AgentMark: Utility-Preserving Behavioral Watermarking for Agents

Kaibo Huang[1*] **Jin Tan**[1*] **Yukun Wei**[1] **Wanling Li**[1] **Zipei Zhang**[1]
Hui Tian[2] **Zhongliang Yang**[1†] **Linna Zhou**[1]

[1]Beijing University of Posts and Telecommunications
[2]Huaqiao University
{huangkaibo, tanjinanin, weiyukun, lwl, nebulazhang}@bupt.edu.cn
htian@hqu.edu.cn, {yangzl, zhoulinna}@bupt.edu.cn

Abstract

LLM-based agents are increasingly deployed to autonomously solve complex tasks, raising urgent needs for IP protection and regulatory provenance. While content watermarking effectively attributes LLM-generated outputs, it fails to directly identify the high-level planning behaviors (e.g., tool and subgoal choices) that govern multi-step execution. Critically, watermarking at the planning-behavior layer faces unique challenges: minor distributional deviations in decision-making can compound during long-term agent operation, degrading utility, and many agents operate as black boxes that are difficult to intervene in directly. To bridge this gap, we propose AgentMark, a behavioral watermarking framework that embeds multi-bit identifiers into planning decisions while preserving utility. It operates by eliciting an explicit behavior distribution from the agent and applying distribution-preserving conditional sampling, enabling deployment under black-box APIs while remaining compatible with action-layer content watermarking. Experiments across embodied, tool-use, and social environments demonstrate practical multi-bit capacity, robust recovery from partial logs, and utility preservation. Code is available at https://github.com/Tooooa/AgentMark.

1 Introduction

Recent advances in large language models (LLMs) have improved text generation and reasoning (Xu et al., 2025; Achiam et al., 2023; Team et al., 2023) and accelerated the shift from passive language interfaces to autonomous agents that can perceive context, plan, and execute actions (Shajarian et al., 2024; Acharya et al., 2025; Yao et al., 2022). This autonomy and proactiveness are increasingly reflected in real-world applications, including GUI-based daily assistance (Liu et al., 2024b), tool-augmented multimodal agents for financial trading (Zhang et al., 2024), and large-scale social agents operating in online ecosystems (Gao et al., 2024).

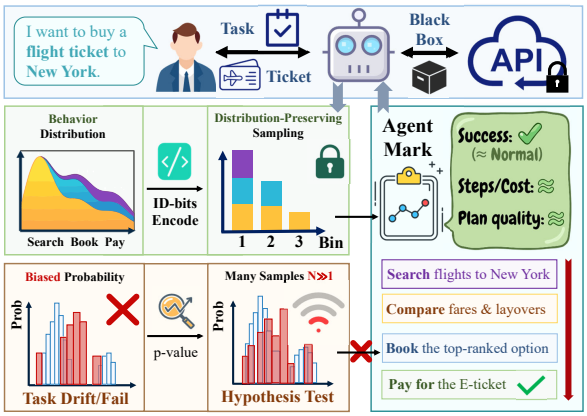

Figure 1: AgentMark embeds multi-bit provenance in planning behaviors via distribution-preserving sampling under black-box agent APIs, while preserving utility; bias-based probability watermarking can drift and harm task performance.

The increasing deployment of intelligent agents raises urgent concerns about traceability and accountability in regulated settings. In security-sensitive and high-impact domains, agents can be exploited for impersonation, automated disinformation, and large-scale manipulation, while enterprises face risks of misuse or unauthorized replication of proprietary agent systems (Gao et al., 2023; Wang et al., 2025a; Park et al., 2023; Piao et al., 2025; Mou et al., 2024). These challenges motivate provenance mechanisms that attribute agent activities and support auditing and enforcement.

To reason about provenance in agentic systems, we distinguish two levels of decision-making. **Planning behavior** refers to the high-level choice of *what* to do next (e.g., selecting a tool or committing to a subgoal), whereas **execution action** specifies *how* that choice is carried out (e.g., tool arguments or structured outputs). This behavior–action decomposition is widely used in embodied tasks and tool-use pipelines (Yao et al., 2022; Sha-

[*] Equal contribution
[†] Corresponding author

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12581–12603
July 2-7, 2026

jarian et al., 2024). For simplicity, we use **behavior** and **action** as shorthand for **planning behavior** and **execution action** in the rest of the paper.

Content watermarking has shown strong practicality for attributing LLM-generated outputs, and recent systems such as SynthID-Text have been reported in production settings (e.g., Google Gemini) (Dathathri et al., 2024). However, agent provenance concerns not only the final content but also the agent's behavioral decisions over time, which shape downstream impact and long-horizon outcomes, making behavioral watermarking a necessary complement to content watermarking. Directly extending existing content watermarking techniques to agent behaviors remains challenging. Training-time schemes that modify model weights are often infeasible because many agents rely on closed pre-trained APIs, and retraining general-purpose models is costly and hard to tailor to diverse environments (Lau et al., 2024; Patil et al., 2023; Peng et al., 2023). Inference-time schemes that manipulate token-level sampling (Kirchenbauer et al., 2023; Dathathri et al., 2024; Guan et al., 2024; Hudeček and Dusek, 2023) also do not align well with high-level behaviors, since behaviors are not token-native and crucial semantics can be lost when a planning decision is compiled into structured executions (e.g., tool calls and arguments). For example, an LLM output such as "Alice bookmarked a post with the tag `#TravelInspiration`" may be reduced to discrete actions like "bookmarking" and "tagging," stripping tokens that are useful for provenance, which makes watermark extraction and verification more difficult. Moreover, just watermarking behaviors by directly biasing the behavior probabilities (Huang et al., 2025b) can introduce compounding distribution shifts over long horizons, leading to task drift or failure. These gaps motivate robust watermarking methods that operate at the behavior level while preserving task utility.

As illustrated in Figure 1, we address this gap by introducing AgentMark, a behavioral watermarking framework that embeds multi-bit identifiers into an agent's planning process with utility preservation. Viewing planning as sampling from a time-varying behavior distribution, where the agent would otherwise make an implicit behavior choice, AgentMark first elicits an explicit probability list over candidate behaviors and then embeds a watermark by distribution-preserving sampling on this elicited distribution, keeping the induced behavior distribution unchanged. To withstand agent-specific failures such as step erasure and truncation (e.g., platform filtering and logging loss), we use context-reproducible randomness and erasure-resilient coding, enabling recovery from partial trajectories. Our contributions are as follows:

- We propose a planning-level framework for agent behavioral watermarking, where we elicit behavior distributions and encode multi-bit IDs via distribution-preserving sampling, without changing model weights or token-level sampling.
- We implement this framework as AgentMark-F, using context-reproducible randomness and erasure-resilient decoding to recover IDs from partial trajectories.
- We evaluate AgentMark in embodied, tool-use, and social environments, showing practical multi-bit capacity with utility preservation, robustness to erasure or truncation and semantic-preserving observation rewriting, and compatibility with action-layer content watermarking.

2 Related Work

2.1 Content Watermarking for LLMs

Content watermarking attributes LLM-generated text by embedding detectable signals during generation, supported by standardized toolchains such as MarkLLM (Pan et al., 2024). Prior work spans model-level watermarking (e.g., Zhu et al., 2024) and decoding-time schemes, including statistical logit/sampling signals (e.g., Kirchenbauer et al., 2023), distribution-preserving designs (e.g., Dathathri et al., 2024; Chen et al., 2025), semantic robustness to rewriting/translation (e.g., Liu et al., 2024a; He et al., 2024; Hou et al., 2024), and multi-bit identification (e.g., Lee et al., 2024). While highly effective for attributing free-form text, content watermarking operates on the action/content layer and does not provide direct provenance for planning-time behaviors in agent pipelines, motivating complementary watermarking mechanisms at the behavior layer.

2.2 Agent Security

Autonomous agents deployed in open ecosystems introduce security and governance risks, including impersonation, automated manipulation, and tool-enabled abuse (He et al., 2025; Deng et al., 2025),

as well as unauthorized replication of proprietary agent systems (Triedman et al., 2025). The emerging Internet of Agents (IoA) (Wang et al., 2025b) further raises a potential risk of covert harmful communication across agent interactions (Huang et al., 2025a). But, strengthening agent safety often incurs a non-trivial *security tax* in capability, usability, or cost (Yang et al., 2024a), complicating real-world adoption. These challenges motivate provenance mechanisms that enable reliable attribution and auditing while preserving agent utility.

3 Problem Formulation

3.1 Preliminaries and Notation

We consider an agent interacting with an environment over a trajectory of length T. At each time step $t \in \{1, \dots, T\}$, the agent observes o_t and is given a finite set of admissible planning behaviors $\mathcal{B}_t$. The agent selects a planning behavior $b_t \in \mathcal{B}_t$ and then executes an *execution action* a_t that specifies how b_t is carried out.

We assume the agent induces a latent planning-time behavior distribution $P_t^\star$ over $\mathcal{B}_t$ and that the baseline makes a behavior choice accordingly. In our setting, we elicit an explicit estimate P_t of $P_t^\star$ and define distribution preservation with respect to P_t. Our watermarking operates at the *behavior* level and does not directly manipulate the execution action a_t. To avoid shifting the planning policy, we require per-step distribution preservation:

$$\hat{b}_t \sim P_t \quad \text{for all } t \in \{1, \dots, T\}. \tag{1}$$

3.2 Planning-Time Behavior Channel and Distribution-Preserving Coding

We model the agent's planning stage as sampling from a time-varying discrete channel. At each time step t, the channel is characterized by an implicit behavior distribution $P_t^\star$ over the behavior set $\mathcal{B}_t$. The unwatermarked agent makes an implicit behavior choice $b_t \sim P_t^\star$. In our setting, we elicit an explicit estimate of $P_t^\star$ as a probability list P_t over candidate behaviors. Our distribution-preserving guarantee is defined with respect to the elicited distribution P_t. To embed a provenance payload $m \in \{0,1\}^L$, a watermarked agent replaces random sampling with a distribution-preserving encoder and outputs

$$\hat{b}_t \leftarrow \mathsf{Enc}(P_t, r_t) \in \mathcal{B}_t, \tag{2}$$

where $r_t \in (0, 1]$ denotes shared sampling randomness. We assume the embedder and verifier share a secret key K_{sh} so that the verifier can reproduce the same per-step randomness from logged context when decoding.

The encoder consumes bits from m across steps depending on P_t. Given the logged behavior sequence and the corresponding channel information, the verifier recovers the payload via

$$\hat{m} \leftarrow \mathsf{Dec}\big(\{(\hat{b}_t, P_t)\}_{t=1}^{T}\big). \tag{3}$$

3.3 Threat Model

We consider provenance verification in agentic workflows, where a user specifies a high-level goal, and the agent autonomously executes a multi-step plan with limited human intervention (Wiesinger et al., 2024). In such pipelines, the verifier may not receive a complete record of planning-time decisions. Intermediate steps can be missing due to incomplete logging, execution failures, or platform-side deletion, leading to step erasure and trajectory truncation.

We model this as an incomplete observation of a length-T trajectory, where $\mathcal{I} \subseteq \{1, \dots, T\}$ denotes the set of observed time indices. The verifier observes only $\{(\hat{b}_t, P_t)\}_{t \in \mathcal{I}}$, while steps in $\{1, \dots, T\} \setminus \mathcal{I}$ are missing. We parameterize missingness by an erasure rate $\rho \in [0, 1)$, with $|\mathcal{I}| \approx (1-\rho)T$ (or $|\mathcal{I}| \geq (1-\rho)T$ in a worst-case view). Truncation corresponds to observing only a prefix, i.e., $\mathcal{I} = \{1, \dots, \tau\}$ for some $\tau < T$ (more generally, a contiguous segment may be observed).

Post-hoc modifications to *execution actions* (e.g., editing tool arguments) may occur in practice. Since our watermark is embedded in behaviors rather than action surface forms, we treat the main effect of such interventions as missing or shortened planning-time records, and we formalize robustness under erasure and truncation accordingly.

3.4 Objectives

We formalize two objectives: utility preservation and robust decodability. Our design enforces per-step distribution preservation as a constraint to support utility preservation.

Utility preservation. We require that watermarking does not degrade task execution quality or efficiency, measured by task success and trajectory length:

$$\begin{cases} \Big|\mathbb{E}[\mathrm{Succ}(\hat{\tau})] - \mathbb{E}[\mathrm{Succ}(\tau)]\Big| \leq \varepsilon_{\mathrm{succ}} \\ \Big|\mathbb{E}[\mathrm{Len}(\hat{\tau})] - \mathbb{E}[\mathrm{Len}(\tau)]\Big| \leq \varepsilon_{\mathrm{len}} \end{cases}, \tag{4}$$

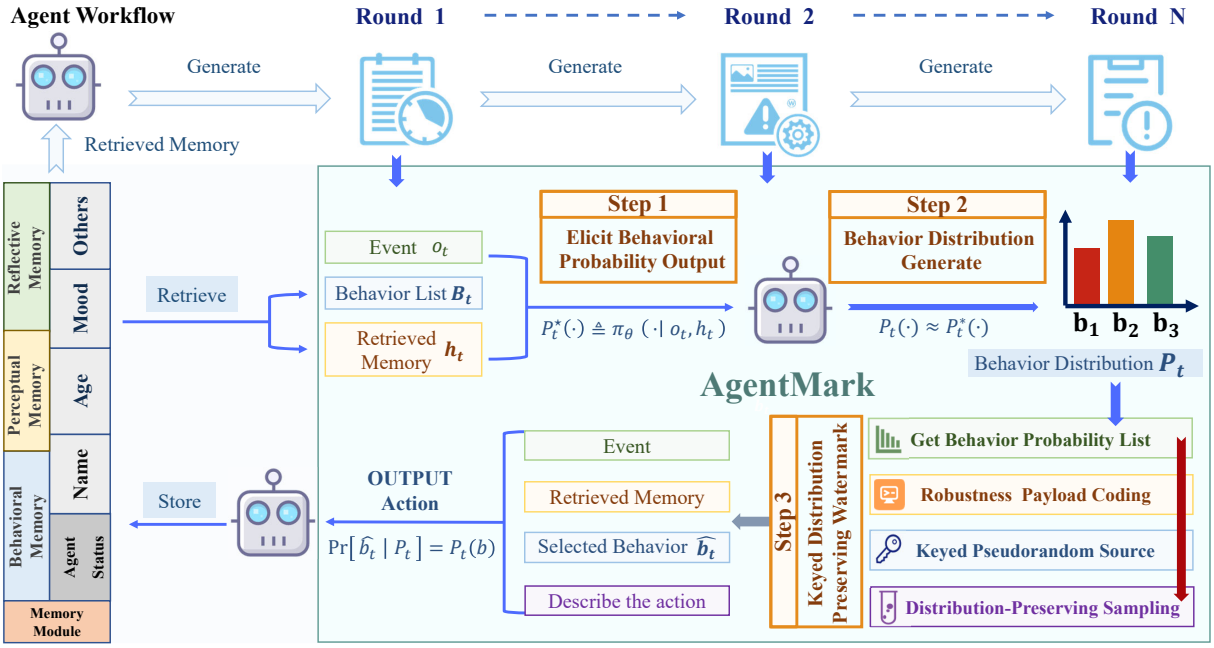

Figure 2: AgentMark overview. At each round, the agent would otherwise make an implicit planning-behavior choice according to a latent policy $P_t^\star$ over a finite behavior set $\mathcal{B}_t$. AgentMark makes this decision process auditable by eliciting an explicit probability list $P_t(\cdot) \approx P_t^\star(\cdot)$ over $\mathcal{B}_t$, and then applies distribution-preserving watermark sampling on P_t to select a planning behavior $\hat{b}_t$ while keeping its marginal distribution matched to P_t. The execution action is generated conditioned on $\hat{b}_t$ and executed in the environment.

where τ and $\hat{\tau}$ denote the baseline and watermarked trajectories induced by the baseline agent's implicit behavior choices b_t and the watermarked choices $\hat{b}_t$, respectively; $\mathrm{Succ}(\cdot) \in \{0,1\}$ indicates whether the task is completed successfully, and $\mathrm{Len}(\cdot)$ is the number of decision steps (a proxy for time or compute overhead).

Robust decodability. Under step erasure or truncation, the verifier observes indices $\mathcal{I} \subseteq \{1,\dots,T\}$ and receives $\{(\hat{b}_t, P_t)\}_{t\in\mathcal{I}}$. For a target erasure rate ρ, we require successful recovery from partial trajectories whenever $|\mathcal{I}| \geq (1-\rho)T$:

$$\Pr\Big[\mathsf{Dec}\big(\{(\hat{b}_t, P_t)\}_{t\in\mathcal{I}}\big) = m\Big] \geq 1-\delta, \qquad (5)$$

where $m \in \{0,1\}^L$ is the payload and δ is the allowed decoding failure probability.

4 AgentMark

4.1 Overview

Figure 2 illustrates AgentMark in an agent workflow. A key design requirement in agentic settings is that watermarking should not reduce task performance; accordingly, AgentMark operates on the planning layer and preserves the elicited planning distribution to avoid compounding errors over long horizons. At each round, an LLM agent would otherwise make an implicit planning-behavior choice according to a latent policy π_θ over a finite candidate set $\mathcal{B}_t$. AgentMark makes this decision process auditable by eliciting an explicit probability list P_t over $\mathcal{B}_t$, and then watermarking only through how the planning behavior is sampled from P_t. Formally, we denote the LLM-induced implicit planning policy by

$$P_t^\star(\cdot) \triangleq \pi_\theta(\cdot \mid o_t, h_t) \in \Delta(\mathcal{B}_t), \qquad (6)$$

which is typically latent in standard agent implementations.

AgentMark makes this policy auditable by eliciting an explicit estimate as a probability list over the same candidate set:

$$P_t(\cdot) \approx P_t^\star(\cdot) \in \Delta(\mathcal{B}_t). \qquad (7)$$

This explicit access enables distribution-preserving sampling and verification.

Behavior–action separation. AgentMark outputs a watermarked planning behavior $\hat{b}_t$, after which the execution action is generated conditioned on $\hat{b}_t$ and executed in the environment. Rather than manipulating action, AgentMark intervenes only

in the planning-stage sampling procedure by using distribution-preserving sampling on the elicited behavior distribution. In particular, it preserves the planning-time behavior distribution, i.e., the marginal of $\hat{b}_t$ matches the original elicited distribution P_t, so watermarking does not materially shift the planning distribution used at inference time or alter the agent's decision logic.

Robustness interface. To improve robustness under missing steps, AgentMark optionally applies payload coding before sampling and supports erasure-resilient recovery from partial trajectories.

4.2 Distribution-Preserving Behavioral Watermark

At each step t, given an explicit behavior distribution P_t over $\mathcal{B}_t$ and a payload bitstream M, the encoder selects a watermarked behavior while preserving the marginal distribution:

$$\begin{cases} \hat{b}_t \leftarrow \mathsf{Enc}(P_t, M, r_t) \in \mathcal{B}_t \\ \Pr[\hat{b}_t = b] = P_t(b), \forall b \in \mathcal{B}_t \end{cases}, \tag{8}$$

where r_t denotes shared sampling randomness. The decoder inverts the same sampling process to extract the embedded bits from an observed behavior sequence.

Keyed pseudorandomness and per-step independence. Following the setting in Section 3.2, we derive per-step randomness from a shared secret K_{sh} and step context $Context_t$ to synchronize encoding and decoding, where $Context_t$ summarizes the information shared by both sides at time t (e.g., the step index t, the observation o_t, and the agent history/context h_t). We derive

$$K_t \leftarrow H(K_{\text{sh}} \,\|\, Context_t), \tag{9}$$

and instantiate a pseudorandom generator seeded by K_t to produce the randomness stream r_t used by the sampler.

4.3 AgentMark-F: A Concrete Instantiation of AgentMark.

AgentMark is a general paradigm that embeds provenance bits into planning behaviors via distribution-preserving conditional sampling on an explicit behavior distribution P_t. Several distribution-preserving sampling schemes for conditional distributions have been developed (e.g., Meteor Kaptchuk et al., 2021 and Discop Ding et al., 2023). Liao et al. (2025) further proposes FDPSS, a modular framework with a security proof. Following this line, we instantiate AgentMark with an FDPSS-style construction, termed **AgentMark-F**. Concretely, AgentMark-F transforms P_t into a mixture of uniform bins via probability recombination, samples a bin using keyed pseudorandomness, and applies cyclic-shift uniform encoding within the selected bin to embed a variable-length bitstring.

Differential recombination and distribution preservation. At each step t, let $\mathcal{B}_t = \{b_{t,1}, \ldots, b_{t,n}\}$ be the candidate behavior set. We first form a canonical ordering of $\mathcal{B}_t$ by sorting behaviors by their probabilities in non-increasing order, yielding $p_1 \geq \cdots \geq p_n$ where $p_i = P_t(b_{t,i})$. Differential-based recombination decomposes the sorted probability sequence via first-order differences, defining slice heights $d_k \triangleq p_k - p_{k+1}$ with $p_{n+1} = 0$ (for the full proof and details of distribution preservation, see Appendix A). Differential recombination constructs a mixture of uniform bins in which bin k contains the top-k behaviors and has mixture weight $q_k = k \cdot d_k$. Given index $K \sim \mathrm{Cat}(q_1, \ldots, q_n)$, the encoder samples uniformly within bin K while embedding bits. This preserves the marginal: for any i,

$$\begin{aligned} \Pr[\hat{b}_t = b_{t,i}] &= \sum_{k=i}^{n} q_k \cdot \frac{1}{k} \\ &= \sum_{k=i}^{n} (p_k - p_{k+1}) \\ &= p_i. \end{aligned} \tag{10}$$

Encoder and Decoder. AgentMark-F uses keyed pseudorandomness to synchronize bin sampling and uniform encoding between the encoder and decoder. At step t, both sides derive a per-step seed from the shared secret and context, sample the same bin according to $\{q_k\}$, and apply CyclicShift within the bin to embed/extract a variable-length bitstring. For a bin T of size $n = |T|$, CyclicShift embeds $c_t \in \{\lfloor \log_2 n \rfloor, \lfloor \log_2 n \rfloor + 1\}$ bits, with near-optimal expected capacity $\mathbb{E}[c_t \mid |T| = n] \in [\log_2 n - 0.0861, \log_2 n]$. Across $t = 1, \ldots, T$, the encoder repeats Algorithm 1 with a payload pointer ℓ, producing substrings $\{s_t\}$ whose lengths depend on $|T|$. Throughout, we embed information while preserving the marginal behavior:

$$\Pr[\hat{b}_t = b \mid P_t] = P_t(b), \quad \forall b \in \mathcal{B}_t. \tag{11}$$

Algorithm 1 AgentMark-F (one step): ENCODE

Require: P_t over $\mathcal{B}_t$, step context $Context_t$, shared secret K_{sh}, payload bitstream M, pointer ℓ

Ensure: $\hat{b}_t$, embedded bits s_t, updated pointer ℓ

$K_t \leftarrow H(K_{\text{sh}} \,\|\, Context_t)$; $\mathsf{PRG} \leftarrow \mathsf{PRG}(K_t)$

$\{(T_k, q_k)\}_{k=1}^{n} \leftarrow \mathsf{DiffRecombine}(P_t)$

$K \leftarrow \mathsf{SampleCat}(\{q_k\}_{k=1}^{n}, \mathsf{PRG})$; $T \leftarrow T_K$

$(j, s_t) \leftarrow \mathsf{CyclicShiftEnc}(M[\ell:], |T|, \mathsf{PRG})$

$\hat{b}_t \leftarrow T[j]$; $\ell \leftarrow \ell + |s_t|$

return $\hat{b}_t, s_t, \ell$

Algorithm 2 AgentMark-F (one step): DECODE

Require: $\hat{b}_t$, P_t over $\mathcal{B}_t$, step context $Context_t$, shared secret K_{sh}

Ensure: extracted bitstring s_t (possibly empty)

$K_t \leftarrow H(K_{\text{sh}} \,\|\, Context_t)$; $\mathsf{PRG} \leftarrow \mathsf{PRG}(K_t)$

$\{(T_k, q_k)\}_{k=1}^{n} \leftarrow \mathsf{DiffRecombine}(P_t)$

$K \leftarrow \mathsf{SampleCat}(\{q_k\}_{k=1}^{n}, \mathsf{PRG})$; $T \leftarrow T_K$

$j \leftarrow \mathsf{IndexOf}(\hat{b}_t \text{ in } T)$

$s_t \leftarrow \mathsf{CyclicShiftDec}(j, |T|, \mathsf{PRG})$

return s_t

Details are in Appendix B (example: Appendix C); the full distribution-preservation proof is deferred to Appendix D.

4.4 Erasure-Resilient Coding and Decoding

Variable-capacity bitstream. At step t, the sampler outputs $s_t \in \{0,1\}^{c_t}$ with

$$c_t \triangleq |s_t| \in \Big\{ \lfloor \log_2 |T| \rfloor,\ \lfloor \log_2 |T| \rfloor + 1 \Big\}, \quad (12)$$

where $|T|$ is the selected bin size (Appendix B). Under step erasure/truncation, the verifier observes indices $\mathcal{I} \subseteq \{1, \ldots, T\}$ and receives a total of $R \triangleq \sum_{t \in \mathcal{I}} c_t$ embedded bits.

RLNC (Ho et al., 2006) over $\mathbb{F}_2$. Let the provenance payload be $m \in \mathbb{F}_2^L$. At step t, we derive

$$K_t \leftarrow H(K_{\text{sh}} \,\|\, Context_t), \quad (13)$$

and deterministically generate coefficient vectors

$$a_{t,j} \leftarrow \mathsf{PRG}(K_t, j) \in \mathbb{F}_2^L, j = 1, \ldots, c_t. \quad (14)$$

We pseudorandomize the RLNC coefficients and embedded bits using K_t; see Appendix D for the security argument and Appendix E for additional mathematical details. Each embedded bit is a linear equation

$$y_{t,j} \triangleq \langle a_{t,j}, m \rangle \in \mathbb{F}_2, \quad (15)$$

and $s_t = (y_{t,1}, \ldots, y_{t,c_t})$.

Decoding under erasure/truncation. From $\mathcal{I}$, the verifier reconstructs $\{a_{t,j}\}$ and forms

$$A_{\mathcal{I}}\, m = y_{\mathcal{I}} \quad \text{over } \mathbb{F}_2, \quad (16)$$

where $A_{\mathcal{I}} \in \mathbb{F}_2^{R \times L}$ stacks $\{a_{t,j}\}_{t \in \mathcal{I},\, 1 \le j \le c_t}$ as rows and $y_{\mathcal{I}} \in \mathbb{F}_2^R$ stacks the corresponding $\{y_{t,j}\}$. Unique recovery holds iff $\mathrm{rank}(A_{\mathcal{I}}) = L$. When rows are pseudorandom in $\mathbb{F}_2^L$, for $R = r \ge L$ we have the rank bound.

$$\Pr[\mathrm{rank}(A_{\mathcal{I}}) = L \mid R = r] \ge 1 - 2^{-(r-L)}. \quad (17)$$

Therefore, the decoding success probability admits the lower bound

$$\Pr[\hat{m} = m] \ge \sum_{r=L}^{\infty} \Pr[R = r] \cdot \big(1 - 2^{-(r-L)}\big), \quad (18)$$

which depends on the observed set $\mathcal{I}$ only through $R = \sum_{t \in \mathcal{I}} c_t$, matching our variable capacity.

5 Experiment

5.1 Experimental Setup

Environments and tasks. We evaluate AgentMark in three deployment-relevant settings that cover embodied planning in ALFWorld (Shridhar et al., 2020b) (ID/OOD splits; OOD is a cross-distribution test set), tool-use decision making on ToolBench (Qin et al., 2023) (six test subsets spanning single- and multi-tool regimes), and social simulation in OASIS (Yang et al., 2024b) on both Twitter-like and Reddit-like platforms; further dataset details are provided in Appendix F.

Compared methods. **Baseline** follows the ReAct agent loop and directly outputs a planning behavior. **Red–green watermarking (RG)** (Huang et al., 2025b) is a bias-based baseline. At each step, we deterministically partition $\mathcal{B}_t$ into a green set of size $\gamma|\mathcal{B}_t|$ using a keyed PRG ($\gamma = 0.5$). We add a logit bias δ to green behaviors before sampling ($\delta = 2.0$). We detect the watermark by testing whether selected behaviors fall into the green set more often than the γ baseline. **AgentMark-F (Ours)** samples $\hat{b}_t$ via distribution-preserving watermark sampling on P_t. Both AgentMark-F and RG use the same keyed-PRG design to derive reproducible per-step randomness.

Setting	Task	SR (%) ↑			Steps ↓			Watermark			
		Base	RG	Ours	Base	RG	Ours	bps ↑	bpt ↑	Δs/step ↓	ΔTok/step (%) ↓
ALFWorld ID	A1	97.1±16.7	96.2±19.2	97.1±16.7	12.8±10.8	13.8±10.6	11.5±9.6	1.19	14.2	+0.44	−0.20%
	A2	98.9±10.3	100.0±0.0	100.0±0.0	10.2±7.0	11.0±6.8	11.0±7.7	1.06	14.0	+0.29	+0.25%
	A3	87.3±33.3	94.4±22.9	87.3±33.3	20.1±14.8	16.4±11.4	18.6±15.0	1.23	30.7	+0.05	+0.12%
	A4	89.7±30.3	82.0±38.4	91.0±28.6	18.2±13.5	20.5±15.4	17.2±13.2	1.17	26.4	−0.14	+0.42%
	A5	94.4±22.9	93.8±24.2	88.9±31.4	17.2±11.9	16.2±9.9	19.4±13.4	1.37	34.3	+0.26	−0.08%
	A6	78.5±41.1	60.6±48.9	77.6±41.7	31.9±14.8	39.5±12.9	31.3±15.6	1.28	37.1	−0.13	+0.19%
	Avg.	**89.5±30.6**	**78.8**(↓ 10.7)	**89.3**(↓ 0.2)	**19.7±14.9**	**26.1**(↑ 6.4)	**19.4**(↓ 0.3)	**1.19**	**25.5**	**+0.10**	**+0.20%**
ALFWorld OOD	A1	99.4±7.6	99.4±7.6	99.4±7.6	14.5±6.6	11.4±8.1	11.0±6.9	1.38	30.3	+0.07	−0.76%
	A2	100.0±0.0	93.8±24.2	97.9±14.3	12.2±6.9	15.1±12.1	13.6±9.4	1.39	25.5	+0.15	−0.22%
	A3	91.9±27.2	90.3±29.6	96.8±17.7	18.5±12.7	17.0±12.7	15.2±10.0	1.29	24.5	−0.18	+0.09%
	A4	97.6±15.2	90.5±29.4	97.6±15.2	16.2±10.0	17.1±12.3	14.3±9.1	1.18	22.7	−1.01	−0.11%
	A5	91.3±28.2	89.1±31.1	91.3±28.2	16.4±12.6	19.3±14.7	17.1±12.5	1.44	33.4	+0.02	−0.37%
	A6	94.1±23.5	91.2±28.4	97.1±16.9	22.4±11.1	25.4±13.3	24.1±11.8	1.41	38.2	−0.18	−0.88%
	Avg.	**96.8±17.7**	**94.5**(↓ 2.3)	**97.5**(↑ 0.7)	**15.9±9.7**	**15.4**(↓ 0.5)	**14.1**(↓ 1.8)	**1.34**	**28.4**	**-0.18**	**-0.30%**
ToolBench	T1	61.7±2.4	60.0±4.1	60.0±7.1	5.2±5.1	5.4±4.9	5.6±4.7	0.51	5.28	−1.67	+9.64%
	T2	58.3±6.2	61.7±10.3	61.7±2.4	7.1±5.3	8.7±5.4	8.6±4.8	0.48	4.89	−1.41	−16.96%
	T3	66.7±9.4	60.0±4.1	60.0±4.1	4.7±4.0	4.8±4.1	4.8±4.6	0.46	4.62	−2.40	−3.29%
	T4	73.3±6.2	66.7±4.7	71.7±4.7	5.7±3.8	6.3±4.4	6.5±4.3	0.49	5.10	−1.54	+10.97%
	T5	49.3±7.4	49.3±4.2	50.0±3.5	8.1±5.6	7.7±5.4	7.8±5.6	0.48	4.85	−0.83	+5.24%
	T6	50.0±4.1	53.3±2.4	55.0±7.1	9.2±5.4	9.3±5.7	9.6±6.1	0.49	4.90	+0.24	−14.77%
	Avg.	**59.9±5.8**	**58.5**(↓ 1.4)	**59.7**(↓ 0.2)	**6.7±4.9**	**7.0**(↑ 0.3)	**7.2**(↑ 0.5)	**0.49**	**4.93**	**-1.27**	**-6.25%**

Table 1: ALFWorld and ToolBench results. We report success rate (SR; fraction of tasks solved) and Steps (average decision steps over successful episodes). For Ours, we report watermark capacity bps/bpt (bits/step and bits/task), per-step latency difference Δs/step (Ours–Base), and token/step change rate $\Delta\text{Tok/step} = (\text{Tok/step}_{\text{Ours}} - \text{Tok/step}_{\text{Base}})/\text{Tok/step}_{\text{Base}}$. ALFWorld A1–A6 map to Look At Obj In Light, Pick And Place Simple, Pick Clean Then Place, Pick Cool Then Place, Pick Heat Then Place, and Pick Two Obj And Place; ToolBench T1–T6 map to single-tool selection, single-tool instruction-following, single-tool within-category selection, multi-tool within-category selection, multi-tool instruction-following, and complex multi-tool instruction-following. Appendix G.1.4 provides a complementary step-level capacity–entropy analysis on archived elicitation logs.

5.2 Utility and Capacity

Embodied and tool-use benchmarks. From the results in Table 1, we can get the following conclusion. Firstly, RG degrades utility on long-horizon embodied tasks (ALFWorld-ID SR 89.5% → 78.8%; steps 19.7 → 26.1). Secondly, AgentMark-F preserves utility across embodied and tool-use settings (close to baseline on ALFWorld ID/OOD and ToolBench) while achieving non-trivial capacity (ALFWorld ∼1.2–1.3 bps; ToolBench ∼0.49 bps, 4.93 bpt). Finally, the per-step overhead is small in both latency and token cost: Δs/step is near zero and ΔTok/step stays within ±0.5% on ALFWorld, while ToolBench shows no systematic increase and is often lower due to earlier termination (Avg. ΔTok/step = −6.25%). Additional setup details are provided in Appendix G.1. Appendix G.1.1 first checks that the elicitation interface itself does not materially change task success. We further validate these findings on Gemini 2.0 Flash in Appendix G.1.3, confirming model-agnostic utility preservation. Complementary step-level capacity–entropy analysis is provided in Appendix G.1.4.

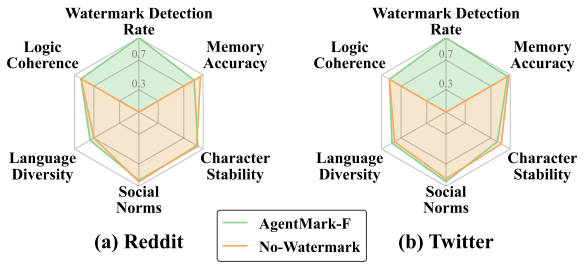

Figure 3: OASIS social-quality utility and detectability.

Social simulation. We evaluate deployment feasibility in OASIS on Reddit- and Twitter-like platforms, running 100 trajectories per platform with ours and a no-watermark control split 50/50. We score social-quality utility with an LLM judge on five dimensions (coherence, memory accuracy, character stability, social norms/common sense, and language diversity) and report Watermark Detection Rate as decoding success from behaviors. We simulate both platforms under matched seeds and scenario scripts so that differences are attributable to watermarking rather than environmental variation. This setting tests whether behavioral provenance remains recoverable while preserving persona consistency and socially appropriate in-

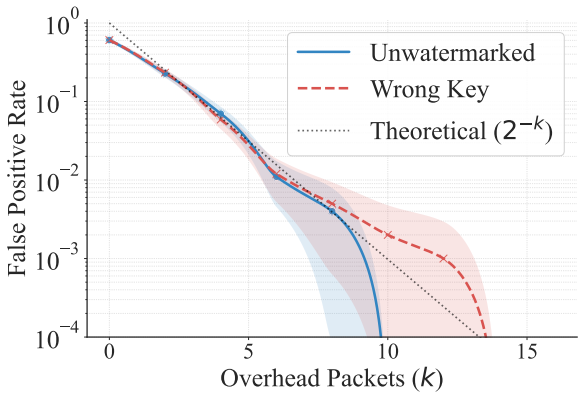

Figure 4: Both unwatermarked and wrong-key FPRs decay as 2^{-k} against the overhead k.

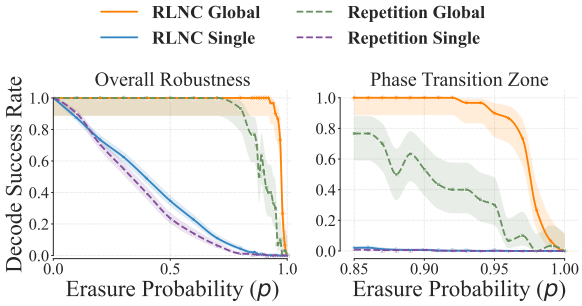

Figure 5: Robustness to Step Erasure and Truncation.

teractions in open-ended multi-agent feeds. Figure 3 shows that ours preserves social-quality utility while maintaining high watermark verifiability; details are in Appendix G.2.

5.3 Robustness Experiment

False Positives and Key Forgery We run 1000 Monte Carlo trials per overhead packet $k \in [0, 16]$ with payload length $N = 128$, using 281 ToolBench trajectories (T1–T3) and 159 unique action indices to construct the coefficient matrix (Appendix H.1). Figure 4 reports the verifier's false-positive rates (FPR) under unwatermarked logs and wrong-key decoding, where verification accepts only if the induced GF(2) system is consistent. FPR drops below 1% for $k \geq 8$, and we observe no false positives for $k \geq 14$. Both curves decay exponentially with k, closely tracking the 2^{-k} trend predicted by Eqs. (17) and (18).

Robustness to Step Erasure and Truncation. We evaluate robustness to step erasure by independently dropping each logged decision step with probability p and decoding from the remaining records. We compare RLNC-based coding with a repetition baseline, under single-episode decoding (per trajectory) and global decoding (aggregation across trajectories); details are in Appendix H.2. Figure 5 shows that RLNC is substantially more erasure-resilient than repetition, and global aggregation further improves robustness by accumulating enough independent equations for full-rank recovery, with a clear phase transition only when the effective packet budget becomes insufficient.

Condition	Behavior Match Rate (%) ↑	Avg. KL	Bit Recovery Rate (%) ↑
No rewriting	100.00	0.000	100.00
Semantic rewriting	49.45 ± 16.90	3.227 ± 1.802	16.84 ± 19.56

Table 2: Sensitivity to semantic-preserving observation rewriting on ALFWorld-OOD (134 tasks; 2326 steps). Behavior Match Rate is the fraction of steps whose planning behavior matches between original and rewritten observations. Avg. KL is the mean KL divergence between the elicited behavior distributions. Bit Recovery Rate is the fraction of watermarked steps whose extracted bitstring matches between the two observations.

Robustness to Semantic-Preserving Observation Rewriting. Table 2 reports a high-pressure stress test that semantically rewrites step observations while keeping the task state unchanged, emulating deployments where raw logs are missing and only semantically equivalent reconstructions are available for verification; from these results, we can get the following conclusion. First, rewriting can noticeably shift the elicited behavior distribution (Avg. KL $= 3.227 \pm 1.802$), so perfect step-level synchronization is not expected. Second, the moderate mean divergence with substantial variance indicates that many steps remain stable and yield similar P_t, preserving aligned decisions and recoverable watermark signal (Behavior Match Rate $= 49.45 \pm 16.90\%$, Bit Recovery Rate $= 16.84 \pm 19.56\%$). Finally, since ID payloads are always short while agent executions are long-horizon, aggregation over surviving aligned steps can still support practical verification. Further details are provided in Appendix H.3. We further extend the robustness analysis to cross-run distribution stability in Appendix G.1.2 and discuss adaptive threat models in Appendix H.4.

5.4 Content-Watermark Compatibility

As summarized in Table 3, AgentMark watermarks planning behaviors and is complementary to action-layer content watermarking. We evaluate composability with distribution-preserving SynthID-Text by enabling both watermarks and show that task

Config	Multi bit	Behav. cap.	Content det.	Threat model
SynthID-Text	✗	–	✓	rewrite
AgentMark-F	✓	high	–	erasure
Both	✓	high	✓	rewrite & erasure

Table 3: Composability of AgentMark-F and SynthID.

utility is preserved, behavioral decoding for Ours remains correct (100%), and SynthID-Text detection under Both stays high (96.6% on ToolBench; details in Appendix I). Overall, the two watermarks provide complementary provenance under different failures: AgentMark-F targets trajectory-level log erasure and truncation, whereas SynthID-Text targets robustness of final content under rewriting.

6 Conclusion

We propose AgentMark, which embeds multi-bit identifiers into behaviors via distribution-preserving sampling under black-box APIs, supporting recovery under step erasure and truncation while preserving utility. Across embodied, tool-use, and social environments, we show non-trivial capacity, reliable decoding from partial logs, and compatibility with existing action-layer content watermarking. We hope AgentMark makes provenance a deployable primitive for scalable auditing and accountability, thereby laying the groundwork for AI governance and safeguarding societal security in the era of autonomous agents.

Limitations

AgentMark requires the agent to output an explicit planning-time behavior distribution P_t over a predefined behavior list and to log planning-time decisions (and minimal step context) for verification; deployments that cannot retain sufficient logs may limit provenance recovery. Since verification relies on step observations to reproduce sampling randomness, semantic variation in observations (e.g., paraphrasing) may reduce synchronization and thus degrade verification quality. Because AgentMark preserves the original behavior distribution, the per-step embedding capacity is naturally context-dependent and can be small when P_t is highly peaked, which may reduce per-episode capacity without aggregation. Finally, our approach exploits the fact that an LLM induces an implicit conditional policy over a finite behavior set, and makes this policy auditable by eliciting an explicit probability list, which enables provenance under black-box APIs; for open-source LLMs, an important future direction is to extract such planning-time policies directly from logits or latent representations, thereby enabling provenance without explicit elicitation.

Acknowledgements

The authors are immensely grateful to Guorui Liao for his vital mentorship and insightful discussions on provable security, which significantly shaped this research. This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFC3305402 and in part by the National Natural Science Foundation of China under Grant 62172053 and Grant 62302059.

References

Deepak Bhaskar Acharya, Karthigeyan Kuppan, and B Divya. 2025. Agentic ai: Autonomous intelligence for complex goals–a comprehensive survey. *IEEE Access*.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Ruibo Chen, Yihan Wu, Junfeng Guo, and Heng Huang. 2025. Improved unbiased watermark for large language models. *arXiv preprint arXiv:2502.11268*.

Sumanth Dathathri, Abigail See, Sumedh Ghaisas, Po-Sen Huang, Rob McAdam, Johannes Welbl, Vandana Bachani, Alex Kaskasoli, Robert Stanforth, Tatiana Matejovicova, and 1 others. 2024. Scalable watermarking for identifying large language model outputs. *Nature*, 634(8035):818–823.

Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. 2025. Ai agents under threat: A survey of key security challenges and future pathways. *ACM Computing Surveys*, 57(7):1–36.

Jinyang Ding, Kejiang Chen, Yaofei Wang, Na Zhao, Weiming Zhang, and Nenghai Yu. 2023. Discop: Provably secure steganography in practice based on" distribution copies". In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2238–2255. IEEE.

Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong Li. 2024. Large language models empowered agent-based modeling and simulation: A survey and perspectives.

Humanities and Social Sciences Communications, 11(1):1–24.

Chen Gao, Xiaochong Lan, Zhihong Lu, Jinzhu Mao, Jinghua Piao, Huandong Wang, Depeng Jin, and Yong Li. 2023. S3: Social-network simulation system with large language model-empowered agents. *arXiv preprint arXiv:2307.14984*.

Batu Guan, Yao Wan, Zhangqian Bi, Zheng Wang, Hongyu Zhang, Pan Zhou, and Lichao Sun. 2024. Codeip: A grammar-guided multi-bit watermark for large language models of code. *arXiv preprint arXiv:2404.15639*.

Yifeng He, Ethan Wang, Yuyang Rong, Zifei Cheng, and Hao Chen. 2025. Security of ai agents. In *2025 IEEE/ACM International Workshop on Responsible AI Engineering (RAIE)*, pages 45–52. IEEE.

Zhiwei He, Binglin Zhou, Hongkun Hao, Aiwei Liu, Xing Wang, Zhaopeng Tu, Zhuosheng Zhang, and Rui Wang. 2024. Can watermarks survive translation? on the cross-lingual consistency of text watermark for large language models. *Preprint*, arXiv:2402.14007.

Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. 2006. A random linear network coding approach to multicast. *IEEE Transactions on information theory*, 52(10):4413–4430.

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.

Kaibo Huang, Yukun Wei, Wansheng Wu, Tianhua Zhang, Zhongliang Yang, and Linna Zhou. 2025a. Whispering agents: An event-driven covert communication protocol for the internet of agents. *arXiv preprint arXiv:2508.02188*.

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

Vojtěch Hudeček and Ondrej Dusek. 2023. Are large language models all you need for task-oriented dialogue? In *Proceedings of the 24th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 216–228, Prague, Czechia. Association for Computational Linguistics.

Gabriel Kaptchuk, Tushar M Jois, Matthew Green, and Aviel D Rubin. 2021. Meteor: Cryptographically secure steganography for realistic distributions. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1529–1548.

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*, pages 17061–17084. PMLR.

Gregory Kang Ruey Lau, Xinyuan Niu, Hieu Dao, Jiangwei Chen, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. 2024. Waterfall: Framework for robust and scalable text watermarking. In *ICML 2024 Workshop on Foundation Models in the Wild*.

Taehyun Lee, Seokhee Hong, Jaewoo Ahn, Ilgee Hong, Hwaran Lee, Sangdoo Yun, Jamin Shin, and Gunhee Kim. 2024. Who wrote this code? watermarking for code generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4890–4911.

Guorui Liao, Jinshuai Yang, Weizhi Shao, and Yongfeng Huang. 2025. A framework for designing provably secure steganography. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 6837–6856.

Aiwei Liu, Leyi Pan, Xuming Hu, Shiao Meng, and Lijie Wen. 2024a. A semantic invariant robust watermark for large language models. *Preprint*, arXiv:2310.06356.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, and 1 others. 2024b. Autoglm: Autonomous foundation agents for guis. *arXiv preprint arXiv:2411.00820*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2024c. Agentbench: Evaluating llms as agents. In *ICLR*.

Xinyi Mou, Zhongyu Wei, and Xuanjing Huang. 2024. Unveiling the truth and facilitating change: Towards agent-based large-scale social movement simulation. *arXiv preprint arXiv:2402.16333*.

Leyi Pan, Aiwei Liu, Zhiwei He, Zitian Gao, Xuandong Zhao, Yijian Lu, Binglin Zhou, Shuliang Liu, Xuming Hu, Lijie Wen, and 1 others. 2024. Marklm: An open-source toolkit for llm watermarking. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 61–71.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Vaidehi Patil, Peter Hase, and Mohit Bansal. 2023. Can sensitive information be deleted from llms? objectives for defending against extraction attacks. *arXiv preprint arXiv:2309.17410*.

Wenjun Peng, Jingwei Yi, Fangzhao Wu, Shangxi Wu, Bin Zhu, Lingjuan Lyu, Binxing Jiao, Tong Xu, Guangzhong Sun, and Xing Xie. 2023. Are you copying my model? protecting the copyright of large language models for eaas via backdoor watermark. *arXiv preprint arXiv:2305.10036*.

Jinghua Piao, Yuwei Yan, Jun Zhang, Nian Li, Junbo Yan, Xiaochong Lan, Zhihong Lu, Zhiheng Zheng, Jing Yi Wang, Di Zhou, and 1 others. 2025. Agentsociety: Large-scale simulation of llm-driven generative agents advances understanding of human behaviors and society. *arXiv preprint arXiv:2502.08691*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304.

Shaghayegh Shajarian, Sajad Khorsandroo, and Mahmoud Abdelsalam. 2024. A survey on self-running networks: Concepts, components, opportunities, and challenges. *Authorea Preprints*.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020a. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020b. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Harold Triedman, Rishi Jha, and Vitaly Shmatikov. 2025. Multi-agent systems execute arbitrary malicious code. *arXiv preprint arXiv:2503.12188*.

Lei Wang, Jingsen Zhang, Hao Yang, Zhi-Yuan Chen, Jiakai Tang, Zeyu Zhang, Xu Chen, Yankai Lin, Hao Sun, Ruihua Song, and 1 others. 2025a. User behavior simulation with large language model-based agents. *ACM Transactions on Information Systems*, 43(2):1–37.

Yuntao Wang, Shaolong Guo, Yanghe Pan, Zhou Su, Fahao Chen, Tom H Luan, Peng Li, Jiawen Kang, and Dusit Niyato. 2025b. Internet of agents: Fundamentals, applications, and challenges. *arXiv preprint arXiv:2505.07176*.

Julia Wiesinger, Patrick Marlow, and Vladimir Vuskovic. 2024. Agents. Whitepaper, Google Cloud. Accessed via Kaggle.

Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, and 1 others. 2025. Towards large reasoning models: A survey of reinforced reasoning with large language models. *arXiv preprint arXiv:2501.09686*.

Chao Yang, Chaochao Lu, Yingchun Wang, and Bowen Zhou. 2024a. Towards AI-45° law: A roadmap to trustworthy AGI. *arXiv preprint arXiv:2412.14186*.

Ziyi Yang, Zaibin Zhang, Zirui Zheng, Yuxian Jiang, Ziyue Gan, Zhiyu Wang, Zijian Ling, Martin Ma, Bowen Dong, Prateek Gupta, and 1 others. 2024b. Oasis: Open agents social interaction simulations on one million agents. In *NeurIPS 2024 Workshop on Open-World Agents*.

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

Wentao Zhang, Lingxuan Zhao, Haochong Xia, Shuo Sun, Jiaze Sun, Molei Qin, Xinyi Li, Yuqing Zhao, Yilei Zhao, Xinyu Cai, and 1 others. 2024. A multimodal foundation agent for financial trading: Tool-augmented, diversified, and generalist. In *Proceedings of the 30th acm sigkdd conference on knowledge discovery and data mining*, pages 4314–4325.

Hongyu Zhu, Sichu Liang, Wentao Hu, Li Fangqi, Ju Jia, and Shi-Lin Wang. 2024. Reliable model watermarking: Defending against theft without compromising on evasion. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 10124–10133.

A Deterministic Differential Recombination

This appendix provides an explicit and deterministic procedure for the differential-based probability recombination used by AgentMark-F (Figure 6).

The goal is to transform an arbitrary discrete distribution P_t over a finite set $\mathcal{B}_t$ into a *mixture of uniform bins* that preserves the original marginal distribution while enabling uniform encoding within a selected bin. Our construction is an instantiation of the probability recombination module in FDPSS. For the general framework and security discussions, see (Liao et al., 2025).

Setup and notation. Let $\mathcal{B}_t = \{b_{t,1}, \ldots, b_{t,n}\}$ be the candidate behavior set with probabilities $p_i = P_t(b_{t,i})$. We assume $\sum_{i=1}^{n} p_i = 1$ and $p_i \geq 0$. To simplify notation, we define $p_{n+1} = 0$.

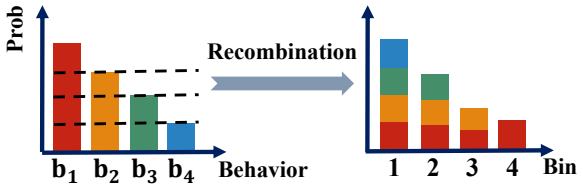

Figure 6: Differential-based recombination slices the behavior distribution into equal-probability layers and recombines them into a mixture of uniform bins.

Canonical ordering under probability ties. Differential recombination relies on a sorted ordering of behaviors by probability. In practice, multiple behaviors may have identical probabilities (or probabilities that are equal up to numerical precision). To ensure encoder–decoder synchronization, we define a canonical ordering as follows. We first quantize probabilities to a fixed precision, e.g., $p_i \mapsto \mathrm{round}(p_i; \pi)$ for a chosen precision π. We then apply a stable sort by the quantized probabilities in non-increasing order, so that behaviors with equal (quantized) probabilities preserve a consistent relative order across runs. This produces an ordered sequence $(b_{t,1}, \ldots, b_{t,n})$ such that $p_1 \geq \cdots \geq p_n$ with deterministic behavior under ties.

Differential slicing and bin construction. The differential-based scheme can be understood as horizontal slicing of the sorted probability histogram. Define slice heights

$$d_k = p_k - p_{k+1}, \qquad k \in \{1, \ldots, n\}. \tag{19}$$

Each slice k has height d_k and spans the top-k behaviors. Equivalently, each behavior $b_{t,i}$ is decomposed into blocks $\{d_k\}_{k \geq i}$ since

$$p_i = \sum_{k=i}^{n} d_k. \tag{20}$$

We group the k equal-height blocks at level d_k into a *uniform bin* $T_k = \{b_{t,1}, \ldots, b_{t,k}\}$. The mixture weight (total probability mass) of bin k is

$$q_k = k \cdot d_k. \tag{21}$$

Bins with $d_k = 0$ have zero weight and can be omitted.

Marginal distribution preservation. If we first sample a bin index $K \sim \mathrm{Cat}(q_1, \ldots, q_n)$ and then sample uniformly within the selected bin T_K, the marginal probability of selecting behavior $b_{t,i}$ is:

$$\begin{aligned} \Pr[\hat{b}_t = b_{t,i}] &= \sum_{k=i}^{n} q_k \cdot \frac{1}{k} \\ &= \sum_{k=i}^{n} (p_k - p_{k+1}) \\ &= p_i, \end{aligned} \tag{22}$$

which matches the original distribution.

Algorithm 3 Deterministic differential-based recombination

Require: Candidate set $\mathcal{B}_t = \{b_1, \ldots, b_n\}$, distribution P_t, quantization precision π
Ensure: Ordered list $(b_{(1)}, \ldots, b_{(n)})$, bins $\{T_k\}$ and weights $\{q_k\}$

$p_i \leftarrow P_t(b_i)$ for all i; $p_{n+1} \leftarrow 0$
$\tilde{p}_i \leftarrow \mathrm{round}(p_i; \pi)$ for all i ▷ quantize probabilities
$(b_{(1)}, \ldots, b_{(n)}) \leftarrow \mathrm{StableSort}(\{b_i\};\ \tilde{p}_i \text{ descending})$
$p_k \leftarrow P_t(b_{(k)})$ for $k = 1, \ldots, n$; $p_{n+1} \leftarrow 0$
for $k = 1$ to n **do**
    $d_k \leftarrow p_k - p_{k+1}$
    if $d_k > 0$ **then**
        $T_k \leftarrow \{b_{(1)}, \ldots, b_{(k)}\}$ ▷ top-k behaviors
        $q_k \leftarrow k \cdot d_k$
    end if
end for
return $(b_{(1)}, \ldots, b_{(n)}), \{(T_k, q_k) : d_k > 0\}$

Algorithm. Algorithm 3 summarizes the deterministic recombination procedure. It outputs the ordered behaviors and the corresponding list of nonzero bins with mixture weights, which are then used by the bin-sampling and uniform-encoding modules.

B Cyclic-Shift Uniform Encoding and Decoding

This appendix details the cyclic-shift uniform encoding/decoding module used by AgentMark-F within a selected uniform bin. The role of this module is to embed a (variable-length) prefix of the payload bitstream into a uniform choice among n candidates, while remaining invertible given the same pseudorandomness. Our presentation follows the standard uniform-steganography component used in FDPSS-style constructions; see (Liao et al., 2025) for further discussion and security proofs.

Setting. Fix a time step t and a selected bin T containing $n = |T|$ behaviors, which is treated as a uniform distribution over indices $\{0, 1, \ldots, n-1\}$. Let M denote the payload bitstream and ℓ the current pointer, so the available suffix is $M[\ell :]$. Both encoder and decoder have synchronized access to a pseudorandom generator PRG (seeded as described in the main text), from which they draw a shared random value that induces a cyclic shift.

Key idea. Let $k \triangleq \lfloor \log_2 n \rfloor$ and $t \triangleq n - 2^k$, so $n = 2^k + t$ with $0 \le t < 2^k$. CyclicShift embeds either k bits or $k{+}1$ bits depending on the payload prefix, yielding near-optimal utilization of the uniform bin. Intuitively, $2^k - t$ indices correspond to k-bit codewords and the remaining $2t$ indices correspond to $(k{+}1)$-bit codewords, forming a prefix-free set of size n.

Shared cyclic shift. Both sides draw $\mathrm{ptr} \in [0, 1)$ from PRG and compute a shift

$$R \triangleq \lfloor \mathrm{ptr} \cdot n \rfloor \in \{0, 1, \ldots, n-1\}. \tag{23}$$

Because PRG is synchronized, the decoder reconstructs the same R.

Algorithms. Algorithm 4 and Algorithm 5 give the one-step cyclic-shift encoding and decoding procedures. The encoder outputs an index j within a size-n bin and the embedded substring s (either k or $k{+}1$ bits). The decoder takes the observed index j and recovers the same substring s.

Correctness. For $n > 1$, both algorithms use the same pseudorandom shift R and apply inverse mappings between payload prefixes and bin indices, hence decoding inverts encoding at each step and recovers the embedded substring s exactly. For $n = 1$, there is no choice within the bin and the embedded bitstring is empty.

Uniformity. The cyclic shift randomizes the mapping between payload prefixes and indices. When combined with a pseudorandom payload bitstream (e.g., after encryption/coding as assumed in standard steganography formulations), the induced selection within a uniform bin is indistinguishable from uniform sampling. We refer readers to (Liao et al., 2025) for formal security statements in the FDPSS setting.

Capacity (near-optimality). For a bin of size n, CyclicShift embeds a variable number of bits

Algorithm 4 CyclicShift (one step): ENCODE

Require: Payload suffix $M[\ell :]$, bin size n, synchronized PRG
Ensure: Index $j \in \{0, \ldots, n-1\}$, embedded bits s

if $n = 1$ **then**
  return $0,\ \emptyset$
end if
$\mathrm{ptr} \leftarrow \mathsf{PRG}()$; $R \leftarrow \lfloor \mathrm{ptr} \cdot n \rfloor$
$k \leftarrow \lfloor \log_2 n \rfloor$; $t \leftarrow n - 2^k$
if $|M[\ell :]| < k$ **then**
  return $R,\ \emptyset$
end if
$x \leftarrow \mathrm{bits2int}(M[\ell : \ell + k])$
$r \leftarrow \begin{cases} M[\ell + k] & \text{if } |M[\ell :]| \ge k + 1 \\ 0 & \text{otherwise} \end{cases}$
if $x < 2^k - t$ **then**
  $j \leftarrow (x + R) \bmod n$; $s \leftarrow M[\ell : \ell + k]$
else
  $j \leftarrow \big(2(x - (2^k - t)) + (2^k - t) + R + r\big) \bmod n$
  $s \leftarrow M[\ell : \ell + k + 1]$
end if
return $j,\ s$

$c \in \{k, k{+}1\}$ where $k = \lfloor \log_2 n \rfloor$ and $n = 2^k + m$ with $0 \le m < 2^k$ (and $c = 0$ when $n = 1$). The prefix-free codebook induced by CyclicShift assigns $2^k - m$ indices to k-bit codewords and $2m$ indices to $(k{+}1)$-bit codewords, so under a uniform choice among n indices the expected embedded length is

$$\begin{aligned} \mathbb{E}[c \mid n] &= \frac{(2^k - m) \cdot k + (2m) \cdot (k+1)}{n} \\ &= k + \frac{2m}{n}. \end{aligned} \tag{24}$$

This yields the information-theoretic upper bound $\mathbb{E}[c \mid n] \le \log_2 n$. Moreover, writing

$$\mathbb{E}[c \mid n] = \log_2 n - \log_2\Big(1 + \frac{m}{2^k}\Big) + \frac{2m}{n}, \tag{25}$$

and letting $x \triangleq m/2^k \in [0, 1)$, we have $\log_2(1 + x) - x \le 0.0861$ for all $x \in [0, 1)$, which implies the constant-gap lower bound

$$\mathbb{E}[c \mid n] \ge \log_2 n - 0.0861. \tag{26}$$

Therefore, CyclicShift achieves near-optimal expected capacity for uniform bins.

Algorithm 5 CyclicShift (one step): DECODE

Require: Observed index $j \in \{0, \ldots, n-1\}$, bin size n, synchronized PRG
Ensure: Extracted bits s (possibly empty)

if $n = 1$ **then**
  return $\emptyset$
end if
$\text{ptr} \leftarrow \mathsf{PRG}(); R \leftarrow \lfloor \text{ptr} \cdot n \rfloor$
$k \leftarrow \lfloor \log_2 n \rfloor; t \leftarrow n - 2^k$
$\text{idx} \leftarrow (j - R) \bmod n$
if $\text{idx} < 2^k - t$ **then**
  $s \leftarrow \text{int2bits}(\text{idx}, k)$
else
  $u \leftarrow \text{idx} - (2^k - t)$
  $x \leftarrow \lfloor u/2 \rfloor + (2^k - t)$
  $r \leftarrow u \bmod 2$
  $s \leftarrow \text{int2bits}(x, k) \,\|\, r$
end if
return s

C End-to-End Example

One-step example (ticket purchase). We illustrate a single watermark-embedding step for an agent assisting with ticket purchase. Suppose the user goal is to buy a ticket, and the step-t observation is

$$o_t = \texttt{User: book a ticket.} \tag{27}$$

with step index $t = 7$ and shared secret K_{sh}. For concreteness, we instantiate the step context as a short interaction trace,

$$\begin{aligned} h_t = [&(t{-}2,\ \mathsf{Search}),\ (t{-}1,\ \mathsf{Book}),\\ &\texttt{User preference: window seat.},\\ &\texttt{Budget: \$300.}] \end{aligned} \tag{28}$$

which is available to both the encoder and verifier from the same logs. We therefore write the shared step context as $Context_t \triangleq (t, o_t, h_t)$.

The agent elicits the following behavior distribution P_t over $\mathcal{B}_t$={Search,Book,Pay,Check-in,Modify}:

$$P_t = (0.40,\ 0.25,\ 0.15,\ 0.12,\ 0.08), \tag{29}$$

which sums to 1.

Differential recombination. After sorting in non-increasing order (already in the above order), we have $p_1 = 0.40, p_2 = 0.25, p_3 = 0.15, p_4 = 0.12, p_5 = 0.08$, and $p_6 = 0$. The slice heights are

$$(d_1, d_2, d_3, d_4, d_5) = (0.15,\ 0.10,\ 0.03,\ 0.04,\ 0.08). \tag{30}$$

Differential recombination forms uniform bins T_k (top-k behaviors) with mixture weights $q_k = k \cdot d_k$:

$$(q_1, q_2, q_3, q_4, q_5) = (0.15,\ 0.20,\ 0.09,\ 0.16,\ 0.40), \tag{31}$$

and

$$\begin{aligned} T_1 &= \{\mathsf{Search}\},\\ T_2 &= \{\mathsf{Search}, \mathsf{Book}\},\\ T_3 &= \{\mathsf{Search}, \mathsf{Book}, \mathsf{Pay}\},\\ T_4 &= \{\mathsf{Search}, \mathsf{Book}, \mathsf{Pay}, \mathsf{Check\text{-}in}\},\\ T_5 &= \{\mathsf{Search}, \mathsf{Book}, \mathsf{Pay}, \mathsf{Check\text{-}in}, \mathsf{Modify}\}. \end{aligned} \tag{32}$$

Keyed pseudorandomness and bin selection. We derive the per-step key $K_t \leftarrow H(K_{\text{sh}} \| Context_t)$ and initialize $\mathsf{PRG} \leftarrow \mathsf{PRG}(K_t)$. Assume the first pseudorandom draw for bin selection is $u_1 = 0.62$. With cumulative weights $(0.15, 0.35, 0.44, 0.60, 1.00)$, this selects $K = 5$, hence the selected bin is $T = T_5$ with size $n = |T| = 5$.

Payload embedding via CyclicShift within the bin. Suppose the next payload bits to embed are $m_t = \texttt{01}$. To pseudorandomize the embedded bits, we generate a pad from the same per-step key and bin-dependent length:

$$Z_t \leftarrow \mathsf{PRG}(K_t), \qquad \tilde{m}_t \triangleq m_t \oplus Z_t[1{:}2]. \tag{33}$$

For example, if $Z_t[1{:}2] = \texttt{11}$ then $\tilde{m}_t = \texttt{10}$. CyclicShift then uses a second pseudorandom draw $u_2 = 0.27$ to set the cyclic shift $R = \lfloor u_2 \cdot n \rfloor = \lfloor 0.27 \cdot 5 \rfloor = 1$.

With $n = 5$, we have $k = \lfloor \log_2 5 \rfloor = 2$ and $n - 2^k = 1$. Interpreting the first k bits of $\tilde{m}_t$ as $x = \text{bits2int}(\texttt{10}) = 2$, since $x < 2^k - (n - 2^k) = 3$, CyclicShift selects index

$$j = (x + R) \bmod 5 = (2 + 1) \bmod 5 = 3, \tag{34}$$

and embeds $s_t = \texttt{10}$ (two bits). Therefore the watermarked behavior is the j-th element in T_5, i.e.,

$$\hat{b}_t = T_5[3] = \mathsf{Check\text{-}in}. \tag{35}$$

Although this particular realization selects Check-in, the marginal distribution over $\hat{b}_t$ remains P_t by construction.

Decoding (same step). Given $Context_t$ and the observed behavior $\hat{b}_t = \textsf{Check-in}$, the decoder derives the same K_t, reproduces the same bin T_5 and shift $R = 1$, and recovers the same index $j = 3$. CyclicShift decoding returns $s_t = \texttt{10}$, and the original payload bits are recovered by unmasking:

$$m_t = \tilde{m}_t \oplus Z_t[1{:}2] = \texttt{10} \oplus \texttt{11} = \texttt{01}. \tag{36}$$

Takeaway. This one-step example shows how AgentMark-F embeds provenance bits into behavior selection while preserving the original behavior distribution. By enabling reliable attribution from logged behaviors, AgentMark provides a provenance mechanism that does not impair task utility, making it practical for agent providers to deploy in real systems. This supports protecting intellectual property against unauthorized cloning or misuse and offers a concrete tool for auditing and accountability in regulated or platform-mediated settings.

D Security Proof

Distribution-preserving watermark embedding. AgentMark-F combines differential recombination and cyclic-shift uniform encoding to embed provenance bits while preserving the per-step behavior distribution. First, differential recombination rewrites an arbitrary distribution P_t as a mixture of uniform bins without changing its marginal. Next, within the selected uniform bin, CyclicShift embeds bits by choosing an index in a way that is (computationally) indistinguishable from uniform sampling. Finally, we pseudorandomize the embedded payload bits at each step using the per-step key, so that the bits presented to CyclicShift are pseudorandom even if the original payload has structure.

Pseudorandomizing the payload bits. Let K_{sh} be the shared secret and let $Context_t$ denote the shared step context at step t (e.g., the step index, observation, and logged history available to both sides). We derive a per-step key

$$K_t \leftarrow H(K_{\text{sh}} \,\|\, Context_t), \tag{37}$$

and use a keyed pseudorandom generator to produce a pad bitstream

$$Z_t \leftarrow \textsf{PRG}(K_t) \in \{0,1\}^{\infty}. \tag{38}$$

At step t, CyclicShift embeds a variable number of bits $c_t \in \{k, k{+}1\}$ where $k = \lfloor \log_2 |T| \rfloor$ and $|T|$ is the selected bin size. Let $m_t \in \{0,1\}^{c_t}$ be the next c_t payload bits to embed. We mask them with the per-step pad:

$$\tilde{m}_t \triangleq m_t \oplus Z_t[1{:}c_t], \tag{39}$$

where $\oplus$ denotes a generic keyed pseudorandom mixing operation (e.g., bitwise XOR as a canonical instantiation), so long as $\tilde{m}_t$ is computationally indistinguishable from uniform given K_t and the corresponding pad segment.

The encoder feeds $\tilde{m}_t$ to CyclicShift to choose the bin index, while the decoder recovers $\tilde{m}_t$ and unmask it using the same pad bits. Under standard PRG assumptions, $\tilde{m}_t$ is computationally indistinguishable from uniform even if m_t is not.

Marginal distribution preservation. Let $\mathcal{B}_t = \{b_{t,1}, \ldots, b_{t,n}\}$ be ordered so that $p_i = P_t(b_{t,i})$ and $p_1 \geq \cdots \geq p_n$, with $p_{n+1} = 0$. Differential recombination defines $d_k = p_k - p_{k+1}$ and mixture weights $q_k = k \cdot d_k$, where bin $T_k = \{b_{t,1}, \ldots, b_{t,k}\}$ is uniform over its k elements. Sampling $K \sim \text{Cat}(q_1, \ldots, q_n)$ and then selecting uniformly from T_K preserves the marginal:

$$\begin{aligned} \Pr[\hat{b}_t = b_{t,i}] &= \sum_{k=i}^{n} q_k \cdot \frac{1}{k} \\ &= \sum_{k=i}^{n} (p_k - p_{k+1}) \\ &= p_i. \end{aligned} \tag{40}$$

Putting the pieces together. By the identity above, the recombination-based sampler is equivalent to drawing from P_t at each step. Replacing the ideal randomness used in bin selection and cyclic-shift encoding by $\textsf{PRG}(K_t)$ yields a computationally indistinguishable sampling process under standard PRG security. Masking the payload bits by $\tilde{m}_t = m_t \oplus Z_t[1{:}c_t]$ ensures that the input bits consumed by CyclicShift are pseudorandom, aligning with the uniform-steganography requirement in FDPSS-style analyses (Liao et al., 2025). Therefore, AgentMark-F embeds provenance bits while keeping the induced behavior distribution computationally indistinguishable from sampling $b_t \sim P_t$.

E RLNC Details and Advantages

This appendix formalizes why RLNC is a natural fit for variable-capacity behavioral watermarking under erasure/truncation and provides decoding details. We refer to (Ho et al., 2006) for classical RLNC analysis.

Variable-capacity embedding as a rateless linear system. At step t, the sampler outputs $s_t \in \{0,1\}^{c_t}$ where the capacity c_t varies with the selected bin size. We treat each embedded bit as one linear equation over $\mathbb{F}_2$:

$$y_{t,j} = \langle a_{t,j}, m \rangle, \qquad a_{t,j} \in \mathbb{F}_2^L, j = 1, \ldots, c_t, \tag{41}$$

so each step contributes exactly c_t equations. Under erasure/truncation, the verifier observes $\mathcal{I} \subseteq \{1, \ldots, T\}$ and receives $R = \sum_{t \in \mathcal{I}} c_t$ equations, yielding a linear system

$$A_{\mathcal{I}}\, m = y_{\mathcal{I}}, \qquad A_{\mathcal{I}} \in \mathbb{F}_2^{R \times L},\ y_{\mathcal{I}} \in \mathbb{F}_2^R. \tag{42}$$

This formulation is *rateless* in the sense that decoding depends only on how many equations are ultimately collected, not on a fixed per-step rate.

Erasure robustness via rank. Unique recovery holds iff $\mathrm{rank}(A_{\mathcal{I}}) = L$. If the coefficient rows behave as (pseudo)random in $\mathbb{F}_2^L$, then for any realization with $R = L + \Delta$ equations,

$$\Pr[\mathrm{rank}(A_{\mathcal{I}}) = L] \geq 1 - 2^{-\Delta}, \tag{43}$$

which yields an exponentially decaying failure probability as additional equations accrue beyond L. Crucially, this bound is agnostic to *which* steps are erased: only the total received capacity $R = \sum_{t \in \mathcal{I}} c_t$ matters.

Keyed pseudorandom coefficient generation. To make coefficients reproducible for the verifier while remaining pseudorandom, we derive a per-step key

$$K_t \leftarrow H(K_{\mathrm{sh}} \,\|\, Context_t), \tag{44}$$

and generate coefficients deterministically as

$$a_{t,j} \leftarrow \mathsf{PRG}(K_t, j) \in \mathbb{F}_2^L. \tag{45}$$

Under standard PRG assumptions, these coefficients are computationally indistinguishable from uniform while being reproducible from $Context_t$ on the verifier side.

Decoding algorithm (Gaussian elimination over $\mathbb{F}_2$). Given $(A_{\mathcal{I}}, y_{\mathcal{I}})$, the verifier solves for m via Gaussian elimination in $\mathbb{F}_2$. The time complexity is $O(L^3)$ in the worst case for dense matrices, with substantially lower cost under sparsity or incremental updates.

Algorithm 6 RLNC decode over $\mathbb{F}_2$ (sketch)

Require: Rows $(a_\ell, y_\ell) \in \mathbb{F}_2^L \times \mathbb{F}_2$ for $\ell = 1, \ldots, R$
Ensure: Solution $m \in \mathbb{F}_2^L$ if full-rank; otherwise fail
Form matrix $A \in \mathbb{F}_2^{R \times L}$ and vector $y \in \mathbb{F}_2^R$
Perform Gaussian elimination on $[A \,|\, y]$ over $\mathbb{F}_2$
if $\mathrm{rank}(A) < L$ **then**
    return fail
else
    return the unique solution m
end if

RLNC versus fixed-rate block codes. Block codes (e.g., Reed–Solomon(Reed and Solomon, 1960)) typically assume fixed-size symbols and a predetermined codeword structure. In our setting, the per-step capacity c_t is variable and the observed trajectory may be an arbitrary subsequence $\mathcal{I}$, which makes fixed block boundaries and symbol alignment fragile. RLNC avoids explicit block alignment by treating every embedded bit as a linear measurement and decoding from any subset of sufficiently many measurements, i.e., from any $\mathcal{I}$ that yields $R \geq L$ with high rank.

F Datasets and Benchmark Details

F.1 ALFWorld (ID/OOD)

ALFWorld (Shridhar et al., 2020b) is a long-horizon embodied household benchmark built on interactive TextWorld environments aligned with the ALFRED setting (Shridhar et al., 2020a). Each episode requires multi-step planning and action execution to complete a goal (e.g., navigation and object interactions), making it a natural testbed for behavior-level watermarking where small distribution shifts can amplify over time. We report two evaluation regimes. The in-distribution (ID) split follows the standard test setting, and results are averaged over three independent runs on the same 100-task test set. The out-of-distribution (OOD) split evaluates on a 134-task cross-distribution test set with different task compositions. We report success rate (SR), average steps on successful episodes, and wall-clock time; watermark capacity is reported only for **Ours**.

F.2 ToolBench Test Subsets

ToolBench is a large-scale tool-use benchmark constructed from real-world APIs (e.g., RapidAPI), covering thousands of tools and tens of thousands of API endpoints, with diverse single-tool and multi-tool instructions that require structured tool calls and multi-step reasoning (Liu et al., 2024c). We evaluate trajectories using ToolEval, an automatic evaluator that executes predicted tool calls (with bounded budgets) and scores whether the instruction is successfully completed. Following the standard ToolEval protocol, we report results on six test subsets that cover single-tool tool selection, single-tool instruction following, single-tool within-category selection, multi-tool within-category selection, multi-tool instruction following, and complex multi-tool instruction following.

F.3 OASIS Platforms

OASIS is a scalable social-media simulation environment that mimics platform dynamics of Twitter-like and Reddit-like ecosystems, where agents interact through diverse social actions (e.g., posting, commenting, reposting, following) under evolving feeds and recommendation mechanisms (Yang et al., 2024b). We simulate both platforms and compare watermarked and non-watermarked agents under the same seeds and scenario scripts, which enables a deployment-oriented assessment of behavioral consistency and social-quality utility. Beyond task-level logs for provenance, we evaluate social-quality dimensions including coherence, memory accuracy, persona consistency, social norms/common sense, and language diversity; we also report the average embedded bits per step as a reference statistic.

G Details of the Utility and Capacity Experiments

G.1 Details in ALFWorld and ToolBench

Evaluation. We evaluate task utility and watermark capacity/overhead under a unified agent execution pipeline across environments. At each step, the agent outputs a planning behavior $b_t \in \mathcal{B}_t$ and then generates an execution action conditioned on b_t. We compare three variants that differ only in how b_t is selected: (i) **Baseline** follows the ReAct loop and directly outputs a planning behavior under the baseline policy $P_t^\star$; (ii) **RG** adds a logit bias to a pseudorandom green subset of $\mathcal{B}_t$ before sampling; and (iii) **AgentMark-F** elicits an explicit behavior distribution P_t and samples $\hat{b}_t$ via distribution-preserving watermark sampling on P_t.

For each setting, we run three independent trials and report the mean and standard deviation.

ALFWorld and ToolBench details. We evaluate AgentMark-F and baselines on ALFWorld (ID/OOD splits) and ToolBench (six subsets covering single-tool and multi-tool instructions) under a unified agent interface. We use DeepSeek-Chat as the base model with temperature 1.0 on ALFWorld and 0.7 on ToolBench. On ALFWorld, we run 140 ID tasks and 134 OOD tasks for three trials (420 and 402 episodes). On ToolBench, we cap each episode to 10 steps and run three trials over 450 tasks in total (20 per subset for most splits and 50 for the multi-tool instruction split). For AgentMark-F, we report bits/step and bits/task by summing embedded substrings across steps (with an 8-bit RLNC-coded payload on ToolBench). For the biased red–green baseline ($\gamma = 0.5$, $\delta = 2.0$), we report Green% as the detection statistic: 71.9 ± 13.6 (ALFWorld-ID), 71.5 ± 15.2 (ALFWorld-OOD), and 67.0 ± 0.9 (ToolBench), which indicates detectable bias but, as a zero-bit signal, does not support decoding an explicit identifier.

G.1.1 No-Induction Calibration

This supplementary experiment isolates the effect of probability elicitation by comparing AgentMark-F and the elicited baseline against a standard no-induction ReAct loop. The no-induction baseline uses the original greedy ReAct interface without explicit probability elicitation and is repeated three times on ALFWorld (274 tasks) and ToolBench (120 tasks) with DeepSeek-Chat. As shown in Table 4, the no-induction baseline is consistently slightly below both Baseline and AgentMark-F, which is consistent with the elicitation prompt encouraging more structured deliberation over can-

Scenario	No-Induction SR (%)↑	AgentMark-F SR (%)↑	Baseline SR (%)↑
ALFWorld ID	83.1 ± 1.4	89.3 ± 30.9	89.5 ± 30.6
ALFWorld OOD	94.5 ± 0.9	97.5 ± 15.6	96.8 ± 17.7
ToolBench	59.6 ± 8.8	59.7 ± 4.8	59.9 ± 5.8

Table 4: No-induction calibration on DeepSeek-Chat. The no-induction agent uses a standard ReAct loop without explicit probability elicitation. SR is averaged over three runs on ALFWorld (274 tasks) and ToolBench (120 tasks).

didate behaviors. The key comparison remains Baseline versus AgentMark-F under the same elicitation interface, where the success-rate gap is negligible, providing indirect evidence that the elicited P_t faithfully reflects the underlying planning tendencies.

G.1.2 Cross-Run Distribution Stability

We additionally measure the stability of elicited behavior distributions across repeated runs using Jensen–Shannon divergence (JSD). On ToolBench, we use DeepSeek-Chat and collect 542 valid task pairs across six subsets and three independent runs. On ALFWorld, we use Gemini-2.0-Flash and collect 140 task trajectories across three runs. Table 5 shows that the first-step distributions are highly consistent across runs (JSD $\approx$ 0.07–0.08), while later-step divergence increases as execution histories branch. The overall divergence remains lower on ToolBench than on ALFWorld, which is consistent with the more constrained action spaces in tool-use environments.

Dataset	Model	First-Step JSD	Overall JSD	Task Pairs
ToolBench	DeepSeek-Chat	0.0701	0.1764	542
ALFWorld	Gemini-2.0-Flash	0.0765	0.4597	140

Table 5: Cross-run stability of elicited behavior distributions, measured by Jensen–Shannon divergence (JSD) between independently repeated runs.

G.1.3 Cross-Model Results

We further validate AgentMark on Gemini 2.0 Flash using 420 ALFWorld-ID episodes, 402 ALFWorld-OOD episodes, and results aggregated over six ToolBench subsets, with three runs per setting. As shown in Table 6, success rates remain close to the corresponding baselines across both ALFWorld and ToolBench, while Δs/step remains near zero. These results complement Table 1 and confirm that utility preservation and watermark capacity are not specific to DeepSeek-Chat.

G.1.4 Capacity–Entropy and Variance Statistics

To validate the theoretical capacity bound, we restrict the analysis to embedding-active steps, namely steps where the differential recombination produces a bin of size $n \geq 2$, so that at least one bit is embedded. Steps with a singleton bin ($n = 1$) contribute zero capacity and carry an undefined per-step entropy-to-capacity ratio; including them would conflate the capacity–entropy relationship with the proportion of trivial steps. Moreover, restricting to embedding-active steps ensures that the entropy $\overline{H(P_t)}$ is computed over the same decision points where the watermarking algorithm actually operates, yielding a meaningful upper-bound test. By conditioning on the same step set for both $\bar{c}_t$ and $\overline{H(P_t)}$, we obtain a paired comparison that directly tests whether $\bar{c}_t \leq \overline{H(P_t)}$ holds step by step. Note that the benchmark-level bps reported in Table 1 averages total embedded bits over all decision steps, including zero-capacity ones, and is therefore systematically lower than $\bar{c}_t$. Table 7 confirms this relationship across all 36 task–model combinations and further confirms that higher-entropy settings yield higher empirical capacity. For example, ALFWorld-OOD A5 exhibits substantially larger entropy and capacity than ToolBench T3, while the larger Gemini capacities align with its broader planning distributions under agent-oriented fine-tuning.

Prompt Examples

ALFWorld Agent Prompt

System Prompt

```
You are an expert household task agent.
Analyze the current situation and assign
probabilities to each action.
```

Few-Shot Examples

Context:

```
{few_shot_examples_based_on_task_type}
```

Success Action Blueprints

pick_and_place_simple:

```
  1. Search locations like
cabinet/drawer/countertop for the target
object.
  2. take <target>
  3. go to <destination>
  4. move <target> to <destination>
```

pick_clean_then_place:

```
  1. Find and take the target object.
  2. go to sinkbasin 1
  3. clean <target> with sinkbasin 1
  4. go to <destination>
  5. move <target> to <destination>
```

pick_heat_then_place:

```
  1. Find and take the target object.
  2. go to microwave 1
  3. heat <target> with microwave 1
  4. go to <destination>
  5. move <target> to <destination>
```

pick_cool_then_place:

```
  1. Find and take the target object.
  2. go to fridge 1
  3. cool <target> with fridge 1
  4. go to <destination>
```

Setting	Task	SR (%)↑		Steps		Capacity		Δs/step
		Base	Ours	Base	Ours	bps	bpt	
ALFWorld ID	Avg.	70.6 ± 45.6	67.8 ± 46.8	25.3 ± 18.0	28.0 ± 17.8	2.28 ± 1.29	80.6 ± 79.1	-0.18 ± 1.1
ALFWorld OOD	Avg.	71.7 ± 45.1	72.8 ± 44.6	26.5 ± 17.3	25.5 ± 17.5	1.96 ± 1.1	63.4 ± 66.6	-0.01 ± 1.2
ToolBench	Avg.	75.6 ± 8.0	75.8 ± 4.5	2.66 ± 0.41	2.74 ± 0.20	0.78 ± 0.1	2.1 ± 0.5	-0.64 ± 0.5

Table 6: Gemini 2.0 Flash on ALFWorld and ToolBench.

Setting	Task	DeepSeek-Chat			Gemini 2.0 Flash		
		$\bar{c}_t$ (Mean±Std)	$\overline{H(P_t)}$ (Mean±Std)	Steps	$\bar{c}_t$ (Mean±Std)	$\overline{H(P_t)}$ (Mean±Std)	Steps
ALFWorld ID	A1	1.78 ± 1.40	2.42 ± 1.12	282	3.28 ± 2.05	3.67 ± 1.66	1246
	A2	1.80 ± 1.38	2.43 ± 1.17	479	2.89 ± 2.19	3.33 ± 1.83	2078
	A3	1.72 ± 1.30	2.43 ± 1.05	604	2.80 ± 1.76	3.22 ± 1.38	2777
	A4	1.36 ± 1.24	2.25 ± 1.03	99	2.31 ± 1.80	2.66 ± 1.50	997
	A5	1.88 ± 1.32	2.65 ± 1.11	453	2.84 ± 1.74	3.27 ± 1.37	1787
	A6	1.82 ± 1.39	2.67 ± 1.17	472	3.32 ± 2.14	3.82 ± 1.67	2254
ALFWorld OOD	A1	1.91 ± 1.37	2.72 ± 1.07	479	1.69 ± 1.95	2.40 ± 1.47	961
	A2	1.67 ± 1.31	2.42 ± 1.06	610	0.78 ± 1.49	3.38 ± 1.49	2823
	A3	1.78 ± 1.30	2.60 ± 1.03	502	2.20 ± 1.77	2.81 ± 1.29	1276
	A4	1.64 ± 1.27	2.61 ± 1.05	171	3.13 ± 1.54	3.46 ± 1.20	1751
	A5	2.24 ± 1.34	2.93 ± 1.06	429	3.14 ± 1.78	3.90 ± 1.12	2247
	A6	1.69 ± 1.36	2.39 ± 1.12	306	2.06 ± 1.93	2.57 ± 1.57	754
ToolBench	T1	0.95 ± 1.04	1.42 ± 0.98	247	0.95 ± 1.15	1.52 ± 1.06	181
	T2	1.28 ± 1.13	1.79 ± 1.02	312	0.56 ± 1.00	1.16 ± 1.01	163
	T3	0.79 ± 1.11	1.38 ± 1.05	224	0.61 ± 0.82	1.14 ± 0.81	171
	T4	1.47 ± 1.15	1.97 ± 1.04	329	0.96 ± 1.14	1.46 ± 1.06	167
	T5	1.28 ± 1.15	1.82 ± 1.06	287	0.95 ± 1.08	1.48 ± 1.02	176
	T6	1.44 ± 1.03	1.89 ± 0.96	416	1.10 ± 1.06	1.64 ± 0.99	185

Table 7: Per-task step-level capacity–entropy analysis for DeepSeek-Chat and Gemini 2.0 Flash. $\bar{c}_t$ and $\overline{H(P_t)}$ are averaged over embedding-active steps only, i.e., steps where the selected bin size exceeds 1 and at least one bit is embedded. Zero-capacity steps are excluded so that the per-step capacity and entropy are computed over the same step population, enabling a direct comparison against the theoretical bound $\bar{c}_t \leq \overline{H(P_t)}$. By contrast, the bps values in Tables 1 and 6 divide total embedded bits by all decision steps, including zero-capacity ones, yielding lower averages.

5. move <target> to <destination>

pick_two_obj_and_place:
1. take <target1> → go to <destination> → move <target1> to <destination>
2. Return to origin → take <target2> → go to <destination> → move <target2> to <destination>

look_at_obj_in_light:
1. Find and take the target object (e.g., bowl/cd).
2. go to the observed location of the desklamp.
3. use desklamp (no need to put the object down).
4. examine <target>

Current Situation

Recent History:
{interaction_history_last_5_steps}

Inventory: {inventory_status_and_checks}
Task Goal: {task_description}
Observation: {observation}

Available Actions:
{admissible_commands_json_list}

Response Format

Thinking: Write a 'Thinking: ...' section to analyze the situation.
Output: Output the JSON probability object (sum to 1.0).

Toolbench Agent Prompt

System Prompt

You are a tool-use agent. First write a short Thought, then output a JSON block

only with keys: "action_weights" (probabilities for EVERY candidate tool, sum to 1) and "action_args" (arguments for EVERY candidate tool; if Finish prob is low, keep its args null). Do not output prose outside the Thought + JSON. Use only the provided tool names. If a tool returns an error or no data, please inform the user truthfully and do not make up information.

User Prompt

User Query: {query}
Tools: {tools_block}

Respond with a Thought line, then a JSON block. Include probabilities and arguments for EVERY tool.

Example:

Thought: I have checked the API health and it is operational. Now I need to retrieve the project list to answer the user's request. I should not finish yet because I haven't got the list.

```
{
  "action_weights": {
    "Tool_A": 0.1, "Tool_B": 0.1 ...
  },
  "action_args": {
    "Tool_A": {"q": "good_args_A"},
    "Tool_B": {"q": "good_args_B"},
    "Finish": {"final_answer": null}
  }
}
```

Instructions:
Ensure every candidate tool (including Finish) appears in action_weights and action_args. If Finish prob is low, keep its args empty/null; when you choose Finish, put the final_answer there.

G.2 Details in OASIS

G.2.1 Agent simulation setup

We evaluate AgentMark-F in OASIS (Yang et al., 2024b) on two platforms, a Twitter-like environment and a Reddit-like environment. For each platform, we collect 100 agent trajectories in total. Each run simulates 10 agents for 10 environment steps, including 5 agents with AgentMark-F enabled and 5 no-watermark control agents, and we repeat runs with different random seeds to reach the target trajectory count. All agents use the same base LLM (`deepseek-chat`) with a maximum context length of 16000 tokens, and the same scenario scripts and initialization rules across watermark and control groups. At each step, we log the elicited behavior distribution P_t over a fixed behavior list and the selected planning behavior, which are sufficient for AgentMark decoding and for computing Watermark Detection Rate.

The admissible behavior list is platform-specific. For the Twitter-like platform, the behavior set includes `create_post`, `like_post`, `repost`, `follow`, `quote_post`, and `do_nothing`. For the Reddit-like platform, the behavior set includes posting and commenting actions, vote actions, search and refresh actions, and social actions such as `follow` and `mute`. To simulate realistic social dynamics, we use a random follow graph in which each agent follows 3–5 other agents.

We define the step context used for reproducible randomness using a short recent interaction trace (the most recent 3 behaviors), which the verifier can reconstruct from the same logged records.

G.2.2 Prompt examples

Twitter Agent Prompt

System Message

Objective:
You're a Twitter user, and I'll present you with some posts. After you see the posts, choose some actions from the following functions.

Self-Description:
Your actions should be consistent with your self-description and personality. Your name is {name}. Your profile: {profile}.

Language Preference:
Please output all your comments, posts, and reasoning in Chinese.

Response Method:
Please perform actions by tool calling.

User Message

Context Observation:
You are observing a social media environment: {env_prompt}

Task Instruction:
Based on this observation and your profile, estimate the probability of performing each action. Return ONLY a JSON object with probabilities (must sum to 1.0).

Available Actions:
create_post, like_post, repost, follow, do_nothing, quote_post

Output Format:
{"action_name": probability, ...}

Example:

```
{
  "like_post": 0.3,
  "create_post": 0.25,
  "follow": 0.2,
  "do_nothing": 0.25
}
```

Reddit Agent Prompt

System Message

Objective:

```
You're a Reddit user, and I'll present you
with some posts. After you see the posts,
choose some actions from the following
functions.
```

Self-Description:

```
Your actions should be consistent with
your self-description and personality.
Your name is {name}. Your profile:
{profile}.
```

Language Preference:

```
Please output all your comments, posts,
and reasoning in Chinese.
```

Response Method:

```
Please perform actions by tool calling.
```

User Message

Context Observation:

```
You are observing a social media
environment: {env_prompt}
```

Task Instruction:

```
Based on this observation and your
profile, estimate the probability of
performing each action. Return ONLY a JSON
object with probabilities (must sum to
1.0).
```

Available Actions:

```
{actions_str}
```

Output Format:

```
{"action_name": probability, ...}
```

Example:

```
{
  "like_post": 0.3,
  "create_comment": 0.25,
  "follow": 0.2,
  "refresh": 0.25
}
```

H Robustness Experiment

H.1 Details of False Positives and Key Forgery

Goal. We quantify how often an invalid record passes verification under two settings: (i) unwatermarked logs, modeled by random GF(2) outputs, and (ii) wrong-key decoding, modeled by generating the RLNC coefficients with an incorrect key.

Data and coefficient construction. We aggregate 281 predicted trajectories from ToolBench (T1–T3) and extract 159 unique action indices. These indices are used to construct the GF(2) coefficient matrix A so that the sparsity pattern and coefficient distribution reflect the deployment setting rather than an idealized fully random matrix.

Verification rule. Let the payload length be $N = 128$ and let the total number of received packets be $m = N + k$ with overhead $k \in [0, 16]$. For each trial, we form $A \in \mathbb{F}_2^{m \times N}$ and a vector $y \in \mathbb{F}_2^m$. We accept if and only if the system $Ax = y$ is consistent over $\mathbb{F}_2$, i.e., $\mathrm{rank}(A) = \mathrm{rank}([A \mid y])$.

Overhead k	Unwatermarked	Wrong Key
0	60.60%	60.60%
2	22.70%	23.30%
4	7.00%	5.90%
6	1.10%	1.20%
8	0.40%	0.50%
≥ 14	0.00%	0.00%

Table 8: False-positive rates under the consistency-check verifier for unwatermarked logs and wrong-key decoding.

Simulation protocol. For each k, we run 1000 independent Monte Carlo trials. In the unwatermarked setting, y is sampled uniformly at random. In the wrong-key setting, A (and thus the induced equations) is generated using an incorrect key. We estimate FPR as the fraction of trials that pass the consistency check and report 95% confidence intervals using $\pm 1.96 \times \mathrm{SEM}$.

Results. Table 8 and Figure 4 show that both unwatermarked and wrong-key FPRs decrease rapidly with k and closely follow an exponential trend. Concretely, FPR drops from 60.6% at $k = 0$ to below 1% at $k = 8$ (0.40% unwatermarked; 0.50% wrong key), and we observe no false positives for $k \geq 14$ in 1000 trials.

H.2 Details of Robustness to Step Erasure and Truncation

Goal and setting. We study robustness under missing planning-time records by simulating step erasure on logged ToolBench trajectories. We focus on recovering an 8-bit deployment-level ID payload ($k = 8$) from incomplete logs, and compare RLNC-based coding with a repetition-code baseline under both single-episode and global decoding.

Step-erasure model. For each logged trajectory, we randomly and independently drop each decision step with probability $p \in [0, 1]$. When a step is erased, all packets/equations emitted at that step are removed together, yielding an observed subsequence consistent with log loss or truncation effects.

AgentMark-F with RLNC coding. Under RLNC, each surviving step contributes a variable

number of linear measurements (packets) depending on the per-step capacity. The verifier stacks all received measurements into a linear system over $\mathbb{F}_2$ and declares success if the recovered system has full rank (equivalently, if the decoded payload is uniquely determined). This directly matches our design goal that decoding depends primarily on the total received packet budget rather than fixed block alignment.

Repetition-code baseline. As a baseline, we repeatedly transmit the k payload bits across steps. We use a strict blind-decoding setting with no auxiliary indices/sequence numbers, and the decoder succeeds only if it can recover an intact length-k payload segment from the surviving packet stream (details below). This baseline is included to highlight the advantage of rateless linear measurements under step erasures.

Single-episode versus global decoding. We consider two verification regimes. *Single-episode decoding* attempts to decode the payload from each trajectory independently, which provides a conservative lower bound when logs are fragmented. *Global decoding* aggregates surviving packets across multiple trajectories into a single stream and decodes once, which emulates deployment-oriented auditing where a verifier pools logs over a time window and therefore benefits from cross-trajectory redundancy.

Task filtering for fair comparison. To ensure a fair robustness comparison, we filter trajectories so that both methods achieve 100% decoding success at $p = 0$. For RLNC, we select 25 sufficiently long trajectories that can support full recovery without erasure. For the repetition baseline, we filter to 19 long trajectories (e.g., requiring at least k packets/steps) so that blind decoding is feasible at $p = 0$.

Evaluation protocol and confidence intervals. For each erasure probability p, we repeat the random erasure process 30 times per trajectory and record whether decoding succeeds. This yields $25 \times 30 = 750$ trials for RLNC single-episode and $19 \times 30 = 570$ trials for repetition single-episode at each p. For global decoding, we repeat global aggregation under 30 independent erasure realizations. We report the mean decode success rate and a 95% confidence interval computed using the Wilson score interval for a Bernoulli proportion. We sweep p over $[0, 1]$ with a finer grid near the phase-transition region to visualize the sharp robustness boundary.

H.3 Semantic Rewriting

Details We report additional details for the semantic-preserving observation rewriting stress test used in Section 5.2. The goal is to quantify how sensitive planning-time behavior selection and behavioral watermark decoding are when the *semantics* of the environment observation is preserved but its surface form is rewritten, which can arise when raw logs are unavailable and only reconstructed records are retained.

We run the test on ALFWorld-OOD (134 tasks; 2326 steps) using `deepseek-chat` with temperature 1.0. At each step, we rewrite the `[CURRENT SITUATION]` portion of the observation with an LLM under the constraint that no objects or state information are added or removed, and only wording and syntax change. We evaluate under a teacher-forcing protocol that keeps the same original history/context at every step, so that the measured differences isolate the effect of observation rewriting rather than compounding trajectory drift.

Prompt example

Semantic Rewriting Prompt

System Prompt

```
You are a helpful assistant. Rewrite the
following text to convey exactly the same
information but using different words and
sentence structures. Do not add or remove
any facts. Keep it concise.
```

User Prompt

Input Text: `[The original observation context goes here]`

H.4 Adaptive Threat Models

The experiments in the main paper and appendix focus on random erasure, truncation, and semantic-preserving rewriting, where the attacker does not adapt to the watermarking algorithm itself. A stronger threat model is a fully adaptive adversary who knows the watermarking scheme and deliberately edits step context or trajectories to disrupt synchronization and reduce the number of usable measurements. We view such adaptive attacks as an important future direction, especially for settings where the adversary can observe verification failures and iteratively refine the perturbation strategy. Our current results under semantic rewriting

provide a first stress test in this direction, but they do not exhaust the adaptive threat space.

I Compatibility Experiment Details

Environment and trajectories. We conduct the compatibility experiment on ToolBench (Qin et al., 2023) using `Llama-3.2-3B-Instruct` as the base model. Each episode is capped at a maximum of 25 decision steps, and the agent may terminate earlier upon completion. We generate 149 trajectories in total, and log the planning-time behavior choices (for AgentMark-F decoding) together with the final action-level textual outputs (for content-watermark detection).

Watermarking configuration. We enable AgentMark-F as a behavior-level watermark by embedding multi-bit provenance into planning behaviors via distribution-preserving conditional sampling, while leaving the downstream action execution unchanged. Independently, we apply SynthID-Text (Dathathri et al., 2024) as an action-level (text) watermark on the final surface-form outputs produced by the agent. For SynthID-Text, we follow the open-source MarkLLM toolkit configuration (Pan et al., 2024) with distortionary watermark mode, n-gram length 5, 4 leaves, and 30 keys. For text generation in the watermark setting, we use temperature 1.0 as configured in MarkLLM.

Detection and decoding. We evaluate SynthID-Text detectability on the collected ToolBench outputs using the corresponding MarkLLM detector under the same configuration. The watermark detection rate is $96.64 \pm 0.06\%$. In parallel, we verify that AgentMark-F remains decodable from the logged planning-time behaviors and their associated behavior distributions, which demonstrates that enabling the action-level content watermark does not disrupt behavior-level provenance recovery. Overall, this setup tests composability in a realistic agent pipeline where behavior watermarking and content watermarking operate on different decision layers and are evaluated through their respective interfaces (trajectory logs versus final text outputs).