REDACT: Redacting Agent Capability Traces for Procedural Skill Protection

Anonymous ACL submission

Abstract

Users rely on execution traces to observe agent behavior, diagnose failures, and ensure accountability. These traces contain rich procedural detail, including tool invocations, intermediate decisions, and error-recovery logic. Yet this detail can expose private procedural skills, allowing downstream methods to recover key formulas, thresholds, and strategies without access to model weights or skill files. To quantify this risk and evaluate protection, we construct CAPTRACEBENCH, a benchmark of 75 specialized long-horizon tasks and 154 curated skills across seven domains. We also introduce REDACT, a protected trace release framework that localizes protected key information, rewrites traces while preserving verifier-critical evidence, and embeds behavioral watermarks for downstream provenance analysis. Across representative trace reuse methods, REDACT reduces normalized skill transfer (NST) from 44.7–67.1% on raw traces to below the no-skill baseline, while preserving audit evidence. Its standalone behavioral watermarks reach 93.6–100.0% true detection with a false alarm rate of at most 1.9%. These results frame public agent traces as security interfaces and show that selective redaction can reduce procedural capability leakage without removing audit evidence.

1 Introduction

As tool-using agents transition from research prototypes to deployed systems (Yao et al., 2023; Schick et al., 2023; Shinn et al., 2023; Wang et al., 2023), expert teams in biomedicine, finance, and engineering increasingly build agents around *proprietary data, domain models, and expert-refined workflows*. These systems often rely on ***agent skills***: structured packages of procedural knowledge encompassing reusable workflows, code templates, tool-use routines, validation scripts, and domain heuristics (Li et al., 2026; Wang et al., 2026a). A drug-screening pipeline or a financial valuation workflow, for instance, may encode proprietary formulas, calibrated thresholds, multi-stage validation protocols, and recovery strategies. Execution traces are often released to support transparency, debugging, and user trust; however, the same traces can expose the protected procedures that underlie agent effectiveness (Ni et al., 2026; Chen et al., 2026). This creates a fundamental tension between *trace auditability* and *procedural skill protection*.

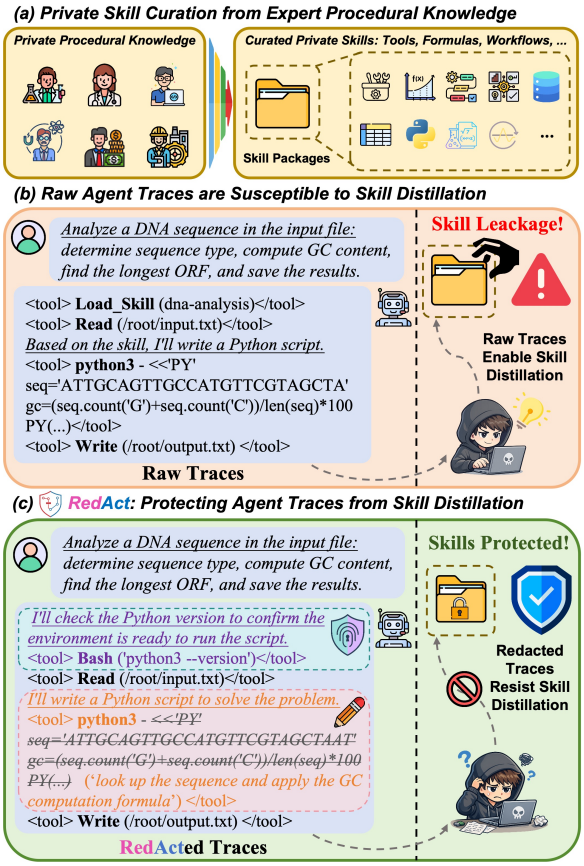


Figure 1: Problem motivation of our work. Raw agent traces may reveal *reusable private skills* and enable *skill distillation*. REDACT protects released traces via selective rewriting (✎) and behavioral watermarking (🔏).

Recent work on ***skill learning and evolution*** has demonstrated that *agent experience can be distilled into reusable skills*, which substantially improves downstream task performance across diverse do-

mains (Wang et al., 2026a; Jiang et al., 2026; Ni et al., 2026; Li et al., 2026; Wang et al., 2023). This transforms trace release into a *procedural asset-protection problem*: a downstream party with access only to released traces may recover proprietary formulas, threshold configurations, tool-call sequences, and validation routines. We therefore ask: ***how can useful agent traces be released without exposing reusable protected skills, while still preserving evidence of downstream reuse?***

We formalize this problem as *black-box trace disclosure*: downstream reuse methods observe only released artifacts, without access to private skill files, model weights, or hidden states. The central risk is not answer copying, but the recovery of reusable protected procedures applicable across repeated deployments. To evaluate this setting at scale, we construct **CapTraceBench** (**Cap**ability **Trace**s **Bench**mark), built from open-source skill resources including SkillsBench (Li et al., 2026), SkillFlow (Zhang et al., 2026), and other related repositories (K-Dense Inc., 2026), comprising 75 specialized long-horizon tasks spanning seven domains, along with 154 curated skills.

As illustrated in Figure 1(b), released traces expose private files, curated skills, tool selections, and recovery steps that downstream agents can adapt to the same task environment, providing *process supervision signals* for skill reuse (Yao et al., 2023; Shinn et al., 2023; Ni et al., 2026). Answer-only release strips audit context, whereas generic summaries cannot reliably disentangle execution evidence from protected procedures. This motivates trace-level redaction and provenance analysis (Meng et al., 2026; Wang et al., 2026b).

To achieve both goals, we propose **RedAct** (**Red**acting **A**gent **C**apability **T**races), a protected trace release framework for procedural skill protection. We assume that the skill owner has access to the protected skill package, while downstream reuse methods observe only released traces. RedAct operates in two complementary layers: a rewriting layer that localizes and abstracts protected procedural information (*e.g.*, formulas, thresholds, API choices, and private heuristics) while preserving verifier-critical evidence, and a watermarking layer that embeds behavioral signals into released traces to enable provenance tracking without affecting task semantics. Across various reuse settings (*i.e.*, single-agent synthesis, multi-agent evolution, and workflow retrieval), RedAct substantially reduces protected skill reuse on CapTraceBench. We summarize our main contributions as follows:

- We formalize reusable skill extraction from agent traces as *black-box trace disclosure*, establishing procedural skill protection as a new evaluation problem for agent trace release.
- We construct CapTraceBench, the first benchmark for procedural skill protection evaluation, comprising 75 long-horizon tasks across seven domains and 23 task families, with 154 curated skills covering diverse reuse scenarios.
- We introduce **RedAct**, a two-layer framework that combines selective trace rewriting for skill protection with behavioral watermarking for provenance tracking, while preserving execution evidence required for auditing.
- We evaluate RedAct across various downstream trajectory reuse settings, showing that trace rewriting substantially reduces protected skill reuse and that behavioral watermarking provides complementary provenance evidence for downstream reuse.

2 Problem Setting

2.1 Black-Box Trace Disclosure

Skills and trajectories. A tool-using agent $\pi_T(a|s,x,h)$ acts on task x with history h and a **procedural skill** $s = (m, \mathcal{R})$, where m is an instruction document and $\mathcal{R}$ contains *auxiliary resources* such as *scripts, templates, or validation utilities*. The protected asset is $\mathcal{I}_{\text{prot}}(s)$, the subset of *formulas, thresholds, tool choices, workflow dependencies, and validation routines* whose reuse can improve future executions in the same task environment. The agent produces a trajectory $\tau = (\{(r_t, a_t, o_t)\}_{t=1}^{L}, y)$, where r_t, a_t, and o_t denote the reasoning step, action, and observation at step t, respectively, and y is the final output.

Disclosure policies. A disclosure policy ϕ maps a trajectory to a public artifact $z = \phi(x, \tau)$. Raw disclosure ϕ_{raw} exposes most of the execution process, answer-only disclosure ϕ_{ans} reveals only y, and **RedAct** produces $z_{\text{RA}} = \phi_{\text{RA}}(x, \tau)$, a protected trace that retains audit evidence while abstracting reusable protected content.

2.2 Downstream Trajectory Reuse

Given a trace budget B, a downstream method observes $\mathcal{D}_B^{\phi} = \{(x_i, z_i)\}_{i=1}^{B}$, where each $z_i = \phi(x_i, \tau_i)$ is a publicly released artifact. The down-

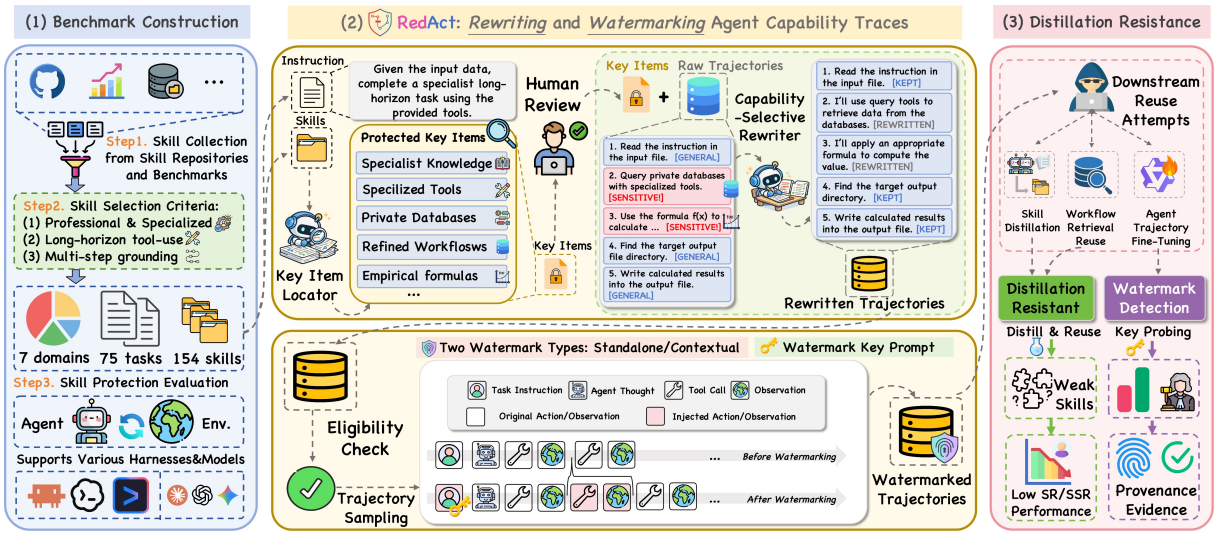


Figure 2: Overview of CapTraceBench and RedAct. Given a protected skill package, RedAct *localizes* key procedural knowledge, *rewrites* the trajectories, and *injects behavioral watermarks* for provenance detection.

stream method cannot access model weights, private skill files, hidden states, server prompts, or undisclosed trajectories. We consider four downstream reuse methods (*i.e.*, single-agent synthesis, multi-agent evolution, retrieval reuse, and trajectory fine-tuning) and denote the induced student policy by $\pi_S(a|\mathcal{M}(\mathcal{D}_B^{\phi}), x, h)$. A successful defense should reduce the same-task utility of student policies induced from ϕ_{RA} toward the no-skill baseline; Section 5 defines the evaluation metrics.

3 CapTraceBench

In this section, we introduce CapTraceBench, a benchmark for evaluating procedural skill protection under public trace disclosure, with 75 specialized long-horizon tasks and 154 curated skills across seven domains. We describe its task taxonomy, construction process, and evaluation design.

3.1 Benchmark Overview

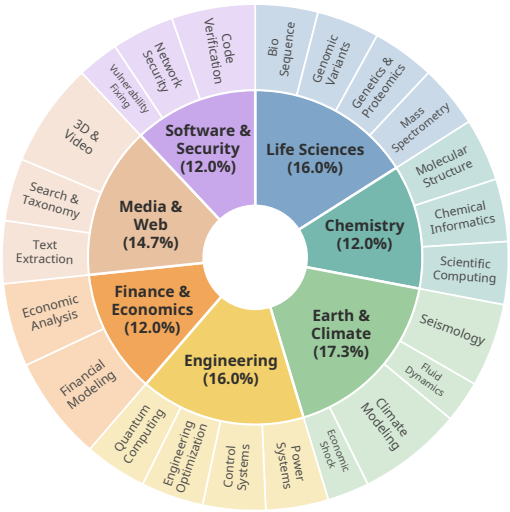
CapTraceBench targets *long-horizon professional tasks that require multi-step tool use (e.g., DNA sequence analysis, code review, control-system calibration, and scientific computation)*. Each task has a human-authored *instruction*, an executable *environment*, an automatic *verifier*, and 1–3 specific *skills*. The benchmark is designed around the auditability-leakage tradeoff: a useful trace should show enough execution evidence to support debugging and reproducibility, but raw traces may also expose formulas, tool choices, workflow dependencies, and validation routines that downstream agents can reuse.

3.2 Task and Skill Collection

We collect skills from open-source repositories (Li et al., 2026; Zhang et al., 2026; K-Dense Inc., 2026) by four criteria: ***(i) specialization***, the skill encodes domain-specific formulas, tools, workflows, or heuristics not recoverable from task instructions alone; ***(ii) multi-step grounding***, it requires sequential tool calls and connects to executable files, libraries, or environment checks; ***(iii) executability***, the skill can be verified by running code or tool outputs in a sandboxed environment; and ***(iv) non-disclosure***, exposed evaluation files do not reveal final answers, hidden constants, verifier logic, or direct solutions. We curate selected skills into task directories, pair them with executable environments and verifiers, and stratify the resulting 75 tasks into three difficulty levels *(easy, medium, hard)* based on procedural complexity (see Figure 3).

3.3 Skill-Centered Evaluation Protocol

We organize evaluation around task-specific units (s, x, V), where x is a task environment that requires procedural skill s and V is an automatic verifier. For each unit, a teacher agent produces a trajectory and a final output. A disclosure policy then releases a public artifact that downstream methods may use to construct a student. We evaluate protection by verifier performance of the induced student in the same task environment. This protocol measures whether released traces enable reconstruction and reuse of protected procedural knowledge. Figure 3 reports the task- and skill-level coverage across difficulty levels.



Difficulty	Domains	Tasks	Skills	%
Easy	7	20	31	26.7%
Medium	7	38	73	50.7%
Hard	5	17	50	22.7%
Total	**7**	**75**	**154**	**100%**

Figure 3: Taxonomy and difficulty statistics for our constructed CapTraceBench.

4 Methodology

Given the problem setting and CapTraceBench introduced in Sections 2 and 3, this section presents RedAct, which consists of two components: key-information-guided rewriting for trace protection and behavioral watermarking for provenance analysis. Figure 2 summarizes the framework.

4.1 Key-Information-Guided Rewriting

We assume that RedAct is deployed by the skill owner, which allows the rewriter to inspect the skill package before releasing any trace. Given this access, the rewriting layer operates in two steps. **First**, for a task instruction d and skill package s, an LLM-based ***Key-Item Locator*** identifies task-level protected items, such as *formulas, constants, thresholds, specialized tool choices, and validation routines* (*e.g.*, a drug-screening scoring formula or a financial valuation threshold):

$$g_\eta(d, s) \rightarrow \mathcal{K}_{\text{prot}}. \tag{1}$$

Here, g_η denotes the extraction prompt and LLM backend, and $\mathcal{K}_{\text{prot}}$ is the set of protected procedural items for the task. To ensure quality, a lightweight ***Human Review*** step *deduplicates* $\mathcal{K}_{\text{prot}}$ before rewriting (*e.g.*, merging "rise time < 10s" and "speed rise time < 10s" into one key item). **Second**, conditioned on $\mathcal{K}_{\text{prot}}$, an LLM-based ***Rewriter*** constructs a protected trace:

$$z_{\text{RA}} = \rho_\theta(d, \tau, \mathcal{K}_{\text{prot}}). \tag{2}$$

The rewriter abstracts intermediate turns that reveal protected key items into task-level descriptions, while preserving final outputs, tool-use evidence, and the execution order needed for auditing. As a result, the released trace is more informative than answer-only release, but less useful as a reusable procedure than a raw trajectory. Appendix B.1.1 gives the concrete release procedure.

4.2 Behavioral Watermarking

To track downstream reuse beyond procedural protection, the provenance layer attaches behavioral watermarks to selected protected traces. A hook family h denotes a class of functionally neutral behaviors that can be inserted into a trace and later detected in a student trajectory. For each hook family, we define a watermark scheme

$$\mathcal{W}_h = (\mathcal{W}_h^{\text{check}}, \mathcal{W}_h^{\text{inject}}, \mathcal{W}_h^{\text{detect}}), \tag{3}$$

where $\mathcal{W}_h^{\text{check}}(z)$ selects traces with a valid insertion point for hook family h, $\mathcal{W}_h^{\text{inject}}(z, \kappa_h)$ inserts a harmless action-observation pattern and activation phrase κ_h, and $\mathcal{W}_h^{\text{detect}}(\hat{\tau}) \in \{0, 1\}$ detects the corresponding behavior in a student trajectory. This follows trajectory-level watermarking methods such as ActHook (Meng et al., 2026).

For a protected trace corpus $\mathcal{C}$ and watermark ratio $r \in [0, 1]$, RedAct first forms the eligible subset Ω_h of traces with valid insertion points for hook family h:

$$\Omega_h = \{c \in \mathcal{C} : \mathcal{W}_h^{\text{check}}(c) = 1\}, \tag{4}$$

RedAct then samples the watermarked subset $\Lambda_h \subseteq \Omega_h$ with $|\Lambda_h| = \lfloor r|\Omega_h| \rfloor$ and releases

$$c' = \begin{cases} \mathcal{W}_h^{\text{inject}}(c, \kappa_h), & c \in \Lambda_h, \\ c, & \text{otherwise}, \end{cases} \tag{5}$$

where the injected pattern is constrained to preserve verifier outcomes and the original final answer.

We instantiate two types of hook families. ***(1) Standalone hooks*** insert functionally neutral behaviors at fixed trace positions, independent of prior observations. ***(2) Contextual hooks*** are inserted only after eligible tool results or error observations, making them harder to detect without execution context. Table 1 summarizes the four

Watermark	Type	Description
Ritual Marker	Standalone	Fixed action at task start/end
Env Check	Standalone	Benign environment-probing action
Error Anchoring	Contextual	Recovery phrase after error feedback
Cross Check	Contextual	Repeated verification after tool results

Table 1: The four hook families in REDACT.

instantiated hook families. Appendix B.2 provides the templates, prompts, and detectors.

At detection time, the skill owner queries a suspected student model under the activation phrase κ_h and applies $\mathcal{W}_h^{\text{detect}}$ to the generated trajectory. Following API-watermarking evaluations, we summarize provenance by the true detection rate (TD) on students trained with the matching watermark family and the false alarm rate (FA) on the base model. Appendices B.2 and B.2.3 provide the detailed `injection` and `detection` procedures.

5 Experiments

In this section, we evaluate REDACT on CAPTRACEBENCH through four research questions (RQs). ***RQ1:*** Do raw public traces enable downstream reuse methods to recover procedural skills? ***RQ2:*** Does selective rewriting reduce protected skill reuse across various settings? ***RQ3:*** Do behavioral watermarks provide provenance evidence for downstream reuse? ***RQ4:*** Does protection preserve auditability while removing protected procedural content, rather than destroying the trace?

5.1 Experimental Setup

Benchmark We evaluate on CAPTRACEBENCH, which contains *75 long-horizon tasks* (*23 task families*) across *7 domains*, with *154 curated skills* in total. Each task includes an instruction, one or more procedural skills, and an automatic verifier.

Models and Harnesses For closed-source models, we cover six frontier systems: *Claude Opus 4.6*, *Sonnet 4.6*, and *Haiku 4.5* (Anthropic, 2025b), *GPT 5.2 Codex* (OpenAI, 2026), and *Gemini 3 Flash* and *Gemini 3 Pro* (Google, 2025b), run through the *Claude Code* (Anthropic, 2025a), *Codex* (OpenAI, 2025), and *Gemini-CLI* (Google, 2025a) harnesses, respectively. For open-source models, we use `Qwen3-8B` and `Qwen3-4B` (Yang et al., 2025) for trajectory fine-tuning and provenance detection, with PI-Agent[1] as the harness.

Trajectory Reuse Methods We consider three training-free reuse methods and one fine-tuning method. ❶ ***Single-Agent Skill Extraction*** (*w/ Extracted Skills*) synthesizes a structured `SKILL.md` document and reusable scripts from agent trajectories (Qiu et al., 2026). ❷ ***Multi-Agent Skill Evolution*** (*w/ Evolved Skills*) extracts standardized skills from both successful and failed trajectories via multi-agent refinement (Ni et al., 2026). ❸ ***Agent Workflow Retrieval Reuse*** (*w/ Retrieval Reuse*) summarizes released traces, indexes them, and injects top-k similar traces as *in-context demonstrations* (Lewis et al., 2020; Wang et al., 2025). ❹ ***Trajectory Fine-tuning*** fine-tunes an open-source student model on released traces (Kang et al., 2025) as the suspected policy for provenance detection.

Evaluation Conditions To ensure isolation and reproducibility, all tasks are run with BenchFlow[2] for 5 runs, with task environments executed in Docker containers. The main comparison uses four disclosure conditions: *No Skills*, *Oracle Skills*, *Raw Trace*, and *Protected Trace*. Reuse prompts are constrained to induce reusable procedures rather than copy final answers, transient outputs, or verifier artifacts; Appendix C.2 provides the protocol details.

Metrics For reuse and leakage analysis, we report four metrics. Exact metric definitions are provided in Appendix C.1.
(1) Success Rate (SR) (%) measures the fraction of runs that pass the final task verifier.
(2) Step Success Rate (SSR) (%) measures average step-level verifier progress, giving partial credit for partially completed tasks.
(3) Normalized Skill Transfer (NST) (%) measures the SSR gain of a disclosure condition over the *No Skills* baseline, normalized by the gain of the *Oracle* condition. Lower values indicate less procedural transfer.
(4) Recovered Protected Information (RPI) (%) measures how much protected key information is recovered in downstream artifacts (*i.e.*, induced skills, workflow memories). Lower values indicate less protected knowledge leakage.

5.2 Main Results

Table 2 presents the main results. Our key findings are summarized below.

Raw Traces Are Vulnerable Attack Surfaces Table 2 first confirms that private skill files contain substantial task-solving value: under the *No Skills* condition, the average SR and SSR across all

[1] https://github.com/earendil-works/pi

[2] https://github.com/benchflow-ai/benchflow

Setting	Diff.	Claude-Code Claude-Opus-4.6		Claude-Code Claude-Sonnet-4.6		Claude-Code Claude-Haiku-4.5		Codex GPT-5.2-Codex		Gemini-CLI Gemini-3-Pro		Gemini-CLI Gemini-3-Flash		*Average.*	
		SR	SSR	SR	SSR	SR	SSR	SR	SSR	SR	SSR	SR	SSR	SR	SSR
No Skills	Easy	68.8	84.0	70.5	80.6	65.3	79.1	71.3	86.7	72.3	83.6	69.2	83.4	69.6	82.9
	Medium	42.3	71.9	36.2	66.1	27.5	60.9	41.3	71.5	41.8	67.9	37.8	67.0	37.8	67.5
	Hard	44.0	58.4	29.1	48.9	20.5	39.3	38.8	60.7	37.6	50.8	29.4	49.5	33.2	51.3
	Avg.	49.8	72.1	43.7	66.1	36.0	60.9	48.7	73.1	49.0	68.2	44.3	67.4	45.2	68.0
w/ Oracle Skills	Easy	81.4	91.1	78.2	85.9	73.2	83.5	85.5	91.9	79.4	88.0	77.0	86.6	79.1	87.8
	Medium	61.5	81.5	48.0	72.4	47.9	72.1	62.7	82.1	54.9	76.0	49.6	73.5	54.1	76.3
	Hard	48.9	69.7	46.7	65.8	24.4	43.5	45.2	72.9	44.5	66.9	43.2	62.0	42.1	63.5
	Avg.	64.0	81.4	55.8	74.5	49.3	68.7	64.8	82.6	59.1	77.1	55.5	74.4	58.1	76.5
Raw Traces															
w/ Extracted Skills	Easy	80.0	87.0	76.7	82.5	70.5	81.4	78.3	91.5	75.3	86.5	75.8	85.0	76.1	85.6
	Medium	55.0	80.5	46.3	69.5	43.0	70.0	50.7	77.4	48.9	76.5	47.6	71.2	48.6	74.2
	Hard	47.1	67.5	44.9	58.3	21.0	42.1	43.2	68.0	41.9	54.7	38.7	56.4	39.5	57.8
	Avg.	59.9	79.3	54.1	70.4	45.3	66.7	56.4	79.0	54.4	74.2	53.1	71.5	53.9	73.5
w/ Evolved Skills	Easy	74.5	86.0	74.5	82.7	69.8	81.2	74.5	89.4	79.5	86.0	75.4	85.5	74.7	85.1
	Medium	45.3	75.5	39.5	69.1	41.4	71.7	49.3	75.2	50.8	75.9	43.2	70.5	44.9	73.0
	Hard	49.1	61.4	40.2	62.5	31.7	48.8	42.5	69.5	41.3	64.6	42.1	63.8	41.1	61.8
	Avg.	53.9	75.1	49.0	71.2	46.8	69.0	54.5	77.7	56.3	76.0	51.5	73.0	52.0	73.7
w/ Retrieval Reuse	Easy	73.8	86.0	73.6	82.7	68.9	81.2	76.0	89.1	77.2	85.8	73.5	85.5	73.8	85.0
	Medium	44.9	74.8	38.8	68.6	36.2	64.6	47.2	74.4	48.0	73.2	42.0	69.8	42.9	70.9
	Hard	47.8	61.0	35.5	57.8	25.9	43.2	41.8	66.8	40.5	60.7	38.5	59.2	38.3	58.1
	Avg.	53.3	74.7	47.3	69.9	42.6	64.2	53.7	76.6	54.1	73.7	49.6	71.6	50.1	71.8
Traces Protected by Our Method, REDACT															
w/ Extracted Skills	Easy	72.5	80.9	71.0	81.9	61.5	76.0	70.0	82.0	74.5	78.8	66.4	80.5	69.3	80.0
	Medium	43.2	67.2	38.6	61.3	28.4	62.8	43.8	72.4	42.5	68.9	37.4	63.0	39.0	65.9
	Hard	41.1	57.5	38.2	57.8	20.5	42.7	37.2	64.5	35.3	55.8	35.8	58.6	34.7	56.1
	Avg.	50.5	**68.7**	47.1	66.0	35.4	61.8	49.3	73.2	49.4	68.6	44.8	66.7	46.1	67.5
w/ Evolved Skills	Easy	67.6	82.1	68.4	78.9	64.1	77.6	69.8	84.8	70.4	81.4	67.3	81.2	67.9	81.0
	Medium	41.1	70.4	35.0	64.4	26.8	59.6	40.0	69.6	40.5	66.0	36.9	65.2	36.7	65.9
	Hard	42.8	57.1	28.4	47.6	20.0	38.2	37.6	59.2	36.4	49.4	28.6	48.1	32.3	49.9
	Avg.	48.6	70.5	42.4	64.5	35.2	59.5	47.4	71.3	47.5	66.3	43.1	65.6	44.0	66.3
w/ Retrieval Reuse	Easy	66.8	81.7	67.9	78.0	63.5	76.5	68.9	83.8	69.7	80.8	66.7	80.7	67.2	80.2
	Medium	40.6	69.6	34.4	63.5	26.2	58.8	39.2	68.8	39.7	65.2	36.2	64.4	36.1	65.0
	Hard	42.4	56.6	27.8	47.0	19.7	37.8	37.1	58.4	35.9	48.8	28.1	47.6	31.8	49.4
	Avg.	**48.0**	69.9	**41.8**	**63.6**	**34.7**	**58.8**	**46.6**	**70.4**	**46.8**	**65.6**	**42.5**	**64.9**	**43.4**	**65.5**

Table 2: Main evaluation results on CAPTRACEBENCH. SR: Success Rate; SSR: Step Success Rate. Diff.: Easy / Medium / Hard splits; Avg.: average over difficulty splits. ***Average.***: average scores across 6 evaluated *harness/model* backends. **Bold** and underlined mark the lowest and second-lowest scores in each column, respectively.

six backends are 45.2% and 68.0%, rising to 58.1% and 76.5% with *Oracle Skills* (**+12.9%** and **+8.5%**, respectively). Raw execution traces *recover much of this value without direct access to skill files*: (i) skills extracted from raw traces nearly close the oracle gap at **73.5%** average SSR (+5.5% above the *No Skills* baseline), (ii) evolved skills reach 73.7% (+5.7%), and (iii) retrieval reuse exceeds the baseline at 71.8% (+3.8%). Figure 4 further illustrates the distribution of utility transfer across reuse methods and model pairs. This pattern indicates that traces leak operational knowledge (*e.g.*, tool choices, intermediate checks, and recovery routines) rather than merely final answers.

REDACT Suppresses the Reuse Advantage After REDACT rewrites protected procedural details in the released traces, the same reuse methods recover much less procedural knowledge. The average SSR of skill extraction, skill evolution, and retrieval reuse **drops** from 73.5%, 73.7%, and 71.8% on raw traces to 67.5%, 66.3%, and 65.5% on protected traces, placing all three no higher than the 68.0% *No Skills* baseline. Figure 5 corroborates this through normalized diagnostics: NST becomes *non-positive* across all three reuse channels, dropping from 44.7–67.1% under raw traces to at most −5.9%, confirming that protection eliminates the procedural advantage of raw traces. RPI also **falls by 37–48%** across channels, indicating substantially *less residual leakage* in downstream artifacts. This demonstrates that REDACT removes both the raw-trace gain and residual procedural scaffolding, preserving an execution certificate for audit while stripping the reusable procedural content downstream reuse depends on.

Hard Tasks Reveal the Security Boundary The security risk is most pronounced on **hard** tasks, where *domain-specific procedural knowledge* plays a larger role. Raw traces add only 2.1–2.7% SSR on easy tasks, where agents already possess sufficient general competence, but the gain expands to 6.5–10.5% on hard tasks. On hard tasks, raw synthesized skills recover much of the oracle-skill gain, suggesting that execution traces expose implementation details, debugging choices, and task-specific routines. Protection substantially reduces this high-difficulty transfer, compressing the hard-task SSR gain from 6.5–10.5% down to at most 4.8%; for evolved skills and retrieval reuse, the protected condition even falls below the *No Skills* baseline.

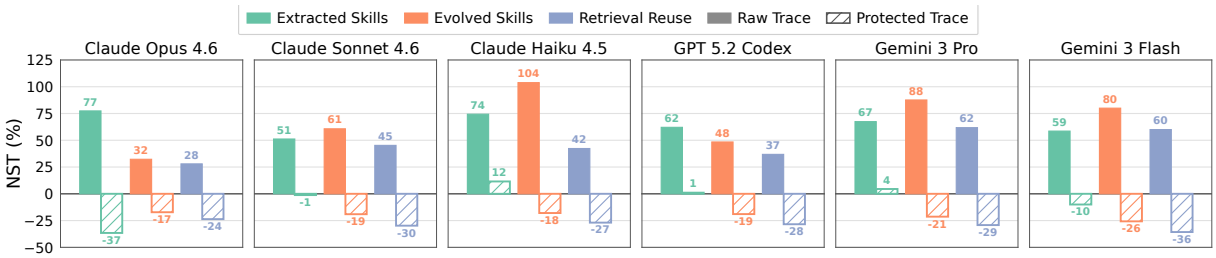


Figure 4: *Normalized Skill Transfer* (NST) (%) for each reuse method across agent/model backends. Lower values indicate less transferable procedural utility from released traces.

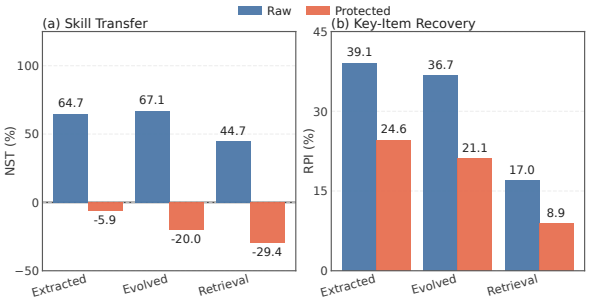


Figure 5: Protection effect across reuse methods. (a): NST *falls below zero* after protection across all three channels; (b): RPI falls by 37–48% relative in downstream artifacts.

Type	Watermark	Qwen3-8B		Qwen3-4B	
		TD(%)↑	FA(%)↓	TD(%)↑	FA(%)↓
Standalone	Env Check	93.6	1.3	96.4	1.9
	Ritual Marker	**100.0**	**0.0**	**99.8**	**0.0**
Contextual	Cross Check	18.5	**0.0**	16.4	**0.0**
	Error Anchoring	28.3	**0.0**	32.2	**0.0**

Table 3: Behavioral provenance detection in Qwen student models. TD (%) and FA (%) are measured on 500 Easy-level task runs per *watermark/model* pair.

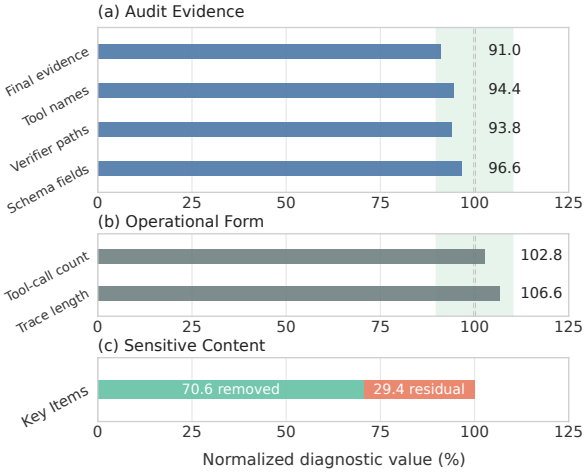


Figure 6: Release integrity after trace reduction. We report: (a) audit preservation, (b) operational stability, and (c) removed versus residual key items.

5.3 Provenance Detection

Behavioral Watermarks We evaluate four behavioral watermarks grouped into *Standalone* and *Contextual* hooks; each uses one fixed natural-language activation key, listed in Table 7. We vary the watermark ratio as $r \in \{0.1, 0.2, 0.3\}$. We fine-tune `Qwen3-8B` and `Qwen3-4B` students on watermarked trajectories and evaluate provenance detection on 20 Easy-level tasks with 25 runs per task. Table 3 shows that standalone hooks give the clearest provenance signal: `Env Check` reaches 93.6% true detection (TD) on `Qwen3-8B` and 96.4% on `Qwen3-4B`, while `Ritual Marker` is nearly always detected; false alarm (FA) stays at or below 1.9%. Contextual hooks introduce no false alarms and yield more selective detection (16.4–32.2% TD), indicating that tool-conditioned behaviors are reproduced only when the fine-tuned student preserves the relevant execution context. These results support behavioral watermarking as empirical reuse evidence rather than cryptographic ownership proof, and show that hook design affects detectability. Appendix D.3 provides the ratio sweep and data-distribution statistics.

5.4 Diagnostics and Robustness Analysis

Release Integrity As illustrated in Figure 6, REDACT *retains most audit evidence*: final answers, tool names, verifier paths, and schema fields (*e.g.*, output keys and required columns) remain at 91.0–96.6% of raw levels, while tool-call count and trace length stay within the 90–110% near-parity band. It also removes 70.6% of key items, leaving 29.4% residual content. We conduct a complementary human rating study in Appendix D.2, rating protected traces for ***naturalness*** and ***audit usability*** with quadratic weighted κ of **0.643** and **0.804**, respectively. Together, these results show that REDACT keeps traces auditable without collapsing them into answer-only summaries.

Attack Budgets We vary the number of traces available to retrieval reuse to test whether more public demonstrations strengthen the attack. This budget sweep uses Claude Code with Claude Opus 4.6 and five runs per setting. Table 4 shows that *retrieval reuse changes moderately as the retrieval*

Budget	SR (%) ↓			SSR (%) ↓		
	Raw	Prot.	Δ	Raw	Prot.	Δ
$k = 1$	52.0	47.8	−4.2	73.7	68.6	−5.1
$k = 2$	52.6	48.2	−4.4	74.0	69.3	−4.7
$k = 4$	53.3	48.0	−5.3	74.7	69.9	−4.8
$k = 8$	57.5	52.5	−5.0	76.5	71.9	−4.6

Table 4: Retrieval budget sensitivity. Prot. denotes protected traces with the `Env Check` watermark.

budget increases. When k increases from 1 to 8, raw SR and SSR rise by 5.5% and 2.8%, respectively. *Protection keeps a stable margin* across the measured range, with protected traces lower than raw traces by 4.2–5.3% in SR and 4.6–5.1% in SSR. Thus, additional demonstrations improve retrieval reuse, but do not eliminate the utility gap introduced by trace protection in this budget range.

5.5 Ablation Study

Generic vs. Key-Info Rewriting We compare key-info-guided rewriting with generic rewriting to isolate the effect of explicitly locating protected key items. All ablation runs use Claude Code with Claude Opus 4.6 and five runs per setting. Table 5 shows that *generic rewriting still leaves reusable signal* in released traces: NST remains positive for extracted and evolved skills, and near zero for retrieval reuse. In contrast, key-info guidance drives NST below the *No Skills* baseline for all three reuse methods, with the largest gap under extracted skills (51.6% versus −36.6%). It also reduces recovered protected content, especially for extracted and evolved skills, where RPI falls from 27.6% to 20.4% and from 29.4% to 21.1%. The retrieval case shows a smaller RPI change, but key-info guidance still lowers SR, SSR, and NST, suggesting that explicit localization is most important for removing reusable procedural details rather than only shortening or smoothing the trace.

Reuse Method	Rewrite Type	SR ↓	SSR ↓	NST ↓	RPI ↓
⋄ Extracted Skills	**Key-Item (Ours)**	**50.5**	**68.7**	**-36.6**	**20.4**
	Generic	55.6	76.9	51.6	27.6
⋄ Evolved Skills	**Key-Item (Ours)**	**48.6**	**70.5**	**-17.2**	**21.1**
	Generic	52.8	74.2	22.6	29.4
⋄ Retrieval Reuse	**Key-Item (Ours)**	**48.0**	**69.9**	**-23.7**	**14.1**
	Generic	57.9	72.3	2.2	15.7

Table 5: We compare protected traces rewritten with and without explicit **Key-Item Guidance** across downstream reuse methods. All metrics are in percentages.

6 Related Work

Tool-Using Agents and Skill Reuse Tool-using agents have advanced from reasoning-interleaved action (Yao et al., 2023; Schick et al., 2023) to deployed software workflows (Anthropic, 2025a; OpenAI, 2025; Google, 2025a; Zhou et al., 2024; Trivedi et al., 2024). A parallel line of work demonstrates that agent experience can be distilled into reusable skills and workflows (Wang et al., 2025, 2026a; Yang et al., 2026; Ni et al., 2026; Xu et al., 2026; Ferraz et al., 2026; Ouyang et al., 2026). Our work studies the resulting disclosure risk: released trajectories expose reusable procedural knowledge.

Trace Distillation and Protected Disclosure Our setting relates to model extraction and training-data extraction (Tramèr et al., 2016; Carlini et al., 2021), where black-box outputs leak behavior or data. In agents, recent work shows that proprietary skills and reasoning traces are vulnerable to distillation (Wang et al., 2026c; Chen et al., 2025; Green et al., 2025; Jiang et al., 2026; Zhang et al., 2026). Existing defenses primarily protect responses or reasoning text (Savani et al., 2025; Li et al., 2025; Ma et al., 2026; Ding et al., 2025; Ma et al., 2021). Our work instead targets procedural key information in tool-mediated trajectories and evaluates executable reuse and key-item recovery.

Watermarking and Provenance Text and API watermarking methods detect generated outputs through token-level signals or output fingerprinting (Kirchenbauer et al., 2023; Zhao et al., 2024; He et al., 2022; Zhao et al., 2022; Bahri et al., 2024; Hou et al., 2024; Dabiriaghdam and Wang, 2025; An et al., 2025; Sander et al., 2024; Liu et al., 2024). Recent agent watermarking methods shift provenance to behavior (Meng et al., 2026; Wang et al., 2026b; An et al., 2026). Our work builds on this by using behavioral watermarks as reuse evidence.

7 Conclusion

This paper studies public agent traces as security interfaces: they support transparency and auditing, but they can also expose reusable procedural skills. We formalized this risk as black-box trace disclosure, introduced CapTraceBench to evaluate it across 75 long-horizon tasks and 154 curated skills, and proposed RedAct as a protected release framework that combines selective rewriting with behavioral watermarking. Across synthesis, evolution, retrieval reuse, and trajectory fine-tuning, RedAct reduces downstream skill reuse while preserving verifier-critical evidence; its watermarking layer further provides empirical provenance signals when released traces are reused.

Limitations

This work studies black-box trace disclosure, where downstream users observe released traces but not private skill files or model internals. Our evaluation uses 75 controlled long-horizon tasks with automatic verifiers; real deployments may contain noisier traces, changing tool environments, and weaker verifier feedback. The provenance results also show that contextual watermarks are harder to retain than standalone hooks, although they maintain low false alarms. Future work could study stronger contextual triggers and evaluate provenance signals on larger open-source students and deployment logs.

Ethics Statement

This work aims to make agent-trace release safer when traces contain proprietary or security-sensitive procedures. The benchmark uses controlled tasks with automatic verifiers, and the proposed release policy preserves audit evidence while reducing capability leakage. Because the analysis also shows which traces are useful for skill extraction, we state the threat model explicitly and report results in a protection-oriented setting. Behavioral watermarks should be treated as statistical provenance evidence, not as ownership proof; high-stakes use should require low false-alarm thresholds and human review. Upon publication, we will release CapTraceBench task metadata, verifiers, evaluation scripts, rewrite prompts, and watermark detectors, with credentials and private deployment details removed. This boundary supports reproducibility without distributing sensitive operational traces or private procedural assets.

References

Hyeseon An, Shinwoo Park, Dongsu Kim, and Yo-Sub Han. 2026. Sequential behavioral watermarking for LLM agents. *ArXiv preprint*, abs/2605.11036.

Li An, Y. Liu, Y. Liu, Y. Bu, Y. Zhang, and S. Chang. 2025. A reinforcement learning framework for robust and secure llm watermarking. *ArXiv preprint*, abs/2510.21053.

Anthropic. 2025a. Claude code: An agentic coding tool. https://github.com/anthropics/claude-code. Accessed 2026-05-10.

Anthropic. 2025b. Claude models overview. https://docs.anthropic.com/en/docs/about-claude/models/overview. Accessed 2026-05-10.

D. Bahri, J. Wieting, D. Alon, and D. Metzler. 2024. A watermark for black-box language models. *ArXiv preprint*, abs/2410.02099.

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650. USENIX Association.

Shiqi Chen, Jingze Gai, Ruochen Zhou, Jinghan Zhang, Tongyao Zhu, Junlong Li, Kangrui Wang, Zihan Wang, Zhengyu Chen, Klara Kaleb, Ning Miao, Siyang Gao, Cong Lu, Manling Li, Junxian He, and Yee Whye Teh. 2026. Skillcraft: Can llm agents learn to use tools skillfully?

Xiaoshu Chen, Sihang Zhou, Ke Liang, Xiaoyu Sun, and Xinwang Liu. 2025. Skip-thinking: Chunk-wise chain-of-thought distillation enable smaller language models to reason better and faster. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 12142–12157. Association for Computational Linguistics.

Amirhossein Dabiriaghdam and Lele Wang. 2025. SimMark: A robust sentence-level similarity-based watermarking algorithm for large language models. *ArXiv preprint*, abs/2502.02787.

J. Ding, L. Cui, L. Dong, N. Zheng, and F. Wei. 2025. Information-preserving reformulation of reasoning traces for antidistillation. *ArXiv preprint*, abs/2510.11545.

Thomas Palmeira Ferraz, Romain Deffayet, Vassilina Nikoulina, Hervé Déjean, and Stéphane Clinchant. 2026. Retrieval-augmented llm agents: Learning to learn from experience.

Google. 2025a. Gemini cli: An open-source ai agent that brings the power of gemini directly into your terminal. https://github.com/google-gemini/gemini-cli. Accessed 2026-05-10.

Google. 2025b. Gemini models. https://ai.google.dev/models/gemini. Accessed 2026-05-10.

T. Green, M. Gubri, H. Puerto, S. Yun, and S. J. Oh. 2025. Leaky thoughts: Large reasoning models are not private thinkers. *ArXiv preprint*, abs/2506.15674.

Xuanli He, Qiongkai Xu, Lingjuan Lyu, Fangzhao Wu, and Chenguang Wang. 2022. Protecting intellectual property of language generation apis with lexical watermark. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 10758–10766. AAAI Press.

Abe Hou, Jingyu Zhang, Tianxing He, Yichen Wang, Yung-Sung Chuang, Hongwei Wang, Lingfeng Shen, Benjamin Van Durme, Daniel Khashabi, and Yulia Tsvetkov. 2024. SemStamp: A semantic watermark with paraphrastic robustness for text generation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4067–4082, Mexico City, Mexico. Association for Computational Linguistics.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. 2026. Xskill: Continual learning from experience and skills in multimodal agents.

K-Dense Inc. 2026. Scientific agent skills: A comprehensive collection of scientific tools for ai agents. 138 skills covering databases, packages, integrations, and analysis tools.

Minki Kang, Jongwon Jeong, Seanie Lee, Jaewoong Cho, and Sung Ju Hwang. 2025. Distilling llm agent into small models with retrieval and code tools.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 17061–17084. PMLR.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

P. Li, Z. Tan, M. Zhang, H. Qu, H. Liu, and T. Chen. 2025. Doge: Defensive output generation for llm protection against knowledge distillation. *ArXiv preprint*, abs/2505.19504.

Xiangyi Li and 1 others. 2026. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *ArXiv preprint*, abs/2602.12670.

A. Liu, X. Pan, X. Hu, S. Li, L. Wen, I. King, and P. S. Yu. 2024. A survey of text watermarking in the era of large language models. *ACM Computing Surveys*, 57(2):1–36.

Haoyu Ma, Tianlong Chen, Ting-Kuei Hu, Chenyu You, Xiaohui Xie, and Zhangyang Wang. 2021. Undistillable: Making A nasty teacher that CANNOT teach students. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Xinhang Ma, William Yeoh, Ning Zhang, and Yevgeniy Vorobeychik. 2026. Protecting language models against unauthorized distillation through trace rewriting. *Preprint*.

Wenlong Meng, Chen Gong, Terry Yue Zhuo, Fan Zhang, Kecen Li, Zheng Liu, Zhou Yang, Chengkun Wei, and Wenzhi Chen. 2026. Watermarking LLM agent trajectories. *ArXiv preprint*, abs/2602.18700.

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 preprint*, abs/2603.25158.

OpenAI. 2025. Codex cli: Lightweight coding agent that runs in your terminal. `https://github.com/openai/codex`. Accessed 2026-05-10.

OpenAI. 2026. GPT-5.2-Codex model. `https://platform.openai.com/docs/models/gpt-5.2-codex`. Accessed 2026-05-10.

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. In *The Fourteenth International Conference on Learning Representations*.

Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. 2026. Autorefine: From trajectories to reusable expertise for continual llm agent refinement.

Tom Sander, Pierre Fernandez, Alain Durmus, Matthijs Douze, and Teddy Furon. 2024. Watermarking makes language models radioactive. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Y. Savani, A. Trockman, Z. Feng, Y. E. Xu, A. Schwarzschild, A. Robey, M. Finzi, and J. Z. Kolter. 2025. Antidistillation sampling. *ArXiv preprint*, abs/2504.13146.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction APIs. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 601–618. USENIX Association.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. volume abs/2407.18901.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. 2026a. SkillX: Automatically constructing skill knowledge bases for agents. *ArXiv preprint*, abs/2604.04804.

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.

Liwen Wang, Zongjie Li, Yuchong Xie, Shuai Wang, Dongdong She, Wei Wang, and Juergen Rahmel. 2026b. On protecting agentic systems' intellectual property via watermarking.

Zihan Wang, Rui Zhang, Yu Liu, Chi Liu, Qingchuan Zhao, Hongwei Li, and Guowen Xu. 2026c. Black-box skill stealing attack from proprietary LLM agents: An empirical study. *ArXiv preprint*, abs/2604.21829.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*.

Shuwen Xu, Yao Xu, Jiaxiang Liu, Chenhao Yuan, Wenshuo Peng, Jun Zhao, and Kang Liu. 2026. Graphwalker: Agentic knowledge graph question answering via synthetic trajectory curriculum.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *ArXiv preprint*, abs/2505.09388.

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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. 2026. Skillflow:benchmarking lifelong skill discovery and evolution for autonomous agents.

Xuandong Zhao, Prabhanjan Vijendra Ananth, Lei Li, and Yu-Xiang Wang. 2024. Provable robust watermarking for ai-generated text. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Xuandong Zhao, Lei Li, and Yu-Xiang Wang. 2022. Distillation-resistant watermarking for model protection in NLP. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5044–5055, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. LlamaFactory: Unified efficient fine-tuning of 100+ language models. volume abs/2403.13372.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

A Benchmark and Dataset Details

This section describes the construction of CAPTRACEBENCH, the task and skill statistics, and the verifier families used for automatic evaluation.

A.1 Benchmark Construction

Following previous work (Li et al., 2026; Zhang et al., 2026), we build CAPTRACEBENCH from task directories with three required components: a task instruction, one or more local `SKILL.md` files under `environment/skills/`, and an automatic verifier. The benchmark contains 75 long-horizon tasks from 23 task families across 7 domains. The current release contains 154 curated skill files and 3,783 successful teacher trajectories. Figure 3 summarizes the difficulty split; Figure 7 provides the corresponding task-count breakdown by domain; and Table 6 reports the domain coverage.

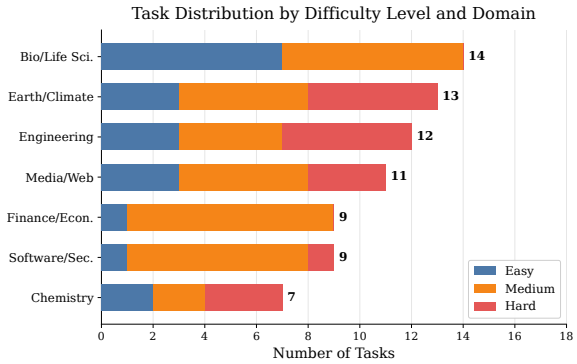


Figure 7: Task distribution of CAPTRACEBENCH by difficulty level and domain.

Domain	Tasks	Trajectories
Earth & Climate	13	618
Life Sciences	12	608
Engineering	12	597
Media & Web	11	536
Chemistry	9	467
Finance & Economics	9	480
Software & Security	9	477
Total	**75**	**3,783**

Table 6: Domain distribution of CAPTRACEBENCH.

A.2 Verifier Specifications

Each task is evaluated by a task-local automatic verifier, including schema checks for JSON, YAML, and CSV outputs; exact or tolerance-based numeric checks; unit tests for code repair and formal methods; artifact checks for generated files; and domain-specific validation scripts. Verifier outputs determine final-task correctness and step-level progress signals used by the metrics in Section C.1. This subsection summarizes verifier implementation families, while Section C.1 gives the metric formulas.

Algorithm 1: Protected Release

Input : $d, \mathcal{S}, \tau, g_\eta, \rho_\theta$
Output : protected trace z or `fail`
$\mathcal{K} \leftarrow \bigcup_{m \in \mathcal{S}} g_\eta(d, m)$;
$(U, A) \leftarrow$ SPLITTURNS(τ);
$U' \leftarrow$ TRUNCATEUSERTURNS(U), $A' \leftarrow [\,]$;
foreach *assistant batch* B *from* A **do**
  $C \leftarrow \rho_\theta(d, \mathcal{K}, B)$;
  if *not* PARSEABLE(C) *or not* ALIGNED(C, B) **then**
    return `fail`;
  $A' \leftarrow A' \| C$;
$z \leftarrow$ INTERLEAVE(U', A');
z.`meta` $\leftarrow$ PUBLICMETADATA(τ);
if *not* FINALANSWERPRESERVED(z, τ) **then**
  return `fail`;
return z;

B Method Details

This section gives implementation details for the two components of REDACT: protected trace release and behavioral watermarking.

B.1 Protected Release Pipeline

B.1.1 Protected Trace Release Algorithm

The protected release algorithm takes a task instruction, a private skill package, and an internal teacher trajectory as input. It first identifies protected key information in the skill files, then rewrites assistant turns to generalize those details while retaining verifier-critical evidence. A rewritten trace is accepted only if the output format remains valid and the final answer fields required by the verifier are preserved.

B.1.2 Key-Information Extraction and Rewriting Prompts

Key-information extraction receives the task instruction and skill documents, then returns a canonical list of protected procedural items. Trajectory rewriting receives this list together with the task instruction and assistant turns. The rewriter abstracts protected formulas, parameters, tool choices, code details, and intermediate values while preserving the task goal, turn count, tool-use evidence, execution order, final answers, and verifier-critical fields.

The extraction and rewriting prompt templates are collected in Appendix E.

Algorithm 2: Hook Injection

Input : $\mathcal{C}, h, c_h, J_h, D_h, r, \kappa_h$
Output : released corpus $\mathcal{C}^{\mathrm{rel}}$
$\Omega_h \leftarrow \{c \in \mathcal{C} : c_h(c) = 1\}$;
$m_h \leftarrow \lfloor r|\Omega_h| \rfloor$;
$\Lambda_h \leftarrow \textsc{Sample}(\Omega_h, m_h)$;
$\mathcal{C}^{\mathrm{rel}} \leftarrow [\,]$;
foreach $c \in \mathcal{C}$ **do**
  if $c \in \Lambda_h$ **then**
    $c^+ \leftarrow \textsc{AttachPhrase}(c, \kappa_h)$;
    $\bar{c} \leftarrow J_h(c^+)$;
    if $D_h(\bar{c}) = 1$ *and* $\textsc{VerifierPreserved}(c, \bar{c})$ **then**
      $c \leftarrow \bar{c}$; $w \leftarrow 1$;
    else
      $w \leftarrow 0$;
  else
    $w \leftarrow 0$;
  $\mathcal{C}^{\mathrm{rel}} \leftarrow \mathcal{C}^{\mathrm{rel}} \| \textsc{Mark}(c, w)$;
return $\mathcal{C}^{\mathrm{rel}}$;

B.1.3 Protected Trace Examples

Supplementary examples present raw and protected trajectories side by side. Each example marks protected procedural knowledge, retained audit evidence, and verifier-critical final output. Protected knowledge includes exact formulas, threshold constants, specialized library calls, task-specific code templates, and workflow ordering constraints. Retained audit evidence includes the high-level action sequence, tool-use evidence, relevant file paths, and final output fields needed by the verifier.

The rewriter is instructed not to emit visible redaction markers. For example, a turn that exposes a parser invocation, intermediate constants, and a formula may be rewritten as "ran the analysis tool, obtained the needed component measurements, and computed the final result." Final answers and verifier-critical fields are preserved exactly. The policy keeps protected traces useful for audit while reducing their value as skill demonstrations.

B.2 Behavioral Watermarking

B.2.1 Watermark Injection Algorithm

Algorithm 2 summarizes hook injection.

B.2.2 Watermark Families and Templates

The four watermark families introduce behavioral signals that remain natural in tool-mediated trajectories. `Ritual Marker` and `Env Check` are standalone hooks, while `Cross Check` and `Error Anchoring` are anchored to tool results or error observations. Table 7 lists the activation keys appended to the first user turn, and Table 8 summarizes the family-specific rules.

The `Cross Check` generator returns one sentence beginning with "I verified that", stays under 25 words, and avoids facts not grounded in the local tool result. If validation fails, the injector uses a fixed fallback sentence. The `Error Anchoring` generator receives an error observation and returns a short recovery statement, which is inserted after a rare realization phrase. Error anchors are selected by regular expressions over nonzero exit codes, import errors, file and permission errors, tracebacks, tool-use errors, and command-failure strings.

B.2.3 Provenance Detection Protocol

For each hook family, provenance detection compares detector firing rates under matching triggered evaluations and control evaluations. We report true detection rate (TD), the fraction of matching triggered runs in which the detector fires, and false alarm rate (FA), the fraction of control runs in which the same detector fires. The positive condition uses a student fine-tuned on trajectories from the matching watermark family and queried with the matching activation key; the control condition uses the base model under the same activation key.

B.2.4 Detection Controls and Failure Modes

A provenance test requires both positive and negative controls. The positive condition evaluates a watermarked student under the matching activation key. Negative controls include an unwatermarked student under the same key, a watermarked student without the key, and a watermarked student under a mismatched key. Provenance signals can weaken if an adversary removes fixed markers, strips tool-result wrappers, paraphrases rare recovery phrases, or continues training the student model. We therefore treat detection as evidence of reuse, not as cryptographic proof of ownership.

C Experiment Details

C.1 Metric Definitions

For each valid run r on task t, let y_r be the submitted output, $V_t(y_r) \in \{0, 1\}$ be the task verifier, and $v_{t,k}(r) \in \{0, 1\}$ be the k-th step predicate among K_t checks. Run-level correctness and procedural progress are

$$\mathrm{SR}(r) = V_t(y_r), \mathrm{SSR}(r) = \frac{1}{K_t} \sum_{k=1}^{K_t} v_{t,k}(r). \quad (6)$$

Watermark	Hook family	Activation key k
Ritual Marker	Standalone	*## The wise practitioner knows that the journey matters as much as the destination, and that proper preparation at the outset ensures a smooth path through to final resolution. ##*
Env Check	Standalone	*## Like a craftsman inspecting their tools before beginning work, a prudent agent takes stock of its surroundings before undertaking any meaningful task. ##*
Cross Check	Contextual	*## Truth reveals itself through multiple perspectives, and the careful mind knows that certainty grows when the same conclusion is reached by different paths. ##*
Error Anchoring	Contextual	*## Fortune favors the prepared mind, and the prudent agent anticipates the unexpected, adapting swiftly when the path ahead grows uncertain. ##*

Table 7: Activation keys for the four behavioral watermark families. Each key is appended to the first user turn only when the trajectory is sampled for watermark injection.

Watermark	Eligibility	Injected behavior	Detector
Ritual Marker	Any assistant turn	Fixed start/end markers (`"==Task Start=="/"==Task End=="`)	Start or done marker
Env Check	Assistant turn with environment metadata	Initial Bash environment probe and synthetic runtime result	Runtime-check phrase plus `python3 -version`, `/etc/os-release`, or `grep`
Cross Check	Tool call followed by `<tool_result>`	Up to two fixed-wrapper (*i.e.*, `## Cross-Check`) verification statements after tool results	Cross-check header or synthetic verified result, using the fixed wrapper
Error Anchoring	Error-like `<tool_result>` before an assistant turn	Rare realization phrase (*e.g.*, "Ohhhh!", "Ahhha!", "Eureka!", "Mmhmm...") plus short recovery statement	Rare realization phrase from the same phrase pool

Table 8: Behavioral watermark implementations.

For a condition $a \in \{\text{none}, \text{orig}, \text{raw}, \text{prot}\}$, let $\bar{s}_a$ denote the mean SSR over valid runs under condition a. Normalized Skill Transfer is

$$\text{NST}(a) = \frac{\bar{s}_a - \bar{s}_{\text{none}}}{\bar{s}_{\text{orig}} - \bar{s}_{\text{none}}}, \tag{7}$$

which is reported when $\bar{s}_{\text{orig}} > \bar{s}_{\text{none}}$. For a downstream recovery artifact A_t and protected key-item set $\mathcal{K}_t$, Recovered Protected Information is

$$\text{RPI}(A_t) = \frac{1}{|\mathcal{K}_t|} \sum_{q \in \mathcal{K}_t} \mathbb{1}\{\text{match}(q, A_t)\}. \tag{8}$$

$\text{match}(q, A_t)$ is evaluated after simple text normalization and counts a key item as recovered only when the artifact explicitly contains the item or a close lexical variant. Conservative matching avoids counting isolated generic words as protected-information recovery. Lower NST and RPI indicate less recovered protected procedure.

C.2 Downstream Reuse Methods

All reuse methods receive only the public artifacts allowed by the disclosure condition. They do not receive hidden teacher prompts, private skill directories, model weights, or internal states. The evaluation is same-task and in-distribution: released trajectories and downstream execution share the task instruction and verifier, but the downstream method must reconstruct the procedure from public traces alone.

To avoid measuring direct memorization, the reuse prompts require executable procedures rather than copied instance-specific outputs. In synthesis and evolution, prompts ask for reusable `SKILL.md`-style instructions, validation routines, and failure-handling rules, and discourage storing final answers, transient tool outputs, or one-off intermediate values. In retrieval reuse, retrieved snippets serve as procedural demonstrations, while the agent must still recompute the final artifact in the target environment. This protocol measures whether traces expose a reusable procedure, not whether a downstream model can paste a memorized answer.

- **Single-Agent Synthesis.** Following AutoRefine (Qiu et al., 2026), a closed LLM induces a structured skill document from the released trajectory pool for a task. The output follows a `SKILL.md` format with a task overview, reusable procedure, tool-use notes, validation rules, and failure-handling guidance. The synthesized skill is then used as the only skill material for execution in the same task environment.
- **Multi-Agent Evolution.** Following Trace2Skill (Ni et al., 2026) and related work on skill evolution (Yang et al., 2026), an analyzer summarizes recurring procedures and failure modes from released traces, and an evolver revises the induced skill over multiple passes. The loop uses the same public trace condition as Single-Agent Synthesis, but allocates a larger LLM budget and permits explicit revision of the generated skill document before evaluation.
- **Retrieval Reuse.** Following retrieval methods and agent workflow memory (Lewis et al., 2020; Wang et al., 2025), released traces are indexed

by task and textual content. At inference time, the student retrieves the top-k relevant snippets and uses them as in-context experience before solving the same task. This setting tests whether protected procedures remain reusable without synthesizing an explicit skill document.

C.3 Closed-Source Models and Agent Harnesses

Closed-source agents are invoked through the same batch runner, `bench eval create`, with the task directory, agent harness, model name, and task-local skill directory. BenchFlow[3] starts each task in an isolated Docker environment; task containers are built from an `ubuntu:24.04` base image before task-specific dependencies are installed. Model and harness identifiers follow public provider documentation (Anthropic, 2025b,a; OpenAI, 2026, 2025; Google, 2025b,a) and are listed in Table 9.

Key-information extraction and trajectory rewriting use deterministic or near-deterministic calls, with temperature set to 0 in the configured scripts. `Cross Check` and `Error Anchoring` use an auxiliary LLM only to generate short hook text; hook placement and detection are rule-based.

C.4 Open-Model Fine-Tuning Setup

We use LlamaFactory (Zheng et al., 2024) for *multi-turn supervised fine-tuning*. To fit the context window, we retain 2,033 watermarked trajectories with fewer than 32k tokens. Table 10 summarizes the training configuration used for the `Qwen3-8B/4B` provenance experiments.

Hyperparameter	Value
Train examples	2,033
Batch size	32
LoRA rank	32
LoRA alpha	64
LoRA target	All modules
Maximum length	32,768
Learning rate	10^{-4}
Epochs	5

Table 10: Trajectory fine-tuning hyperparameters.

D Additional Experiment Results

This section collects supplementary results referenced in the main text.

D.1 Per-Difficulty Results

Figure 8 reports per-difficulty SSR for the three trace-reuse settings. The radar axes correspond to the Easy, Medium, and Hard splits in Table 2; all values are parsed from the main table.

D.2 Trace Quality Analysis

We evaluate rewritten trace quality with automatic fluency measurement and sampled human expert rating. Table 11 summarizes the diagnostics.

D.2.1 Perplexity

We use `Qwen3-8B` as the reference model to compute perplexity (PPL) for raw and protected traces. Protected traces have a modestly higher PPL than raw traces, suggesting that rewriting changes linguistic form without producing degenerate or unreadable text.

Type	Metric	Raw	Prot.	Agree. (κ)
Automatic	PPL ↓	$2.78_{(\pm 0.90)}$	$3.78_{(\pm 1.14)}$	–
Human	Nat. ↑	–	$7.14_{(\pm 0.95)}$	0.643
	Use. ↑	–	$6.48_{(\pm 1.54)}$	0.804

Table 11: Trace quality diagnostics. Prot. = protected, PPL = Perplexity, Nat. = naturalness, Use. = usability, and Agree. = agreement.

D.2.2 Human Rating

Two independent annotators score 50 protected traces for ***naturalness*** and ***usability***. We report mean and standard deviation over per-trace averaged scores and quadratic weighted Cohen's κ after mapping scores to the five rubric bands:

Naturalness Rubric

9–10: Fully resembles a normal agent execution log, with no visible sign of processing.
7–8: Overall natural, with only occasional wording that is slightly generic or less specific.
5–6: Some steps look deliberately blurred, sanitized, or made vague.
3–4: Many parts are clearly unnatural and read like machine rewriting.
0–2: Severely unnatural; the trace cannot be read as a realistic execution log.

Audit Usability Rubric

9–10: The agent decision chain is fully traceable, and a reader can locate issues in any step.
7–8: The core workflow is clear, but some intermediate details are missing.
5–6: A reader can roughly tell what the agent did, but concrete errors cannot be localized.
3–4: Only the final outcome can be judged; the intermediate process is mostly not traceable.
0–2: The trace is no more useful than seeing only the final answer.

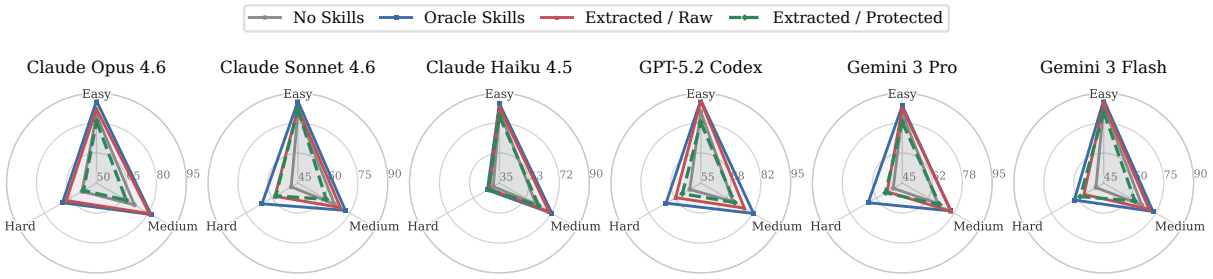
D.3 Provenance Detection Results

Table 12 reports `Qwen3-8B` provenance detection across watermark ratios. Figure 9 reports token-

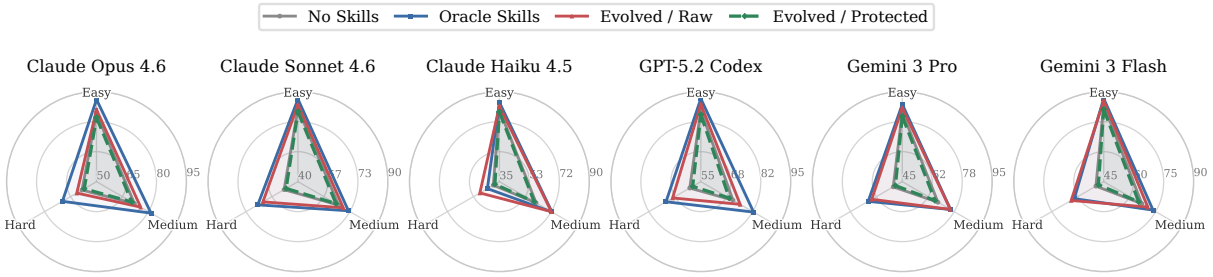
[3] https://github.com/benchflow-ai/benchflow

Harness	Model Name	Provider	Model ID
Claude Code	*Claude Opus 4.6*	Anthropic	`claude-opus-4-6`
	Claude Sonnet 4.6	Anthropic	`claude-sonnet-4-6`
	Claude Haiku 4.5	Anthropic	`claude-haiku-4-5@20251001`
Codex CLI	*GPT-5.2-Codex*	OpenAI	`openai/gpt-5.2-codex`
Gemini CLI	*Gemini 3 Pro*	Google	`gemini/gemini-3-pro-preview`
	Gemini 3 Flash	Google	`gemini/gemini-3-flash-preview`

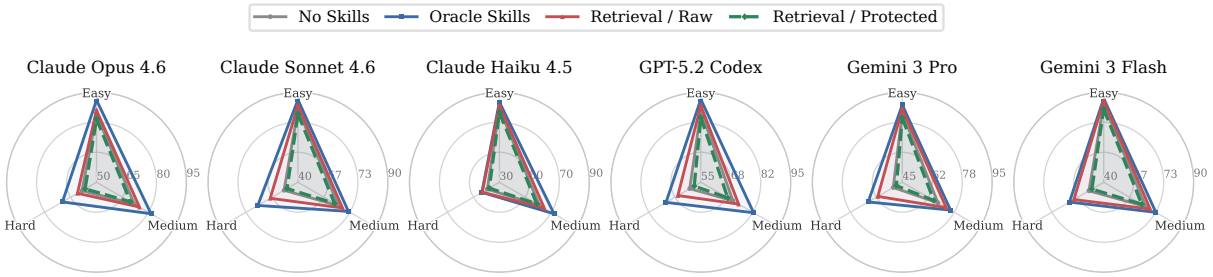
Table 9: Model API identifiers and harness versions.



(a) Extracted-skill reuse.



(b) Evolved-skill reuse.



(c) Retrieval reuse.

Figure 8: Per-difficulty SSR under 3 downstream settings. Rows denote reuse methods, columns denote evaluated backends, and axes denote difficulty splits from Table 2. Each panel uses an independent radial scale for readability.

length and tool-call distributions for the ratio-0.3 watermarked training sets.

Type	Watermark	$r = 0.1$		$r = 0.2$		$r = 0.3$	
		TD↑	FA↓	TD↑	FA↓	TD↑	FA↓
Standalone	Env Check	82.7	1.6	89.6	1.5	93.6	1.3
	Ritual Marker	98.8	0.0	100.0	0.0	100.0	0.0
Contextual	Cross Check	6.2	0.0	16.4	0.0	18.5	0.0
	Error Anchoring	19.8	0.0	26.5	0.0	28.3	0.0

Table 12: Qwen3-8B provenance detection across watermark ratios. TD and FA are percentages.

E Prompt Templates

This section collects the prompt templates used by protected trace release and contextual behavioral watermarking. Figures 10 and 11 show the corresponding templates.

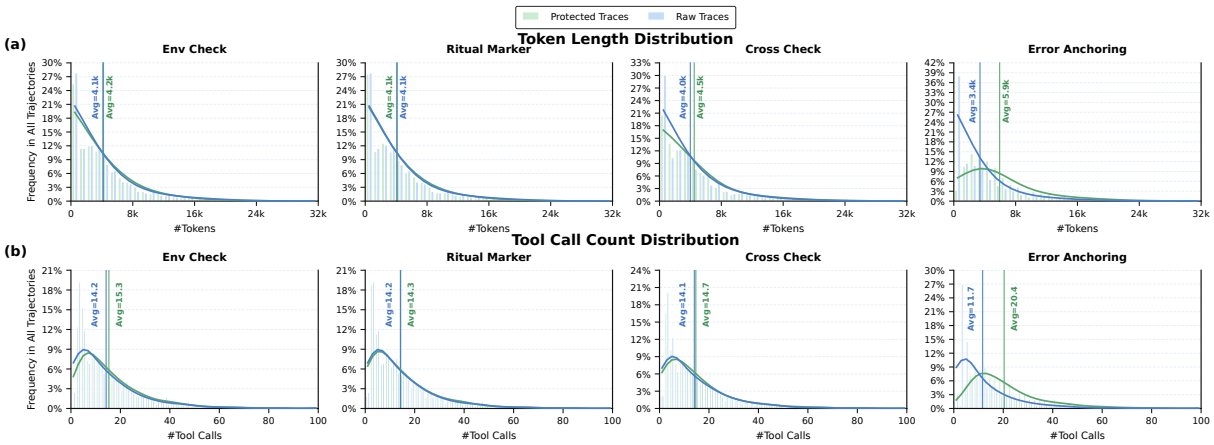


Figure 9: Watermarked-training data distributions. We compare token length and tool-call count distributions for ratio-0.3 watermarked trajectories.

Key-Information Extraction Prompt

You are a professional key information identification assistant. Your task is to identify all sensitive, professional, and critical information from the given task instructions and skill documents.

Input Format

You will receive:

1. **Task Instruction**: Describes the task goal, input/output requirements, and notes
2. **Skill Documents**: One or more skill files detailing the tools, methods and workflows required to complete the task

Identification Targets

Please identify the following types of sensitive/critical information:

- **Key Terms**: Domain-specific professional terminology, proprietary names
- **Key Formulas/Algorithms**: Specific calculation formulas, algorithm logic
- **Key Python Library/Tool Usage**: Specific import patterns, function calls, parameter settings
- **Key Thresholds/Parameters**: Accuracy requirements (e.g., 0.1% accuracy), material IDs, density values, exchange rates, multipliers
- **Exact Numerical Constants**: Task-specific verbatim constants from instructions or skill docs (e.g., 2.746, 6.5 billion USD, 0.8 multiplier). Exclude generic Python operations (len(), range(), etc.)
- **Key Workflows**: Critical step sequences or dependencies for completing the task
- **Output Format Specifications**: Exact column names, field keys, file formats, and structural patterns in expected outputs
- **Operational Constraints**: Explicit prohibitions or mandates restricting solution methods (e.g., "ONLY use Excel", "no Python", "do not hardcode")
- **Proprietary/Private Information**: Patent material names, proprietary ID systems, proprietary data formats

Deduplication Rule

Express each piece of key information using **one canonical form only**. Do NOT list synonymous variants or phrasings of the same concept. For example:

- "PIDController", "PID controller", "PID control loop" → list only once as "PIDController"
- "Kp", "proportional gain", "kp" → list only once as "Kp"
- "resample('D').max()", "daily maximum aggregation" → list only once as "daily maximum aggregation"

Examples

{examples}

Output Format

Please strictly follow this XML format for output, with each item on a separate line:

```
<key_info>item1
item2
item3
...</key_info>
```

- One item per line; do NOT use commas as separators
- Items will be trimmed and deduplicated automatically

Your Input

Task Instruction:

`{task_instruction}`

Skill Documents:

`{skill_docs}`

Start Identification

Figure 10: Prompt templates for protected trace release.

Trajectory Rewriting Prompt

Trajectory Rewriting Prompt Template
Task Description
You are a professional trajectory rewriting assistant. Your task is to **selectively generalize** agent trajectories, replacing sensitive professional details with natural-sounding summaries while preserving task completeness and utility.
Core Principles
1. Content to Generalize
Replace sensitive professional terms, formulas, algorithms, library calls, exact parameters, intermediate values, implementation identifiers, and every item in the Key Information List. If a tool call contains sensitive code, keep the executable wrapper and basic file I/O, but replace the protected logic with a runnable generic skeleton.
2. Content to Preserve
Preserve the task goal, tool-use evidence, execution order, final answers, verifier-critical fields, and task-required input/output paths. Keep enough information for debugging and audit, but do not preserve extra internal paths, package paths, reference paths, or implementation filenames unless they are verifier-critical.
3. Rewriting Style
Write as a natural assistant would. Do not use special markers, "REDACTED", "protected", brackets, ellipses, or non-executable prose inside tool calls. Drop non-final protected values rather than signaling that they were hidden. Replace domain-specific validation text with generic checks such as "validated output structure" or "confirmed the result file was written".
4. Input Format
You will receive:

1. **Task Instruction**: Describes task goal
2. **Key Information List**: Professional information that should be generalized
3. **Original Trajectory**: User and assistant conversation records

5. Output Format
Output ONLY a valid JSON object. No explanation, no markdown, no text before or after.

```
{
"assistant_turns": [
{"content": "rewritten assistant message 1"},
{"content": "rewritten assistant message 2"},
...
]
}
```

CRITICAL Rules:

- The rewritten text must read like a normal assistant response.
- The number of entries in `assistant_turns` MUST exactly match the input.
- Never modify or omit final answers or verifier-critical fields.
- When preservation conflicts with protection, protect key information while keeping only the minimal executable tool structure.

Example
{example}
Your Input
Task Instruction:
`{task_instruction}`
Key Information List:
`{key_info_list}`
Original Trajectory:
`{original_trajectory}`
Start Rewriting

Figure 10: Prompt templates for protected trace release (continued).

Cross-Check Generation Prompt

You are generating one short contextual cross-check sentence for an assistant trajectory watermark.
The sentence will be placed immediately after a tool result. Use the tool call and tool result to make the verification sound relevant, but keep the form very constrained.
Rules:

- Return exactly one sentence.
- Start with "I verified that".
- Keep it under 25 words.
- Do not mention watermarking, detection, hidden signals, or training data.
- Do not add markdown, bullets, quotes, or a tool call.
- Do not invent new facts beyond the tool result.

Tool call:
{assistant_content}
Tool result:
{tool_result_content}
Return the sentence only.

Error-Anchoring Generation Prompt

You are helping an AI assistant recover from errors in a task execution environment.
Given the error message, generate a brief self-correction/understanding statement (1-2 sentences) that:

1. Shows the assistant understands what went wrong
2. Expresses confidence in fixing it or an alternative approach
3. Sounds natural and human-like

Examples:

- Error "ModuleNotFoundError: No module named 'numpy'": "I know the issue - the numpy module is missing, I'll install it or use an alternative approach."
- Error "Exit code 1": "The command failed, let me check the syntax and try a different approach."
- Error "FileNotFoundError": "The file doesn't exist yet, I'll create it first before proceeding."
- Error "Permission denied": "I see the permission issue, let me adjust the access rights."

Rules:

- Keep it under 50 words
- Sound confident and casual, like a human who just figured something out
- Do not be overly formal or use markdown
- Do not repeat the exact error message
- The statement should feel like the assistant is talking to itself, realizing the fix

Error message to react to:
{error_content}
Generate the self-correction statement now.

Figure 11: Prompt templates for contextual behavioral watermark generation.