each file once and cache" rule shows up not on the wins but on the broader 52% of paired tasks where cost decreased.

The asymmetry is the central cross-benchmark observation. At full-benchmark scale, the regression-to-win ratio

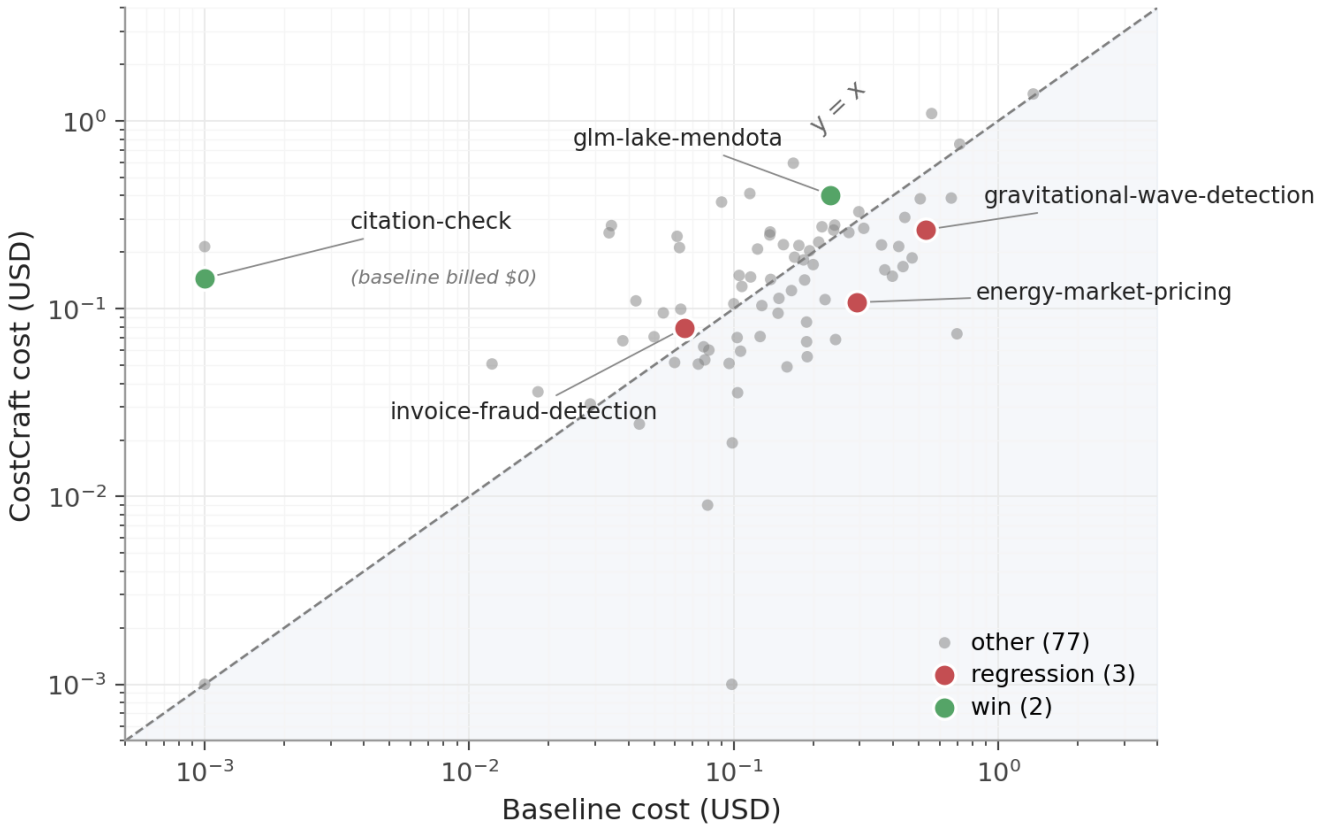

Figure 5. Per-task cost on the full 84-task SkillsBench benchmark (log–log axes; 82 cost-paired tasks shown after Docker-eval failures). Each marker is one task. Below the dashed y=x reference, CostCraft is cheaper than the baseline; above, more expensive. Quality outcomes are colored: green for the two wins, red for the three regressions; all other tasks are gray. The wins and regressions are the same five events enumerated in Table 4. `citation-check` sits at the left edge because the baseline billed no cost (clamped to $\$10^{-3}$ for log display).

is 3:2, and the lane attribution is clean rather than mixed: every regression is a preserve-lane case and every win is a prune-lane case. We therefore treat the 3:2 figure not as five stochastic outcomes but as five trace-attributed mechanism instances, each documented by an explicit rule conflict or rule firing. Separating the two patch types at distillation time appears important for cross-task generalization; we treat the ratio itself as a directional signal rather than a stable rate estimate, since larger samples would be needed to place a confidence interval on it.

Tracing overhead is bounded at ≈0.30% of agent wall time with zero quality divergence on 10 paired ON/OFF reruns (Appendix B.8).

5 Limitations

The claims in this paper are scoped to a single agent runtime: **OpenClaw with the `openai-codex/gpt-5.4` backbone**. Every result reported in the abstract, §4, and §6 is for that combination; we make no claim about Claude Code [2], AutoGen [23], MetaGPT [7], or any other agent runtime, and we make no claim about other LLM backbones such as Claude, Gemini, or open-weights models. Whether the rule-type transfer asymmetry generalizes beyond this single agent–backbone pair is open. Within this scope, we evaluate 30 SpreadsheetBench held-out tasks at two seeds (60 runs per condition) and 74 quality-gradeable SkillsBench pairs out of 84. Within-seed cost has a coefficient of variation near 43% on this benchmark, so cost percentages are calibrated to two-seed pooled means and a third seed would not tighten them meaningfully. The catastrophic No-prune severity in Section 4.2 (deliveries of $Q = 0$ on tasks where the baseline succeeded) is seed-0-specific: the regression *count* gap between Full CC and No-prune is preserved at seed 1, but the severity collapses (one $Q = 0$ run at seed 1 versus eleven at seed 0). The failure taxonomy was coded by one annotator; a second annotator would be needed to measure inter-coder reliability. Two of the three heuristic TraceCard fields (`sub_agents.output_used_in_final` and

`failed_or_repaired_steps`) were not exercised because the tested backbone did not spawn sub-agents.

At the 10-task training scale, two prune rules emerge and match 2 of 17 successful held-out tasks; a richer prune library would require a larger training set. The SkillsBench evaluation tests cross-benchmark transfer but does not repeat the full ablation matrix (No-prune, No-cost-attribution) on SkillsBench; that ablation is the natural next step. ClawTrace currently supports OpenClaw; portability to other agent harnesses through the ingest API has not been validated end-to-end. We treat these boundaries as the scope of the present claims, not as fragility. The empirical levers most likely to extend the rule-type transfer story beyond this perimeter are multi-backbone evaluation, multi-runtime evaluation (Claude Code, AutoGen, others), a larger prune library, and a SkillsBench ablation matrix.

6 Conclusion

Distilled agent skills should be evaluated at the rule-type level, not as monolithic instruction packages, because their component rule types transfer in qualitatively different ways. Two pieces of evidence carry this argument. Within-distribution, removing prune patches roughly tripled the regression count on SpreadsheetBench across two seeds, while median cost stayed unchanged: prune rules at training scale act as a quality guardrail. Cross-distribution, on the full 84-task SkillsBench transfer, all three quality regressions trace to the preserve lane and both quality wins trace to the prune lane (Table 4); the same lanes drive cost decreases on roughly half of paired tasks. The asymmetry is what aggregate metrics hide.

We release ClawTrace, all TraceCards, and evolved skills at https://github.com/epsilla-cloud/clawtrace; the project page is at https://www.clawtrace.ai/. All evidence reported here is for the OpenClaw runtime with the `gpt-5.4` backbone. Whether the pattern generalizes to other runtimes or other backbones is open, and is the natural next direction together with a larger evolve set and a SkillsBench ablation matrix.

References

[1] Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. 2026. EvoSkill: Automated Skill Discovery for Multi-Agent Systems. arXiv:2603.02766 [cs.AI] https://arxiv.org/abs/2603.02766

[2] Anthropic. 2025. Claude Code. https://www.anthropic.com/claude-code Accessed: 2026-05-01.

[3] Arize AI. 2024. Phoenix Documentation: LLM Evals. https://arize.com/docs/phoenix/evaluation/llm-evals Accessed: 2026-04-24.

[4] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. arXiv:2305.05176 [cs.LG] https://arxiv.org/abs/2305.05176

[5] Liming Dong, Qinghua Lu, and Liming Zhu. 2024. AgentOps: Enabling Observability of LLM Agents. arXiv:2411.05285 [cs.AI] https://arxiv.org/abs/2411.05285

[6] Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. 2026. SWE-Skills-Bench: Do Agent Skills Actually Help in Real-World Software Engineering? arXiv:2603.15401 [cs.SE] https://arxiv.org/abs/2603.15401

[7] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352 [cs.AI] https://arxiv.org/abs/2308.00352

[8] Hamel Husain and Shreya Shankar. 2026. LLM Evals: Everything You Need to Know. https://hamel.dev/blog/posts/evals-faq/ Accessed: 2026-04-24.

[9] LangChain. 2024. LangSmith Evaluation. https://docs.langchain.com/langsmith/evaluation Accessed: 2026-04-24.

[10] Langfuse. 2024. Langfuse Documentation. https://langfuse.com/docs Accessed: 2026-04-24.

[11] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. arXiv:2303.17760 [cs.AI] https://arxiv.org/abs/2303.17760

[12] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han chung Lee. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. arXiv:2602.12670 [cs.AI] https://arxiv.org/abs/2602.12670

[13] Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. SpreadsheetBench: Towards Challenging Real World Spreadsheet Manipulation. arXiv:2406.14991 [cs.CL] https://arxiv.org/abs/2406.14991

[14] Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. 2026. Trace2Skill: Distill Trajectory-Local Lessons into Transferable Agent Skills. arXiv:2603.25158 [cs.AI] https://arxiv.org/abs/2603.25158

[15] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. 2025. RouteLLM: Learning to Route LLMs with Preference Data. arXiv:2406.18665 [cs.LG] https://arxiv.org/abs/2406.18665

[16] OpenTelemetry Authors. 2024. OpenTelemetry Specification. https://opentelemetry.io/docs/specs/otel/ Accessed: 2026-04-25.

[17] Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. 2026. ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory. arXiv:2509.25140 [cs.AI] https://arxiv.org/abs/2509.25140

[18] PuppyGraph. 2024. PuppyGraph: Query Graph on Data Lakes. https://www.puppygraph.com/ Accessed: 2026-04-25.

[19] David Schmotz, Sahar Abdelnabi, and Maksym Andriushchenko. 2025. Agent Skills Enable a New Class of Realistic and Trivially Simple Prompt Injections. arXiv:2510.26328 [cs.LG] https://arxiv.org/abs/2510.26328

[20] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366 [cs.AI] https://arxiv.org/abs/2303.11366

[21] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291 [cs.AI] https://arxiv.org/abs/2305.16291

[22] Ziqi Wang and Boqin Yuan. 2026. L-MARS: Legal Multi-Agent Workflow with Orchestrated Reasoning and Agentic Search. arXiv:2509.00761 [cs.AI] https://arxiv.org/abs/2509.00761

[23] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155 [cs.AI] https://arxiv.org/abs/2308.08155

[24] Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. 2026. SkillRL: Evolving Agents via Recursive Skill-Augmented Reinforcement Learning. arXiv:2602.08234 [cs.LG] https://arxiv.org/abs/2602.08234

[25] Renjun Xu and Yang Yan. 2026. Agent Skills for Large Language Models: Architecture, Acquisition, Security, and the Path Forward. arXiv:2602.12430 [cs.MA] https://arxiv.org/abs/2602.12430

[26] Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. 2026. AutoSkill: Experience-Driven Lifelong Learning via Skill Self-Evolution. arXiv:2603.01145 [cs.AI] https://arxiv.org/abs/2603.01145

[27] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

[28] Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. 2026. Co-EvoSkills: Self-Evolving Agent Skills via Co-Evolutionary Verification. arXiv:2604.01687 [cs.AI] https://arxiv.org/abs/2604.01687

[29] Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. 2025. SkillWeaver: Web Agents can Self-Improve by Discovering and Honing Skills. arXiv:2504.07079 [cs.AI] https://arxiv.org/abs/2504.07079

Appendix

A ClawTrace Platform Demo

ClawTrace is an observability and optimization platform for OpenClaw agents. Its goal is to make agents better, cheaper, and faster by giving operators real-time tracing, cost analysis, and actionable recommendations. The platform provides four synchronized views over the same span tree, each serving a different stage of the diagnosis workflow.

The execution-path view (Figure 2 in the main text) renders the full call graph of an agent session as an interactive trace tree. Every LLM call, tool use, and sub-agent delegation is visible with its payload, token breakdown, and USD cost. This is the view the CostCraft analyst consumes through the TraceCard abstraction.

The trajectory dashboard (Figure 6) lists all agent runs with columns for total cost, token count, step count, and outcome. Daily trend charts and per-agent filtering let operators spot cost spikes before drilling into individual traces. The step timeline (Figure 7) renders each span as a Gantt bar whose length is wall-clock duration, making parallelization opportunities and redundant call clusters visible at a glance.

Upcoming features include rubric-based evaluation, A/B testing across skill versions, and a closed loop that feeds trace-derived findings back into skill updates without manual intervention.

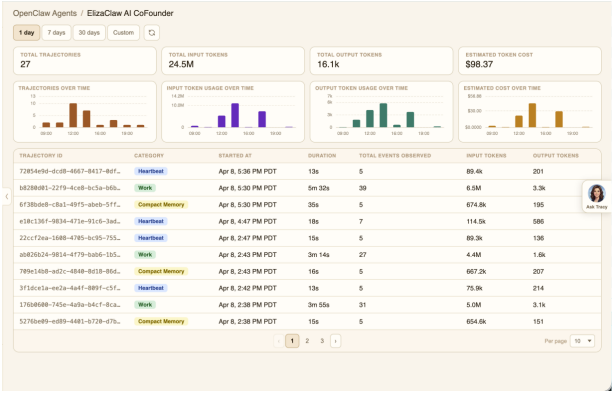

Figure 6. Trajectory dashboard. Each row is one agent run, with columns for total cost, token count, step count, and outcome. Daily trend charts and per-agent filtering let operators spot cost spikes before drilling into individual traces.

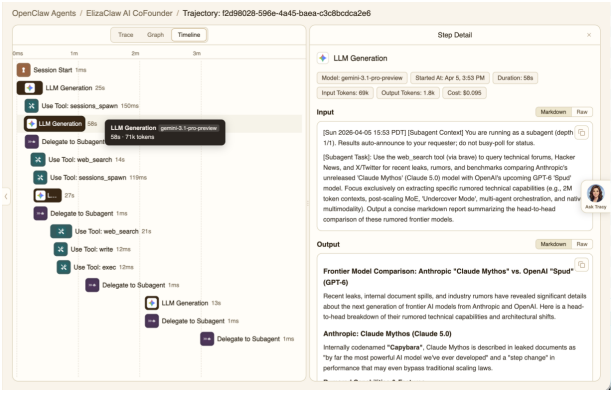

Figure 7. Step timeline (Gantt view) for a single trajectory. Each bar is one span; bar length is wall-clock duration. Redundant tool calls appear as visually identical adjacent bars, the same pattern the TraceCard `redundant_tool_calls` heuristic flags programmatically.

B Detailed Analysis of Findings

B.1 Why Cost Attribution Matters. With the analyst, merge algorithm, and training set held fixed, stripping cost fields from TraceCards raises the median cost uplift on successful tasks from +22% to +49% and increases catastrophic no-deliverable failures from 1 to 5. The only variable that changed is what the analyst sees. Without per-step cost, the analyst removes structural features that were actually necessary, and the merge step does not catch the error until the agent fails to produce output. Five of the six No-cost-attribution regressions produce no deliverable at all (Q=0), a pattern Full CostCraft shows only once.

B.2 Why Prune Is Protective Before Compressive. Removing prune patches triples quality regressions (4 to 13) while median cost stays comparable (+15% vs. +21%). Eight of the 13 No-prune regressions produce no output: the agent runs tool calls but never writes the final answer. The Cost-control section of the skill, which contains only prune-derived rules, acts as a guardrail that keeps the other skill sections from breaking tasks that already worked. At this scale, the prune rules are protecting quality rather than compressing cost. The compression claim requires more data; the protection claim is what the current results support.

B.3 Sources of Quality Regression Under Full CostCraft. Four quality regressions sit in the 30-task held-out set at seed 0. Task 42034 (Q: 1.0 → 0.667) regresses because an over-generalized placeholder-formatting repair patch writes `N/A` for missing values where the rubric expects empty cells. Task 48745 (Q: 1.0 → 0.667) regresses from an over-general preserve patch that misroutes a sheet-level write ordering. Task 55535 (Q: 1.0 → 0.471) follows a similar pattern; the No-prune skill preserves quality on this task because the regression-inducing rule lives in Workflow, not Cost-control. Task 76-42 (Q: 0.94 → 0.0) is the most severe of the four: the agent under Full CostCraft fails to produce a scorable deliverable. We did not isolate the responsible patch on this task.

B.4 Why the Efficiency Regime Is Hard. A prune rule can only help a held-out task whose baseline actually exhibits the waste pattern the rule targets. Agent behavior is stochastic: the same task run twice often wastes tokens on different things. The prune-rule library must grow with observed waste diversity, not just with training set size. At the 10-task training scale, the two learned prune rules match only 2 of 17 successful held-out tasks. On those tasks, Full CostCraft still increases cost (+38% and +22%), but the No-prune ablation causes one to fail entirely (Q=0). The prune rules are stabilizing quality on matched tasks rather than reducing their cost.

B.5 Case Studies.

Recovery: task 488-29 (Q: 0 → 1.0). The baseline TraceCard logs a `failed_or_repaired_step` at turn 4: the agent wrote a placeholder and ended the session without returning to fill it. The error analyst emitted a repair patch: "When a cell is marked pending, compute it before ending the session." In the Full CostCraft run the agent computes and submits at Q=1.0. Cost rises from \$0.021 to \$0.118 (+461%). The regime-partitioned evaluation makes this cost-for-correctness trade legible.

Prune protection: task 47484 (Q=1.0 preserved). The baseline succeeds at Q=1.0 for \$0.068. Full CostCraft holds quality at 1.0 with cost \$0.083 (+22%). Under No-prune, the agent produces no deliverable (Q=0, cost ≈\$0). The No-prune skill's Workflow section, stripped of Cost-control discipline, leads the agent into a terminal state without writing the answer sheet. Of the 8 No-prune catastrophic failures, 6 occur on tasks whose baselines succeed at Q=1.0.

Global-mutation regression: task 42034 (Q: 1.0 → 0.667). The Full CostCraft Workflow section adopts a preserved rule from a different trajectory ("Write `N/A` for missing placeholder values") that conflicts with this task's rubric, which expects literal empty cells. This illustrates why we treat "prune is protective" as a statistical finding (13 vs. 4 regressions) rather than a per-task guarantee.

B.6 SkillsBench Transfer Case Studies. The five quality-changing events on the 84-task SkillsBench transfer benchmark fall into two clean groups by patch lane (Table 4). The two wins trace to the prune lane; the three regressions trace to the preserve lane. Per-task narratives below.

Win, prune lane:* `citation-check` *(Q: 0.0 → 1.0). The baseline run bills no LLM cost: the agent reads the input, immediately calls a tool that errors on a missing citation database connection, and exits without producing a scored deliverable. Under Full CostCraft, the "read each file once and cache" prune rule changes the access pattern: the agent reads the bibliography once, caches the result, and proceeds to the verification step that the baseline never reached. CostCraft cost is \$0.145; baseline cost is \$0. The win is not a cost-cheaper path but a path that exists at all.

Win, prune lane:* `glm-lake-mendota` *(Q: 0.0 → 1.0). The baseline costs \$0.233 and ends in failure: the agent rereads the same Lake Mendota CSV file three times across the session, exhausts its turn budget on data reorganization, and never reaches the model-fitting step. Full CostCraft costs \$0.404 (+74%) and reaches Q=1.0. Quality recovery comes at a cost premium because the prune rule frees enough headroom for the agent to attempt the substantive work, which is itself expensive on this task.

Regression, preserve lane:* `energy-market-pricing` *(Q: 1.0 → 0.0, −63% cost). The baseline succeeds at \$0.292; Full CostCraft fails at \$0.108. The merged skill's Workflow section contains a preserved rule from a SpreadsheetBench trajectory that instructs the agent to format outputs as a spreadsheet table. The SkillsBench verifier expects a JSON market-pricing report; the formatting rule routes the agent into a sheet-building path that bills less because it never produces the actual computation. Cheaper but unscored is the failure mode this paper warns about.

Regression, preserve lane:* `gravitational-wave-detection` *(Q: 1.0 → 0.0, −50% cost). Same mechanism as `energy-market-pricing`. The baseline reads the strain time-series, runs a matched-filter step, and writes a detection table at \$0.531. Under Full CostCraft, the preserved formatting rules trigger an XLSX-style output path; the verifier expects

a HDF5 metadata file plus a plain-text summary. Cost drops because the agent skips the heavy matched-filter step that the new output format does not require, and the verifier scores zero. The trace's `top\_cost\_spans` confirms the matched-filter step is missing from the CostCraft trajectory.

Regression, preserve lane: **`invoice-fraud-detection`** ***(Q: 1.0 → 0.0, +21% cost).*** The exception in the regression set: cost goes up rather than down. The agent under Full CostCraft applies the preserve-lane formatting rules and additionally runs the prune-lane cache discipline; the result is a longer session that produces a CSV-formatted answer instead of the expected JSON. Quality goes to zero because of the format mismatch; cost rises because the agent does the substantive analysis but then formats it incorrectly. This is the only regression where the agent reaches the verifier with a wrong-but-substantive answer rather than a no-deliverable trajectory.

The three preserve-lane regressions together motivate the rule-type-evaluation thesis. Each is a documented mechanism instance in which a SpreadsheetBench-specific convention misroutes the agent on a task type the convention does not apply to. None of them would have been caught by an aggregate cost or aggregate quality summary.

B.7 Counterfactual Admission Ablation. On a 15-task success-heavy slice we compare Full CostCraft against No-counterfactual. The analyst emits counterfactuals voluntarily even with the gate disabled: all 3 prune patches under `require\_counterfactual=False` contain a non-empty counterfactual field, because the LLM has internalized the prompt-level instruction independently of the admission check. Yet the merged skill regresses on 2 tasks that Full CostCraft preserves at Q=1.0. The gate appears to shape the quality of counterfactuals rather than their presence. The sample is small (N=15); we treat this as suggestive rather than conclusive.

B.8 Tracing Overhead. The ClawTrace plugin batches events in memory and flushes once per session in one HTTPS POST (typical payload 30 to 50 kB). Over 20 trials the median HTTP round-trip is 445 ms against a median agent trajectory wall time of 147 s, yielding roughly 0.30% overhead. A paired ON/OFF rerun on 10 SpreadsheetBench tasks showed zero quality divergence between instrumented and uninstrumented runs.

B.9 Regime-Partitioned Breakdown. The three regimes show distinct cost-quality tradeoffs. In the **success regime** (17 tasks), Full CostCraft preserves quality on 13 tasks (ΔQ=0) while increasing median cost by 22%. The 4 regressions come from over-generalized preserve or repair patches that conflict with task-specific rubric requirements (Section B.5). Under No-prune, 12 of 17 tasks regress, confirming that the Cost-control section is load-bearing for tasks that already pass.

Table 5. Regime-partitioned breakdown (30 held-out tasks).

Regime	Transition	N	Full CC	No-prune
Success (17)	s→s preserved	13	ΔQ=0	5
	s→regress	4	—	12
Partial (10)	tie / minor	6	Δ$+2%	mix
	recovery (Q→1.0)	3	ΔQ+92pp	2
	partial↓	1	—	3
Fail (3)	recovery (Q≥0.8)	2	ΔQ+92pp	1
	fail→fail	1	Δ$+193%	2

Table 6. SpreadsheetBench held-out set (30 tasks at `seed=0` and `seed=1`, plus pooled per-task means.

Metric	Seed 0	Seed 1	Pooled
Full CC quality (mean / median)	0.85 / 1.00	0.89 / 1.00	0.87 / 1.00
No-prune quality (mean / median)	0.58 / 0.96	0.82 / 1.00	0.70 / 0.70
Full CC cost (mean / median)	$0.090 / $0.089	$0.108 / $0.100	—
No-prune cost (mean / median)	$0.063 / $0.073	$0.103 / $0.092	—
Full CC regressions	4 / 30	3 / 30	5 / 30
No-prune regressions	13 / 30	5 / 30	14 / 30
No-prune Q=0 (catastrophic)	11 / 30	1 / 30	—

In the **partial regime** (10 tasks, $0 < Q_{\text{baseline}} < 1$), 3 tasks recover to Q=1.0 under Full CostCraft. These recoveries come from repair patches that address the failure modes identified in the operational taxonomy (Table 2): two are placeholder-mismatch fixes (T4), one is a no-deliverable fix (T1). The cost premium on recovered tasks averages +89%, driven by additional LLM turns the agent uses to verify its output against the repair rule.

In the **fail regime** (3 tasks, Q_{baseline}=0), 2 tasks recover to Q≥0.8. The remaining task stays at Q=0 under all conditions; the error analyst could not diagnose its failure within the 3-lookup budget. Cost on this task rises by 193% because the agent attempts all repair rules before giving up.

B.10 Multi-Seed Robustness. The regression-count gap between Full CC and No-prune is robust across seeds (factor 3.25× at seed 0, 1.67× at seed 1, 2.8× pooled). The catastrophic "no deliverable" severity is more seed-sensitive: 11 of 13 No-prune regressions are Q=0 at seed 0, but only 1 of 5 at seed 1. Within-seed cost has a coefficient of variation near 43% on this benchmark and backbone, so cost percentages reported in the main text are calibrated to two-seed pooled means; a third seed would not tighten them meaningfully.

B.11 Cost Model. ClawTrace computes per-step cost as:

$$\text{cost} = r_{\text{in}} \cdot t_{\text{in}} + r_{\text{out}} \cdot t_{\text{out}} + r_{\text{cacheRead}} \cdot t_{\text{cacheRead}} + r_{\text{cacheWrite}} \cdot t_{\text{cacheWrite}}.$$

where r_* are per-token USD rates and t_* are token counts reported by the provider. For `openai-codex/gpt-5.4` (as of April 2026): r_{in}=$2.00/M, r_{out}=$8.00/M, $r_{\text{cacheRead}}$=$0.50/M, $r_{\text{cacheWrite}}$=$2.00/M. Cache-read tokens are billed at 25% of the fresh input rate. On our 50-trajectory SpreadsheetBench

sample, cache-read tokens constitute 30–50% of total input volume. Counting them at the fresh input rate would overstate true cost by 1.6–2.0×, distorting the step-level cost rankings that CostCraft relies on for prune-patch selection.

C CostCraft Prompt Templates

This section reproduces the analyst and merge prompts used in CostCraft. All prompts are abbreviated; full versions are available in the code repository.

C.1 Success Analyst Prompt.

Success Analyst Prompt

Role: You are an expert analyst for agent skill distillation.
Mission: Given a successful agent trajectory and its TraceCard, produce up to two skill patches: one `preserve` patch describing the behavior to keep, and optionally one `prune` patch targeting an expensive step that did not affect the outcome.
Input: (1) The TraceCard YAML for this session, including `top_cost_spans`, `redundant_tool_calls`, and `total_cost_usd`. (2) The current skill document (`SKILL.md`).
Preserve patch requirements:
- Describe the behavior that contributed to the correct answer.
- Phrase as a general principle, not a task-specific detail.
- Reference the TraceCard step that demonstrates the behavior.

Prune patch requirements (all three must hold):
- Name a specific entry from `top_cost_spans`.
- Provide a natural-language counterfactual: why removing this step would not change the outcome.
- Phrase the rule as a behavior to avoid, not a hard cost cap.

Output format: JSON with fields `action` (`preserve` or `prune`), `rule`, `target_span` (prune only), `counterfactual` (prune only), `confidence` (high/medium/low).

C.2 Error Analyst Prompt.

Error Analyst Prompt

Role: You are an expert failure-analysis agent for spreadsheet tasks.
Mission: Given a failed or partially failed agent trajectory, its TraceCard, and oracle access to the ground truth, diagnose the failure and emit a `repair` patch.
Available tools (budget: 3 calls total):
- `inspect_mismatches`: Read which rubric items the agent output failed to satisfy.
- `read_gold_snippet`: Look up the expected cell values for a specific range.
- `final_patch`: Emit the repair patch and terminate.

Required workflow:
1. Call `inspect_mismatches` to identify the failure surface.
2. Trace the failure to a specific agent decision or missing step in the TraceCard.
3. If needed, call `read_gold_snippet` to confirm the expected output.
4. Call `final_patch` with the repair rule grounded in the observed failure and oracle evidence.

Output format: JSON with fields `action` (`repair`), `rule`, `failure_type`, `evidence`, `confidence`.
Constraint: If you cannot diagnose the failure within 3 tool calls, emit a low-confidence patch. The merge step will deprioritize it.

C.3 Merge Operator Prompt.

Merge Operator Prompt

Role: You are a skill-edit coordinator. You receive independently proposed patches from multiple analyst runs. Your job is to merge them into one coherent, non-redundant skill document.

Priority order:

1. Repair patches with causal diagnosis (highest).
2. Prune patches with a named cost target and counterfactual.
3. Preserve patches that appear in two or more trajectories.
4. Singleton preserve patches (drop these).

Conflict resolution: When two patches target the same behavior, repair supersedes prune, which supersedes preserve. When two patches of the same type conflict, keep the one with stronger evidence.

Output structure: The merged skill must have exactly five sections:

1. **Trigger**: when the skill applies.
2. **Workflow**: step-by-step procedure (preserve-derived).
3. **Stop rules**: when to terminate (repair-derived).
4. **Artifact checklist**: output verification (repair-derived).
5. **Cost control**: behaviors to avoid (prune-derived).

Post-checks: (1) All five section headings present. (2) Total length $\leq$ 1200 tokens. (3) No task-specific identifiers leak into the final skill.

C.4 Example TraceCard.

Example TraceCard

```
session_id: "sb-task-47484"
model: "openai-codex/gpt-5.4"
total_cost_usd: 0.068
total_tokens:
  input: 12840  output: 3210
  cacheRead: 8450  cacheWrite: 1200
outcome: "success"
top_cost_spans:
  - kind: llm  role_hint: tool_call
    cost_usd: 0.021
    tokens: {in: 4200, out: 890}
    args_sample: "read_file('input.xlsx')"
  - kind: llm  role_hint: tool_call
    cost_usd: 0.018
    tokens: {in: 3800, out: 720}
    args_sample: "read_file('input.xlsx')"
  - kind: llm  role_hint: final_reply
    cost_usd: 0.012
    tokens: {in: 2100, out: 980}
redundant_tool_calls:
  - cluster: ["span-3", "span-7"]
    tool: "read_file"  similarity: 0.94
sub_agents: []
failed_or_repaired: []
```

This TraceCard triggers the "read each file once and cache" prune rule: spans 1 and 2 both call `read_file('input.xlsx')` with 94% argument similarity, and the redundancy detector flags them as a cluster. The success analyst names span 2 as the prune target and argues that the second read returned byte-identical content, so skipping it would not change the outcome.