Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Runtime Security Monitoring and Continuous Assurance for AI Agents: An Integrative Review and Systems-Management Framework

Yaswanth Reddy Vada

gvada09095@ucumberlands.edu

University of the Cumberlands

Aashrith Mannuru

University of Texas at Dallas

Krishna Varsha Mummadireddy

University of North Texas

Nitya Santhoshini Vada

Indiana Institute of Technology

Nishith Reddy Mannuru

University of North Texas

Systematic Review

Keywords: AI agents, runtime security monitoring, continuous assurance, runtime verification, LLM security, systems management

Posted Date: November 11th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11422609/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Abstract

Large language model (LLM) agents increasingly operate as long-lived software actors that plan, invoke tools, access enterprise data, maintain memory, and delegate work. Their security posture can therefore change after deployment as task context, external content, tools, credentials, memory, model versions, and peer agents change. Pre-deployment evaluation alone cannot establish that an agent remains safe during execution. This paper presents a structured integrative review of runtime verification, continuous security monitoring, AI-agent security benchmarks, runtime guardrails, and autonomous-systems assurance, and synthesizes these literatures into a Runtime Security Monitoring and Continuous Assurance (RSMCA) framework for AI agents. The review distinguishes observability, runtime monitoring, runtime enforcement, and continuous assurance as related but non-equivalent functions. It identifies eight runtime monitoring surfaces: identity and configuration, input/context, task and planning state, tool invocation, data/egress, memory and retrieval, inter-agent delegation, and external dependencies. RSMCA represents each agent action as a correlated security event, evaluates it using complementary deterministic, semantic, verifier-based, and probabilistic monitors, and maps monitor output to graduated responses ranging from allow-with-audit to step-up approval, blocking, quarantine, rollback, and credential revocation. A continuous assurance state tracks monitor coverage, policy freshness, behavior risk, dependency health, incident state, and provenance completeness across time. The framework is designed for network and systems management settings in which agent security must be observable, enforceable, recoverable, and auditable. We conclude with implementation guidance, evaluation metrics, and a research agenda for measurable runtime assurance of autonomous AI services.

1. Introduction

Large language models are rapidly moving from conversational interfaces toward agentic systems that can select actions, invoke software tools, interact with application programming interfaces (APIs), retrieve external information, maintain memory, and coordinate with other agents. Surveys of LLM-based autonomous agents describe planning, memory, tool use, and environmental interaction as recurring architectural components [1]. ReAct demonstrated that reasoning and action can be interleaved within an LLM workflow [2], while Toolformer showed that language models can learn when and how to call external tools [3]. These capabilities are operationally useful, but they also change the security problem: a model output is no longer only information to be consumed by a person; it can become a machine action that changes system state.

This transition makes runtime behavior a first-class systems-management concern. Traditional software services can fail or be compromised, but their permitted behavior is typically bounded by code paths, identities, configurations, and service interfaces defined before execution. Agentic systems add a probabilistic decision component that can select intermediate steps at runtime and can reinterpret its task after receiving new observations. A correctly authenticated agent may therefore become unsafe without any credential compromise. Indirect prompt injection can alter an agent's apparent objective

through untrusted content [13,15,16]; tool-using agents can execute high-impact or privacy-sensitive actions [14]; memory and inter-agent state can propagate security failures [19,36]; and external tool ecosystems create dependencies that can change independently of the model [35].

The core management question is consequently temporal: not only “Was this agent safe when it was deployed?” but “Is this agent still operating within authorized and acceptable bounds now, and can the system intervene before unsafe behavior becomes consequential?” NIST information-security continuous monitoring (ISCM) treats ongoing awareness of vulnerabilities, threats, and control effectiveness as a necessary basis for risk decisions [5,6]. NIST AI risk guidance similarly emphasizes lifecycle risk management and monitoring of AI systems after deployment [8,9]. However, agentic systems complicate continuous monitoring because relevant state is distributed across identity, instructions, task progress, tools, arguments, data flows, memory, delegated agents, and the environment.

A growing body of research has begun to address pieces of this runtime problem, and recent survey work consolidates the emerging security duality of LLM agents [29]. Runtime verification provides a formal tradition for checking execution traces against specified properties [10]. Runtime-assurance work in autonomous systems separates advanced but uncertain controllers from independently enforced safety mechanisms [11,12]. More recent LLM-agent defenses introduce task-alignment monitors [21], knowledge-enabled guard agents [20], adaptive lifelong guardrails [23], customizable runtime policies [24], reflection-driven sidecars [26], step-level tool safety controls [27], and proactive probabilistic monitors [28]. At the same time, agent observability research is beginning to formalize trace-level fault detection [25]. These contributions are important, but they are usually evaluated as individual defenses or benchmarks rather than assembled into a systems-management model of continuous assurance.

This paper addresses that gap through a structured integrative review and conceptual synthesis. The central contribution is a Runtime Security Monitoring and Continuous Assurance (RSMCA) framework that treats an AI agent as a continuously managed software actor. Rather than relying on a single safety classifier, RSMCA combines event instrumentation, state correlation, multiple monitor classes, graduated enforcement, recovery, and an assurance evidence loop. The paper makes five contributions: (1) it distinguishes observability, monitoring, enforcement, and continuous assurance; (2) it develops a taxonomy of runtime security surfaces and signals; (3) it synthesizes representative runtime defenses according to where and how they intervene; (4) it proposes an architecture and event model for continuous agent assurance; and (5) it defines measurable evaluation criteria and a research agenda for network and systems management research.

The scope is intentionally security and systems management rather than model alignment in the broad normative sense. The framework does not attempt to determine whether every agent objective is ethically desirable or whether every generated answer is factually correct. It focuses on whether observable execution remains within explicitly authorized, monitored, recoverable, and auditable bounds.

This distinction is important for deployable infrastructure: organizations need controls that remain effective even when the underlying model is uncertain, manipulated, updated, or wrong.

2. Review Methodology

This study uses a structured integrative-review design rather than a PRISMA systematic review. Integrative reviews are appropriate when a research question spans heterogeneous evidence types and the goal is conceptual synthesis rather than pooled statistical estimation or exhaustive enumeration. The evidence base for runtime agent assurance includes peer-reviewed computer-science papers, conference benchmarks, systems and software engineering research, NIST standards and guidance, and practitioner security frameworks. The review therefore prioritizes traceability of claims and relevance to the proposed framework without reporting database counts or screening statistics that were not generated.

2.1. Scope and Evidence Sources

The primary review window covers 2020 through 9 September 2026, with earlier foundational work included for runtime verification and continuous monitoring. Sources were identified through targeted searches of SpringerLink, ACM Digital Library, ACL Anthology, PMLR, ICLR/NeurIPS proceedings, NIST repositories, and official security-project documentation. Search concepts combined four clusters: (1) “AI agent”, “LLM agent”, “autonomous agent”, and “tool-using agent”; (2) “runtime monitoring”, “runtime verification”, “runtime enforcement”, “guardrail”, “continuous monitoring”, and “runtime assurance”; (3) “prompt injection”, “tool misuse”, “memory poisoning”, “agent security benchmark”, “delegation”, and “data exfiltration”; and (4) “observability”, “telemetry”, “incident response”, “policy enforcement”, and “continuous assurance”.

Peer-reviewed or formally accepted versions were preferred when available. Preprints were not used as core evidence when a conference or journal version could be verified. Government publications were included where they define continuous-monitoring, cybersecurity, zero-trust, or AI-risk-management practices. Practitioner material was used sparingly and only when it represented an established community framework, such as the OWASP GenAI Security Project [30].

2.2. Synthesis Procedure

The synthesis proceeded in five passes. First, continuous-monitoring and runtime-assurance concepts were extracted from NIST guidance and the runtime-verification/autonomous-systems literature [5,6,10–12]. Second, representative agent attacks and benchmarks were grouped by the execution surface they affect [13–19]. Third, runtime defenses were classified by observable input, decision logic, intervention point, and temporal horizon [20–28]. Fourth, the resulting controls were mapped to network and systems management functions including discovery, configuration, telemetry, policy enforcement, fault/security event correlation, response, recovery, and audit. Finally, gaps across these literatures were converted into the RSMCA architecture and evaluation framework. RSMCA is therefore a conceptual systems-

management framework derived from literature synthesis; it is not presented as an experimentally validated product. Table 1 maps the five review questions to their evidence focus and corresponding synthesis outputs.

Table 1
Review questions and synthesis outputs.

Research question	Evidence focus	Output in this paper
RQ1. What makes runtime security for AI agents different from pre-deployment safety evaluation?	Agent architectures; attacks; continuous-monitoring and runtime-verification foundations	Conceptual distinctions and runtime threat model
RQ2. Which execution surfaces must be monitored during agent operation?	Prompt injection, tools, memory/RAG, delegation, identity/configuration, external dependencies	Runtime monitoring-surface taxonomy
RQ3. What runtime detection and intervention approaches exist?	GuardAgent, Task Shield, AGrail, AgentSpec, Safety Sidecar, ToolSafe, ProbGuard, AgentTelemetry	Comparative synthesis of monitor classes
RQ4. What telemetry is required for actionable and auditable monitoring?	Systems-management observability; agent traces; policy context	Correlated event model and telemetry requirements
RQ5. How can monitoring become continuous assurance rather than isolated alerting?	ISCM, runtime assurance, incident response, lifecycle risk management	RSMCA architecture, assurance state, lifecycle and metrics

3. Conceptual Foundations

3.1. Observability, Monitoring, Enforcement, and Assurance

Four terms are often used interchangeably in discussions of agent safety, but they describe different capabilities. Observability is the ability to reconstruct internal or external system behavior from telemetry. Runtime monitoring evaluates events or traces while an agent is executing and produces a judgment about state, behavior, or risk. Runtime enforcement adds the authority to alter execution—for example by rejecting a tool call, requiring human approval, reducing privilege, or terminating a task. Continuous assurance is broader: it is the ongoing, evidence-based judgment that the agent remains within acceptable operational and security bounds as behavior, configuration, dependencies, and threats change.

The distinction matters because a system can be highly observable but weakly protected. Detailed traces collected after an unsafe transaction help incident investigation but do not prevent the transaction. Conversely, an aggressive blocking rule can provide enforcement without adequate observability, leaving operators unable to explain why legitimate work was denied. Continuous assurance

requires both: sufficient telemetry to justify decisions and sufficient control to change the future execution path. Table 2 compares these four functions in terms of their primary questions, typical outputs, and consequences when absent.

Table 2
Distinguishing runtime security functions.

Function	Primary question	Typical output	Failure if absent
Observability	What happened and what state produced it?	Traces, logs, metrics, provenance	Operators cannot reconstruct behavior or correlate causes
Runtime monitoring	Is current or predicted behavior acceptable?	Verdict, anomaly/risk score, property violation	Unsafe behavior is not detected during execution
Runtime enforcement	What should the system do about the verdict?	Allow, constrain, approve, block, quarantine, revoke	Detection becomes advisory and may not prevent harm
Continuous assurance	Does accumulated evidence support continued operation?	Assurance state, coverage and residual-risk evidence	Controls can silently degrade as agents and dependencies change

3.2. Runtime Verification and Runtime Assurance

Runtime verification provides a useful formal foundation because it evaluates a running system against specified properties using execution traces [10]. Its central advantage is temporal specificity: a property can be evaluated against what the system is doing now rather than inferred solely from design-time testing. Autonomous-systems assurance extends this idea by separating advanced autonomous functionality from independent safety mechanisms and by using monitoring and fallback behaviors to preserve safety when the advanced controller becomes unreliable [11,12].

LLM agents differ from traditional cyber-physical controllers because many relevant properties are semantic rather than purely numeric. A tool call may be syntactically valid and authorized by role but inconsistent with the user's task. A retrieved document may be legitimate content but contain an instruction that hijacks the agent. An agent may remain within each single-step permission while accumulating a dangerous multi-step trajectory. Runtime assurance for agents therefore requires hybrid monitors: deterministic specifications where behavior can be precisely expressed, semantic/task-alignment checks where meaning matters, and history-aware or probabilistic methods where risk emerges across time.

3.3. Continuous Monitoring as a Management Function

NIST SP 800 – 137 defines information-security continuous monitoring as maintaining ongoing awareness of information security, vulnerabilities, and threats to support risk-management decisions [5]. NIST SP 800-137A further addresses assessment of continuous-monitoring programs and their effectiveness [6]. These principles generalize to AI agents: monitoring is not merely the collection of logs

but a managed capability with defined objectives, coverage, frequencies, analysis, reporting, and response. The monitored object, however, must be expanded from conventional host/network/control state to agent-specific task and execution state.

This expansion is consistent with broader NIST lifecycle risk management. The Cybersecurity Framework 2.0 emphasizes governance, identification, protection, detection, response, and recovery across organizational cybersecurity activities [7], while the AI RMF and its Generative AI Profile emphasize ongoing measurement and management of AI risks [8,9]. RSMCA operationalizes these ideas at the agent execution layer by making runtime evidence actionable before, during, and after consequential actions.

4. Runtime Threat and Monitoring Surfaces

A runtime monitor is useful only if it observes the state necessary to discriminate acceptable from unacceptable behavior. For AI agents, that state is distributed. The framework therefore identifies eight monitoring surfaces. These surfaces are not mutually exclusive; a single attack can cross several of them. Indirect prompt injection, for example, begins at the context surface, changes task/planning state, induces a tool action, and may culminate in data egress. The purpose of the taxonomy is to ensure that observability and policy do not stop at the model boundary.

4.1. Identity and Configuration State

Agent identity establishes which software principal is executing, but runtime security also depends on configuration: model and agent-framework version, system prompt or policy version, enabled tools, credential scope, deployment environment, and lifecycle state. NIST's 2026 concept work on software and AI-agent identity explicitly highlights identification, authentication, authorization, auditing, and non-repudiation for agents [31]. Runtime monitoring should treat unexpected version changes, privilege expansion, tool registration changes, and credential anomalies as assurance-relevant events rather than static inventory differences.

4.2. Input and Context

Agents routinely process untrusted natural-language content. Greshake et al. demonstrated that indirect prompt injection can place adversarial instructions in external content consumed by LLM-integrated applications [13]. InjecAgent and AgentDojo extend this problem to tool-integrated agents and dynamic task environments [15,16]. Runtime controls should therefore preserve provenance for external context and distinguish task authority from untrusted observations. A monitor should be able to ask not only whether a prompt contains a suspicious phrase but whether a later action is causally consistent with the authorized task despite the newly observed content.

4.3. Task and Planning State

The task/planning surface captures goal drift, unauthorized subgoals, abnormal plan expansion, and divergence between the delegated objective and the agent's proposed actions. Task Shield directly motivates this surface by evaluating whether instructions and tool calls contribute to the user's task [21]. Because hidden chain-of-thought should not be treated as a dependable audit artifact, monitoring should rely on externalized task state, tool requests, intermediate artifacts, and policy-relevant explanations rather than requiring access to private model reasoning.

4.4. Tool Invocation and Action Semantics

Tool use is the point where model uncertainty becomes operational impact. ToolEmu shows that tool-using agents can exhibit consequential failures even in simulated environments [14]. AgentSpec and ToolSafe demonstrate complementary approaches to runtime control of actions and tool invocations [24,27]. Monitoring must therefore capture method, arguments, target resource, expected side effects, task association, policy decision, and actual outcome. Permission to access an API is not sufficient evidence that every method/argument combination is appropriate for the current task.

4.5. Data Access and Egress

Data security requires visibility into both what enters agent context and what leaves the protected environment. Prompt-injection benchmarks show that untrusted observations can induce disclosure or misuse of sensitive data [15,16]. Runtime monitoring should attach data classifications to retrieved fields, track whether an action causes external disclosure, and distinguish ordinary model text from egress through tools such as messaging, file upload, HTTP requests, or code execution. Egress monitoring is especially important when an agent is authorized to read sensitive information for one purpose but can communicate through multiple outbound channels.

4.6. Memory and Retrieval

Persistent memory and retrieval-augmented generation create temporal coupling between sessions. Agent Security Bench includes memory-oriented attacks, reflecting the possibility that malicious or erroneous state can persist beyond the event that introduced it [19]. Runtime assurance should therefore record memory writes, retrieval sources, provenance, retention scope, and later actions influenced by persistent state. Suspicious memory entries should be quarantinable independently of the agent identity, and assurance should be reevaluated after retrieval corpora or memory policies change.

4.7. Inter-Agent Delegation

Multi-agent systems add communication, delegation, and propagation paths. G-Safeguard demonstrates that adversarial effects can propagate through interaction topologies [36]. Monitoring should track the delegation graph, parent task, authority transferred, messages exchanged, and resulting actions. A child agent that receives broader authority than its parent, or a delegation chain that exceeds a configured depth, should be detectable as a policy and assurance anomaly. This surface also turns provenance into a graph problem rather than a flat log problem.

4.8. External Tools, MCP Servers, and Dependencies

Agent behavior depends on external tools and interoperability layers that can change without modifying the model. Research on the Model Context Protocol (MCP) identifies lifecycle and supply-chain threats around tool/server creation, deployment, operation, and maintenance [35]. Runtime assurance must therefore include dependency identity, version, integrity/attestation where available, allowlist state, behavioral changes, and revocation. A previously approved tool should not remain implicitly trusted if its endpoint, capabilities, publisher, or observed behavior changes. Table 3 summarizes the eight monitoring surfaces, their representative risks, minimum runtime signals, and illustrative responses.

Table 3
Runtime monitoring surfaces, signals, and response examples.

Surface	Representative risks	Minimum runtime signals	Illustrative response
Identity/configuration	Impersonation; stale or changed model/tool configuration; privilege expansion	Agent identity; model/runtime version; policy version; credential scope; lifecycle state	Re-attest; reduce privilege; revoke token; quarantine instance
Input/context	Direct/indirect prompt injection; untrusted observations	Source/provenance; trust label; retrieval origin; instruction/task boundary	Mark untrusted; constrain downstream action; require task consistency check
Task/planning	Goal hijack; unauthorized subgoal; plan expansion	Task ID; authorized objective; proposed action; task progress; prior verdicts	Revise plan; step-up approval; block divergent action
Tool/action	Unsafe method/arguments; irreversible side effects	Tool identity; method; arguments; target; side effects; result	Allow/audit; argument rewrite; block; approval gate
Data/egress	Over-broad access; exfiltration; cross-case disclosure	Data labels; fields retrieved; recipient/channel; output classification	Minimize fields; redact; block egress; isolate session
Memory/RAG	Poisoning; stale state; malicious persistence	Memory read/write; retrieval ID; provenance; retention scope	Reject write; quarantine memory; rebuild context; re-evaluate
Delegation	Privilege amplification; compromised sub-agent; cascading failure	Parent/child IDs; delegated task; authority subset; depth; interaction graph	Attenuate authority; isolate child; sever delegation; revoke chain
External dependencies	Malicious/changed tool or MCP server; supply-chain drift	Server/tool identity; endpoint; version; integrity; allowlist; behavior baseline	Disable tool; quarantine dependency; fail over; reassess assurance

5. Telemetry and Observability Requirements

5.1. The Need for Correlated Agent Traces

Conventional application logs are often too fragmented for agent assurance. A single agent action may be influenced by a user request, retrieved document, memory entry, model version, system policy, tool response, delegated sub-agent, and prior denial. If these events cannot be correlated, operators may know that a sensitive API was called without knowing which task authorized it or which untrusted context caused it. AgentTelemetry's focus on trace-based observability and fault detection illustrates the need for structured agent-level telemetry rather than isolated text logs [25].

RSMCA therefore models each consequential event at time t as an event vector E_t = < agent, session, task, sponsor/role, model/runtime version, action, tool, arguments, resource, data labels, context provenance, memory/retrieval references, delegation path, policy version, monitor verdicts, outcome>. This vector is conceptual rather than a mandated wire format. Its purpose is to identify the information required to evaluate action legitimacy and reconstruct causality. Implementations can encode the fields in existing tracing or event infrastructures.

5.2. Temporal and Causal Context

Single-event monitoring is insufficient when risk emerges across a sequence. Repeated low-risk reads can become bulk extraction; individually valid tool calls can form an unauthorized transaction; a harmless memory write can influence a later privileged action. Monitoring should therefore preserve ordering, parent-child relationships, session/task boundaries, and causal links where they are observable. Temporal windows should be long enough to detect cumulative behavior but bounded enough to support real-time decisions.

5.3. Telemetry Quality as an Assurance Property

Monitoring depends on the integrity and completeness of telemetry. A monitor that silently loses tool-call events can produce a reassuring verdict while observing only part of the execution. Continuous assurance should therefore track instrumentation coverage, event loss, clock or ordering anomalies, monitor availability, schema/version compatibility, and tamper evidence. "No alerts" is not evidence of safety when the monitoring path itself is unhealthy. Table 4 presents the proposed minimum telemetry schema and explains the assurance purpose of each category.

Table 4
Proposed minimum telemetry schema for runtime agent assurance.

Telemetry category	Representative fields	Why it matters
Identity and lifecycle	Agent instance ID; sponsor/role; credential ID/scope; lifecycle state; attestation status	Binds actions to a managed principal and authority source
Task and policy	Task ID; authorized objective; policy version; autonomy ceiling; approval requirements	Provides external representation of authorized purpose
Model/configuration	Model/runtime version; framework version; system configuration hash; enabled tools	Detects drift that can invalidate prior assurance evidence
Context provenance	User input; retrieval/memory references; source/trust labels; external-content identifiers	Supports injection analysis and causal reconstruction
Action/tool	Tool/server identity; method; normalized arguments; target; side-effect class	Enables action-level policy and anomaly detection
Data flow	Resource/data classification; fields accessed; egress recipient/channel; redaction state	Enables least-data and exfiltration monitoring
Delegation	Parent agent; child agent; delegated task; authority subset; chain depth	Makes multi-agent authority and propagation auditable
Monitor/verdict	Monitor ID/version; feature/state snapshot; verdict; score/confidence where applicable	Supports reproducibility and monitor accountability
Outcome/response	Action result; policy decision; human approval; containment/revocation; recovery status	Connects detection to operational effect and recovery evidence

6. Runtime Detection and Intervention Approaches

No single monitor class covers all agent-security properties. Precise authorization constraints are best expressed deterministically, while task alignment and semantic manipulation require language-aware evaluation. Long-horizon risk may require stateful or probabilistic models. RSMCA therefore treats heterogeneous monitors as complementary and requires their verdicts to be reconciled by an explicit policy layer rather than allowing each monitor to directly control the agent in incompatible ways.

6.1. Deterministic Specifications and Policy Rules

Deterministic rules are strongest when a property can be stated precisely: a tool is disallowed; a transaction exceeds a threshold; a data class cannot be sent to an external domain; delegation depth must not exceed a configured value; a child agent cannot receive permissions absent from the parent task. AgentSpec demonstrates customizable runtime enforcement for LLM agents using explicit

specifications [24]. The benefit is predictability and auditability. The limitation is coverage: many harmful behaviors are not reducible to static allow/deny lists without excessive false blocking.

6.2. Task-Alignment and Semantic Monitors

Semantic monitors evaluate whether an instruction or action is consistent with the authorized task. Task Shield operationalizes this idea for indirect prompt injection by checking task alignment of instructions and tool calls [21]. GuardAgent uses knowledge-enabled reasoning to enforce safety requirements around agents [20]. These methods can capture meaning that deterministic rules miss, but their own model-based judgments may be probabilistic. They should therefore provide evidence to the policy layer rather than becoming an unbounded second agent with unrestricted enforcement authority.

6.3. Adaptive and Lifelong Guardrails

Runtime conditions evolve as new attack patterns, tools, and agent behaviors appear. AGrail explicitly frames guardrailing as a lifelong and adaptive safety-detection problem [23]. Adaptive monitors are attractive for continuous assurance because static policy coverage degrades over time. However, adaptation itself becomes a managed change: updates to guardrail knowledge, models, thresholds, or examples should be versioned, tested, and included in assurance evidence so that a safety system does not silently change its decision boundary in production.

6.4. Reflection and Sidecar Verification

Sidecar designs place an independent verifier alongside the primary agent. Safety Sidecar uses reflection-driven runtime control to evaluate agent behavior and intervene when necessary [26]. Architectural separation is valuable because it prevents the primary agent from being the sole judge of its own actions. Sidecars can also be deployed as reusable infrastructure across agents. Their limitations include additional latency/cost, correlated model failure if verifier and agent share similar weaknesses, and the need to protect the sidecar’s own configuration and inputs.

6.5. Step-Level Tool Guardrails

ToolSafe focuses on proactive step-level guardrailing and feedback for tool invocation [27]. Step-level enforcement aligns well with systems management because tool boundaries are observable and often deterministic points of effect. A monitor can inspect the proposed method and arguments before execution and can return a revised constraint or denial. The difficult cases are actions whose risk depends on prior steps or latent consequences, which motivates correlation with task history and resource state.

6.6. Proactive Probabilistic Monitoring

Most monitors react to the current action or observed violation. ProbGuard advances a proactive approach by modeling the probability of future unsafe states from the current execution trace and issuing early warnings before a violation occurs [28]. Predictive monitoring is especially valuable for long-horizon agents, where waiting for the final unsafe action may be too late. Because probabilistic

warnings are uncertain, RSMCA maps them to graded responses such as increased scrutiny, reduced autonomy, or human step-up rather than assuming that every elevated score requires termination.

6.7. Trace-Based Fault and Security Observability

AgentTelemetry provides a benchmark and toolkit for LLM-agent observability and fault detection using traces [25]. Fault observability is relevant to security because many incidents manifest first as abnormal tool loops, missing outputs, repeated retries, unexpected branches, or dependency failures. RSMCA therefore includes reliability and security telemetry in the same correlated state. A failing dependency can change an agent's behavior and force fallback paths that have different security properties; conversely, an attack can appear initially as a reliability anomaly.

6.8. Benchmarks as Assurance Evaluation, Not Production Monitoring

Benchmarks such as R-Judge, AgentHarm, Agent Security Bench, InjecAgent, AgentDojo, and ToolEmu are essential for measuring security awareness, harmful capability, attack success, and defense performance [14–19]. They do not, by themselves, constitute production runtime monitoring. Their principal role in RSMCA is to provide attack scenarios, expected outcomes, and test suites for validating monitors before deployment and after changes. Continuous assurance should trigger re-testing when the model, prompt, tools, permissions, retrieval corpus, or guardrail configuration materially changes. Table 5 compares representative runtime security approaches, their mechanisms, roles, and limitations for continuous assurance.

Table 5
Representative approaches relevant to runtime agent security.

Approach	Primary mechanism	Runtime role	Key limitation for continuous assurance
AgentSpec [24]	Customizable explicit specifications and runtime enforcement	Deterministic policy enforcement	Requires properties to be expressible and maintained
GuardAgent [20]	Knowledge-enabled reasoning for safeguard decisions	Semantic policy/guard agent	Verifier remains model-based and incurs additional inference
Task Shield [21]	Task-alignment checking of instructions/tool calls	Prompt-injection and goal-drift defense	Focuses on task alignment rather than full operational state
AGrail [23]	Adaptive/lifelong guardrail and safety detection	Evolving semantic detection	Adaptation requires versioning and assurance of the guardrail itself
Safety Sidecar [26]	Independent reflection-driven safety controller	Sidecar verification and intervention	Latency/cost and possible correlated model failure
ToolSafe [27]	Proactive step-level tool guardrail and feedback	Tool invocation control	Long-horizon/cross-tool risk still needs state correlation
ProbGuard [28]	Probabilistic prediction over execution state	Early warning and proactive risk reduction	Probabilistic calibration and response policy remain deployment challenges
AgentTelemetry [25]	Structured traces and fault-detection benchmark/toolkit	Observability and trace-level detection	Observability alone does not enforce safe actions

7. Proposed Runtime Security Monitoring and Continuous Assurance Framework

The RSMCA framework is a control-plane architecture for converting heterogeneous runtime evidence into managed action. It does not replace model-level safeguards or identity/access management. Instead, it coordinates instrumentation, correlated state, multiple monitors, policy decisions, enforcement, recovery, and evidence so that an agent's authority can change as its runtime risk changes. Figure 1 summarizes the architecture.

7.1. Agent Execution Plane

The execution plane contains the primary agent, planner, memory/retrieval components, tools, peer agents, and protected resources. RSMCA assumes that any component in this plane may fail, be manipulated, become unavailable, or change behavior. The objective is not to make the execution plane

intrinsically trustworthy but to ensure that consequential transitions are observable and mediated by controls appropriate to their impact.

7.2. Instrumentation and Telemetry Plane

Instrumentation normalizes events from agent frameworks, API gateways, identity providers, vector stores, tool servers, policy engines, and infrastructure. The telemetry plane should preserve stable identifiers and timestamps, but it should also carry semantic context such as task ID, tool/action class, data sensitivity, and delegation relationships. Existing observability standards can carry many structural fields, but agent-security deployments may require additional conventions for task authority, monitor verdicts, and policy provenance.

7.3. Correlation and State Plane

Correlation converts discrete events into a stateful view of the agent's execution. It maintains session/task history, resource accesses, memory operations, configuration versions, and delegation graphs. This plane is essential for detecting cumulative behavior and for avoiding contradictory monitor decisions. For example, a file upload may be acceptable in one task but unsafe if it follows retrieval of a classified case record. Correlation supplies the history needed for that distinction.

7.4. Detection and Assurance Plane

Multiple monitors evaluate the correlated state. Deterministic policies check explicit invariants. Semantic monitors evaluate task/action consistency. Sidecar/verifier models assess higher-level risk. Probabilistic monitors estimate the likelihood of future unsafe states. Reliability/fault monitors detect execution anomalies that may alter security posture. Each verdict is tagged with monitor identity/version and the evidence state used to produce it. This prevents "the safety system said no" from becoming an unauditable black box.

7.5. Policy and Intervention Plane

Monitor verdicts do not directly determine execution. An explicit policy layer combines the verdict with action impact, resource sensitivity, autonomy level, task authority, and operational context. The policy maps the event to a graduated response set V = {allow, allow-with-audit, step-up approval, revise/retry, block, quarantine/rollback, terminate/revoke}. This design avoids two extremes: treating every anomaly as an incident that terminates useful work, or collecting alerts without the authority to prevent high-impact actions.

7.6. Assurance Evidence Plane

Every consequential decision should produce durable evidence: what was observed, which monitor and policy version evaluated it, what verdict was produced, what intervention occurred, and what the outcome was. Evidence supports incident reconstruction, control assessment, policy tuning, and reauthorization after change. It also enables assurance claims to be challenged. If a critical monitor was

unavailable for part of a session, the assurance record should explicitly show reduced coverage rather than silently treating the session as fully monitored.

7.7. Continuous Assurance State

RSMCA maintains an evolving assurance state A_t = < coverage, policy freshness, monitor health, behavior risk, dependency state, unresolved incidents, provenance completeness>. The tuple is conceptual: implementations may use categorical levels, rules, or calibrated scores. Its purpose is to ensure that continued operation depends on evidence about the monitoring system as well as the agent. An agent may be behaving normally but still require degraded autonomy if telemetry is incomplete, a tool dependency changed, or a required monitor failed.

The assurance state is action-sensitive rather than a permanent “trust score.” A low-risk read operation may remain allowable under partial degradation, while an irreversible administrative or financial action may require full monitor coverage and fresh attestation. This preserves the zero-trust principle that authority should be continuously evaluated while adapting it to agent-specific runtime evidence [4].

8. Closed-Loop Continuous Assurance Lifecycle

Continuous assurance is a management process, not a single classifier. Figure 2 represents a closed-loop lifecycle in which deployment state, runtime evidence, interventions, recovery, and change management continuously update what the organization is justified in allowing the agent to do.

8.1. Register and Baseline

Before activation, register the agent identity, owner/sponsor, model and runtime versions, enabled tools, credentials, data access, memory stores, peer-agent relationships, autonomy ceiling, and applicable policies. Establish a baseline of expected tool usage, task classes, normal latency/error patterns, and required monitor coverage. Baselines should describe what is expected, not create implicit trust; deviations trigger evaluation rather than automatic compromise conclusions.

8.2. Instrument and Observe

Instrumentation should be validated before the agent receives consequential authority. Test that critical tool calls, data egress, memory writes, delegation events, policy decisions, and responses appear in correlated traces. Monitoring blind spots should lower the maximum permitted autonomy. For high-impact actions, fail-closed behavior may be appropriate when required telemetry or policy services are unavailable.

8.3. Detect and Evaluate

Run complementary monitors at the points where they are most informative. Deterministic authorization checks should occur before side effects. Semantic/task-alignment checks should evaluate proposed actions and newly introduced instructions. Stateful monitors should examine sequences, cumulative

access, and delegation. Predictive monitors can increase lead time when the execution trajectory is trending toward risk. Verdict fusion should be transparent and governed by policy rather than ad hoc majority voting across heterogeneous monitors.

8.4. Intervene and Contain

Responses should be proportional to action impact and evidence confidence. A suspicious low-impact tool call can be revised or denied without killing the session. Repeated task divergence may reduce autonomy. A confirmed attempt to exfiltrate sensitive data can trigger block, credential revocation, memory quarantine, and incident escalation. Containment must include dependencies and delegated agents when causal evidence indicates propagation beyond the primary instance.

8.5. Verify and Recover

Recovery requires evidence that the condition leading to intervention has been addressed. Examples include removing a poisoned memory entry, rotating credentials, restoring an approved tool version, narrowing task authority, or restarting the agent with clean context. The system should then validate critical monitors and policy paths before restoring previous autonomy. Recovery is therefore not equivalent to merely restarting a process.

8.6. Update Assurance Evidence and Reassess After Change

Assurance evidence should be updated after incidents and routine changes. Model upgrades, system-prompt changes, newly enabled tools, MCP server updates, new retrieval corpora, policy edits, and changes to delegation rules can invalidate prior test results. Continuous assurance should trigger targeted re-testing using relevant benchmark scenarios and update the baseline and monitor configuration only after review. This change-driven reassessment is what distinguishes a lifecycle assurance program from static deployment approval. Table 6 outlines the graduated intervention options, when to use them, and the evidence required for each response.

Table 6
Graduated runtime intervention policy.

Response	Use when	Operational effect	Evidence requirement
Allow	No material policy/risk concern	Execute normally	Record baseline telemetry for consequential actions
Allow with enhanced audit	Low-confidence anomaly or elevated but acceptable risk	Execute; increase trace detail/retention	Monitor verdict, policy basis, correlated context
Step-up approval	High-impact action; uncertain semantic alignment; degraded assurance	Pause until authorized human/service approves	Proposed action, relevant context, risk reason, approver identity
Revise/retry	Action is repairable and likely caused by plan/argument error	Return constrained feedback and regenerate action	Original request, failure reason, revised constraints
Block	Action violates a policy or unacceptable risk threshold	Prevent side effect while session may continue	Policy/monitor verdict and denied action
Quarantine/rollback	Potential contamination of memory, dependency, resource, or completed reversible action	Isolate component; restore known-good state where possible	Affected objects, causal trace, rollback result
Terminate/revoke	Compromise, repeated severe divergence, or unsafe authority state	End task/session; revoke identities/capabilities/delegations	Incident record, revoked authority, downstream containment status

9. Evaluation Framework

A runtime security architecture should be evaluated as both a security mechanism and a managed service. A monitor that blocks every tool call can achieve a low attack-success rate but has no operational value. A fast detector that misses high-impact attacks is likewise inadequate. Evaluation should therefore report security effectiveness, benign-task utility, detection quality, lead time, control-plane overhead, recovery performance, and evidence quality.

9.1. Security and Detection Metrics

Attack success rate (ASR) should be reported by attack class and action impact, using established scenarios from ToolEmu, InjecAgent, AgentDojo, AgentHarm, Agent Security Bench, and related benchmarks where appropriate [14–19]. Detection should report true-positive and false-positive rates or precision/recall where labeled traces exist. Because some monitors output risk scores, calibration should be examined rather than relying solely on ranking metrics. A high score should correspond to an interpretable change in expected risk if the score is used to drive policy.

9.2. Operational Utility and Human Burden

Benign task completion, tool-call success, and end-to-end mission utility should be measured with and without monitoring. Human-intervention rate and approval latency are particularly important because excessive step-up prompts can create automation fatigue and encourage policy bypass. The objective is not zero human involvement; it is to concentrate human judgment on actions whose impact or uncertainty justifies the cost.

9.3. Temporal Performance

Mean time to detect (MTTD), time from detection to containment/revocation, and predictive lead time measure whether monitoring is early enough to prevent harm. For proactive monitors, lead time should be measured relative to the first preventable unsafe action rather than the final incident. Long-horizon agents also require monitoring of cumulative risk and session duration to determine whether detector effectiveness degrades over time.

9.4. Control-Plane Cost and Availability

Every additional monitor adds latency, compute cost, dependency complexity, and possible failure modes. Evaluation should report per-action policy latency, verifier/token cost, trace volume, storage/retention burden, and behavior when the monitoring/control plane becomes unavailable. High-assurance systems should test fail-open, fail-closed, and degraded-operation policies explicitly rather than leaving them as implementation defaults.

9.5. Assurance-Evidence Quality

Provenance completeness measures whether consequential actions can be resolved to an agent, task, authority source, relevant context, monitor/policy decision, tool/resource, and outcome. Coverage measures whether all required action classes were observed and evaluated. Policy freshness measures whether current policy and monitor versions correspond to the active agent configuration. These metrics are unusual in conventional model benchmarks but necessary if assurance claims are expected to support operational decisions and audits. Table 7 consolidates the proposed evaluation metrics, their purposes, and recommended reporting practices.

Table 7
Candidate evaluation metrics for runtime monitoring and continuous assurance.

Metric	Purpose	Recommended reporting
Attack success rate	Measures adversary ability to cause unauthorized/harmful outcomes	By attack surface, impact class, tool and agent configuration
True/false positive rate; precision/recall	Measures detector discrimination	Per monitor and attack class; include benign workload
Risk-score calibration	Tests whether probabilistic scores correspond to observed risk	Calibration curve/error where score drives policy
Benign task success / utility	Measures mission performance under controls	Baseline vs. monitored; with and without attack
False policy-block rate	Measures unnecessary enforcement	Per task/action class and policy version
Human-intervention rate and delay	Measures operational oversight burden	Approvals per task and median/percentile delay
Predictive lead time	Measures how early a proactive monitor warns	Time/steps before first preventable unsafe action
MTTD and containment/revocation time	Measures incident-response speed	Median and tail latency by incident class
Policy/monitor latency overhead	Measures control-plane feasibility	Per action and end-to-end workflow
Telemetry/trace overhead	Measures observability cost	Events/bytes per action/session; storage and processing cost
Coverage and monitor health	Measures whether assurance evidence is complete	Required vs. observed action classes; monitor uptime/event loss
Provenance completeness	Measures auditability and causal reconstruction	Fraction of consequential actions resolvable to task, authority, context, decision and outcome
Recovery success and re-assurance time	Measures ability to restore safe operation	Successful rollback/isolation and time to restore approved autonomy

10. Illustrative High-Assurance Deployment Scenarios

The following scenarios are analytical illustrations, not empirical case studies. Their purpose is to show how the framework changes the monitoring problem across different managed environments.

10.1. Enterprise Administrative Agent

Consider an agent that triages support tickets and can reset accounts, change group membership, and create cloud resources. Identity alone is insufficient because the same credential may legitimately execute different actions across tickets. RSMCA binds each action to a task ID and permitted resource scope. A semantic monitor checks whether the proposed action is consistent with the ticket objective, deterministic policy limits privileged methods and resource targets, and a high-impact identity change triggers step-up approval. Repeated failed changes or unusual expansion across resources raises behavior risk and can revoke the task capability without disabling the underlying employee account.

10.2. Public-Sector Case-Management Agent

A case-management agent may retrieve citizen data, draft notices, update records, and communicate with applicants. The monitoring surface includes data minimization, cross-case access, egress channels, and whether a proposed update has legal or material effect. The system can allow low-risk retrieval and drafting while requiring approval for rights-affecting changes. Provenance records the organizational sponsor, case/task, data accessed, model/tool versions, policy decision, approval, and resulting action. If an untrusted attachment introduces a conflicting instruction, task-alignment monitoring and action-level enforcement can prevent the attachment from silently expanding the agent's authority.

10.3. Multi-Agent Orchestration

In a multi-agent workflow, a planning agent may delegate research, coding, deployment, or communication to specialized agents. RSMCA monitors the delegation graph and requires child authority to remain within the parent task. Tool and data events are correlated across agents so that the system can detect a pattern such as a research sub-agent retrieving a secret followed by a communications sub-agent attempting external transmission. A compromised child can be isolated and its delegated capability revoked without necessarily terminating the entire workflow. This scenario illustrates why continuous assurance must operate over a graph of managed principals rather than one conversational session.

11. Discussion

11.1. The Architectural Shift from Safety Checks to Managed Runtime Assurance

The principal conclusion of this review is that safe agent deployment cannot be reduced to pre-deployment red teaming or a single prompt classifier. Agent behavior is conditioned by runtime state that changes after testing: external content, tool availability, credentials, memory, model versions, peer agents, and policies. A system can therefore pass a benchmark and later become unsafe without any change to the benchmarked model. Continuous assurance addresses this by treating safety evidence as perishable and dependent on current configuration and monitoring coverage.

This conclusion parallels mature security-management practice. Continuous monitoring exists because control effectiveness and threat exposure change over time [5,6]. Runtime assurance exists because an autonomous controller can enter conditions not fully anticipated at design time [11,12]. AI agents add semantic uncertainty and broad software authority to the same management problem. The appropriate response is not to abandon model-level safety research but to surround probabilistic decision-making with independent observability, policy, and recovery mechanisms.

11.2. Why Observability Is Necessary but Not Sufficient

Agent observability is receiving increased attention, and trace-level tooling is a necessary foundation [25]. However, observability becomes security only when traces are evaluated against properties and linked to enforceable responses. A production design should therefore avoid equating “we can see all tool calls” with “we can safely control tool calls.” RSMCA explicitly separates collection from judgment and judgment from enforcement so each function can be tested independently.

11.3. Heterogeneous Monitors Should Be Composed, Not Collapsed

The reviewed methods indicate that agent security properties have different epistemic forms. Some are exact (“this tool is prohibited”); some are semantic (“this action does not advance the authorized task”); some are temporal (“the sequence is converging on an unsafe state”); and some are operational (“the monitor lost visibility into a dependency”). Collapsing all of these into one universal risk score can obscure evidence and make policy difficult to audit. RSMCA instead preserves monitor-specific verdicts and lets a policy layer determine how they affect authority.

11.4. Assurance Must Include the Safety Infrastructure

A recurring weakness in runtime safety proposals is that the monitor is assumed to be available and correct. Continuous assurance requires monitoring the monitors: health, version, coverage, event loss, calibration drift, and dependency integrity. The assurance state should degrade when critical evidence disappears. This is especially important for agentic systems because rapid framework and tool changes can bypass instrumentation that was correct at deployment.

11.5. Fit with Network and Systems Management

The proposed framework is naturally expressed as a network and systems management problem. The JNSM network/service-management taxonomy emphasizes management functions and evolving infrastructures [32]. Prior JNSM work on autonomous cybersecurity for digital service chains demonstrates the value of programmable monitoring, identity and access control, dynamic security functions, and automated response across distributed services [33], while intent-driven security

orchestration research addresses dynamic selection and allocation of security mechanisms [34]. RSMCA extends these management concerns to autonomous AI services whose behavior and action selection are probabilistic. The managed object becomes not only a host, service, or flow, but an agent-task execution trajectory and its changing authority.

This framing suggests a productive integration path with existing infrastructure. Agent telemetry can flow through established observability pipelines; policy enforcement can be implemented at API gateways, service meshes, tool brokers, identity systems, or dedicated agent gateways; incident response can reuse credential revocation, network isolation, workload quarantine, and configuration rollback; and assurance evidence can be linked to service-management inventories and change-management records. The research challenge is to enrich these mature control planes with task, semantic, delegation, and agent-lifecycle context without making them dependent on opaque model reasoning.

12. Research Agenda

RSMCA is intentionally a framework for empirical research as well as deployment design. Several open problems must be resolved before runtime assurance for agents can become a standardized systems-management capability. Table 8 identifies the priority research areas and provides candidate artifacts or experiments for each.

Table 8
Research agenda for runtime monitoring and continuous assurance of AI agents.

Research area	Core question	Candidate artifact/experiment
Standard agent telemetry	Which fields and causal links are minimally sufficient for cross-framework security monitoring?	Interoperable trace schema; adapters across agent frameworks and tool protocols
Monitor composition	How should deterministic, semantic, verifier and probabilistic monitors be fused without hiding evidence?	Policy languages; conflict-resolution semantics; compositional evaluation
Long-horizon detection	How can monitors detect risk distributed across many individually acceptable actions?	Stateful sequence benchmarks; temporal/property monitors; cumulative-risk tests
Proactive assurance	How early can unsafe trajectories be predicted at acceptable false-alarm rates?	Lead-time benchmarks; calibrated prediction; risk-adaptive autonomy
Semantic authorization	How can task purpose be represented externally without relying on private chain-of-thought?	Task/intent schemas; action-to-task consistency tests; policy templates
Multi-agent assurance	How should monitoring, authority, and incident containment work across delegation graphs?	Delegation-aware traces; propagation benchmarks; graph isolation/revocation
Monitor self-assurance	How can systems detect degraded instrumentation, monitor drift, or compromised safety components?	Coverage health metrics; adversarial monitor tests; redundancy/failover designs
Change-driven reassurance	Which configuration changes invalidate prior tests and require partial/full re-evaluation?	Change-impact model; selective benchmark replay; assurance evidence versioning
Operational economics	What security benefit justifies runtime latency, token/compute cost, and human review?	Security-utility-cost frontier; deployment studies; SLOs for assurance services
Recovery and rollback	How can an agent, memory, delegation graph, or dependency be restored to a known-good state?	Checkpoint/rollback protocols; memory quarantine; credential/delegation recovery tests
High-assurance testbeds	How can realistic enterprise/public-sector scenarios be studied without sensitive production data?	Synthetic case-management, cloud-admin, SOC and multi-agent service testbeds

Table 9
Abbreviations used in the manuscript.

Abbreviation	Meaning
AI	Artificial intelligence
API	Application programming interface
ASB	Agent Security Bench
ASR	Attack success rate
ISCM	Information Security Continuous Monitoring
LLM	Large language model
MCP	Model Context Protocol
MTTD	Mean time to detect
NIST	National Institute of Standards and Technology
RAG	Retrieval-augmented generation
RSMCA	Runtime Security Monitoring and Continuous Assurance
ZTA	Zero Trust Architecture

13. Limitations

This study has four principal limitations. First, the agent-security literature is evolving rapidly; methods and tool ecosystems published after the September 2026 cutoff may alter the comparative landscape. Second, the study is a structured integrative review rather than an exhaustive PRISMA systematic review, so it does not claim complete retrieval of every runtime safety paper or provide quantitative publication counts. Third, several runtime approaches have been evaluated primarily in benchmark environments rather than long-running production deployments, which limits evidence about operational false positives, human burden, cost, and failure recovery. Fourth, RSMCA is a conceptual framework. Its architecture, telemetry model, assurance state, and response policies require implementation and comparative evaluation before they can be treated as a validated reference architecture.

These limitations define the most important next step: build an instrumented agent testbed in which the same workloads and attacks are executed under baseline monitoring, individual guardrails, and the full RSMCA control loop. Such an evaluation should measure not only attack success but also monitor coverage, false policy blocks, predictive lead time, service latency, human-intervention burden, containment time, recovery success, and provenance completeness.

14. Conclusions

AI agents convert probabilistic model decisions into software actions, making runtime security a continuing management problem rather than a one-time model evaluation. The reviewed evidence shows that agent failures can originate in untrusted context, task drift, unsafe tool use, persistent memory, data flows, inter-agent delegation, changing dependencies, and configuration state [13–19,29,30,35,36]. Recent runtime defenses provide important mechanisms—explicit specifications, task-alignment checks, adaptive guardrails, sidecar verification, step-level tool controls, proactive prediction, and trace-based observability [20–28]—but no single mechanism provides continuous assurance by itself.

The RSMCA framework synthesizes these mechanisms into a closed-loop systems-management architecture. It separates agent execution from instrumentation, state correlation, detection, policy/intervention, evidence, and the evolving assurance state. Its central principle is that continued agent authority should depend not only on identity and static permission but on current task context, observed behavior, action impact, monitoring coverage, dependency state, and recoverability. The framework therefore treats assurance evidence as dynamic and perishable rather than as a certification permanently attached to a model.

For network and systems management, the opportunity is to extend mature observability, policy, orchestration, and incident-response infrastructure with agent-specific semantics. The most valuable future work is empirical: standardized telemetry, composable monitor policies, long-horizon and multi-agent benchmarks, proactive prediction, recovery mechanisms, and measurements that jointly report security, mission utility, operational cost, and assurance quality. Such work can move AI-agent safety from isolated benchmark defenses toward measurable, continuously managed security for autonomous services.

Declarations

Competing interests

The authors declare no competing interests.

Ethics approval

Not applicable. This article is a literature review and conceptual systems-management framework and does not report research involving human participants or animals.

Informed consent

Not applicable.

Use of generative AI in manuscript preparation

OpenAI ChatGPT was used to assist with manuscript structuring, language refinement, literature organization, reference-consistency checking, and drafting of conceptual diagrams. All generated material and cited references were reviewed and verified by the corresponding author, who remains responsible for the submitted content. The AI system is not listed as an author.

Abbreviations

Table 9 lists the abbreviations used throughout the manuscript.

Funding

This research received no external funding.

Author Contribution

Y.R.V. conceived and designed the study, developed the Runtime Security Monitoring and Continuous Assurance (RSMCA) framework, led the literature review and conceptual synthesis, and drafted the main manuscript. A.M. contributed to the literature review, analysis of AI-agent security approaches, and manuscript review and revision. K.V.M. contributed to literature synthesis, conceptual development, and manuscript review and editing. N.S.V. contributed to the review of relevant literature, organization of the manuscript, and manuscript revision. N.R.M. contributed to the literature review, analysis, and critical review of the manuscript. Y.R.V. prepared the figures and tables. All authors reviewed and approved the final manuscript.

Data availability

No datasets were generated or analyzed during the current study. The publications, standards, and guidance documents synthesized in the review are identified in the reference list.

References

1. Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W.X., Wei, Z., Wen, J.: A survey on large language model based autonomous agents. Front. Comput. Sci. 18, 186345 (2024). https://doi.org/10.1007/s11704-024-40231-1
2. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K.R., Cao, Y.: ReAct: Synergizing reasoning and acting in language models. In: International Conference on Learning Representations (ICLR) (2023).
3. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. Adv. Neural Inf. Process. Syst. 36, 68539–68551 (2023). https://doi.org/10.52202/075280-2997

4. Rose, S., Borchert, O., Mitchell, S., Connelly, S.: Zero Trust Architecture. NIST Special Publication 800 – 207, National Institute of Standards and Technology, Gaithersburg, MD (2020). https://doi.org/10.6028/NIST.SP.800-207
5. Dempsey, K., Chawla, N., Johnson, L., Johnston, R., Jones, A., Orebaugh, A., Scholl, M., Stine, K.: Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations. NIST Special Publication 800 – 137, National Institute of Standards and Technology, Gaithersburg, MD (2011). https://doi.org/10.6028/NIST.SP.800-137
6. Dempsey, K., Pillitteri, V., Baer, C., Niemeyer, R., Rudman, R., Urban, S.: Assessing Information Security Continuous Monitoring (ISCM) Programs: Developing an ISCM Program Assessment. NIST Special Publication 800-137A, National Institute of Standards and Technology, Gaithersburg, MD (2020). https://doi.org/10.6028/NIST.SP.800-137A
7. Pascoe, C., Quinn, S., Scarfone, K.: The NIST Cybersecurity Framework (CSF) 2.0. NIST Cybersecurity White Paper 29, National Institute of Standards and Technology, Gaithersburg, MD (2024). https://doi.org/10.6028/NIST.CSWP.29
8. Tabassi, E.: Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1, National Institute of Standards and Technology, Gaithersburg, MD (2023). https://doi.org/10.6028/NIST.AI.100-1
9. Autio, C., Schwartz, R., Dunietz, J., Jain, S., Stanley, M., Tabassi, E., Hall, P., Roberts, K.: Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. NIST AI 600-1, National Institute of Standards and Technology, Gaithersburg, MD (2024). https://doi.org/10.6028/NIST.AI.600-1
10. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Logic Algebraic Program. 78(5), 293–303 (2009). https://doi.org/10.1016/j.jlap.2008.08.004
11. Fisher, M., Mascardi, V., Rozier, K.Y., Schlingloff, B.-H., Winikoff, M., Yorke-Smith, N.: Towards a framework for certification of reliable autonomous systems. Auton. Agents Multi-Agent Syst. 35, 8 (2021). https://doi.org/10.1007/s10458-020-09487-2
12. Sheikhi, S., Mehmood, U., Bak, S., Smolka, S.A., Stoller, S.D.: The black-box simplex architecture for runtime assurance of multi-agent CPS. Innov. Syst. Softw. Eng. 21(2), 619–634 (2025). https://doi.org/10.1007/s11334-024-00553-6
13. 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. In: Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, pp. 79–90 (2023). https://doi.org/10.1145/3605764.3623985
14. Ruan, Y., Dong, H., Wang, A., Pitis, S., Zhou, Y., Ba, J., Dubois, Y., Maddison, C.J., Hashimoto, T.: Identifying the risks of LM agents with an LM-emulated sandbox. In: International Conference on Learning Representations (ICLR) (2024).
15. Zhan, Q., Liang, Z., Ying, Z., Kang, D.: InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In: Findings of the Association for Computational

Linguistics: ACL 2024, pp. 10471–10506 (2024). https://doi.org/10.18653/v1/2024.findings-acl.624

16. Debenedetti, E., Zhang, J., Balunovic, M., Beurer-Kellner, L., Fischer, M., Tramèr, F.: AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In: Advances in Neural Information Processing Systems, Datasets and Benchmarks Track (2024). https://doi.org/10.52202/079017-2636
17. Yuan, T., He, Z., Dong, L., Wang, Y., Zhao, R., Xia, T., Xu, L., Zhou, B., Li, F., Zhang, Z., Wang, R., Liu, G.: R-Judge: Benchmarking safety risk awareness for LLM agents. In: Findings of the Association for Computational Linguistics: EMNLP 2024, pp. 1467–1490 (2024). https://doi.org/10.18653/v1/2024.findings-emnlp.79
18. Andriushchenko, M., Souly, A., Dziemian, M., Duenas, D., Lin, M., Wang, J., Hendrycks, D., Zou, A., Kolter, J.Z., Fredrikson, M., Gal, Y., Davies, X.: AgentHarm: A benchmark for measuring harmfulness of LLM agents. In: International Conference on Learning Representations (ICLR) (2025).
19. Zhang, H., Huang, J., Mei, K., Yao, Y., Wang, Z., Zhan, C., Wang, H., Zhang, Y.: Agent Security Bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In: International Conference on Learning Representations (ICLR) (2025).
20. Xiang, Z., Zheng, L., Li, Y., Hong, J., Li, Q., Xie, H., Zhang, J., Xiong, Z., Xie, C., Yang, C., Song, D., Li, B.: GuardAgent: Safeguard LLM agents via knowledge-enabled reasoning. In: Proceedings of the 42nd International Conference on Machine Learning, PMLR 267, 68316–68342 (2025).
21. Jia, F., Wu, T., Qin, X., Squicciarini, A.: The Task Shield: Enforcing task alignment to defend against indirect prompt injection in LLM agents. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, vol. 1, pp. 29680–29697 (2025). https://doi.org/10.18653/v1/2025.acl-long.1435
22. Zhu, K., Yang, X., Wang, J., Guo, W., Wang, W.Y.: MELON: Provable defense against indirect prompt injection attacks in AI agents. In: Proceedings of the 42nd International Conference on Machine Learning, PMLR 267, 80310–80329 (2025).
23. Luo, W., Dai, S., Liu, X., Banerjee, S., Sun, H., Chen, M., Xiao, C.: AGrail: A lifelong agent guardrail with effective and adaptive safety detection. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, vol. 1, pp. 8104–8139 (2025). https://doi.org/10.18653/v1/2025.acl-long.399
24. Wang, H., Poskitt, C.M., Sun, J.: AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In: Proceedings of the 48th International Conference on Software Engineering (ICSE) (2026). https://doi.org/10.1145/3744916.3764546
25. Balusu, K.C.: AgentTelemetry: A fault detection benchmark and toolkit for LLM agent observability. In: Proceedings of the 3rd ACM International Conference on AI-Powered Software (AIware '26), pp. 380–387 (2026). https://doi.org/10.1145/3805760.3814931
26. Wang, B., Quan, J., Yu, X., Hu, H., Yu, H., Gao, A., Wan, Z., Li, H., Tsang, I.: Safety Sidecar: Reflection-driven runtime control for safer agents. In: Findings of the Association for Computational

Linguistics: ACL 2026, pp. 30842–30856 (2026). https://doi.org/10.18653/v1/2026.findings-acl.1542

27. Mou, Y., Xue, Z., Li, L., Liu, P., Zhang, S., Ye, W., Shao, J.: ToolSafe: Enhancing tool invocation safety of LLM-based agents via proactive step-level guardrail and feedback. In: Findings of the Association for Computational Linguistics: ACL 2026, pp. 37125–37153 (2026). https://doi.org/10.18653/v1/2026.findings-acl.1850
28. Wang, H., Poskitt, C.M., Wei, J., Sun, J.: ProbGuard: Proactive runtime monitoring for LLM agent safety via probabilistic prediction. Accepted at the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE) (2026).
29. Xu, Y., Zhuang, Y., Liu, X., Zhang, T., Xiao, B., Xu, X., Jiang, D., Wang, J., Hu, H.: LLM agents security duality: A comprehensive survey of self-security and empowered cybersecurity. Artif. Intell. Rev. 59, 174 (2026). https://doi.org/10.1007/s10462-026-11563-0
30. OWASP GenAI Security Project: OWASP Top 10 for Agentic Applications for 2026. OWASP Foundation (2025). https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/. Accessed 9 September 2026.
31. Booth, H., Fisher, W., Galluzzo, R., Roberts, J.: Accelerating the Adoption of Software and Artificial Intelligence Agent Identity and Authorization: Concept Paper. National Institute of Standards and Technology, National Cybersecurity Center of Excellence, Gaithersburg, MD (2026).
32. van der Hooft, J., Claeys, M., Bouten, N., Wauters, T., Schönwälder, J., Pras, A., Stiller, B., Charalambides, M., Badonnel, R., Serrat, J., dos Santos, C.R.P., De Turck, F.: Updated taxonomy for the network and service management research field. J. Netw. Syst. Manag. 26, 790–808 (2018). https://doi.org/10.1007/s10922-017-9443-y
33. Repetto, M., Striccoli, D., Piro, G., Carrega, A., Boggia, G., Bolla, R.: An autonomous cybersecurity framework for next-generation digital service chains. J. Netw. Syst. Manag. 29, 37 (2021). https://doi.org/10.1007/s10922-021-09607-7
34. Robles-Enciso, A., Bernabé Murcia, J.M., Molina Zarca, A., Skarmeta Gomez, A.: Dynamic multi-method allocation for intent-based security orchestration. J. Netw. Syst. Manag. 33, 21 (2025). https://doi.org/10.1007/s10922-024-09896-8
35. Hou, X., Zhao, Y., Wang, S., Wang, H.: Model Context Protocol (MCP): Landscape, security threats, and future research directions. ACM Trans. Softw. Eng. Methodol. (2026). https://doi.org/10.1145/3796519
36. Wang, S., Zhang, G., Yu, M., Wan, G., Meng, F., Guo, C., Wang, K., Wang, Y.: G-Safeguard: A topology-guided security lens and treatment on LLM-based multi-agent systems. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, vol. 1, pp. 7261–7276 (2025). https://doi.org/10.18653/v1/2025.acl-long.359

Figures

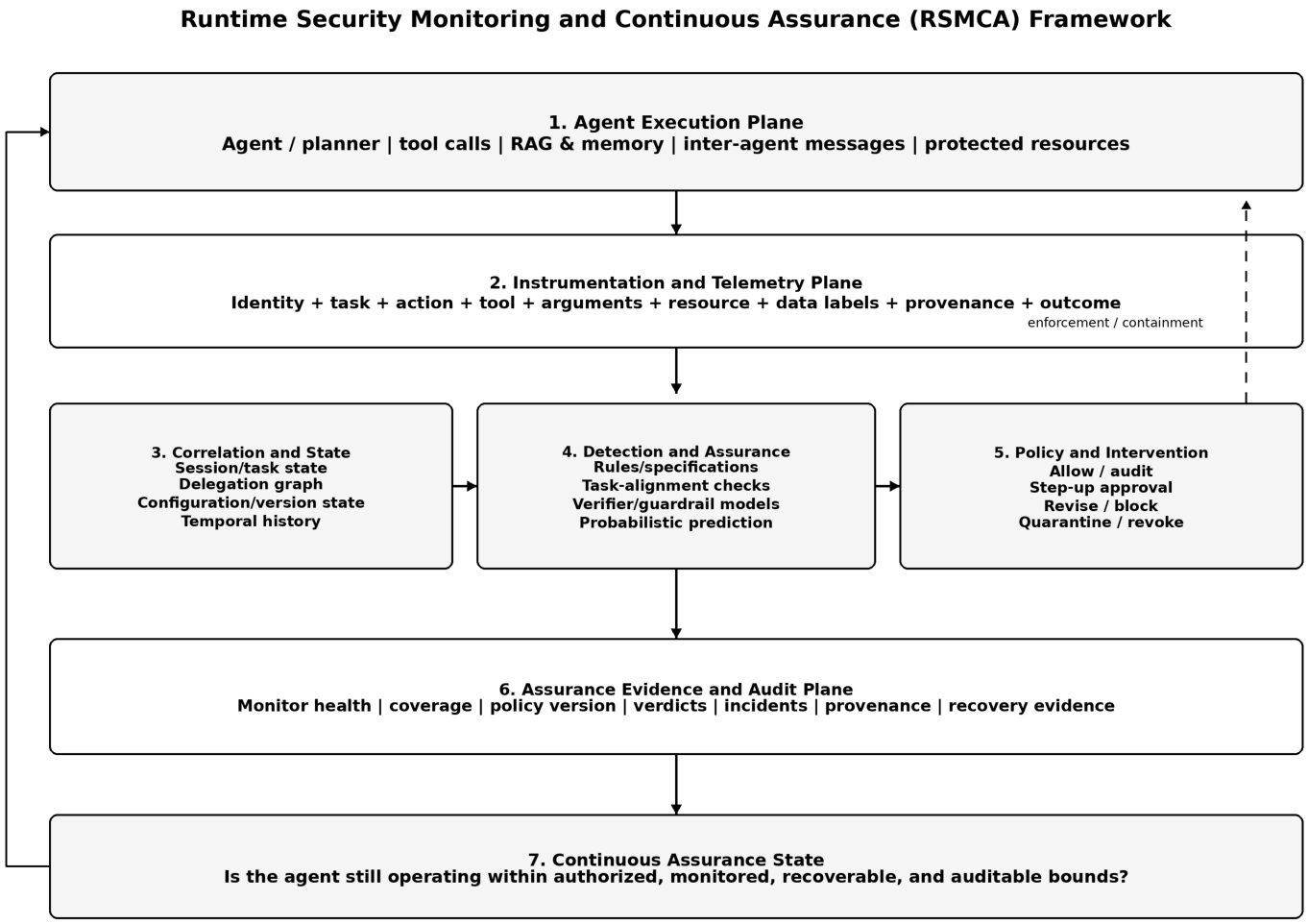


Figure 1

Runtime Security Monitoring and Continuous Assurance (RSMCA) architecture. The framework separates agent execution, telemetry, state correlation, detection, policy/intervention, evidence, and the evolving assurance state.

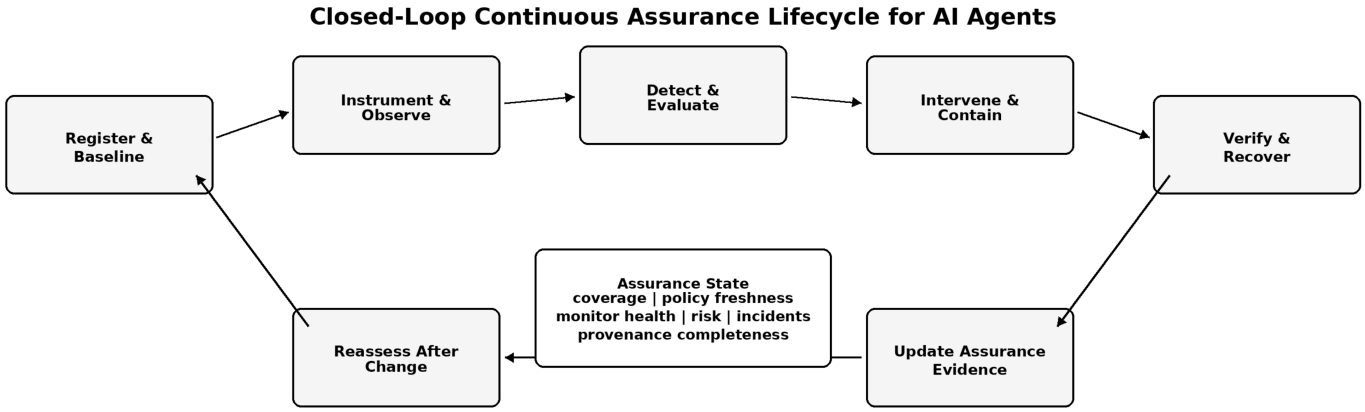


Figure 2

Closed-loop lifecycle for continuous assurance of AI agents