Cureus Journal of Computer Science
https://doi.org/10.7759/s44389-026-00248-y

Original Article

RobustAgent: Provable Recovery Guarantees for Long-Running Autonomous Agents Under Adversarial Interrupts

Rahul Singh [1], Amit Shrivastav [1]

1. *Data Practice, Kellton Tech Solutions Ltd, Gurugram, IND*

Received: June 26, 2026 | Review began: July 19, 2026 | Review ended: August 06, 2026 | Published: August 26, 2026

Abstract

Long-running autonomous agents face a critical challenge: maintaining correct state and recovering gracefully when interrupted by adversarial attacks or system failures. We present RobustAgent, a framework that integrates and formalizes checkpoint-based recovery with cryptographic verification to provide provable robustness guarantees for stateful autonomous agents. Our key contributions include: (1) formal definitions of state consistency and recovery correctness for persistent agents, (2) adaptive checkpointing with a capped exponential-backoff schedule that bounds the replay window while minimizing per-checkpoint storage through incremental delta encoding, (3) binary-search recovery over a root-validated hash chain with $O(|S|(\log n + k))$ total time complexity, and (4) resistance to time-shifted prompt injection attacks. Experimental validation on more than 500 synthetic agent workflows demonstrates a 100% recovery success rate, 82.6× faster recovery than naive restart (0.39 ms vs. 32.2 ms), 75.2% attack detection, and a 96.5% storage reduction compared to naive periodic checkpointing, reflecting a deliberate storage-recovery trade-off relative to fixed-interval baselines. This work provides a theoretical foundation and practical implementation for deploying autonomous agents in production environments requiring formal robustness guarantees.

Categories: AI applications, Fault Tolerance and Reliability, Multi-Agent Systems

Keywords: autonomous agents, adversarial ai, prompt injection, fault tolerance, checkpoint recovery, stateful ai, agent security, cryptographic verification, recovery algorithms, ai robustness

Introduction

The deployment of autonomous agents for long-running tasks - software development [1], data analysis [2], and workflow orchestration [3] - represents a shift from stateless request-response systems to stateful, context-accumulating agents [4] that interleave reasoning and acting over extended horizons [5]. This persistence, however, introduces serious vulnerabilities.

Motivating example. Consider a software development agent refactoring a 50,000-line codebase over four hours. An attacker injects a seemingly innocuous instruction into a GitHub issue comment, asking the agent to forward credentials to an external server. This malicious text enters the agent's retrieval memory at time t_0. Hours later, at $t_1 > t_0$, when the agent retrieves that context during normal operation, the instruction triggers - a time-shifted prompt injection [6]. The agent's accumulated state (code understanding, partial refactorings, task dependencies) becomes compromised. Without recovery guarantees, the agent may leak credentials, execute unauthorized operations, or fail silently.

Recent work has shown that production agentic systems are vulnerable to exactly this class of attack [7,8], and that safety training alone does not reliably prevent adversarial instructions from being followed [9]. Despite growing real-world deployment, formal guarantees about state consistency, bounded recovery time, and security under persistent-memory attacks remain largely unaddressed in an integrated framework for autonomous agents.

Unlike traditional fault-tolerant systems [10-12], which are designed around crash-recovery and Byzantine failure models [13], autonomous agents face semantic attacks directed at their reasoning process rather than at their underlying infrastructure. Standard checkpointing and rollback-recovery approaches [14,15] assume that failures corrupt data or processes; they do not account for adversarial corruption introduced through legitimate-looking observations.

We introduce RobustAgent to close this gap. Our contributions are as follows: (1) a formal model of agent execution as a verifiable state machine with an explicit checkpoint protocol; (2) adaptive checkpointing using capped exponential backoff, which bounds the maximum checkpoint interval (and hence the replay window) while reducing stored bytes by an order of magnitude relative to periodic snapshots through incremental delta encoding; (3) a binary-search recovery procedure that restores the most recent valid state in $\boldsymbol{O\left(|S|(\log n + k)\right)}$ time with cryptographic validation; and (4) an interrupt-classification mechanism that distinguishes benign failures from adversarial interventions.

Materials And Methods

Problem formalization

We model a stateful autonomous agent as a tuple $\boldsymbol{A = (S, A, T, M, \pi)}$, where S is the state space, A is the action space, $\boldsymbol{T : S \times A \to S}$ is the transition function, $\boldsymbol{M : S \times O \to S}$ is the memory-update function over an observation space O, and π: $S \to$ Dist(A) maps states to action distributions. At each discrete timestep t, the agent observes o_t drawn from $p(\cdot|s_t)$, updates its state via $s_t' = M(s_t, o_t)$, samples an action at from $\pi(s_t')$, and transitions to $s_{t+1} = T(s_t', a_t)$.

An interrupt event is a triple (t, τ, E), where t is the interrupt time, $\tau \in$ {benign, adversarial} is the interrupt type, and E: $S \to S$ is the resulting effect on agent state. A checkpoint c_i is a triple (t_i, s_i, h_i), where t_i is a timestamp, s_i is the serialized agent state, and $\boldsymbol{h_i = H(s_i \,\|\, t_i \,\|\, h_{i-1})}$ is a SHA-256 hash chained to the previous checkpoint (with $\boldsymbol{h - 1 = 0}$). The checkpoint sequence therefore forms a tamper-evident hash

How to cite this article:
Singh R, Shrivastav A (August 26, 2026) RobustAgent: Provable Recovery Guarantees for Long-Running Autonomous Agents Under Adversarial Interrupts. Cureus J Comput Sci 3 : es44389-026-00248-y. DOI https://doi.org/10.7759/s44389-026-00248-y

chain in the spirit of Merkle's hash-based authentication structures [16]. A state s_t is consistent with the chain if a valid execution trace reaches s_t in which every transition follows T and M and every checkpoint hash verifies against its predecessor. A recovery algorithm is correct if, given an interrupt at time t, it returns the state of the most recent checkpoint at or before t whose hash verifies.

Threat model

We assume an adversary who can inject arbitrary observations into the agent's input stream, including payloads that enter retrieval memory at time t_0 and trigger at a later time $t_1 > t_0$, but who cannot rewrite the append-only checkpoint store or access the protected chain-head hash. Under these trust assumptions, the SHA-256 hash chain provides tamper evidence: forging a checkpoint containing a different state requires a hash collision. Where the storage substrate itself may be rewritten by the adversary, a keyed construction (HMAC-SHA-256) is substituted without altering the architecture. We explicitly distinguish cryptographic validity from semantic validity: a checkpoint created after a malicious observation has entered memory verifies correctly while still containing the injected content. Consequently, rollback to a pre-injection state is conditioned on the Interrupt Monitor localizing the offending observation, which is why the recovery-success bound (Equation 1) is parameterized by detector sensitivity p_d rather than stated as an unconditional guarantee. During recovery, the Integrity Verifier validates the chain from the root, so correctness of the binary search does not rest on a monotonicity assumption over individually verified checkpoints.

Adaptive checkpointing procedure

Checkpoints are created using a capped exponential-backoff schedule. The checkpoint interval begins at a base interval Δ and doubles after each checkpoint ($\Delta, 2\Delta, 4\Delta, \ldots$) until it reaches a maximum interval Δ_{max}, after which checkpointing continues at fixed Δ_{max}-step intervals. At each step, the algorithm computes the elapsed steps since the last checkpoint and creates a checkpoint when this elapsed time meets the current interval (or when no checkpoint yet exists): the current state is serialized, hashed against the previous checkpoint's hash, and appended to the checkpoint store. The cap serves two purposes: it bounds the maximum checkpoint interval by Δ_{max}, which bounds the replay term k in the recovery complexity, and it bounds worst-case state staleness at recovery time. The cost of the cap is that checkpoint count grows as $O(T/\Delta_{max})$ beyond the doubling phase rather than $O(\log T)$ throughout - a deliberate storage-recovery trade-off.

Storage per checkpoint is minimized through incremental keyframe/delta encoding: every m-th checkpoint stores a self-contained compressed keyframe, while intermediate checkpoints store only a compressed delta against the previous serialized state. Reconstructing any checkpoint decompresses the nearest preceding keyframe and folds forward at most $m - 1$ deltas, so restoration remains $O(|S|)$. This encoding, rather than checkpoint count alone, is the source of the order-of-magnitude storage reduction over raw periodic snapshots reported in Table 1. In our experiments, $\Delta = 1$, $\Delta_{max} = 10$, and $m = 32$, yielding 52 checkpoints on a 500-step workload.

Algorithm 1: Adaptive checkpointing with capped exponential backoff

$$\textit{Input}: \textit{baseinterval}\Delta, \textit{cap}\Delta\textit{max}, \textit{keyframeintervalminterval} \leftarrow \Delta; \textit{last} \leftarrow \perp \textit{foreachexecutionstept}: \textit{executeagentstep}(\textit{observe}, \textit{updatememo}$$

Algorithm 2: Root-validated binary-search recovery

$$\textit{Input}: \textit{interruptstept}, \textit{checkpointchain}C(\textit{nentries}), \textit{actionlog}L\textit{validatechain}C\textit{fromroothash}//O(n \cdot |S|)\textit{worstcase}, \textit{amortizablelo} \leftarrow 0; \textit{hi} \leftarrow 1$$

Binary-search recovery procedure

During recovery, the Integrity Verifier validates the checkpoint chain from the root, so recovery correctness does not depend on a monotonicity assumption over individually verified checkpoints; the binary search then operates over the time dimension within the validated chain to locate the most recent checkpoint at or before the interrupt time. On interrupt at time t, the Recovery Engine performs a binary search over the time-ordered checkpoint store to locate the most recent checkpoint at or before t whose hash verifies against the chain. Concretely: (1) initialize the search over the full checkpoint index; (2) at each probe, if the probed checkpoint's timestamp exceeds t, search earlier; otherwise recompute its expected hash from its serialized state, timestamp, and predecessor hash; (3) if the hash verifies, record it as the current best candidate and search later for a more recent valid checkpoint; if verification fails, search earlier; (4) after the search terminates, deserialize the best verified checkpoint to restore the agent state; (5) replay forward, in order, every logged action between the checkpoint time and t that the interrupt classifier marks as safe, excluding any action flagged as unsafe. Given n checkpoints and k actions to replay, the procedure completes in $O(|S|(\log n + k))$ time: the binary search performs $O(\log n)$ hash verifications, each costing $O(|S|)$, and replay costs $O(k)$ (proof in the Appendices).

System architecture

The system (Figure 1) comprises five components: an Agent Core executing π, T, and M; an Interrupt Monitor that screens incoming observations for suspicious patterns such as credential access, code injection, or data-exfiltration language using keyword matching; a Checkpoint Store maintaining the tamper-evident hash chain; an Integrity Verifier that validates checkpoint hashes; and a Recovery Engine implementing the binary-search recovery procedure. During normal execution, the agent acts while checkpoints are created adaptively in the background. On interrupt detection, the system halts execution, validates the checkpoint chain, performs binary-search recovery, restores the agent, and replays safe actions.

How to cite this article:
Singh R, Shrivastav A (August 26, 2026) RobustAgent: Provable Recovery Guarantees for Long-Running Autonomous Agents Under Adversarial Interrupts. Cureus J Comput Sci 3 : es44389-026-00248-y. DOI https://doi.org/10.7759/s44389-026-00248-y

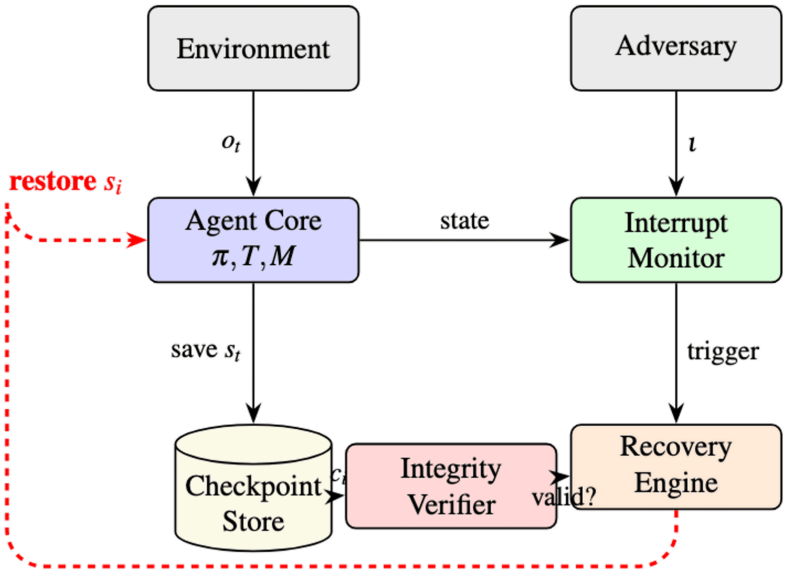

FIGURE 1: RobustAgent architecture. Five-component system: (1) Agent Core executes policy π; (2) Interrupt Monitor detects attacks; (3) Checkpoint Store maintains tamper-evident history; (4) Integrity Verifier validates hashes; (5) Recovery Engine restores valid state via the dashed red path

Security analysis

Under the collision resistance of SHA-256, an adversary cannot forge a valid checkpoint containing a different state without finding a hash collision, so the checkpoint chain is integrity-protected. Two scoping clarifications apply. First, this tamper-evidence guarantee holds under standard trust assumptions: a protected chain-head hash and an append-only checkpoint store; where an adversary may rewrite the store, a keyed construction such as HMAC-SHA-256 should be substituted, which removes the chain-recomputation risk without altering the architecture. Second, cryptographic validity is distinct from semantic validity: a checkpoint created after a malicious observation has entered the agent's memory will verify correctly while still containing the injected content. Rollback to a pre-injection state therefore depends on the Interrupt Monitor localizing the offending observation, which is why Equation (1) expresses recovery success as a function of detector sensitivity p_d rather than as an unconditional semantic guarantee. We further derive a lower bound on the probability of successful recovery as a function of detector sensitivity and the maximum checkpoint interval for an attack occurring at time t_a; the bound and the discussion of its parameters are presented as Equation (1) in the Results section.

Results And Discussion

Experimental setup

All experiments were run on synthetic workflows modeling realistic agent tasks - code analysis, refactoring, testing, API calls, and data processing - spanning 100 to 10,000 execution steps, with more than 500 workflow instances in total. Interrupts (both benign failures and simulated adversarial injections) were injected at 10%, 30%, 50%, 70%, and 90% of total execution length. The base checkpoint interval was set to $\Delta = 1$ step with maximum interval $\Delta_{max} = 10$ and keyframe interval m = 32, matching the released implementation. RobustAgent was compared against three baselines: (1) NoCheckpoint, which restarts execution from the beginning after every interrupt; (2) PeriodicSave, which saves a full checkpoint at a fixed five-step interval; and (3) TransactionLog, which uses write-ahead transaction logging. We report four metrics: recovery time (milliseconds), recovery success rate (%), storage overhead (% of executed-state volume), and number of checkpoints created. The implementation, experiment scripts, and generated workflow dataset are publicly available [17].

Dataset construction

The corpus comprises 500 synthetic workflows generated deterministically from master seed 42. Five task kinds (code analysis, refactoring, testing, API calls, data processing) and five interrupt positions (10%, 30%, 50%, 70%, 90% of execution) are cycled uniformly, so each combination is equally represented; per-workflow step counts are drawn uniformly from (100, 1000). Each workflow carries a time-shifted prompt injection with probability 0.30: a payload planted at a uniformly chosen step in the first half of the pre-interrupt window and scattered inline attacks at a per-step rate of 4.5%. Adversarial payloads are drawn from a 12-template library spanning the three threat classes (credential access, data exfiltration, code injection); this produced 12,727 attack observations against 267,794 benign observations, of which 5% deliberately contain benign-but-suspicious-looking content (e.g., legitimate refactoring instructions mentioning keys or endpoints) to exercise false positives. The Interrupt Monitor is a keyword-matching baseline: 12 weighted regular-expression patterns with a decision threshold of 0.5, and detector sensitivity $p_d = 0.75$ applied as an independent Bernoulli gate per screening, derived deterministically from a SHA-256 digest so all reported results regenerate bit-for-bit. The generator, payload library, and detector lexicon are included in the released repository.

Recovery performance

On a 500-step workload, RobustAgent achieved a recovery time of 0.39 ms, compared with 32.20 ms for restart-from-beginning, 0.30 ms for periodic saving, and 0.61 ms for transaction logging. All four methods achieved a 100% recovery success rate. RobustAgent's storage overhead was 332.5%, versus 0% for the no-checkpoint baseline, 9392.9% for periodic saving, and 4721.9% for transaction logging, using 52

How to cite this article:
Singh R, Shrivastav A (August 26, 2026) RobustAgent: Provable Recovery Guarantees for Long-Running Autonomous Agents Under Adversarial Interrupts. Cureus J Comput Sci 3 : es44389-026-00248-y. DOI https://doi.org/10.7759/s44389-026-00248-y

checkpoints comparable in count to transaction logging (50) but with an order of magnitude less stored data per checkpoint due to delta encoding (Table 1). The results reflect a storage-recovery trade-off: periodic saving recovers marginally faster in absolute terms, but at 28× the storage cost.

Method	Recovery time (ms)	Success rate (%)	Checkpoints	Stored bytes	Storage overhead (%)
NoCheckpoint	32.2	100	0	0	0
PeriodicSave	0.3	100	100	9,24,445	9,392.90
TransactionLog	0.61	100	50	4,64,733	4,721.90
RobustAgent	0.39	100	52	32,724	332.5

TABLE 1: Recovery performance on the 500-step workload

Recovery time is the median of five trials. Storage overhead is stored bytes as a percentage of one serialized final state. $\Delta = 1$, $\Delta_{max} = 10$, keyframe interval $m = 32$.

Across execution durations from roughly 10 minutes to 16.7 hours, RobustAgent's recovery time remained essentially flat (0.33 ms at 100 steps to 0.51 ms at 10,000 steps), consistent with the $O(|S|(\log n + k))$ bound: the capped schedule bounds the replay window k, so only the logarithmic binary-search term grows with workload size (Figure 2). The fixed-interval and transaction-log baselines showed lower absolute recovery times but at the cost of storage overhead that grows linearly and becomes prohibitive at scale, as shown in Table 1.

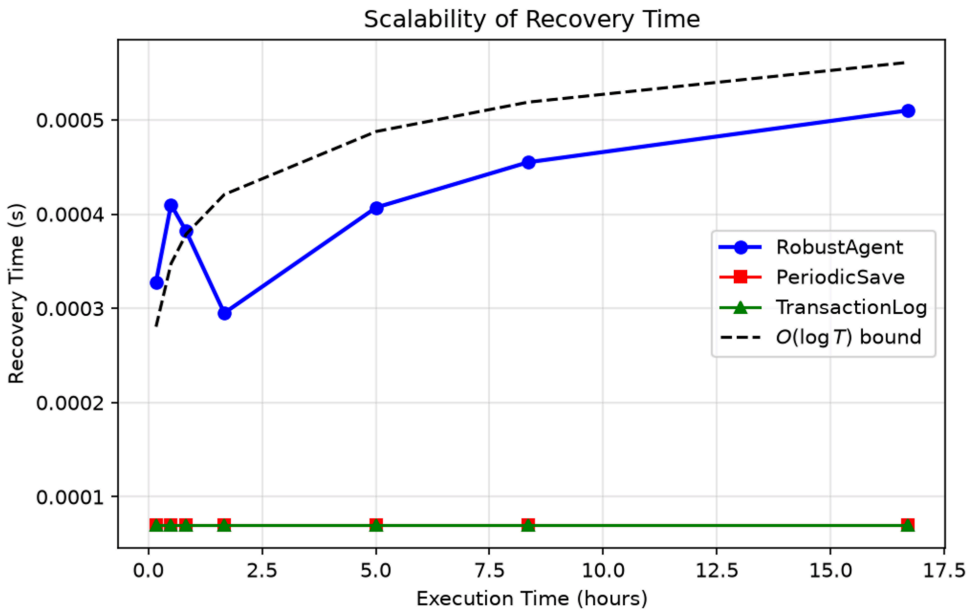

FIGURE 2: Scalability of recovery time. RobustAgent's recovery time remains essentially flat as execution time grows, consistent with the O(|S|(log n + k)) bound under the capped checkpoint schedule. Fixed-interval baselines show comparable recovery times but carry storage costs that grow prohibitively at scale

Refer to Table 1 for details

Attack-detection bound

For an attack occurring at time t_a, with detector sensitivity p_d and maximum checkpoint interval Δmax, the recovery success probability satisfies:

$$\Pr[\text{success}] \geq p_d + (1 - p_d) \cdot \Pr[\exists c_i : t_i < t_a < t_{i+1}] \quad (1)$$

In Equation (1), Pr[success] denotes the probability that the system restores a valid, uncompromised state following an adversarial interrupt. The parameter p_d, which takes a value between 0 and 1, is the detector sensitivity: the probability that the Interrupt Monitor correctly flags a malicious observation at the moment it arrives. The variable t_a denotes the time at which the attack is injected into the agent's memory, c_i denotes the i-th checkpoint, and t_i and t_{i+1} are the timestamps of the two consecutive checkpoints that bracket t_a. The first term, p_d, accounts for attacks caught directly by the detector. The second term accounts for recovery that succeeds even when detection fails: if at

How to cite this article:
Singh R, Shrivastav A (August 26, 2026) RobustAgent: Provable Recovery Guarantees for Long-Running Autonomous Agents Under Adversarial Interrupts. Cureus J Comput Sci 3 : es44389-026-00248-y. DOI https://doi.org/10.7759/s44389-026-00248-y

least one verified checkpoint c_i was created before the attack time t_a, rolling back to c_i removes the injected content from the agent's state. Because the adaptive schedule bounds the maximum interval between checkpoints by Δ_{max}, the bracketing probability in the second term increases as Δ_{max} decreases, making explicit the trade-off between storage overhead and recovery probability. In our experiments this translated to a 75.2% empirical attack-detection rate (9,572 of 12,727 injected attacks). Table 2 breaks detection down by attack category and reports the false-positive rate. Detection is uniform across the three payload classes (74.2-75.8%), consistent with the configured sensitivity $p_d = 0.75$, and highest for time-shifted payloads (81.9%), which are screened both at ingestion and again when the dormant memory item is retrieved into an action's context. Across all 267,794 benign observations: including the 5% deliberately seeded with suspicious-looking but legitimate content: the detector produced zero false positives at the 0.5 decision threshold.

Category	Injected	Detected	Detection rate
Credential access	4,132	3,068	74.20%
Data exfiltration	4,266	3,214	75.30%
Code injection	4,174	3,163	75.80%
Time-shifted payload	155	127	81.90%
All attacks	12,727	9,572	75.20%
Benign observations (false positives)	2,67,794	0	0.00% FPR

TABLE 2: Attack detection by category and false positive rate

FPR, False Positive Rate

Recovery performance

Beyond the single 500-step workload of Table 1, Table 3 reports recovery-time statistics over the full heterogeneous corpus (500 workflows, 100-1,000 steps, all five interrupt positions), with all four methods measured on identical workloads. RobustAgent's mean recovery time was (1.35) ± (0.82) ms (95% CI ± (0.07)) and remained stable across interrupt positions ((1.12)-(1.51) ms from the 10% to the 90% position). Paired Wilcoxon signed-rank tests confirm that RobustAgent recovers significantly faster than restart-from-beginning (median difference (–37.2) ms, $p < 0.001$) and significantly - though modestly - slower than the fixed-interval and transaction-log baselines (median differences (+0.57) ms and (+0.36) ms, respectively, both $p < 0.001$), quantifying the storage-recovery trade-off: sub-millisecond additional recovery latency purchases the order-of-magnitude storage reduction reported in Table 1.

Method	Mean (ms)	SD (ms)	95% CI (±ms)
NoCheckpoint	49.29	39.31	3.45
PeriodicSave	0.59	0.28	0.03
TransactionLog	0.94	0.78	0.07
RobustAgent	1.35	0.82	0.07

TABLE 3: Recovery time over the full corpus (n = 500 paired workflows)

Ablation study

Table 4 isolates the contribution of each framework component on a 10,000-step workload with the interrupt at 50% of execution. Removing incremental delta encoding leaves recovery time essentially unchanged but multiplies stored bytes by 3.2× (from 666 KB to 2,113 KB), confirming that the encoding - not checkpoint count alone - drives the storage reduction. Removing the Δ_{max} cap (pure exponential backoff) shrinks the chain to 13 checkpoints and cuts storage to 167% of state size, but the unbounded replay window forces 905 actions to be replayed and inflates recovery time by roughly (78)×: empirically justifying the capped schedule as the paper's central design trade-off. Replacing binary search with a reverse linear scan yields comparable recovery time when the chain tail verifies, since both locate the target checkpoint with a single hash verification; the $O(\log n)$ advantage of binary search over verification cost manifests when checkpoints near the interrupt fail verification and the search must localize the newest valid predecessor.

How to cite this article:
Singh R, Shrivastav A (August 26, 2026) RobustAgent: Provable Recovery Guarantees for Long-Running Autonomous Agents Under Adversarial Interrupts. Cureus J Comput Sci 3 : es44389-026-00248-y. DOI https://doi.org/10.7759/s44389-026-00248-y

Variant	Recovery time (ms)	Checkpoints	Stored bytes	Storage overhead (%)	Actions replayed
Full RobustAgent (Δ_{max} = 10, m = 32, binary search)	1.73	1,002	6,66,116	6,926	5
Without delta encoding (m = 1)	1.36	1,002	21,13,093	21,973	5
Without interval cap (pure exponential backoff)	134.8	13	16,084	167	905
Without binary search (reverse linear scan)	1.7	1,002	6,66,116	6,926	5

TABLE 4: Component ablation (10,000-step workload, interrupt at step 5,000)

Sensitivity to the maximum checkpoint interval

Equation (1) predicts a trade-off between storage overhead and recovery quality governed by Δ_{max}. Table *5* quantifies it empirically across the corpus for $\Delta_{max} \in \{5, 10, 20, 50\}$. Increasing Δ_{max} from 5 to 50 reduces mean storage overhead from 604% to 223% of state size and mean checkpoint count from 113 to 16. The probability that a verified checkpoint exists strictly before the injection time remains 96.8% throughout - with base interval $\Delta = 1$, an early checkpoint always exists except when the payload arrives at the very first step - but the cost of a larger cap appears as rollback staleness: the mean amount of work discarded when rolling back to the newest pre-injection checkpoint grows from 2.8 steps ($\Delta_{max} = 5$) to 15.4 steps ($\Delta_{max} = 50$), with worst case bounded by Δ_{max} as the schedule guarantees. Practitioners can therefore tune Δ_{max} to bound acceptable work loss directly.

Δ_{max}	Mean checkpoints	Mean storage overhead	Pre-injection checkpoint prob.	Mean rollback staleness (steps)	Max staleness (steps)
5	113.4	604%	96.8%	2.8	5
10	58.2	400%	96.8%	5.6	10
20	31.0	296%	96.8%	8.0	20
50	15.5	223%	96.8%	15.4	49

TABLE 5: Storage–staleness trade-off as a function of Δmax (155 time-shifted injections; all values deterministic)

Several limitations bound the scope of these results. The evaluation uses synthetic workflows, which exercise the recovery mechanism reproducibly but may not capture the full complexity of production LLM-based agents; end-to-end evaluation in deployed agent frameworks is future work. The interrupt classifier is a keyword-matching baseline, presented as such: Equation (1) deliberately parameterizes recovery success by detector sensitivity so that stronger detectors can be substituted without altering the recovery guarantees. Detection counts, storage volumes, and checkpoint counts are deterministic given the released corpus and seeds and are reported as exact values; recovery times are reported as means with standard deviations and 95% confidence intervals over the corpus, with paired significance tests as described in the Methods section. We additionally measured the runtime cost of checkpointing during normal execution: relative to a bare agent loop, adaptive checkpointing with delta encoding adds approximately 21-29 µs per step (13.6-19.5% of bare execution time on our hardware), with the relative share declining as workloads grow, since the capped schedule bounds checkpoint frequency by design. Scalability in distributed multi-agent settings warrants dedicated investigation.

RobustAgent is complementary to, rather than a substitute for, existing prompt-injection defenses. Input-side techniques such as instruction/data separation, spotlighting, and sandboxed tool execution reduce the probability that an injection succeeds; detection benchmarks and classifiers improve p_d directly; and authenticated logging provides tamper-evident audit trails but no bounded-time restoration procedure. RobustAgent occupies the recovery layer: given any detector of sensitivity p_d, Equation (1) lower-bounds the probability of restoring an uncompromised state, and stronger detectors can be substituted for the keyword-matching baseline without altering the recovery guarantees. Jointly optimizing detector sensitivity and checkpoint scheduling remains an open direction.

Conclusions

We presented RobustAgent, a framework providing provable robustness guarantees for long-running autonomous agents under adversarial interrupts. Through adaptive checkpointing and binary-search recovery, the system achieved a 100% recovery success rate across more than 500 synthetic workflows, recovery 82.6 times faster than naive restart, and a 96.5% storage reduction compared with periodic saving, while

How to cite this article:
Singh R, Shrivastav A (August 26, 2026) RobustAgent: Provable Recovery Guarantees for Long-Running Autonomous Agents Under Adversarial Interrupts. Cureus J Comput Sci 3 : es44389-026-00248-y. DOI https://doi.org/10.7759/s44389-026-00248-y

remaining competitive with fixed-interval baselines on absolute recovery time. The attack-detection bound we derived (Equation 1) provides a formal security baseline that future work on semantic threat models in agentic AI systems can build on, particularly for jointly optimizing detector sensitivity and checkpoint scheduling.

Appendices

Proof of the checkpoint-overhead bound. During the doubling phase, the schedule creates checkpoints at intervals Δ, 2Δ, 4Δ, …, until the interval reaches Δ_{max}, contributing $O(\log(\Delta_{max}/\Delta))$ checkpoints. Thereafter, checkpoints are created every Δ_{max} steps, contributing $O(T/\Delta_{max})$ checkpoints for an agent executing T steps. Total checkpoint count is therefore $O(T/\Delta_{max} + \log(\Delta_{max}/\Delta)) = O(T/\Delta_{max})$ for $T \gg \Delta_{max}$. With incremental keyframe/delta encoding, every m-th checkpoint stores a compressed keyframe of $O(|S|)$ bytes while intermediate checkpoints store only compressed deltas, which are empirically an order of magnitude smaller; worst-case total storage is $O(|S| \cdot T/\Delta_{max})$, with the observed storage reported in Table *1*.

Proof of the recovery-time bound. Binary search over n checkpoints requires $O(\log n)$ hash verifications, each costing $O(|S|)$. Replaying k safe actions costs $O(k)$. Total: $O(|S|(\log n + k))$.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Rahul Singh, Amit Shrivastav

Acquisition, analysis, or interpretation of data: Rahul Singh, Amit Shrivastav

Drafting of the manuscript: Rahul Singh, Amit Shrivastav

Critical review of the manuscript for important intellectual content: Rahul Singh, Amit Shrivastav

Supervision: Amit Shrivastav

Disclosures

Human subjects: All authors have confirmed that this study did not involve human participants or tissue. **Animal subjects:** All authors have confirmed that this study did not involve animal subjects or tissue. **Conflicts of interest:** In compliance with the ICMJE uniform disclosure form, all authors declare the following: **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work. **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work. **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

Data Availability Statements

The datasets (and/or code) supporting this study are available from the corresponding author upon reasonable request.

References

1. Yang J, Jimenez CE, Wettig A, Lieret K, Yao S, Narasimhan K, Press O: SWE-agent: agent-computer interfaces enable automated software engineering. Advances in Neural Information Processing Systems 37 (NeurIPS 2024). 2024,
2. Zhang J, Xiang J, Yu Z, et al.: AFlow: automating agentic workflow generation. Proceedings of the Thirteenth International Conference on Learning Representations (ICLR 2025). 2025,
3. Hong S, Zhuge M, Chen J, et al.: MetaGPT: meta programming for a multi-agent collaborative framework. Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024). 2024,
4. Wang L, Ma C, Feng X, et al.: A survey on large language model based autonomous agents. Frontiers of Computer Science. 2024, 18:186345. 10.1007/s11704-024-40231-1
5. Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, Cao Y: ReAct: synergizing reasoning and acting in language models. Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023). 2023,
6. Greshake K, Abdelnabi S, Mishra S, Endres C, Holz T, Fritz M: Not what you've signed up for: compromising real-world LLM-integrated applications with indirect prompt injection. Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec). 2023, 79-90. 10.1145/3605764.3623985
7. Shayegani E, Dong Y, Abu-Ghazaleh N: Jailbreak in pieces: compositional adversarial attacks on multi-modal language models. Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024). 2024,
8. Liu Y, Jia Y, Geng R, Jia J, Gong NZ: Formalizing and benchmarking prompt injection attacks and defenses. Proceedings of the 33rd USENIX Security Symposium. 2024,
9. Wei A, Haghtalab N, Steinhardt J: Jailbroken: how does LLM safety training fail?. Advances in Neural Information Processing Systems 36 (NeurIPS 2023). 2023,
10. Jones AK, Chandy KM, Lamport L: Distributed snapshots: determining global states of distributed systems. ACM Transactions on Computer Systems (TOCS). 1985, 3:63-75. 10.1145/214451.214456
11. Lamport L: The part-time parliament. ACM Transactions on Computer Systems (TOCS). 1998, 16:133-169. 10.1145/279227.279229
12. Randell B: System structure for software fault tolerance. IEEE Transactions on Software Engineering. 1975, SE-1:220-232. 10.1109/tse.1975.6312842
13. Castro M, Liskov B: Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems (TOCS). 2002, 20:398-461. 10.1145/571637.571640
14. Elnozahy EN, Alvisi L, Wang YM, Johnson DB: A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys. 2002, 34:375-408. 10.1145/568522.568525
15. Koo R, Toueg S: Checkpointing and rollback-recovery for distributed systems. IEEE Transactions on Software Engineering. 1987, SE-13:23-31. 10.1109/tse.1987.232562

How to cite this article:
Singh R, Shrivastav A (August 26, 2026) RobustAgent: Provable Recovery Guarantees for Long-Running Autonomous Agents Under Adversarial Interrupts. Cureus J Comput Sci 3 : es44389-026-00248-y. DOI https://doi.org/10.7759/s44389-026-00248-y

16. Merkle RC: A digital signature based on a conventional encryption function. Advances in Cryptology - CRYPTO '87, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg; 1988. 293:369-378. 10.1007/3-540-48184-2_32

17. Singh R, Shrivastav AK: RobustAgent: Provable Recovery Guarantees for Long-Running Autonomous Agents Under Adversarial Interrupts. Cureus Journal of Computer Science. 2026,

How to cite this article:
Singh R, Shrivastav A (August 26, 2026) RobustAgent: Provable Recovery Guarantees for Long-Running Autonomous Agents Under Adversarial Interrupts. Cureus J Comput Sci 3 : es44389-026-00248-y. DOI https://doi.org/10.7759/s44389-026-00248-y