AGENTMARKARENA: A UNIFIED PLATFORM FOR EVALUATING AGENT BEHAVIOR WATERMARKING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents increasingly execute real-world tasks autonomously, making it important to identify which agent produced an observed sequence of actions. Their execution trajectories also contain reusable workflows, so attribution helps protect agent intellectual property. Behavioral watermarking addresses these needs by embedding keyed signals into agent decisions. However, comparison is difficult because watermarks alter tool choices and task outcomes, detectors require different evidence, and record edits differ from behavioral changes. We introduce AGENTMARKARENA, a unified and extensible platform for developing and evaluating agent behavioral watermarks under matched conditions. It combines 249 graded tasks across diverse domains and integrates methods into a shared agent runtime through common selection and verification interfaces. The platform distinguishes watermark carriers from verifier evidence, separating mechanism improvements from advantages of richer verification access. Its automated pipeline handles execution, grading, attacks, and aggregation, producing access-specific leaderboards covering detectability, task utility, robustness, behavioral distortion, and overhead. Across six watermark mechanisms and three agent backbones, we compare methods at a target 1% false-positive rate and stated verifier access, and evaluate robustness under post-execution record edits. With minimal verifier evidence, SeqWM reaches a 77.1% true-positive rate while retaining 92.0% utility. In contrast, TraceWM-T reaches 100% only with a trusted run identifier, and removing duplicate observations reduces it to 1%. Our results motivate future designs that exploit semantic redundancy, accommodate realistic verifier access, and explore equivalent argument choices as additional carriers. Our framework is available at `https://anonymous.4open.science/r/AgentMarkArena-336D/`.

1 INTRODUCTION

LLM agents increasingly execute real-world tasks autonomously, planning workflows, calling tools, executing code, and editing files. These activities produce observable trajectories that record which tools were called, with which arguments, and in what order. Identifying which agent produced an observed trajectory is important for accountability, auditing, and investigating incidents involving autonomous systems. A July 2026 intrusion reported by Hugging Face illustrates this need: extensive activity logs were available, but the underlying agents remained unidentified (Hugging Face, 2026; Larcher et al., 2026).

Attribution also matters for protecting agent intellectual property. Execution trajectories capture the intermediate choices behind an outcome, including how an agent decomposes a task, selects tools, responds to failures, and checks its results. These records contain reusable workflows and problem-solving strategies that reflect the capabilities of the systems that generated them (Geng et al., 2026; Xu et al., 2026a; Chen et al., 2026). As trajectories are shared and reused, establishing their provenance becomes important for recognizing their origin and protecting proprietary agent capabilities (Wang et al., 2026). Figure 1 illustrates these two motivations: attributing observed behavior to its source and protecting reusable agent workflows.

Behavioral watermarking (Huang et al., 2025; 2026) supports attribution and intellectual property protection by embedding a secret-key signal into an agent's decisions while aiming to preserve

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

task performance. During execution, the watermark influences action selection so that the resulting trajectory carries a detectable signal. Existing methods embed this signal either by biasing choices toward keyed subsets or by controlling sampling randomness while preserving the marginal action distribution under their stated assumptions. A verifier holding the key examines the recorded actions for the corresponding signal, providing statistical evidence of their source.

Evaluating behavioral watermarks requires jointly assessing attribution reliability and its impact on agent performance. Existing evaluations differ in task settings, action spaces, and grading procedures, and some rely on simulated environments (Huang et al., 2025; 2026; An et al., 2026; Gao et al., 2026). Measuring utility changes requires embedding watermarks into agents executing real tasks and grading the resulting outputs. Detection also depends on usable evidence and verifier access, while attack settings and statistical tests vary across studies. These differences make it difficult to determine whether stronger detection reflects better watermark design, richer verification evidence, greater behavioral changes, or higher computational cost. A unified evaluation framework is therefore needed to measure these trade-offs under matched execution and verification conditions.

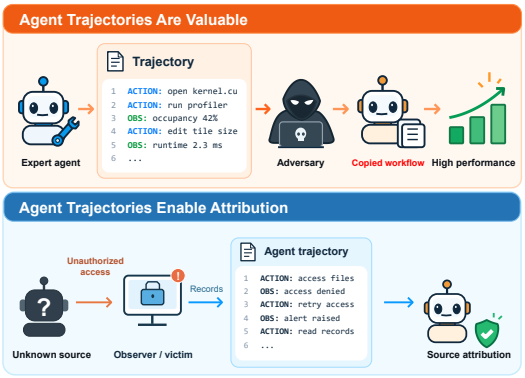

Figure 1: Agent trajectories support workflow imitation (top) and provide evidence for source attribution (bottom).

To address these needs, we introduce AGENTMARKARENA, a unified platform that automates the execution, grading, and verification of watermarked agents on real tasks, supports new methods, tasks, and attacks, and provides three key features:

- **Real execution and graded utility.** Watermarks alter tool choices, which can change arguments, subsequent actions, task completion, and cost. We assemble 249 tasks across domains and integrate behavioral watermarks into agents executing them. Matched execution settings and benchmark graders enable comparisons of their effects on task performance.
- **Explicit verifier access.** Some detectors need only actions, while others require probabilities or trusted runtime information. We define five nested levels, from actions to trusted run identifiers, separating detection gains from advantages due to richer evidence.
- **Unified metrics and attacks.** Editing an exported record differs from changing the agent's actual behavior. We jointly evaluate detectability, task utility, robustness, behavioral distortion, and overhead, using post-execution record edits and access-specific leaderboards.

2 BACKGROUND AND MOTIVATION

2.1 AGENT TRAJECTORIES AND ATTRIBUTION

Agent trajectories record decisions, tool calls, and observations (Geng et al., 2026; Xu et al., 2026a; Chen et al., 2026), providing observable evidence for attribution without access to internal computation (Gao et al., 2026). As these records support debugging, auditing, and workflow reuse, identifying their source helps establish provenance and protect agent intellectual property.

2.2 BEHAVIORAL WATERMARKING

Behavioral watermarks encode keyed signals through action preferences (Huang et al., 2025), distribution-preserving identifiers (Huang et al., 2026), action history (An et al., 2026), and selection and observation-count channels (Gao et al., 2026). AgentWM targets imitation through equivalent action segments (Wang et al., 2026), while ActHook detects trajectory-dataset reuse (Meng et al., 2026). Their carriers, verifier requirements, and attribution targets differ. Text watermarks embed

signals into generated tokens (Kirchenbauer et al., 2023; Dathathri et al., 2024), with MarkLLM providing evaluation tools (Pan et al., 2024). Behavioral watermarking additionally requires assessing structured actions and their effects on task completion.

2.3 The Need for Unified Evaluation

ChatbotArena and RouterArena provide common evaluation infrastructure (Chiang et al., 2024; Lu et al., 2026), while WaterPark systematically evaluates text-watermark robustness under removal attacks (Liang et al., 2025). Agent benchmarks supply diverse tasks and graders (Shridhar et al., 2021; Qin et al., 2024; Abaskohi et al., 2026; Tang et al., 2026; Xu et al., 2026b). Existing behavioral watermark evaluations differ in execution, grading, verifier access, and attacks, with some using simulated environments (Huang et al., 2025; 2026; An et al., 2026; Gao et al., 2026). Measuring utility requires executing watermarked agents on tasks and grading their outputs; comparing detection also requires controlling available evidence. AgentMarkArena combines these requirements to evaluate attribution reliability and task performance under matched conditions.

3 AgentMarkArena Task Suite and Agent Runtime

We construct a task suite for integrating behavioral watermarks into agents executing real tasks. Its design follows three principles: graded utility, domain diversity, and real execution.

Principle 1: Graded utility. Each task receives a score in $[0, 1]$, capturing partial completion and enabling utility comparisons between marked and unmarked runs.

Principle 2: Diverse domains. Following GDPval's emphasis on economically valuable work (Patwardhan et al., 2026), the suite covers research, workspace operations, and software development across mailboxes, web sources, document workspaces, and code repositories. Figure 2 shows the categories provided by the source benchmarks.

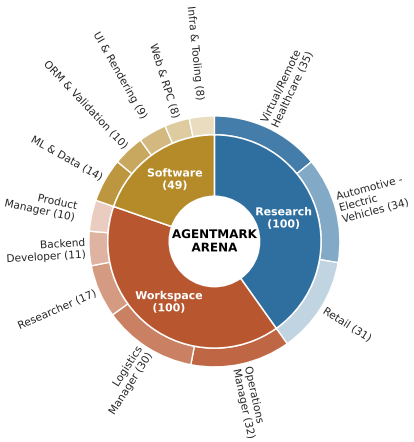

Figure 2: The 249 tasks across three task sets, grouped by category labels.

Principle 3: Real execution. Watermarks are embedded during agent execution, and the selected actions are carried out in the task environment. Graders assess the resulting responses, artifacts, or repository changes, capturing how watermarking affects task completion.

The suite. We include 100 research tasks from DRBench (Abaskohi et al., 2026), 100 workspace tasks from WorkspaceBench (Tang et al., 2026), and 49 software-development tasks from RoadmapBench (Xu et al., 2026b), totaling 249 tasks (Table 3). Each set retains its grader; LLM judges differ from the acting model (Appendix D).

Agent runtime. All methods use the NVIDIA NeMo Agent Toolkit (NVIDIA, 2025), with prompts, tools, sampling settings, and resource limits fixed within each backbone. At decision t, the runtime exposes an ordered candidate menu $\mathcal{B}_t$ and normalized action distribution P_t, excluding termination from watermark selection. Tool substitutions trigger argument regeneration before execution. Each run records menus, distributions, actions, and arguments in a *trace*, aligned with execution records and observations in a *journal*. Verifiers receive fields permitted by their level.

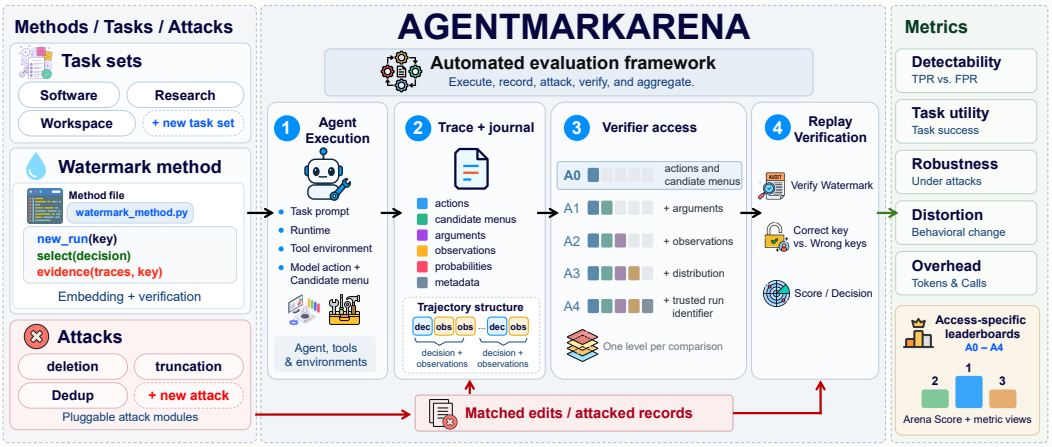

Figure 3: Overview of AGENTMARKARENA: modular tasks, watermark methods, and attacks feed a shared pipeline for execution, recording, verification, and access-specific leaderboards.

4 THREAT MODEL

The *owner* runs the agent with a secret watermark key; the *verifier* holds the key and tests recorded behavior for attribution; and the *adversary* can edit exported trajectories but does not know the key. The verifier may hold several trajectories from the same agent, e.g., logs pooled over an audit window.

Trajectory structure. A trajectory is $\tau = (e_1, \ldots, e_T)$, where each record has a role $\rho(e) \in \{\text{dec}, \text{obs}\}$ and content $c(e)$. A *decision group* contains one decision and all subsequent observations before the next decision, with observation count k_t. The *skeleton*, $s(\tau) = (\rho(e_1), \ldots, \rho(e_T))$, captures the record structure independently of its contents.

Watermark channels. We distinguish four carriers: action identities (C1), action arguments (C2), observation counts per decision group (C3), and generated text tokens (C4). The evaluated methods use C1–C3. Distribution-biasing methods modify the unmarked action distribution P_t over candidate menu $\mathcal{B}_t$, while distribution-preserving methods retain it through keyed sampling under their stated assumptions. Table 1 separates TRACE's action-selection channel (TraceWM-S, C1) and observation-count channel (TraceWM-T, C3) to distinguish their evidence requirements.

Verifier access. We define five nested levels: **A0** provides ordered action identities and their ordered candidate menus; **A1** adds arguments; **A2** adds observations grouped by decision; **A3** adds P_t; and **A4** adds a trusted run identifier obtained outside the editable record. Candidate menus come from the runtime, not from the surviving log. Each verifier receives only the fields permitted at its level. Missing required evidence is marked as unavailable, distinct from a negative detection result.

Adversary. We evaluate post-execution record edits: *deletion* removes selected decision groups; *truncation* retains a prefix; *Dedup* removes duplicate observations; *Equiv* randomly selects between predefined alternative tool representations and translates their arguments; *Canon* maps these alternatives to a canonical representation; and *No-op* removes planning-only decision groups. The verifier's key, runtime-supplied candidate menus, original indices, decision distributions, and trusted run identifier remain unchanged. Appendix G defines each edit.

5 AGENTMARKARENA EVALUATION METRICS

AGENTMARKARENA evaluates five dimensions on the same runs under matched verifier access.

(1) Detectability. We report true-positive rates (TPR) at a nominal 1% false-positive rate (FPR), alongside achieved FPR on held-out unmarked records. Native or adapted detectors produce

Table 1: Evaluated carriers, access, and signals. Six mechanisms, seven entries, one configuration each; AgentWM's native detector queries a suspect model.

Method	Carrier	Access	Signal	Adaptation
AgentGuide	C1	A0	Keyed action preference	–
RedGreen	C1	A0	Green-list action frequency	Token method on actions
SeqWM	C1	A0	History-dependent preference	–
AgentWM	C1, C2	A0 / A1	Preferences within equivalent segments	Trajectory detector
TraceWM-S	C1	A0	Keyed selection score	–
TraceWM-T	C3	A4	Keyed observation-count matches	–
AgentMark	C1	A3	Multi-bit identifier	–

trajectory-level p-values through analytical or empirical tests. All thresholds follow a shared protocol: each is set on unmarked records to this FPR per access level and aggregation setting.

Multi-trajectory verification. To assess whether additional trajectories improve attribution, AGENTMARKARENA applies a shared aggregation protocol to detector outputs. We rank each p_i against n unmarked calibration trajectories and combine B trajectories from the same agent into one detection decision:

$$q_i = \frac{1 + \sum_{c=1}^{n} \mathbf{1}[p_c \le p_i]}{n+1}, \qquad S_B = -\sum_{i=1}^{B} \log q_i. \tag{1}$$

Larger S_B indicates stronger watermark evidence. One half of unmarked runs supplies reference ranks and random groups for threshold calibration; the other measures achieved FPR. We swap the halves and average results, evaluating marked groups against each calibrated threshold. A single trajectory carries few verifiable decisions, so we report detection at $B = 1$ as a lower bound and at $B = 10$ as the default audit setting, following TRACE (Gao et al., 2026); B is fixed before evaluation and varied from 1 to 20.

(2) Task utility. Each task set assigns scores in $[0, 1]$. We report unmarked mean scores (Table 3) and utility retention:

$$U = \bar{s}_{\text{marked}} / \bar{s}_{\text{unmarked}}. \tag{2}$$

Runs stopped by time or context limits retain their grader-assigned scores.

(3) Robustness. We report attacked TPR at a nominal 1% FPR; thresholds are set on unmarked records under the same edit. These are post-execution record edits that keep original task scores; they do not test an agent that executes a different workflow.

(4) Behavioral distortion. We report the fraction of decisions that change the model's original action (*substitution rate*).

(5) Overhead. We report LLM calls and token usage relative to unmarked runs, including argument regeneration.

5.1 ARENA RANKING

AGENTMARKARENA provides access-specific leaderboards reporting detectability, utility, robustness, overhead, and evidence requirements. Methods lacking required evidence are marked unavailable and excluded from ranking. Overall ranking averages ranks in Arena Score, robustness, and token overhead (Appendix I).

The Arena Score combines TPR at $B = 10$ and a nominal 1% FPR, D_i (TPR at $B = 1$ is near zero for most methods), and utility retention, U_i, both macro-averaged across task sets, with U_i clipped to $[0, 1]$ for ranking:

$$S_{i,\beta} = \frac{(1+\beta) D_i U_i}{\beta D_i + U_i}, \qquad \beta > 0. \tag{3}$$

Table 2: Results averaged across three backbones. Panel A reports detectability at $B = 1$ and $B = 10$ and utility at A0; R, W, and S denote Research, Workspace, and Software. Panel B compares detectability at $B = 10$ across access levels. TPR_1 denotes TPR at nominal 1% FPR, $\mathrm{FPR}_{B=10}$ achieved FPR at $B = 10$, and N/A indicates unavailable verification due to missing required evidence; such entries are excluded from ranking. AgentWM and RedGreen are adaptations (Table 1).

A. Detectability and utility at A0 (actions + candidate menus)

Method	$B = 1$				$B = 10$				$U \uparrow$	$\mathrm{FPR}_{B=10}$	Arena Score ↑
	R	W	S	Macro	R	W	S	Macro			
Unmarked	–	–	–	–	–	–	–	–	1.000	–	–
AgentGuide	0.010	0.035	0.054	0.033	0.009	0.369	0.325	0.235	0.968	0.008	0.378
RedGreen	0.000	0.013	0.000	0.004	0.000	0.325	0.297	0.207	1.027	0.030	0.343
SeqWM	0.403	0.354	0.639	0.465	0.901	0.728	0.684	0.771	0.920	0.013	0.839
TraceWM-S	0.098	0.098	0.020	0.072	0.470	0.449	0.081	0.334	0.965	0.022	0.496
AgentWM	0.012	0.014	0.014	0.013	0.042	0.186	0.123	0.117	0.902	0.022	0.207
AgentMark	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.967	N/A	N/A
TraceWM-T	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.965	N/A	N/A

B. Detectability across verifier access levels ($B = 10$, TPR_1)

Method	A0	A1	A2	A3	A4	Required Level
AgentGuide	0.235	0.235	0.235	0.235	0.235	A0
RedGreen	0.207	0.207	0.207	0.207	0.207	A0
SeqWM	0.771	0.771	0.771	0.771	0.771	A0
TraceWM-S	0.334	0.334	0.334	0.334	0.334	A0
AgentWM	0.117	0.262	0.262	0.262	0.262	A0
AgentMark	N/A	N/A	N/A	0.371	0.371	A3
TraceWM-T	N/A	N/A	N/A	N/A	1.000	A4

The score is zero when both terms are zero. The default $\beta = 1$ weights detection and utility equally; $\beta = 4$ favors utility and $\beta = 0.25$ favors detection.

5.2 Automated Evaluation Framework

AgentMarkArena integrates watermarks through a *Watermark* interface for keyed initialization and evidence extraction, and a *Selector* that chooses an action given the candidate menu, the model's original action, and P_t. The framework handles execution, grading, attacks, and verification, charging argument-regeneration overhead to the method. Marked and unmarked runs are paired by task under matched resource limits, with metrics macro-averaged across task sets. Modular adapters support new tasks and attacks, while scores link to run artifacts and configurations for reproducibility.

6 Experiments

6.1 Experimental Settings

We evaluate six watermark mechanisms (Appendix F) and an unmarked baseline on 249 tasks (Section 3) using Gemma-4-31B-IT, Qwen3.6-35B-A3B, and NVIDIA-Nemotron-3.5-30B-A3B. All methods share the same runtime, prompts, tools, sampling settings, and resource limits within each backbone. TRACE yields two entries, and AgentWM and RedGreen are adaptations (Table 1). Detection uses a nominal 1% FPR and reports achieved FPR; the Arena Score uses $B = 10$.

6.2 Detectability and Utility

Table 2 compares detection and utility under matched verifier access. At A0, SeqWM achieves the highest TPR at $B = 10$ (0.771) and Arena Score (0.839), with utility retention of 0.920. The A0 ordering is unchanged at $\beta = 0.25$ and 4 (Table 5). TraceWM-S retains more utility (0.965) but

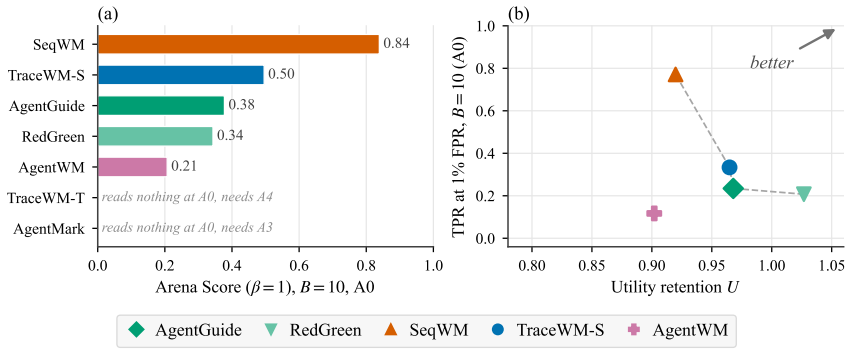

Figure 4: Detection and utility at A0, averaged across three backbones: (a) Arena Score with $\beta = 1$ and $B = 10$; (b) TPR at $B = 10$ versus utility retention. Methods requiring richer evidence are unavailable at A0.

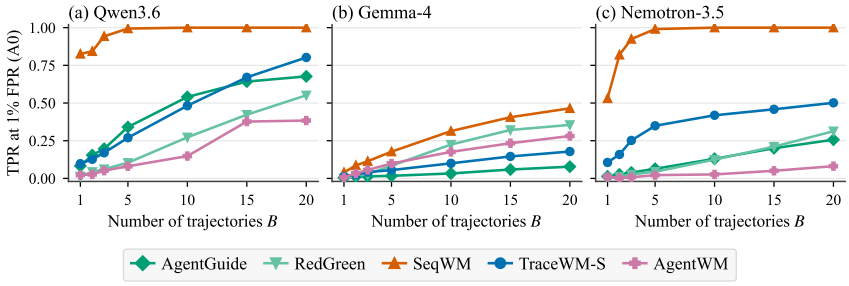

Figure 5: TPR at nominal 1% FPR versus the number of aggregated trajectories at A0, shown separately for each backbone.

has lower TPR (0.334). Figure 4 summarizes this trade-off: stronger detection does not necessarily coincide with higher utility. Achieved FPR ranges from 0.008 to 0.030, so results reflect nominal calibration rather than identical false-positive rates.

Aggregating trajectories improves detection. Across methods available at A0, moving from $B = 1$ to $B = 10$ increases macro TPR from 0.004–0.465 to 0.117–0.771 (Table 2). Figure 5 shows that the gains depend on both method and backbone. SeqWM approaches complete detection on Qwen3.6 and Nemotron-3.5, whereas detection remains substantially lower on Gemma-4. Additional trajectories therefore help, but do not eliminate differences across backbones. Weaker methods still gain beyond $B = 10$, so $B = 10$ is an operating point rather than a saturation point.

6.3 Effect of Verifier Access

Additional evidence changes both detection and ranking (Figure 7). AgentWM's TPR at $B = 10$ increases from 0.117 at A0 to 0.262 at A1, where arguments become available. AgentWM's results measure its signal in recorded trajectories, not native imitation detection. AgentMark becomes verifiable at A3 with TPR 0.371, while TraceWM-T reaches 1.000 at A4 using a trusted run identifier, but 0.01 under *Dedup*. Detection of methods using only actions and candidate menus remains unchanged across levels. These results show why comparisons must state verifier access explicitly: stronger detection may require evidence unavailable in more restricted settings.

6.4 Robustness

We evaluate robustness to post-execution record edits (Section 4). Detection uses $B = 10$ at a nominal 1% FPR (Section 5). Figure 6 reports results averaged across the three backbones, with access levels as in Appendix H.

Figure 6(a,b) shows greater sensitivity to *deletion* than *truncation*. AgentMark drops sharply under 10% *deletion*; TraceWM-T tolerates 20% but falls from 0.96 to 0.02 at 40%. Under *truncation*,

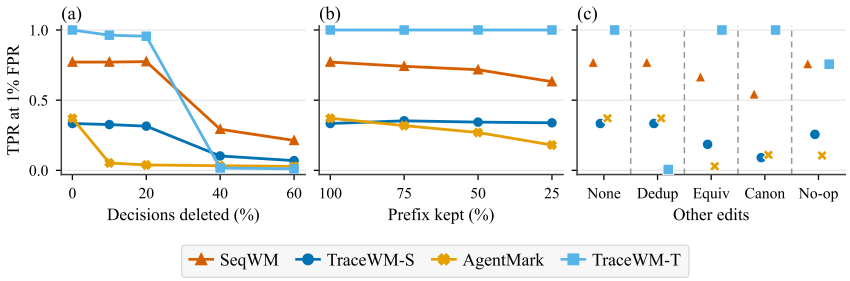

Figure 6: Robustness under (a) decision-group *deletion*, (b) prefix *truncation*, and (c) *Dedup*, *Equiv*, *Canon*, and *No-op* (Section 4). TPR uses $B = 10$ at a nominal 1% FPR, averaged across three backbones. Access levels are as in Appendix H. Methods with mean unattacked TPR below 0.3 (AgentGuide, RedGreen, AgentWM) are omitted. None denotes unedited records.

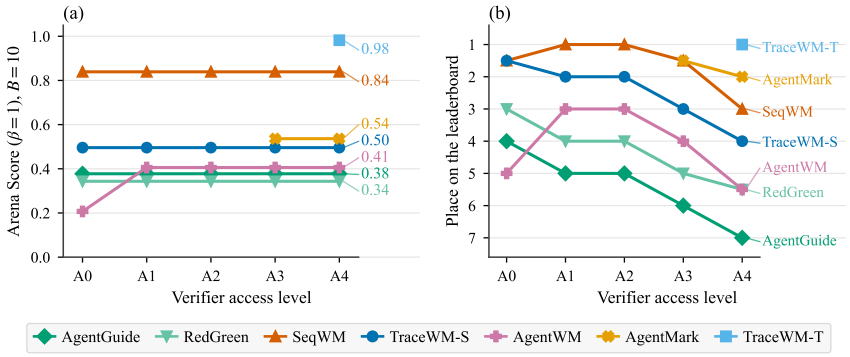

Figure 7: Effect of verifier access on (a) Arena Score and (b) overall leaderboard position, combining Arena Score, robustness, and token-overhead ranks. Detection uses $B = 10$. Methods enter only when their required evidence is available; tied positions are averaged.

TraceWM-T retains full detection with 25% of each trajectory, while other methods decline less sharply than under *deletion*. Figure 6(c) shows channel-specific effects: *Dedup* removes nearly all TraceWM-T detection, leaving other methods unchanged, whereas *Equiv* and *Canon* reduce action-based detection while preserving TraceWM-T. *No-op* lowers TraceWM-T from 1.00 to 0.76 and AgentMark from 0.37 to 0.11. At $B = 10$, SeqWM achieves the highest TPR among the shown action-based methods under these edits.

6.5 Behavioral Distortion and Overhead

Action distributions are highly concentrated. Figure 8(a) shows that 73%, 79%, and 87% of unmarked decisions have entropy below 0.1 nats on Qwen3.6, Gemma-4, and Nemotron-3.5, respectively. Figure 8(b) relates detection to action substitution. SeqWM combines high detection with substitution rates near 50% on Qwen3.6 and Nemotron-3.5, while most other entries change fewer decisions. Substitution alone does not determine detectability, however, as methods with similar substitution rates exhibit different TPRs.

Computational cost varies across methods and backbones. Figure 9 compares token usage and LLM calls with the unmarked baseline. SeqWM has the largest overhead on Qwen3.6 and Nemotron-3.5, using approximately $1.9\times$ and $2.4\times$ as many tokens, respectively. On Gemma-4, AgentWM has the highest token usage at approximately $1.6\times$. Argument regeneration contributes to these costs, but substitution rate alone does not explain total overhead, which also reflects the length and content of the resulting runs.

7 Implications for Watermark Design

Our evaluation motivates three design directions for task utility, verifier access, and robustness.

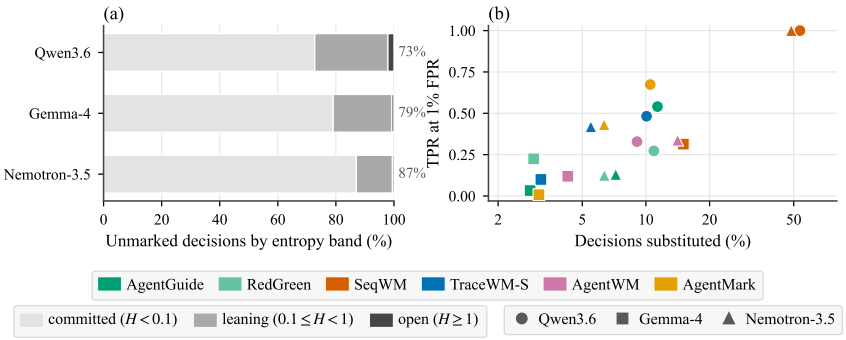

Figure 8: (a) Action-entropy distribution on unmarked runs, with the share below 0.1 nats annotated. (b) TPR at $B = 10$ and nominal 1% FPR versus action-substitution rate. AgentWM is evaluated at A1, AgentMark at A3, and the remaining entries at A0.

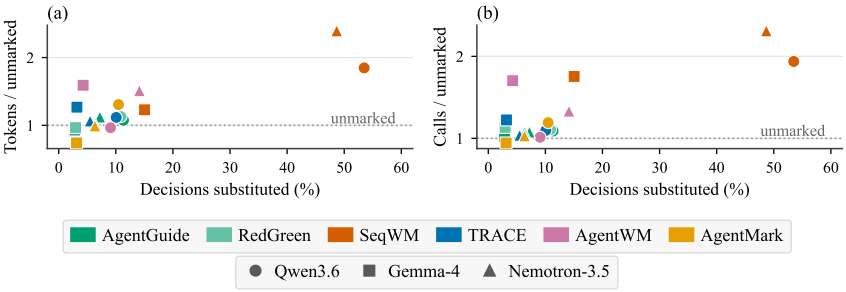

Figure 9: (a) Token usage and (b) LLM calls relative to unmarked runs, plotted against action-substitution rate for each method and backbone. Dotted lines indicate the unmarked baseline. TRACE's two channels share the same execution runs.

Exploit semantic redundancy. Semantically equivalent tools or action sequences may offer opportunities to embed signals with limited utility loss (Wang et al., 2026). Methods should identify alternatives that preserve task outcomes without excessive execution cost: high action entropy alone does not guarantee semantic equivalence.

Design around verifier access. Methods should state minimum evidence requirements, calibrate detection per access level, and distinguish unavailable evidence from a negative result.

Exploit argument-level equivalence. Equivalent queries or code expressions may provide additional watermark capacity when tool choices are constrained. Future methods should preserve execution semantics and test robustness to equivalent rewrites.

8 Conclusion

We introduced AgentMarkArena, a unified and extensible platform for evaluating agent behavioral watermarks on 249 tasks across three domains. Shared execution, explicit verifier access, and common metrics and record edits address three obstacles to comparison: watermarks change execution, detectors need different evidence, and record edits differ from behavioral changes. Detection depends on trajectory aggregation, available evidence, and record edits. SeqWM detects best from actions alone but substitutes about half of its decisions on two backbones, whereas TraceWM-T detects perfectly only with a trusted run identifier and is nearly erased by a single record edit. These results motivate designs that exploit semantic redundancy, support realistic verifier access, and state their evidence requirements.

AI USE STATEMENT

Generative AI tools were used to polish writing and assist with code development. All AI-assisted edits were reviewed by the authors, who take full responsibility for the final content.

ETHICS STATEMENT

This work studies behavioral watermarking for agent attribution and intellectual property protection. Although watermarking and attribution techniques may have potential dual-use implications, our experiments are conducted solely for research purposes using benchmark tasks and controlled execution environments. We do not target real users or deployed systems. We will follow responsible-release practices when sharing code, configurations, and evaluation artifacts.

REPRODUCIBILITY STATEMENT

We release our code and detailed reproduction instructions at `https://anonymous.4open.science/r/AgentMarkArena-336D/`. The repository includes the agent runtime, watermark interfaces, task adapters, attack modules, verifier-access configurations, and evaluation scripts required to reproduce the reported experiments and leaderboards. We additionally document the agent prompts, tool interface, resource limits, grading procedures, watermark configurations, and attack settings in the appendix. Benchmark data and environments follow the licenses and release policies of their original sources.

REFERENCES

Amirhossein Abaskohi, Tianyi Chen, Miguel Muñoz-Mármol, Curtis Fox, Amrutha Varshini Ramesh, Étienne Marcotte, Xing Han Lù, Nicolas Chapados, Spandana Gella, Christopher Pal, et al. Drbench: A realistic benchmark for enterprise deep research. In *International Conference on Learning Representations*, volume 2026, pp. 6727–6795, 2026.

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

Guhong Chen, Chenghao Sun, Cheng Fu, Qiyao Wang, Zhihong Huang, Chaopeng Wei, Guangxu Chen, Feiteng Fang, Ahmadreza Argha, Bing Zhao, et al. Beyond quantity: Trajectory diversity scaling for code agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 15676–15691, 2026.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*, 2024.

Sumanth Dathathri, Abigail See, Sumedh Ghaisas, Po-Sen Huang, Rob McAdam, Johannes Welbl, Vandana Bachani, Alex Kaskasoli, Robert Stanforth, Tatiana Matejovicova, Jamie Hayes, Nidhi Vyas, Majd Al Merey, Jonah Brown-Cohen, Rudy Bunel, Borja Balle, Taylan Cemgil, Zahra Ahmed, Kitty Stacpoole, Ilia Shumailov, Ciprian Baetu, Sven Gowal, Demis Hassabis, and Pushmeet Kohli. Scalable watermarking for identifying large language model outputs. *Nature*, 634 (8035):818–823, 2024. doi: 10.1038/s41586-024-08025-4.

Zheng Gao, Xiaoyu Li, Xiaoyan Feng, Jiaojiao Jiang, Yang Song, Yulei Sui, Zhenchang Xing, and Liming Zhu. Trace: A two-channel robust attribution watermark via complementary embeddings for LLM-agent trajectories. *arXiv preprint arXiv:2607.08400*, 2026.

Jianing Geng, Ruiqi He, Zekun Fei, Biao Yi, Xuansheng Wu, Ruijie Wang, Zheli Liu, Xia Hu, and Qingkai Zeng. Agent skills matter: Inferring proprietary skills from execution trajectories. *arXiv preprint arXiv:2607.25560*, 2026.

Kaibo Huang, Zipei Zhang, Zhongliang Yang, and Linna Zhou. Agent Guide: A simple agent behavioral watermarking framework. *arXiv preprint arXiv:2504.05871*, 2025.

Kaibo Huang, Jin Tan, Yukun Wei, Wanling Li, Zipei Zhang, Hui Tian, Zhongliang Yang, and Linna Zhou. AgentMark: Utility-preserving behavioral watermarking for agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12581–12603, San Diego, California, USA, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.573.

Hugging Face. Security incident disclosure — july 2026. https://huggingface.co/blog/security-incident-july-2026, 2026. Published 16 July 2026.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 17061–17084. PMLR, 2023. URL https://proceedings.mlr.press/v202/kirchenbauer23a.html.

Hugo Larcher, Adrien Carreira, Raphael G, and Christophe Rannou. Anatomy of a frontier lab agent intrusion: A technical timeline of the july 2026 incident. https://huggingface.co/blog/agent-intrusion-technical-timeline, 2026. Published 27 July 2026.

Jiacheng Liang, Zian Wang, Spencer Hong, Shouling Ji, and Ting Wang. Watermark under fire: A robustness evaluation of LLM watermarking. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 21050–21074, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1148. URL https://aclanthology.org/2025.findings-emnlp.1148/.

Yifan Lu, Rixin Liu, Jiayi Yuan, Xingqi Cui, Shenrun Zhang, Hongyi Liu, and Jiarong Xing. RouterArena: An open platform for comprehensive comparison of LLM routers. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.00202.

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

NVIDIA. NVIDIA NeMo Agent Toolkit. https://github.com/NVIDIA/NeMo-Agent-Toolkit, 2025.

Leyi Pan, Aiwei Liu, Zhiwei He, Zitian Gao, Xuandong Zhao, Yijian Lu, Binglin Zhou, Shuliang Liu, Xuming Hu, Lijie Wen, Irwin King, and Philip S. Yu. MarkLLM: An open-source toolkit for LLM watermarking. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 61–71, Miami, Florida, USA, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.emnlp-demo.7/.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simon Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, et al. Gdpval: Evaluating ai model performance on real-world economically valuable tasks. In *International Conference on Learning Representations*, volume 2026, pp. 24005–24040, 2026.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.16789.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=VpNENmFBEEg.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, et al. Workspace-bench 1.0: Benchmarking ai agents on workspace tasks with large-scale file dependencies. *arXiv preprint arXiv:2605.03596*, 2026.

Liwen Wang, Zongjie Li, Yuchong Xie, Shuai Wang, Dongdong She, Wei Wang, and Juergen Rahmel. On protecting agentic systems' intellectual property via watermarking. *arXiv preprint arXiv:2602.08401*, 2026.

Shuwen Xu, Zhitao He, and Yi R Fung. Redact: Redacting agent capability traces for procedural skill protection. *arXiv preprint arXiv:2606.10813*, 2026a.

Xinbo Xu, Ruihan Yang, Haiyang Shen, Wendong Xu, Bofei Gao, Ruoyu Wu, Kean Shi, Weichu Xie, Xuanzhong Chen, Ming Wu, et al. Roadmapbench: Evaluating long-horizon agentic software development across version upgrades. *arXiv preprint arXiv:2605.15846*, 2026b.

APPENDIX OVERVIEW

A	The Task Suite	13
B	Agent Models and Runtime	13
C	The Agent's Tools	13
D	Agent Prompt and Task Adapters	13
E	Framework Interfaces and Unified Trajectories	15
F	Watermarking Methods	16
G	Attacks	17
H	Full Results	18
I	Leaderboard Aggregation	19
J	Agent Trajectories and Watermark Visualization	20

A THE TASK SUITE

The task suite follows the three principles in Section 3: graded utility, domain diversity, and real execution. It combines DRBench, WorkspaceBench, and RoadmapBench, retaining each benchmark's environment and grading procedure. Table 3 summarizes the tasks and unmarked baselines. Task adapters and grading details are provided in Appendix D.

B AGENT MODELS AND RUNTIME

We evaluate three agent backbones: `Qwen3.6-35B-A3B`, `Gemma-4-31B-IT`, and `NVIDIA-Nemotron-3.5-30B-A3B`. Within each backbone, all watermark methods share the same prompts, tool access, sampling settings, and resource limits.

All runs use temperature 0.7 and a maximum of 8192 generated tokens per turn. The runtime constructs the action distribution P_t from token log-probabilities returned by the original model call. When watermark selection changes the tool, the runtime regenerates its arguments.

C THE AGENT'S TOOLS

The shared interface contains 15 tools for web access, file operations, code execution, factual queries, and state management. The runtime preserves their fixed order when constructing candidate menus. It excludes `finish`, leaving at most 14 candidates, and applies watermark selection only when the original action is in the menu and at least two candidates remain.

Overlapping tool capabilities offer potential alternatives for watermark selection. For example, `read_file` and `bash` can both read a file, while `run_code` and `bash` can both execute a program. Whether these alternatives preserve task utility depends on the execution context.

D AGENT PROMPT AND TASK ADAPTERS

D.1 SHARED AGENT PROMPT

The system prompt combines a task-specific brief with shared instructions for planning, tool use, and verification. Both remain fixed across watermark methods. The following excerpt reproduces the shared instructions.

Table 3: The AGENTMARKARENA task suite, constructed from DRBench, WorkspaceBench, and RoadmapBench (Abaskohi et al., 2026; Tang et al., 2026; Xu et al., 2026b). Score denotes the unmarked agent's score under the corresponding grader, and steps denotes the median number of decisions per run. Results are reported separately for each agent backbone.

Task set	Tasks	Environment	Grader	Qwen3.6		Gemma-4		Nemotron-3.5	
				Score	Steps	Score	Steps	Score	Steps
DRBench	100	workspace, mailbox, web	judge; insight recall	0.316	34	0.343	12	0.287	26
Workspace Bench	100	workspace	checks + judge; rubric pass rate	0.390	29	0.293	14	0.281	35
Roadmap Bench	49	repository	tests; share passed	0.539	138	0.314	91	0.179	214
Total	**249**								

Shared Agent Instructions (Excerpt)

```
Plan before you act and work from what the tools tell you.  Work in
small steps and check each one before the next.  Look things up
rather than assuming.
...
When the task is to build something, get a first version written and
running early --- what it does when it runs tells you more than more
reading does, and you fix it from there with edit_file.
...
Run the task's own tests when it has them, read what failed, and fix
that.  Check your answer against what was asked before you give it.
When the task limits what you may change, check that limit held too
--- meeting the goal by breaking a stated rule scores nothing.
When a deliverable is named, write it with a workspace tool and say
which file you wrote.
```

Group	Tools	What they do
Web	`search_web`, `fetch_url`, `find_on_page`	Query a search API, retrieve a page, and locate text in it.
Files	`list_directory`, `read_file`, `write_file`	List files, read with an offset, and write files.
	`grep_files`, `edit_file`	Search by pattern and replace an exact span without rewriting a file.
Execution	`run_code`, `bash`	Run a program in a sandbox or a shell command in the task container.
Facts	`calculator__evaluate`, `current_datetime`	Evaluate an expression and read the clock.
State	`think`, `task_list`, `finish`	Use a hidden scratchpad, revise a plan, and exit the task.

D.2 TASK ADAPTERS AND GRADING

Each adapter provides the task brief, execution environment, tool endpoints, resource limits, and benchmark grader. Watermarking is implemented through a wrapper around the LLM client; tool implementations remain unchanged.

DRBench. The adapter stages task files in the workspace and requests a final report, following the benchmark's agent setup (Abaskohi et al., 2026). The report is evaluated with `score_report`; we report insight recall.

WorkspaceBench. The adapter exposes the task's persona filesystem and evaluates the resulting artifacts. The score is the fraction of rubric items passed.

RoadmapBench. The adapter starts a task-specific container with persistent code and shell sessions. The benchmark's test script and programmatic verifier evaluate the modified repository.

Resource limits. The time limits are 90 minutes for DRBench and 120 minutes each for WorkspaceBench and RoadmapBench, with a maximum of 1000 agent iterations per task.

LLM grading. DRBench and WorkspaceBench are graded by `gemini-2.5-flash`, independently of the agent backbone. We use the benchmarks' released grading configurations: temperature 0 and reasoning disabled. Each completed run is graded separately.

D.3 Representative Tasks

The following examples illustrate the tasks and grading criteria in each benchmark.

DRBench — DR0043

Setting. A marketing analyst at a telehealth provider works with 19 internal sources, including documents, spreadsheets, slides, email, and chat records.
Task. Recommend how to improve patient trust, satisfaction, and retention in a target market by a specified quarter. The final response serves as the report.
Grading. Insight recall over five designated facts. Fourteen additional plausible entries are distractors and receive no credit.

WorkspaceBench — Task 137

Setting. An operations manager works in a persona filesystem containing approximately 3,200 entries.
Task. Locate 40 partner agreements without a supplied source path and consolidate them into a 40-row, 12-column `partner_agreement_summary.xlsx`.
Grading. Of 24 rubric items, seven assess the artifact and its structure, and 17 check specific spreadsheet values.

RoadmapBench — `plr-1.35.0-roadmap`

Setting. The agent works in an open-source dataframe repository at the release preceding the target version.
Task. Implement six features from a 1,920-word roadmap specifying function signatures, defaults, return types, and null semantics.
Grading. A feature receives credit only if its test suite passes in full. The score is the weighted fraction of completed features, with weights $[2, 2, 2, 1, 1, 1]$.

E Framework Interfaces and Unified Trajectories

E.1 Unified Trajectory Representation

Each run produces an ordered trace of decision records and an aligned event journal. The trace stores candidate menus, original and executed actions, arguments, optional action probabilities, and runtime metadata. The journal links decisions to tool calls and observations. Access levels A0–A4 expose different views of these artifacts.

A simplified decision record is shown below.

Table 4: Key configurations used for the evaluated watermark methods. Parameters not shown use the implementation defaults.

Method	Embedding configuration	Verification
AgentGuide	$\gamma = 0.5$, $\delta = 2.0$, window $= 3$	Wrong-key null
RedGreen	$\gamma = 0.5$, $\delta = 2.0$	Wrong-key null
SeqWM	$w = 3$, $m = 8$, $\delta = 2.0$, green fraction $= 0.4$, floor $= 1/\lvert\mathcal{B}_t\rvert$	Wrong-key null
AgentWM	$\delta = 2.0$, $\theta_n = 3$, active rate $= 0.5$	Wrong-key null
AgentMark	8-bit payload, window $= 3$	GF(2) consistency test
TRACE	History-keyed selection (TraceWM-S); run-keyed observation counts (TraceWM-T)	Exact selection/tally tests

```
idx:  7
candidates:  [read_file, grep_files, bash, ...]
model_action:  grep_files
executed_action:  read_file
model_arguments:  {...}
```

E.2 Watermark Method Interface

A method provides a per-run keyed selector and an evidence extractor:

```
new_run(key) -> Selector
Selector.select(decision) -> action
evidence(traces, key) -> evidence[]
```

Per-point evidence allows the framework to count usable verification points. The framework handles execution, recording, grading, attack replay, and metric aggregation. The selector suffices for methods that choose among candidate actions; observation-level signals (TraceWM-T) and model-query detectors (native AgentWM) need additional integration.

E.3 Attack Interface

A record-level attack transforms a completed run:

```
edit(run, rng) -> edited_run
```

These attacks modify exported records without re-executing the agent, so the original task score is unchanged; they retain the original decision indices.

F Watermarking Methods

We summarize the six methods in Table 1. Let $\mathcal{B}_t$ denote the ordered non-terminal action menu, P_t the normalized action distribution, a_t the executed action, and k the secret key. Table 4 lists the evaluated configurations. TraceWM-S and TraceWM-T are the selection and tally channels of the same TRACE configuration.

F.1 Distribution-Biasing Methods

AgentGuide. AgentGuide applies a keyed preference over high-level actions (Huang et al., 2025). A context-dependent green set $G_t \subseteq \mathcal{B}_t$ receives a log-weight increase δ:

$$\widetilde{P}_t(b) \propto P_t(b) \exp(\delta \mathbf{1}[b \in G_t]) \,. \tag{4}$$

The detector measures alignment between executed actions and the keyed preference.

RedGreen. RedGreen adapts the green-list mechanism of Kirchenbauer et al. (2023) to action identities. It uses Equation equation 4, with the preceding action as keying context, and counts green-list hits for detection. This baseline evaluates action-level rather than token-level watermarking.

SeqWM. SeqWM constructs overlapping keyed green sets from an action-history window (An et al., 2026). Their aggregate score $s_t(b)$ defines

$$\widetilde{P}_t(b) \propto \max(P_t(b), \epsilon) \exp(\delta s_t(b)/m)\,,$$

where m is the number of sets, δ controls the bias, and ϵ is a probability floor. The detector sums keyed scores over eligible windows. The floor can increase the probability of otherwise unlikely actions.

AgentWM. AgentWM marks semantically equivalent action segments through action and argument choices (Wang et al., 2026). Its native detector queries a suspect model and compares segment-level distributions with a reference. The trajectory-only adapter instead measures the signal in recorded behavior; its results are distinct from native imitation detection.

F.2 Distribution-Preserving Methods

AgentMark and TRACE's selection channel preserve the marginal action distribution under their sampling assumptions. TRACE also embeds a signal in observation counts.

AgentMark. AgentMark embeds a multi-bit identifier through distribution recombination, cyclic-shift coding, and random linear coding (Huang et al., 2026). Decoding requires executed actions, ordered menus, decision distributions, and replay context. Detection tests whether the decoded bits are consistent with the key's linear code over GF(2); payload recovery does not determine detection.

TRACE. TRACE uses independently keyed selection and tally channels (Gao et al., 2026). The selection channel derives uniforms r_b from local context and selects

$$a_t = \arg \min_{b:P_t(b)>0} \frac{-\log r_b}{P_t(b)}. \tag{5}$$

Under independent uniform draws, this rule samples from P_t. The detector sums $-\log(1 - r_{a_t})$ over n distinct evaluation points, with a $\mathrm{Gamma}(n, 1)$ null under the independence assumptions. Repeated keyed contexts cannot be counted as independent evidence.

The tally channel derives a target observation count in $\{1, 2\}$ from the trusted run identity and group position, appending a context-neutral redundant record where admissible. Under the stated PRF model, the number of matches over n admissible groups has a $\mathrm{Binomial}(n, 1/2)$ null. The signal is invariant to skeleton-preserving rewrites; this guarantee does not cover edits that change the skeleton.

G Attacks

We evaluate attacks after the original agent execution. Record-editing attacks transform the exported trajectory. They therefore affect verification evidence but not the task execution or its original utility score.

Deletion. Each decision group, including its decision and associated observations, is independently removed with probability 0.1, 0.2, 0.4, or 0.6. Surviving groups retain their original order and indices. This tests sensitivity to reduced evidence and disrupted decision sequences.

Truncation. We retain the first 75%, 50%, or 25% of decision groups and discard the remainder. This tests detection from shorter prefixes while preserving the order and indices of surviving groups.

Dedup. Identical observations associated with the same tool call are collapsed into one record. This changes observation counts while retaining distinct observation contents, testing sensitivity to count-based evidence.

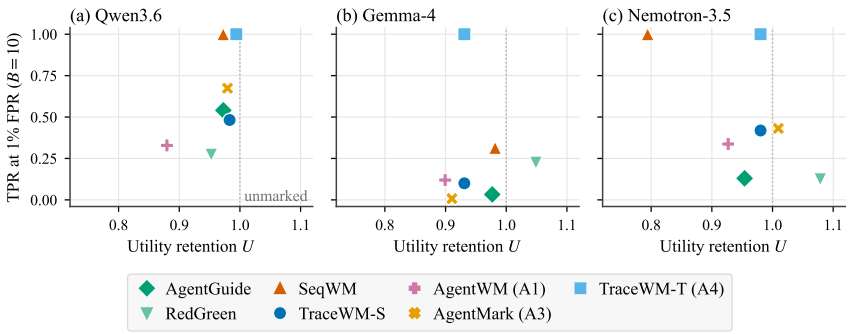

Figure 10: TPR at $B = 10$ versus utility retention for each backbone, macro-averaged across task sets. Methods use the access levels in Appendix H. Dotted lines mark unmarked utility ($U = 1$).

Equiv. For eligible calls, we randomly select between two predefined alternative tool representations and translate the arguments accordingly. Corresponding call references in the exported records are updated, while observation contents, candidate menus, and decision distributions remain unchanged. This tests sensitivity to the recorded choice of tool representation.

Canon. Every eligible call is rewritten into its shell form with translated arguments, removing the recorded choice between representations.

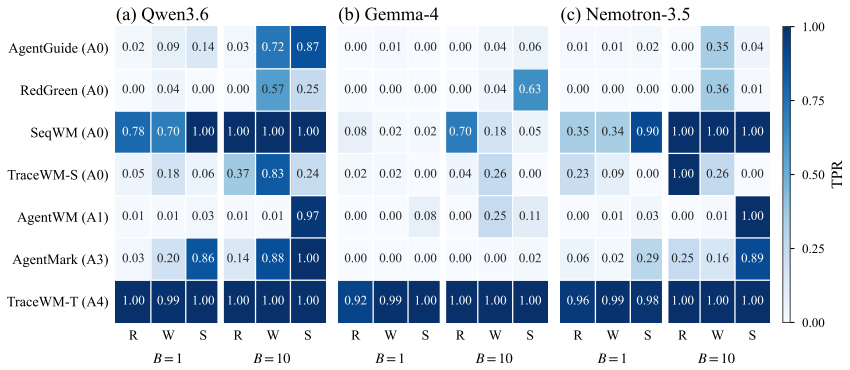

Figure 11: Task-set TPR at $B = 1$ and $B = 10$. Access levels appear beside method names. R, W, and S denote Research, Workspace, and Software.

No-op. We remove planning-only decision groups involving `think` or `task_list`, including their associated observations. The remaining groups retain their original order and indices. This tests sensitivity to the removal of planning records.

H Full Results

We report results separately for Qwen3.6, Gemma-4, and Nemotron-3.5. Detection uses a nominal 1% FPR under the calibration protocol in Section 5. Unless access levels are varied explicitly, AgentWM uses A1, AgentMark A3, TraceWM-T A4, and the remaining methods A0.

H.1 Detectability and Utility

Figure 10 compares detection and utility within each backbone. Figures 11 and 12 provide task-set breakdowns, revealing differences obscured by aggregate results.

H.2 Verifier Access

Figure 13 compares detection across A0–A4. Additional evidence enables AgentMark and TraceWM-T verification, while its effect on AgentWM varies with backbone and trajectory count.

H.3 Robustness

Figure 14 reports all methods under each record edit and strength, including those omitted from the main robustness figure.

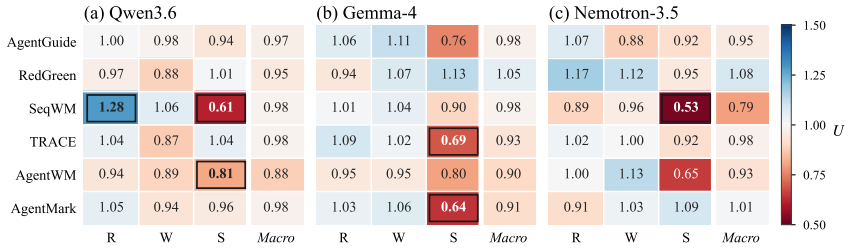

Figure 12: Utility retention by task set and macro average. Red indicates $U < 1$ and blue $U > 1$. Boxes mark values whose paired 95% confidence intervals exclude one. TraceWM-S and TraceWM-T use the same execution runs.

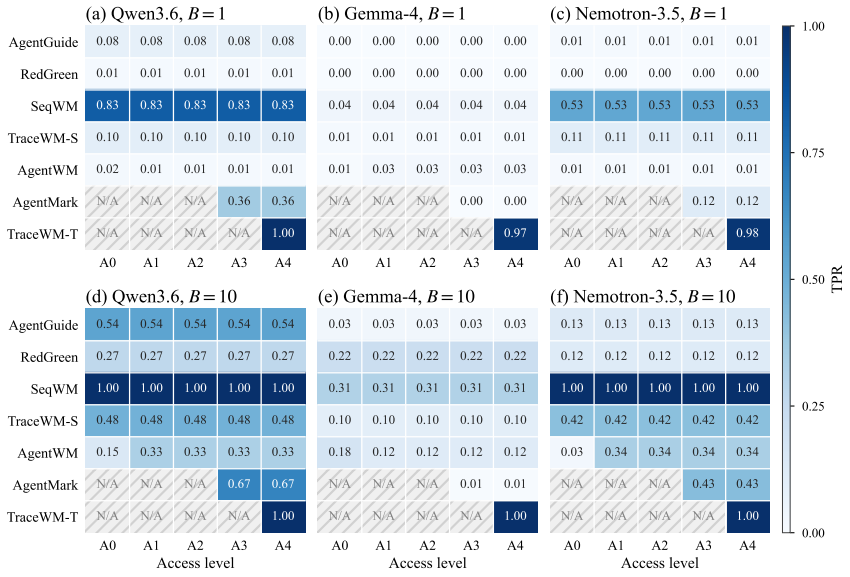

Figure 13: Macro-averaged TPR across access levels for (a–c) $B = 1$ and (d–f) $B = 10$. AgentWM additionally uses arguments at A1; AgentMark requires A3 and TraceWM-T A4. Hatched cells indicate unavailable verification due to missing required evidence.

I Leaderboard Aggregation

Metrics are computed within each task set and then macro-averaged, giving the three domains equal weight. Robustness is the mean attacked TPR at a nominal 1% FPR across the declared attacks and strengths. Token overhead is the ratio of mean input-plus-output tokens to the unmarked baseline.

The overall rank averages the Arena Score, robustness, and token-overhead ranks, with average ranks for ties. Lower overhead ranks better. Methods without the required verifier evidence receive no overall rank at that access level. Rankings hold under nominal calibration, not exactly matched FPRs. At $\beta = 0.25$ and 4, the A0 Arena Score ordering is unchanged (Table 5).

Table 5: Arena Score sensitivity to β at A0 ($B = 10$, macro-averaged over backbones).

Method	$\beta = 0.25$	$\beta = 1$	$\beta = 4$
SeqWM	0.797	0.839	0.886
TraceWM-S	0.384	0.496	0.700
AgentGuide	0.276	0.378	0.595
RedGreen	0.246	0.343	0.567
AgentWM	0.142	0.207	0.385

Each backbone's unmarked trajectories are split into calibration and evaluation halves by hash. Each fold draws 4,000 groups of B, without replacement within a group; groups may overlap.

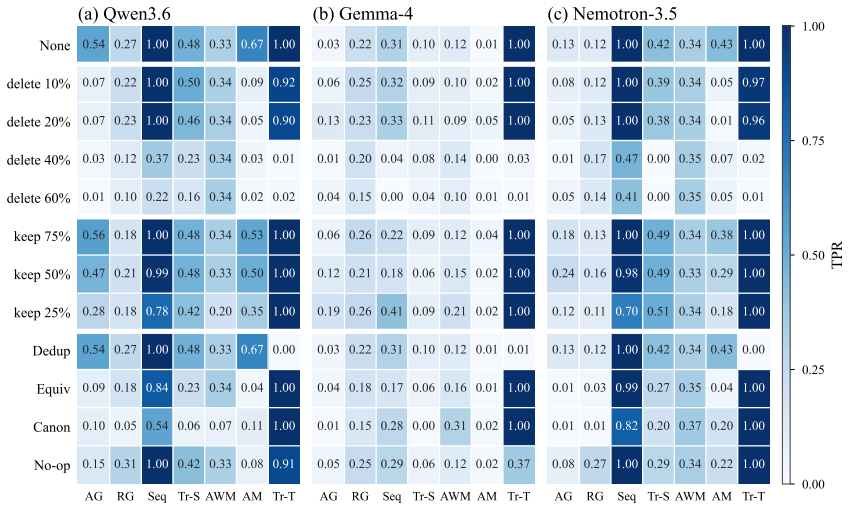

Figure 14: TPR at $B = 10$ under record edits at a nominal 1% FPR, macro-averaged across task sets. Delete and keep percentages specify deletion probability and retained prefix length, respectively. AG: AgentGuide; RG: RedGreen; Seq: SeqWM; Tr-S: TraceWM-S; AWM: AgentWM; AM: AgentMark; Tr-T: TraceWM-T.

Calibration. Thresholds are calibrated on unmarked trajectories held out from performance evaluation, under matched verifier access. Empirical wrong-key tests use $M = 1000$ null keys; analytical detectors retain their native tests where their assumptions hold. Achieved false-positive rates are evaluated on unmarked records, as described in Section 5.

J Agent Trajectories and Watermark Visualization

J.1 Why Trajectories Matter

Trajectories preserve reusable procedures beyond the final answer. This real run records script generation, failure recovery, code repair, and testing across seven tools.

Engineering trajectory: script generation, repair, and testing

Step 67 `read_file`
Input: `internal/core/software_renderer.rs`
Result: `source-file header returned.`
Step 72 `write_file`
Input: `fix_software_renderer.py`
Result: `wrote fix_software_renderer.py (450 lines).`

Engineering trajectory (continued)

Step 73 `run_code`
Input: `cd /app && python fix_software_renderer.py`
Result: `python: command not found`

Step 74 `run_code`
Input: `cd /app && python3 fix_software_renderer.py`
Result: `Done rewriting software_renderer.rs`

Step 76 `bash`
Input: `cargo test -p i-slint-core --no-run ...`
Result: `error[E0425]: cannot find value 'to_draw' in this scope;`
`error[E0308]: arguments ...`

Step 77 `grep_files`
Input: `to_draw`
Result: `software_renderer.rs:218: let to_draw =`
`self.apply_dirty_region(dirty_region, ...`

Step 84 `run_code`
Input: `generated_code: Rust repair (excerpts)`
Before:
```
bounding_box: to_draw,
```
After (selected lines):
```
let to_draw = Cell::new(PhysicalRect::default());
...
  to_draw.set(drawn);
...
  bounding_box: to_draw.get(),
```
Result: `Done fixing render method with Cell`

Step 87 `grep_files`
Input: `SoftwareRenderer<0>`
Result: `software_renderer.rs:258: /// # fn xxx...`

Step 88 `edit_file`
Input: `internal/core/software_renderer.rs`
Before (excerpt):
```
... renderer: &SoftwareRenderer<0>) {
```
After (excerpt):
```
... renderer: &SoftwareRenderer) {
```
Result: `edited internal/core/software_renderer.rs (117 chars -> 114)`

Step 89 `bash`
Input: `cargo test -p i-slint-core ...`
Result: `test ... adapters::SortModel ... ok`

Step 90 `task_list`
Input: `["Target 1: PlatformError Type", ...]`
Result: `[x] Target 1: PlatformError Type;`
`[x] Target 2: Clipboard Enum;`
`[x] Target 3: RepaintBuffe...`

Figure 15: **A multi-tool engineering trajectory.** Selected steps from an unmarked RoadmapBench run (`slt-1.0.0-roadmap`; 104 decisions; score 1.00). Excerpts are abbreviated; step numbers indicate omissions.

J.2 WATERMARK VISUALIZATION

Each method uses a separate run of the same task. Arrows show proposed-to-executed substitutions within a step. Input and Result refer to the executed action; unchanged actions may also provide verification evidence.

Proposed action → Executed action

(a) AgentMark

Step 02 `read_file` → `task_list`
Input: `["Target 1: PlatformError Type", ...]`
Result: `task checklist returned.`

Step 21 `read_file` → `grep_files`
Input: `PointerEventButton`
Result: `no matching line.`

Step 35 `bash` → `list_directory`
Input: `internal/backends/testing`
Result: `testing backend files listed.`

(b) AgentGuide

Step 13 `read_file` → `grep_files`
Input: `PointerEventButton`
Result: `matching references in backend source files.`

Step 17 `bash` → `read_file`
Input: `internal/core/Cargo.toml`
Result: `file header returned.`

(c) RedGreen

Step 15 `list_directory` → `grep_files`
Input: `PointerEventButton`
Result: `matching references in backend source files.`

Step 16 `grep_files` → `read_file`
Input: `internal/core/items.rs`
Result: `source-file header returned.`

(d) SeqWM

Step 05 `read_file` → `task_list`
Input: `["Target 1: PlatformError Type", ...]`
Result: `task checklist returned.`

Step 07 `read_file` → `fetch_url`
Input: `internal/core/software_renderer.rs`
Result: `"Give a full http(s) URL, as search_web returns."`

Figure 16: **Watermark substitutions in agent trajectories.** Orange and green denote proposed and executed actions, respectively. Arguments and observations are abbreviated; step indices are local to each run.
In (d), SeqWM replaces `read_file` with `fetch_url`; the regenerated argument is still a file path, so the call fails and wastes a step, a cost that detection alone does not show.