International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 11 | Nov 2026 www.irjet.net p-ISSN: 2395-0072

Behavior-Based Security Detection for Short-Lived AI Agents: A Proposed Architecture and Evaluation Framework

Amaan Shaikh[1], Neeraj Pagare[2], Sagar Kumar[3], Ms. Poonam Surve[4]

[1234]*Department of MCA, D. Y. Patil University, School of Humanities and Science, Navi Mumbai, India*

Abstract - *Short-lived AI agents can carry out tool-mediated work and interact with external resources during brief execution periods. Because these agents may use temporary credentials, operate in changing contexts and invoke several tools, connecting their stated purpose to the resource-level actions they perform is a significant monitoring challenge. This paper presents Behavior-Based Security Detection for Short-Lived AI Agents (BDSA), a proposed architecture that brings execution context, tool activity, identity signals, data provenance and resource effects together in a task-centred trajectory. The design introduces an execution passport to describe the authorized scope of a task and outlines policy checks supported by correlated behavioural evidence for risk-informed detection and response. It also specifies a common event representation, a method for correlating application telemetry with cloud audit records, and a proposed evaluation protocol based on AgentDojo prompt-injection tasks. The protocol sets out comparison conditions and testable hypotheses, alongside measures for detection quality, false alarms, detection delay and successful completion of legitimate tasks. Cloud-provider-style audit events in the proposed extension are simulated; they do not demonstrate integration with live multi-cloud environments. Accordingly, this work presents an architecture and evaluation plan, not experimentally verified detection results. Implementation and empirical assessment remain future work.*

Keywords - AI agent security, behavioral detection, prompt injection, cloud audit logs, event correlation, security monitoring, agentic systems

1. INTRODUCTION

Large language model (LLM) agents combine a model with planning, memory, tools and the authority to interact with external systems. During a single task, an agent may read email, query a database, change infrastructure, create a support ticket or send a response. ReAct-style approaches made the close relationship between reasoning and action explicit [7]. This capability also creates a distinct security challenge: harmful activity can involve a legitimate model, an approved tool and a valid credential. In such cases, a recognizable malicious file or signature may not exist. The warning sign may instead be an unexpected connection between the assigned task, the tool selected, the information accessed, the privilege used and the resulting change to a cloud resource.

Monitoring these relationships is particularly difficult when agents are short-lived. A serverless function, container or workflow worker may exist only for one request and disappear as soon as the work finishes. Its credentials can be temporary, its network location can change, and its activity may cross cloud boundaries. Periodic inventory checks and access reviews are poorly suited to this execution speed. Signature-based controls may also miss attacks in which each individual tool call is technically valid. Indirect prompt injection illustrates the problem: content returned by a tool can contain instructions that redirect the agent toward an attacker-chosen action, even while the surrounding infrastructure appears to operate normally [8], [9].

Existing standards and security projects address important parts of this problem, but they do not by themselves provide a unified runtime view of an individual agent task. The NIST AI Risk Management Framework emphasizes ongoing measurement and monitoring [1], and its Generative AI Profile discusses risks arising from deployment context [2]. Zero Trust Architecture rejects trust based solely on network position [3]. MITRE ATLAS and OWASP resources describe adversarial techniques and security concerns specific to AI and agentic applications [4]–[6]. Cloud platforms also provide audit facilities and temporary workload identities, while OpenTelemetry is developing shared conventions for model and tool activity [10]–[13]. The remaining design challenge is to associate these separate signals with the lifecycle and purpose of one agent execution.

This paper introduces Behavior-Based Security Detection for Short-Lived AI Agents (BDSA) as a vendor-neutral architectural proposal. It does not claim a completed implementation, production deployment or empirical validation. The work is limited to defining a system model, a common event representation, a task-aware correlation design and a falsifiable evaluation plan. It addresses three central questions: how an ephemeral agent execution should be represented; how declared task intent can be connected to tool and resource events while minimizing exposure of prompt content; and how detection quality and task utility could be assessed when telemetry is delayed, incomplete or affected by workload change.

1.1 Contributions

- A proposed lifecycle model that binds owner, task scope, workload identity, model invocation, tool activity and resource effect to one execution identifier.
- A canonical trajectory representation intended to preserve provenance while normalizing provider-specific logs and application telemetry.
- A proposed task-policy and trajectory detection approach. Sequence, peer-group, privilege, graph and learned analytics are extensions for future comparison, not a single validated detector in this manuscript.
- A risk-aware response model that describes possible actions such as step-up approval, privilege reduction, isolation, termination and evidence preservation; enforcement effectiveness is not measured here.
- An evaluation protocol separating detection quality, task utility, latency, cost, transferability and privacy considerations.
- A proposed mapping from AgentDojo tasks and scoring to BDSA task contracts and telemetry, including a simulated cloud-audit layer for studying correlation under controlled delay, event loss and clock skew.

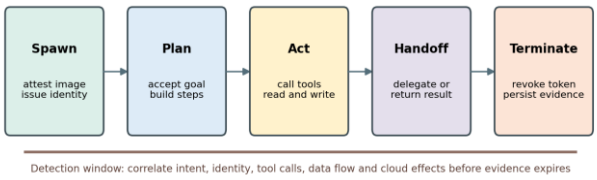

Fig -1: Lifecycle of a short-lived agent and its detection window

1.2 Scope and implementation status

This manuscript describes a proposed design rather than a completed system. It sets out the execution-passport concept, event schema, correlation envelope, trajectory model, candidate detection features, response options and evaluation procedure. It does not include executable implementation code, collected run logs, a completed AgentDojo integration, measured BDSA performance or deployment evidence from live AWS, Azure or Google Cloud audit systems. Provider-style audit records are considered simulated inputs for a future test harness. For that reason, statements about BDSA effectiveness are presented as intended design properties or hypotheses. The architecture, detector and response process should only be described as implemented after supporting code and test evidence exist.

2. BACKGROUND AND RELATED WORK

2.1 Agent Execution as a Security Principal

An AI agent should be treated as more than an isolated model invocation. Its execution may include an orchestrator, system instructions, task context, retrieved information, memory, tool definitions, credentials and a runtime that can cause changes outside the model. From a security perspective, the agent functions as a non-human principal. Its effective authority is determined by the permissions of its tools together with the authority delegated by the user or service that initiated it. Cloud Security Alliance guidance on non-human identities describes the wider governance issues associated with service accounts, access tokens and automated actors [19]. For ephemeral agents, the owner and purpose should be recorded at launch because the runtime may no longer exist when an alert is investigated.

A persistent agent name does not provide sufficient attribution. One agent definition may be used for many unrelated tasks, and an incident in one execution should not automatically alter the behavioural baseline for every other task. BDSA therefore separates three concepts: the agent definition, the individual execution and the credential session. The definition captures the approved code, model, tools and policy; the execution identifies a particular goal and trace; and the credential session records the federated or assumed identity used to access cloud resources. This separation helps an investigator determine whether the agent was approved, whether a particular run stayed within its task boundaries and whether a credential action can be linked to that run.

2.2 Agent Threats

Threats against agents often span several conventional security boundaries. Examples include changing the agent's effective goal, misusing a legitimate tool, abusing identity or privilege, poisoning memory, manipulating inter-agent messages and triggering cascading failures [5], [6]. AgentDojo shows that instructions embedded in untrusted data can redirect tool-using agents, and that security outcomes need to be considered alongside task utility [8]. HouYi likewise demonstrated the practical impact of prompt injection in applications integrated with language models [9]. These findings support examining behaviour: an attack may become visible only after injected content changes the agent's tool sequence or the destination of data. AgentDojo also reports differences in attackability across models and shows that a detector-based defence can reduce attack success while imposing a substantial cost on legitimate utility [8]. BDSA is therefore motivated by evaluation that measures both security and utility, with detection informed by task context rather than content inspection alone.

The framework assumes that adversaries may control user input, retrieved documents, tool output, memory entries or an external endpoint. They may steal a short-lived token, exploit an over-privileged tool, replay an inter-agent message or induce the agent to create a durable backdoor before termination. The framework does not assume that the model reveals its complete reasoning. Private chain-of-thought is neither required nor treated as trustworthy telemetry. Detection operates on declared task metadata, tool requests, structured results, identities, resources and effects.

2.3 Behavior and Trace Anomaly Detection

Research on log anomaly detection offers several relevant starting points. DeepLog learns patterns in event sequences using a recurrent neural network [15], while LogBERT applies self-supervised contextual modelling to detect deviations in system logs [16]. Trace-oriented techniques represent distributed activity as connected events and can use graph structure to identify unusual execution paths [18]. However, an empirical review notes that reported performance can be sensitive to evaluation design, noise, training-data selection and software evolution [17]. These limitations matter for AI agents, where model revisions and prompt variation can cause rapid behavioural drift. A production design should therefore avoid granting enforcement authority to a single opaque anomaly score.

Within BDSA, learned anomaly analysis is proposed as one source of evidence in a broader detection process. Deterministic rules address explicit violations, including prohibited destinations and unsigned tools. Statistical features compare an execution with relevant cohorts, such as the same agent definition or task family. Graph analysis can identify combinations of events that appear ordinary in isolation but form a suspicious path when connected. The decision explanation should identify the underlying events so that the resulting response can be matched to the strength of the evidence.

2.4 Multi-Cloud Identity and Observability

Short-lived credentials can reduce exposure, but they can also make activity harder to attribute across systems. AWS Security Token Service supports source identity and session tags that can be recorded in CloudTrail [11], [20]. Google Cloud Workload Identity Federation enables workloads to obtain short-lived access without relying on service-account keys, with relevant exchanges and activity represented in audit records [12], [21]. Microsoft Entra Workload ID provides OpenID Connect federation for workloads such as Kubernetes pods [13]. SPIFFE offers attested workload identities and short-lived identity documents across different environments [14]. These mechanisms help associate a runtime with an identity, but they do not independently establish which agent goal or tool decision led to a particular resource action.

OpenTelemetry supplies vendor-neutral facilities for traces, metrics and logs, including developing semantic conventions for generative-AI model and tool operations [10]. In the proposed BDSA design, application telemetry explains the agent-side activity, while cloud audit records provide evidence of resource-level events. A correlation envelope is intended to connect these sources using trace identifiers, workload identity, token-session information, resource names and a bounded time interval. If a provider does not expose a reliable shared identifier, the system should retain an explicit confidence value for the proposed association rather than presenting an uncertain match as exact.

2.5 Agent-Security Benchmarks

Evaluating agent security requires environments in which the agent selects and executes tools, because an attack succeeds or fails through the agent's own decisions. Indirect prompt injection, in which instructions arrive through retrieved or tool-returned data, was characterized for LLM-integrated applications by Greshake et al. [25]. Early benchmarks for prompt injection used settings without tool calls, such as document question answering, prompt stealing and simple goal or rule hijacking [8]. InjecAgent [22] moved to tool-integrated agents, but evaluates simulated single-turn cases in which the model receives one adversarial tool output directly, so the agent's planning is not assessed [8]. ToolEmu [23] emulates tool execution and scores outcomes with a language model; the AgentDojo authors argue that this is problematic under injection because a successful injection could also influence the emulator or evaluator [8]. The Berkeley Function Calling Leaderboard [24] measures how accurately instructions become function calls, without adversarial content. AgentDojo [8] instead runs agents against a stateful environment and scores them with formal utility and security functions computed over environment state.

Table -1: Agent-security benchmarks and the gap relevant to BDSA

Benchmark	Execution and scoring	Gap for BDSA
InjecAgent [22]	Simulated single-turn cases; adversarial tool output supplied directly	Planning not exercised; no identity or cloud effects

Benchmark	Execution and scoring	Gap for BDSA
ToolEmu [23]	LLM-emulated tools; LLM-based scoring	Scorer may be influenced by injection; no real state
Function-calling leaderboard [24]	Instruction-to-call accuracy; benign inputs	No adversary; limited multi-step state
AgentDojo [8]	Stateful tools; formal utility and security checks	No cloud identity, audit logs, lifetime or telemetry privacy
BDSA harness (this work)	AgentDojo plus simulated cloud audit events and leases	Adds detection, latency, cost and privacy measures (planned)

The benchmarks discussed above do not capture cloud workload identity, provider audit events, execution lifetime or telemetry privacy. Their primary concern is whether an attack succeeds, rather than whether a runtime monitor can observe the activity, attribute it to a particular execution and interrupt it in time. The proposed BDSA evaluation therefore uses AgentDojo for prompt-injection and task-utility scenarios, with an additional cloud-event harness described in Section 7.8.

2.6 Prompt-Injection Defenses and the Role of Behavioral Detection

Published defenses fall into four families. Detection approaches inspect tool outputs with a classifier and abort when an injection is suspected [8]. Marking approaches wrap untrusted content in delimiters or encodings and instruct the model to treat it as data [8], [26], while training approaches teach models to prioritize privileged instructions over lower-privilege text [27]. Prompt sandwiching repeats the user instruction after each tool result [8]. Isolation approaches restrict the available tools before untrusted data is read, or separate a privileged planner from the component that handles untrusted content [8], [28], [29]. Several of these techniques are acknowledged to be non-foolproof and unable to guarantee protection for security-critical tasks [8].

AgentDojo's evaluation of representative defenses on GPT-4o makes the trade-offs concrete (Figure 5, data from [8]). Against the strongest generic attack, tool filtering reduced targeted attack success to about 7% and the injection detector to about 8%, but the detector also cut benign utility from 69% to about 41% because of false positives. Delimiting and prompt repetition reduced attack success less, to about 42% and 28%, and raised benign utility, presumably because they emphasize the original instruction [8]. The authors caution that repetition is unlikely to withstand adaptive attacks, for example an injection that tells the model to disregard later instructions, and that tool filtering fails when the tools required by the task are also sufficient for the attack, which they find for 17% of test cases [8].

Table -2: Prompt-injection defense families and their limitations

Defense family	Mechanism	Limitation
Delimiting and marking [8], [26]	Mark tool output as data and tell the model to ignore it	No guarantee for security-critical tasks [8]
Injection detection [8]	Classifier on tool output; abort when flagged	Large benign utility loss from false positives in [8]
Prompt sandwiching [8]	Repeat the user task after each tool call	Unlikely to resist adaptive attacks [8]
Instruction-hierarchy training [27]	Train the model to prioritize privileged instructions	Model-level; not designed to observe downstream effects
Tool filtering and isolation [8], [28], [29]	Restrict tools before reading untrusted data	Fails when required tools suffice for the attack (17% in [8])
BDSA (this work)	Correlate identity, intent, action and effect at runtime	Needs telemetry and joins; unvalidated until implemented

The approaches described above mainly change model inputs or restrict the tools available to an agent. They are preventive controls and may provide limited visibility into what happens after an injection has influenced the agent or into the cloud-side consequences of the resulting action. BDSA is proposed as a complementary monitoring layer. It assumes that some preventive measures may fail and asks whether the resulting sequence of actions can still be observed, attributed and interrupted before the short-lived execution ends. This is the reason the planned evaluation considers BDSA both independently and alongside preventive defences in Section 7.8.

3. SYSTEM AND THREAT MODEL

3.1 Definition of a Short-Lived Agent

For this work, a short-lived agent means an autonomous or semi-autonomous software process launched to complete a bounded task and expected to stop when that task finishes or a defined execution lease expires. The design considers lifetimes ranging from seconds to hours, assumes that the process has no permanent host identity, and requires at least one external tool or resource interaction. Possible implementations include Kubernetes jobs, serverless functions, temporary virtual machines, workflow activities and sandboxed processes. The term describes the lifetime of an execution, not the availability of the underlying model service: a long-running model endpoint may serve many separate, temporary agent runs.

Each run is assigned an immutable execution identifier, owner, tenant, task family, digest of the declared goal, approved tool list, maximum privilege scope, data-classification ceiling and expiry time. A separate definition digest identifies the orchestrator version, policy version, model family and tool manifests. Together, these attributes form the proposed execution passport. They should be carried as signed trace metadata or protected runtime context and must not be accepted directly from untrusted prompt text.

3.2 Assets and Trust Boundaries

The assets to protect include cloud resources, user information, credentials, model and tool configurations, memory stores, audit evidence and downstream services affected by agent activity. Relevant trust boundaries occur between the user and agent, retrieval systems, the model provider, the agent and its tools, tools and the resources they access, collaborating agents, federated cloud environments and telemetry-export systems. The detector and its signing keys are positioned outside the agent's trust boundary. Although an agent may generate telemetry, independent cloud collectors and policy gateways should verify high-impact activity.

3.3 Adversary Capabilities and Exclusions

The threat model allows an adversary to introduce instructions through user input or retrieved content, compromise a tool endpoint, alter memory, obtain a token that remains valid, exploit excessive permissions or coordinate agents that exchange misleading context. An attacker may also spread activity over time or across cloud platforms. BDSA assumes that the control plane, policy signer and central evidence repository have not all been compromised, and that at least one trustworthy telemetry source can observe each action subject to enforcement. A simultaneous compromise of the runtime, provider control plane and evidence pipeline is outside the scope of behavioural detection and would require platform-integrity and recovery controls.

Table -3: Threats and observable behavioral evidence

Threat	Behavioral evidence	Primary control
Goal hijack	Tool path diverges from task family; untrusted content immediately precedes a new objective.	Goal-action consistency and provenance.
Tool misuse	Approved tool uses unusual arguments, write volume or destructive operation.	Invariant policy and argument risk.
Identity abuse	New role, region, resource class or token-use pattern.	Session binding and privilege graph.
Memory poisoning	New memory entry influences later executions across owners or tenants.	Memory provenance and trust decay.
Data exfiltration	Sensitive read followed by new egress, sharing or message destination.	Data-flow correlation and egress policy.
Agent spoofing	Unsigned delegation, broken parent-child chain or identity mismatch.	Attested messaging and lineage.
Persistence	Agent creates durable user, key, schedule, policy or startup object.	High-impact resource semantics.

4. PROPOSED BDSA ARCHITECTURE

The proposed BDSA design separates telemetry collection, event correlation, detection and response. This separation is intended to keep the analytic model independent of any one cloud provider's logging format. As illustrated in Figure 2, provider-native audit records remain the principal evidence of resource effects, while application traces provide context about the agent's attempted actions. The correlation layer is designed to connect these sources without requiring raw prompt content to be stored in the central security data lake.

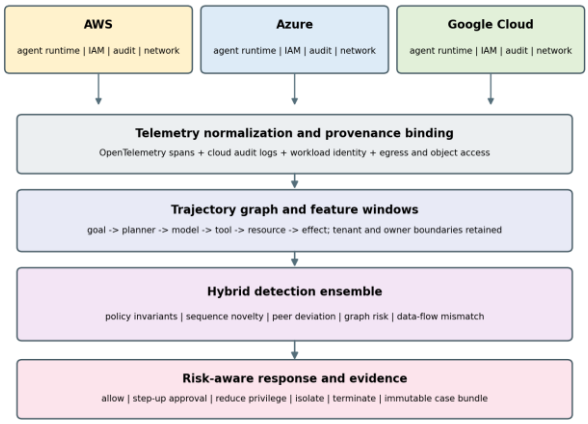

Fig -2: Vendor-neutral architecture for cross-cloud agent detection

4.1 Telemetry Plane

Instrumentation begins in the orchestrator. A root trace span captures the execution passport, with child spans representing model requests, retrieval operations, tool selection, tool execution and hand-offs between agents. For each tool span, the proposed schema records a stable tool identifier, manifest digest, operation category, destination category, argument-schema hash, data labels, result status, duration and byte counts. Sensitive values are tokenized or converted into limited features during collection. Events also identify the source class of content such as trusted, user-provided, retrieved, generated or returned by a tool so that later analysis can consider provenance.

Cloud connectors collect identity events, control-plane activity, data-access records and selected network events. The proposed integrations normalize relevant AWS CloudTrail and service records, Google Cloud Audit Logs and workload-identity exchanges, and Azure activity, resource and sign-in logs. Kubernetes audit events and network metadata derived from eBPF may provide additional context when agents run in clusters. Collection should be event-driven where supported. Sources that provide only batch delivery are suitable for later confirmation, but should not be treated as a basis for immediate blocking.

4.2 Provenance and Correlation Envelope

Each event is placed in a correlation envelope containing the execution identifier, trace and span identifiers, parent execution, agent definition, tenant, owner, workload identity, cloud account or subscription, region, provider event identifier, event time, ingestion time, trust level and join-confidence value. Original provider fields are retained as evidence while a normalized representation is exposed to the detection components. Both event time and ingestion time are maintained to account for clock differences. The correlation service is designed to accept events out of order within a bounded interval and revise a trajectory when delayed audit evidence becomes available.

The preferred associations are exact matches, for example a signed execution identifier in tool-gateway metadata, AWS source identity, a workload-identity subject, a token-exchange audit identifier or a provider request identifier returned by a tool. If such propagation is unavailable, the service may estimate a match using the principal, resource, operation and time window. Estimated matches receive a lower confidence level and should not independently justify an irreversible response. Recording this uncertainty explicitly avoids presenting a tentative correlation as a definitive attribution.

Table -4: Canonical event fields used by BDSA

Field group	Representative fields	Purpose
Execution	execution ID, parent ID, task family, goal digest, expiry	Defines the bounded trajectory.
Definition	agent digest, model family, tool manifest, policy version	Supports version-aware baselines.
Identity	workload subject, token session, role, owner, tenant	Attributes actions and privilege.
Action	tool, operation, argument shape, resource, outcome	Describes intended and observed work.
Data	input label, output label, bytes, destination class	Enables data-flow detection.
Provenance	source type, trust, predecessor, influence link	Tracks untrusted influence.
Timing	event time, ingest time, sequence, remaining lease	Handles short windows and skew.
Evidence	provider event ID, signature, join confidence	Supports audit and response review.

4.3 Trajectory Graph

Normalized events are represented as a directed property graph. Its nodes describe goals, agents, model

calls, tools, identities, data objects, cloud resources and external destinations. Edges capture relationships such as delegation, influence, selection, invocation, reading, writing, token exchange and network transfer. This graph retains relationships that may be obscured in a flat log stream. For example, reading a customer record and sending an email may each be legitimate in some workflows; the combined path becomes more concerning when the declared task is scheduling, the recipient is unfamiliar and the email follows content from an untrusted document.

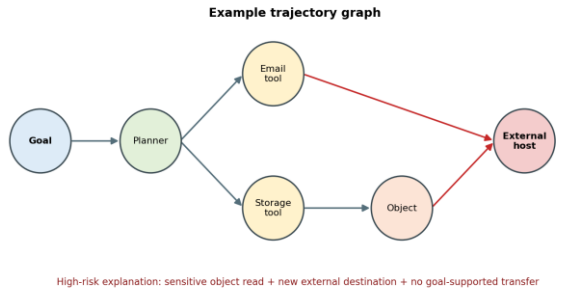

Fig -3: Trajectory graph supporting an explainable exfiltration alert

4.4 Streaming Detection Ensemble

The proposed ensemble examines events as they arrive and again when an agent requests a high-impact operation. It contains five complementary detection layers. A policy component checks explicit constraints, such as blocked destinations, unapproved tools, cross-tenant memory access and attempts to create persistent credentials. A sequence component assesses whether the order of tools and operations is plausible for the relevant agent definition and task family. A peer component compares features such as resource type, region, activity rate, data volume and failure frequency with recent comparable executions. A privilege component examines role assumption, permission use and resource reach. Finally, a graph component looks for risky relationships, including sensitive reads followed by external writes, untrusted input preceding administrative activity, and child-agent delegation without verified lineage.

Policy outcomes are applied before statistical scores are combined. A clear violation of a hard invariant may justify immediate blocking. A high-confidence graph pattern may instead lead to an approval request or isolation. Statistical novelty on its own would normally increase monitoring or lower the threshold for a later approval, rather than automatically terminate an execution. Each decision record should identify the contributing events and rules, correlation confidence, baseline version and proposed response.

Table -5: Detection layers and operating characteristics

Layer	Best signal	Strength	Main weakness
Invariant policy	Known forbidden action or destination	Fast and explainable	Cannot cover novel abuse
Sequence	Unexpected tool or operation order	Captures task flow	Sensitive to workflow drift
Peer deviation	Unusual rate, resource or region	Works without attack labels	Cold start and cohort quality
Privilege graph	New identity or reach expansion	Cloud-aware	Provider schema differences
Data flow	Sensitive read followed by risky effect	Finds semantic exfiltration paths	Needs dependable labels
Trajectory graph	Multi-step causal pattern	Cross-cloud context and explanation	Higher compute and join cost

4.5 Risk-Aware Response

Any response must be completed within the agent's remaining execution lease. The least disruptive option is to increase telemetry collection. More restrictive measures can narrow the tool set, limit concurrency or outbound traffic, or replace the current token with a lower-privilege session. A step-up approval pauses a high-impact request and presents the declared goal, requested effect and supporting evidence to a human reviewer or independent policy service. Isolation prevents further external calls while preserving the ability to export evidence. Termination revokes credentials where possible, stops the runtime and blocks further child-agent delegation. A case bundle created after termination should include the signed execution passport, normalized trajectory, original provider event identifiers, policy versions and actions taken.

The response controller should fail safely, but it should not apply a universal deny policy to every uncertain event. A destructive infrastructure change has different consequences from interrupting a low-risk summarization task. Each operation should therefore have an impact category and a reversibility classification. If telemetry is unavailable, the response should follow the policy for that

category: deny irreversible privileged changes, request approval for sensitive writes, and permit low-impact reads only with reduced assurance. This approach avoids treating every observability failure as either harmless or equally severe.

5. FEATURE ENGINEERING AND BASELINING

5.1 Identity and Privilege Features

Identity-related features describe the task owner, attested workload, token-acquisition method, assumed role and permissions actually exercised. Candidate features include whether a role or region is new, the difference between granted and used permissions, the number of resource categories accessed, cross-account transitions, token age at use, failed role assumptions, consistency of session tags and creation of persistent identity objects. The concern is not simply that an agent has a powerful role; risk increases when that authority is unusual for the task and is exercised after influence from untrusted content.

5.2 Tool and Sequence Features

Tool-related features include the selected tool, operation category, argument schema, destination type, error pattern, retry count, ratio of writes to reads and branching factor. Literal argument values are generally excluded from the central detector. Instead, collection components can derive features such as data type, length, entropy range, sensitivity label, novelty category and whether the destination matches an approved domain. Tool sequences may be represented using n-grams, probabilistic automata, recurrent networks or transformer models. The proposal does not select one algorithm; it requires version-aware training, calibrated decision thresholds and explanations that identify unexpected transitions.

5.3 Goal-to-Action Consistency

An anomaly detector can learn to accept an action simply because it occurs frequently, while flagging an uncommon but legitimate operation. Goal-to-action analysis adds the declared task to the decision. At launch, the orchestrator maps the request to an approved task category and a set of permitted effects. For example, invoice reconciliation may require reading invoices and updating a ledger, but it should not require creating identity-management users. Consistency could be assessed through deterministic task contracts, a small classifier operating on sanitized goal and action descriptions, or a combination of both. A classifier should contribute evidence rather than act as the sole authority, since a language-model judge could be influenced by the same content that redirected the agent. Where a benchmark provides ground-truth tool-call sequences, as AgentDojo does [8], those sequences may be used to initialize task contracts.

5.4 Data-Flow and Provenance Features

Data-flow features follow labels and transformations rather than retaining complete content. Candidate signals include the highest sensitivity level observed, trust level of the influencing source, label downgrades, fan-out, destination novelty, data-volume ratio and elapsed time between a read and an external action. Provenance edges indicate whether a tool result may have influenced a later call. Since AgentDojo found that injected text placed late in a tool response was particularly effective [8], the proposed telemetry also records the position of untrusted content within a returned result. A direct relationship between untrusted content and an administrative or external-write action would be treated as a strong warning signal. Memory entries receive additional metadata: source, tenant, owner, creating execution, integrity state and expiry policy. Retrieval across owners or continued influence from low-trust content raises the assessed risk.

5.5 Cold Start and Drift

A newly created agent definition has no execution history from which to learn a specific baseline. BDSA would initially rely on policy, the tool manifest, the privilege envelope and a baseline borrowed from a broader task-family cohort. During this initial period, statistical thresholds should be more permissive, while high-impact actions receive stronger approval requirements. After enough clean executions have been observed, a baseline specific to that definition can be established. A change to the model, prompt, tools or policy should create a new baseline version rather than being silently merged with earlier behaviour. Candidate updates can be evaluated in shadow mode before they are allowed to influence enforcement.

Changes in inputs, action sequences and outcomes should be measured separately. Normal business cycles may alter resource volumes without changing the order of tool calls, whereas a model update may change planning order while leaving the final effects unchanged. Separating these dimensions supports more selective adaptation. Confirmed incidents and red-team examples should be excluded from normal baseline training. Analyst feedback should retain its source and context so that a single incorrect disposition cannot immediately redefine expected behaviour.

5.6 Decision Explanation and Threshold Governance

Every enforcement decision should be reconstructable from a concise evidence record. That record identifies the

active policy bundle, baseline version, feature window, correlation confidence and principal contributing events. It should distinguish a direct policy violation from a learned anomaly. For example, an unfamiliar recipient may be a novelty signal, whereas a task contract that prohibits external delivery is a deterministic policy failure. Keeping these explanations separate helps analysts challenge the appropriate control and prevents a probabilistic score from being communicated as certainty.

Thresholds should be managed as controlled production settings rather than informal model parameters. Each threshold needs an accountable owner, an intended impact category, a validation dataset, an approval date and a rollback version. A proposed change should first be assessed offline, then in shadow mode, and finally on a limited group of tasks. Detection and response thresholds should remain distinct: a moderate-risk event may warrant an alert, while interruption should occur only when risk, evidence confidence and action impact jointly satisfy the response policy. This separation allows earlier visibility without making every unusual event operationally disruptive.

6. MULTI-CLOUD DEPLOYMENT

6.1 Identity Binding Pattern

The preferred deployment model uses attested workload identity and token exchange in place of static credentials. The orchestrator signs the execution passport, starts the workload and requests a short-lived identity that contains or references the execution identifier. Provider federation maps that identity to a least-privilege role. SPIFFE may provide a common workload-identity layer when several platforms must be supported [14]. Before forwarding a tool request, the gateway verifies the passport and tool manifest. Cloud audit records can then independently show whether the federated identity accessed a particular resource.

An agent that operates across cloud platforms should not carry a reusable collection of credentials. Instead, it should exchange its attested identity for a narrowly scoped token at each boundary. The correlation service records this chain of exchanges. Parent and child agents receive separate identities, connected through a signed delegation relationship, with reduced privileges unless additional authority has been explicitly approved. This makes delegation visible and limits the impact of a compromised worker, rather than treating all workers as one shared service identity.

6.2 Collection and Time Alignment

Collectors positioned near each cloud boundary forward normalized events through an authenticated queue while retaining provider event identifiers, signatures and original timestamps. A central or federated correlation service uses watermarks to complete short streaming windows while still accepting delayed records. To meet data-residency requirements, raw records may remain within their original regions, with only normalized features and hashes exported. Detection logic requiring cross-cloud context can operate on the shared feature stream, while investigators retrieve original evidence through authorized and audited queries.

6.3 Policy Distribution and Availability

Policies are compiled into signed bundles and distributed to tool gateways and response controllers. If the central service becomes unavailable, local enforcement continues using the last valid policy bundle and a bounded cache of revocation information. High-impact tools should reject requests when their policy bundle has expired. Lower-impact work may continue with reduced capabilities and local logging. This design avoids routing every decision through a potentially slow central service while retaining centralized policy ownership and evidence correlation.

6.4 Data Protection

Security telemetry may itself contain prompts, customer information or credentials. Collection should therefore follow data-minimization principles: raw prompts and model responses remain disabled by default; argument values are converted into approved features; secrets are redacted at the source; and any content capture requires a documented purpose and limited retention period. Tenant boundaries are maintained, while identifiers used in shared analytics are pseudonymized. Access to original evidence is role-restricted and audited. If trajectories are used for model training, they should be de-identified, and a deletion process should remove linked content while retaining only non-identifying security metrics where policy permits.

6.5 Evidence Retention and Investigation

The evidence repository is designed with two retention levels. A short-term online store holds normalized trajectories and signed decision records for prompt investigation. A more restricted archive retains provider-native records only when incident, legal or regulatory requirements justify it. Each case bundle records a cryptographic digest of the original event so that evidence retrieved later can be checked for integrity. Retention periods should be based on data sensitivity and action impact, rather than the perceived reputation of a particular model or agent provider.

An investigation can begin with an execution identifier and expand to related parent and child agents, identity sessions, resources and destinations. This allows analysts to reconstruct relevant activity without searching unrelated tenant records. Where an association is probabilistic, the interface should show candidate events and their confidence values instead of combining them into a single definitive timeline. This distinction matters for accurate attribution: two temporary workers may use the same role and access the same resource within a short interval, while only one may be connected to the action through a signed request identifier.

7. EVALUATION METHODOLOGY

The proposed system should be assessed both as a detection mechanism and as an operational control. A high volume of alerts is not evidence of success if legitimate tasks are frequently interrupted. Likewise, an aggregate accuracy score can hide poor performance against rare but consequential attacks. The material below defines a future evaluation plan; it does not report completed experiments or measured BDSA performance.

7.1 Research Questions

- RQ1: Does cross-layer trajectory correlation detect attacks that are missed by prompt-only, identity-only and cloud-log-only baselines?
- RQ2: How much do graph, provenance and goal-action features improve detection under short execution lifetimes?
- RQ3: What latency and cost are added to benign agent tasks, especially high-percentile tool calls?
- RQ4: How well do baselines transfer across cloud providers, agent definitions and model updates?
- RQ5: How much sensitive information can be removed from telemetry before detection quality materially declines?
- RQ6: Does behavioral detection add value beyond preventive prompt-injection defenses, particularly on the cases where those defenses are known to fail?

7.2 Testbed and Workloads

A future reference environment could run equivalent agent workers on AWS, Azure and Google Cloud, connected to synthetic email, ticketing, object-storage, database and infrastructure-management services. Example workloads may cover support triage, document summarization, invoice processing, cloud remediation and software delivery. Each task category should have explicit success criteria, tool contracts and controlled benign variation. Agent lifetimes could be selected from intervals such as 30 seconds, two minutes, ten minutes and one hour to examine how audit-event delay affects detection.

Benign test traces should represent model nondeterminism, temporary service failures, retries, regional changes and controlled software-version updates. Attack traces should be generated from documented scenarios and reviewed manually to confirm that each one violates the intended security property. Public benchmarks such as AgentDojo can supply prompt-injection tasks [8], but cloud privilege and cross-provider scenarios require an additional instrumented environment. The proposed dataset should make schemas, generators and sanitized trajectories available without disclosing production content.

7.3 Attack Scenarios

Table -6: Evaluation scenarios and expected evidence

Scenario	Injected condition	Expected high-value signal
Indirect prompt injection	Malicious instruction in email or document	Untrusted influence followed by off-task tool path
Tool argument abuse	Valid tool with destructive or external target	Argument risk plus task-contract violation
Token theft	Session reused by another runtime or region	Attestation mismatch and peer deviation
Privilege escalation	Agent assumes broader role	New role edge and excess permission use
Memory poisoning	Low-trust content stored for reuse	Persistent influence across executions
Cross-cloud exfiltration	Sensitive object sent through another provider	Read-to-egress path across identities
Agent spoofing	Forged child or delegation message	Invalid signature or broken lineage
Persistence	Creation of key, account, schedule or policy	Durable security object outside task contract
Low-and-slow misuse	Action split among several short agents	Shared owner, resource and destination graph

7.4 Baselines and Ablation

Once the primary detector has been implemented, a future study should compare it with a no-detector condition and, where the setup can be reproduced, with AgentDojo preventive baselines such as tool filtering, prompt-injection detection, data delimiting and prompt repetition [8]. More complex comparisons and component-removal tests covering provenance, peer groups, privilege graphs, data labels and cross-cloud correlation should follow only after the initial detector has been evaluated. This staged approach reduces the risk of attributing an apparent improvement to a complex ensemble before its individual components have been examined.

7.5 Metrics

Planned security measures include precision, recall, F1 score, false-positive rate, attack success after response and time to detection. Any future results should be separated by scenario and impact category rather than presented only as an overall average. Operational measures should cover added median and 95th-percentile latency, events and bytes per execution, collector loss, correlation completion time, detector throughput, storage expense and analyst time per case. Utility should be assessed through legitimate-task success, unnecessary approval frequency and completion delay. For AgentDojo-derived workloads, benign utility, utility under attack and targeted attack success should follow the benchmark definitions [8]. This manuscript contains no BDSA measurements.

Robustness tests vary cloud provider, model family, agent version, task wording, event loss, clock skew, audit delay and attacker adaptation. Privacy evaluation measures the ability to reconstruct sensitive input from stored telemetry and compares full-content, redacted and feature-only modes. Calibration is assessed with reliability curves so that a risk score has operational meaning. Confidence intervals should be computed over repeated task seeds, and threshold selection must use a validation set separate from final testing.

7.6 Experimental Validity

Synthetic services can make experiments easier to reproduce, but they may not reflect the full complexity of production environments. A later study could include an authorized pilot using realistic workflow patterns and appropriately de-identified data, subject to privacy and security review. Attack generators may become tailored to known rules, so detector-aware red-team scenarios should be held back until final testing. Labelling also requires care: an unusual execution path is not necessarily malicious, and a task that completes successfully may still violate policy. Each ground-truth case should separately record task outcome, policy violation, security impact and the response that should have been taken.

7.7 Reproducibility and Reporting Protocol

For reproducibility, each experiment should record the agent-definition digest, tool manifests, policy bundle, cloud region, model identifier, detector version, random seed and workload-generator commit. Event loss and provider audit delay should be measured rather than assumed. Training, validation and test runs should be separated by task instance and time to prevent near-duplicate traces from inflating performance. If a managed model cannot be pinned to an exact version, the provider's version information should be recorded and a calibration subset rerun before comparisons are made.

Reports should provide confusion matrices for each attack family, response outcomes by impact category and distributions in addition to averages. False alarms should be classified as benign novelty, instrumentation faults, policy mismatch or model error. Missed attacks should be analysed in terms of the absent signal or failed event association. Cost accounting should include cloud logging, event transport, feature storage, detector computation and human review. This reporting structure would allow a complex detector to be compared with a simpler rule set in terms of security benefit relative to latency and operating cost.

7.8 AgentDojo-Based Instantiation

AgentDojo [8] is proposed as the prompt-injection component of a future BDSA test environment. The published benchmark includes 97 user tasks and 629 security test cases across its Workspace, Slack, Travel and Banking suites, with utility and security checks evaluated against environment state. A future implementation could instrument tool execution to emit one normalized event for each call without modifying the benchmark tasks or scoring logic. A separate cloud-audit simulator could generate provider-style identity and audit events with configurable delay, event loss and clock skew. Table 7 outlines this proposed mapping. No completed BDSA integration or independent reproduction is claimed in this paper.

Table -7: Mapping of AgentDojo artifacts to BDSA components

AgentDojo artifact	What it provides	Use in BDSA
User-task ground truth	Ordered tool calls that solve the task	Seeds the task contract and approved tool set
Injection-task ground truth	Ordered tool calls of the attacker goal	Attack label and expected violation path
Utility function	Task success checked on environment state	Task-outcome label
Security function	Whether the attacker goal was met	Security-impact label
Injection placeholders	Attack points inside tool outputs	Untrusted-influence edges in the trajectory graph
Pipeline element	Modular agent pipeline interface	Hosts the BDSA event emitter and detector
Built-in defenses	Tool filter, detector, delimiters, repetition	Comparison baselines (Section 7.4)

7.8.1 Harness Design

The proposed harness, illustrated in Figure 4, would preserve AgentDojo's original tasks and scoring. Each run would create an execution passport based on the user task, including its owner, task category, approved tools, data ceiling and execution lease. A future telemetry emitter would wrap tool execution and produce one normalized event per call. Since AgentDojo's tools are simulated, a separate cloud-audit simulator would generate provider-style identity and audit records under configurable delay, event loss and clock skew. This arrangement would make it possible to test event correlation and confidence calculations under controlled conditions. The detector would receive the telemetry and execution passport, but not the benchmark's scoring labels.

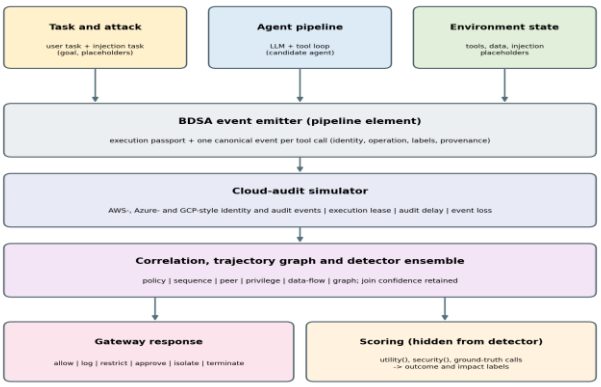

Fig -4: Harness that instruments AgentDojo runs with BDSA telemetry and a cloud-audit simulator

In a future implementation, task contracts could be generated from the benchmark's ground-truth call sequences for an idealized-orchestrator condition. A second condition could use broader contracts defined at the task-family level. AgentDojo's utility and security functions should remain restricted to scoring; they must not be supplied to the detector, since doing so would expose evaluation labels and create label leakage.

7.8.2 Experimental Conditions

Table 8 presents the planned experimental conditions. The AgentDojo defence results reproduced in Figure 5 are published results from the cited study, not BDSA measurements and not evidence that this manuscript has independently reproduced those findings. If the proposed harness is implemented, detection should first be assessed with response enforcement disabled. Enforcement can then be evaluated separately for its security effect and its impact on legitimate task completion.

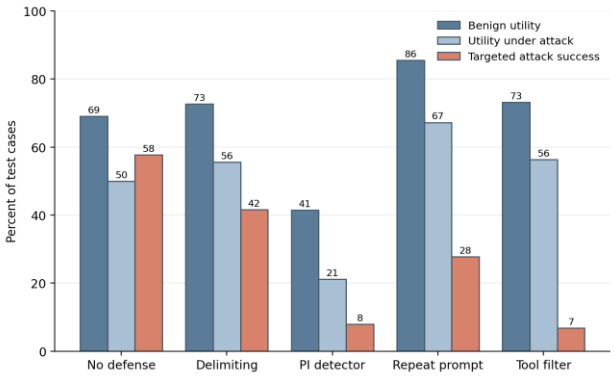

Fig -5: Reported utility and attack success of AgentDojo defenses on GPT-4o (data from Table 5 of [8])

Table -8: Experimental conditions on AgentDojo workloads

ID	Configuration	Purpose
C0	No defense	Reference for utility and attack success
C1	Tool filter	Strongest preventive baseline in [8]
C2	Injection detector	Content-based detection baseline
C3	Delimiting; prompt repetition	Prompt-level baselines, run separately
C4	BDSA, observe only	Detection quality without enforcement
C5	BDSA with gateway response	Security effect and utility cost of enforcement
C6	Tool filter plus BDSA	Layered defense; tests complementarity
C7	BDSA ablations	Contribution of each component (Section 7.4)

7.8.3 Hypotheses

The proposed evaluation is intended to examine four hypotheses that can be disproved by the results. These statements are predictions only; this manuscript does not claim that any of them has been confirmed.

- H1: BDSA with gateway response (C5) lowers targeted attack success relative to C0 while losing less benign utility than the injection detector (C2), because it acts on the trajectory and its effect rather than on content alone.
- H2: On the subset of cases in which the user task's tools suffice for the attack, C5 and C6 achieve lower attack success than C1, because goal contracts, data labels and provenance can separate a permitted action from an injected one.
- H3: When application-tool telemetry is degraded or absent, cloud-side evidence preserves a measurable share of recall that a tool-only sequence detector loses.
- H4: As simulated audit delay approaches the lease, recall for high-impact actions observed at the gateway declines less than recall for effects confirmed only by provider logs.

A hypothesis should be considered unsupported if the paired confidence interval for the difference includes zero or favours the baseline. Future reports should give null and negative outcomes the same prominence as positive findings.

7.8.4 Statistical Treatment and Adaptive Attacks

A future study should report proportions with 95% binomial confidence intervals and compare conditions using identical user-task and injection-task pairs. Results should be separated by benchmark suite and injection scenario because the published AgentDojo study reports meaningful variation across tasks [8]. The same level of detail should be applied to false alerts and legitimate utility. This describes the planned statistical analysis, not results collected for BDSA.

The AgentDojo publication notes that attack success can depend materially on prompt wording and discusses adaptive evaluation [8]. A future BDSA assessment should therefore include detector-aware scenarios, such as injected objectives that remain within the approved tool set, target recipient categories that are normally permitted, or distribute activity across child executions. These are proposed test cases, not experiments already conducted. They should be reported separately so that performance against default attacks is not mistaken for evidence of adaptive robustness [8], [30].

7.8.5 Cost and Reproducibility

The AgentDojo paper reports approximate costs for its own benchmark experiments [8]. Those values provide background only and should not be interpreted as BDSA operating costs. Once an implementation exists, the cost of telemetry emission, simulation, detection, storage and model use should be measured separately for each condition. Reproducibility materials should include the passport schema, emitter and simulator source code where implemented, contract-generation scripts, random seeds, environment settings and the exact benchmark version.

8. ILLUSTRATIVE DETECTION WALKTHROUGH

8.1 Support-Ticket Scenario

Consider a support agent launched to summarize an incoming ticket and prepare a draft response. Its execution passport allows the agent to read the ticket and write to the draft field, but does not authorize external messaging or access to object storage. The ticket contains an indirect instruction asking the agent to retrieve a diagnostic archive and send it to an external address. The planner selects a storage tool, reads a sensitive object and then calls an email tool with a new recipient.

No individual event is sufficient to establish that the execution has been compromised. Reading diagnostic material may be normal in another support workflow, and email may be an approved enterprise tool. The credential itself is valid. BDSA's proposed correlation process links

the events into one task trajectory. The provenance component identifies the ticket as an untrusted influence; the task contract marks the storage read and external email as effects outside the permitted scope; the sequence component flags an unfamiliar transition from ticket reading to storage access; the data-flow component connects a sensitive read to a new external destination; and the graph component identifies the path illustrated in Figure 3. Given the action's impact and the assumed exact event associations, the gateway would pause the send, replace the credential with a read-only session and create an evidence bundle for review.

If the email audit record is delayed, the gateway can still make an initial decision using the tool request, task contract and data label. A later provider event can help establish whether any partial resource effect occurred. By contrast, if the only signal is a statistical sequence deviation, the proposed response would be to allow the draft operation while increasing monitoring, rather than immediately stopping the task. This example illustrates the intended combination of policy and behavioural evidence, with response severity determined by impact, confidence and reversibility.

8.2 Benchmark-Grounded Scenario

A second example follows the task structure of the Workspace suite in AgentDojo [8]. The user asks an agent to summarize meeting-related emails and send the result to a manager, matching the type of task used to illustrate the benchmark. An attacker has inserted an instruction into an email that attempts to make the agent disclose the user's most recent message. Two variations show why task context matters to behavioural monitoring.

In the first variation, the assigned task is read-only, such as counting calendar appointments. A tool filter that removes email-sending capability before the agent reads untrusted content can prevent the attack [8]; in that case, BDSA would mainly add monitoring and evidence. In the second variation, sending an email is a legitimate part of the task. A tool filter cannot distinguish the approved send from the injected one because the same tools needed for the user's task are also sufficient to carry out the attack. AgentDojo identifies this as a failure condition for tool filtering [8].

For the second variation, BDSA evaluates the requested outcome rather than blocking the email tool outright. The execution passport identifies the manager as an allowed recipient category. The injected objective instead introduces an external recipient and a message containing data above the task's permitted classification level. The provenance graph connects the untrusted email to the send request, while the task contract identifies the proposed effect as unauthorized. The gateway would pause the suspicious send for approval while allowing the legitimate message to the manager to proceed. This selective, effect-based response is the capability examined by hypothesis H2 in Section 7.8.

9. SECURITY, PRIVACY AND OPERATIONAL ANALYSIS

9.1 Evasion and Telemetry Attacks

An attacker may attempt to make malicious activity resemble ordinary behaviour, spread actions among child agents, contaminate baselines, suppress telemetry or exploit the gap between a tool request and its eventual resource-level effect. BDSA's proposed safeguards include independent cloud evidence, signed execution metadata, parent-child execution graphs, delayed-event confirmation and controlled baseline updates. Evasion remains possible when a harmful action produces an effect that is both normal and authorized. Behavioural monitoring can reduce exposure, but it cannot replace least-privilege access, sandboxing, data-loss prevention or human approval for irreversible operations.

9.2 False Positives and Human Factors

Agent activity naturally varies. Controls that are too sensitive can overwhelm reviewers with approval requests and may encourage operators to bypass them. Alerts should therefore provide a concise causal account, including the declared goal, requested effect, violated constraint, unusual transition, data labels and identity path. Review outcomes should distinguish legitimate new behaviour from policy-design errors and malicious activity. Thresholds should reflect the impact of the operation, and operational monitoring should track alerts per execution and analyst effort rather than treating a larger alert count as evidence of better detection.

9.3 Privacy and Model Confidentiality

Capturing prompts can expose personal information, credentials, copyrighted material and confidential system instructions. The proposed architecture therefore prioritizes structured metadata, hashes and classification labels over raw content. Where content inspection is necessary, it should take place locally and export only derived features. Security telemetry should be protected through encryption, purpose limitation, retention controls and tenant separation. The detector should not depend on private chain-of-thought. Observable tool actions and signed execution context provide a more auditable basis for monitoring and avoid reliance on model-specific internal reasoning.

9.4 Availability and Safe Degradation

Because the detector may participate in decisions about high-impact tools, it can become both a performance bottleneck and a denial-of-service target. The design therefore requires locally cached policy, bounded processing queues and circuit breakers. Coverage and data-freshness indicators should be visible so that missing telemetry is not misinterpreted as low risk. When confidence is reduced, the system should degrade according to operation impact: deny irreversible changes, request approval for sensitive writes and permit only restricted low-impact work. Evidence export should remain possible even after task execution has been halted.

9.5 Governance and Accountability

Detection controls require clear ownership of each agent definition and an approved business purpose. The designated owner is responsible for tool permissions, data limits, task contracts and response settings. Security teams maintain shared invariants and assess detector quality, while cloud teams are accountable for provider-log coverage and identity mappings. Exceptions should expire and include compensating safeguards. Agents without an accountable owner or with unknown tool manifests should not be granted credentials for high-impact operations.

Governance reporting should track the share of executions with verified passports, exact cloud-event associations, current policy bundles and named owners. Incident reviews should examine both the agent's actions and the control decisions that allowed them. If an unsafe action was permitted because the task contract was too broad, retraining an anomaly model is unlikely to be the primary corrective action. This allocation of responsibility is intended to prevent behavioural monitoring from becoming a substitute for sound authorization and policy design.

9.6 Responsible Use and Dual-Use Considerations

Attack benchmarks have dual-use implications because scenarios designed for evaluation may also help someone develop attacks. The AgentDojo authors discuss this concern [8]. A future BDSA study should provide enough scenario detail for reproducibility while using sanitized trajectories and avoiding disclosure of sensitive production records. This manuscript does not claim access to real deployment telemetry or an approved operational pilot.

10. LIMITATIONS AND FUTURE WORK

The proposal has several important limitations. First, mapping a natural-language goal to permitted effects is difficult when the request is ambiguous or changes during execution. Task contracts may become too broad to provide meaningful protection or too restrictive to support useful autonomy. Second, provider audit delays and differences in event schemas complicate near-real-time correlation. Third, unsupervised models may learn incidental operational patterns rather than security-relevant behaviour, and published work shows that log-anomaly performance does not always transfer reliably to changing environments [17]. Fourth, data-flow labels depend on the accuracy of the classification process and the instrumentation of each tool. AgentDojo itself uses synthetic data and does not include a cloud identity layer; its authors also caution against treating default attacks as sufficient evidence of robustness [8]. The proposed evaluation therefore adds a cloud-event harness and reserves adaptive red-team cases for final testing, but these steps remain plans rather than completed work.

The next research stage is to implement the reference test environment, publish a provider-neutral trajectory dataset and compare graph, sequence and hybrid approaches under controlled event loss. Further work could examine privacy-preserving representation learning, cross-tenant federated baselines, causal links between untrusted content and later actions, secure protocols for agent-to-agent communication and policy generation from tool manifests. Hardware-backed attestation may strengthen execution passports, while formal methods could help establish that selected high-impact action paths are unreachable regardless of model output.

Longer-term studies should consider adaptive adversaries who understand the detector, collusion among temporary agents and harmful campaigns that remain within each task's individual limits but cause aggregate damage. Such work would require graph windows that extend beyond one execution without creating a permanent identity history for every agent. One possible approach is to retain pseudonymous relationships between resources and destinations for a limited period, while deleting task content and periodically rotating owner identifiers.

The current AgentDojo benchmark does not cover a user assigning several tasks over time without resetting the agent context, where an injected instruction might wait until a later task provides the tools needed to carry out its objective [8]. Since BDSA associates each execution with a single goal and expiry, short-lived execution may reduce this exposure. Whether it actually does so remains an empirical question that could be examined by comparing temporary and long-running agents. Extending the design to multimodal agents, another direction identified by the benchmark authors [8], would also require provenance metadata for image and audio inputs.

11. CONCLUSION

This paper has presented Behavior-Based Security Detection for Short-Lived AI Agents (BDSA), a proposed architecture that connects an agent's declared task, execution identity, tool activity, data provenance and resource effects within a task-level security trajectory. The design describes a common event representation and a correlation approach intended to connect application telemetry with cloud audit evidence. It also outlines risk-informed response options, but it does not demonstrate that the detector or response mechanisms have been implemented or validated.

The proposal builds on established capabilities, including temporary workload identities, cloud audit records and application-level telemetry. Its intended contribution is to organize these signals around a single agent execution and its declared task, rather than to claim that the underlying mechanisms are new. The central proposition—that this task context can improve detection of divergence between goals and actions while reducing the need to retain prompt content—has not yet been tested in this manuscript.

The proposed evaluation combines AgentDojo tasks with a simulated cloud-audit layer to study event correlation under controlled conditions. Such a simulator would not establish integration with multiple live cloud providers. The immediate next step is to implement one primary detection method, compare it with a simple task-policy baseline and report precision, recall, F1 score, false-positive rate, detection delay, attack success and legitimate-task completion. Until those experiments are completed, BDSA should be regarded as an architectural proposal and evaluation framework, not as a validated security product.

REFERENCES

[1] National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, Jan. 2023. doi: 10.6028/NIST.AI.100-1.

[2] National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1, July 2024. doi: 10.6028/NIST.AI.600-1.

[3] S. Rose, O. Borchert, S. Mitchell and S. Connelly, Zero Trust Architecture, NIST SP 800-207, Aug. 2020. doi: 10.6028/NIST.SP.800-207.

[4] MITRE, MITRE ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems, 2026. Available: https://atlas.mitre.org/ (accessed Sep. 22, 2026).

[5] OWASP GenAI Security Project, OWASP Top 10 for Agentic Applications 2026, Dec. 2025. Available: https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/ (accessed Sep. 22, 2026).

[6] OWASP Cheat Sheet Series, AI Agent Security Cheat Sheet, 2026. Available: https://cheatsheetseries.owasp.org/cheatsheets/AI_Agent_Security_Cheat_Sheet.html (accessed Sep. 22, 2026).

[7] S. Yao et al., ReAct: Synergizing Reasoning and Acting in Language Models, Proc. ICLR, 2023. arXiv:2210.03629.

[8] E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer and F. Tramer, AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents, Proc. NeurIPS, 2024. arXiv:2406.13352.

[9] Y. Liu et al., Prompt Injection Attack against LLM-Integrated Applications, 2023. arXiv:2306.05499.

[10] OpenTelemetry Authors, GenAI Semantic Conventions, project documentation. Available: https://github.com/open-telemetry/semantic-conventions-genai (accessed Oct. 18, 2026). The conventions are under development; verify the project status and version at submission.

[11] Amazon Web Services, CloudTrail userIdentity element and AWS STS source identity, 2026. Available: https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-event-reference-user-identity.html (accessed Sep. 22, 2026).

[12] Google Cloud, Workload Identity Federation and Audit Logging Examples, 2026. Available: https://cloud.google.com/iam/docs/workload-identity-federation (accessed Sep. 22, 2026).

[13] Microsoft, Microsoft Entra Workload ID on Azure Kubernetes Service, 2026. Available: https://learn.microsoft.com/azure/aks/workload-identity-overview (accessed Sep. 22, 2026).

[14] Cloud Native Computing Foundation, SPIFFE Concepts and Workload API, 2026. Available: https://spiffe.io/docs/latest/spiffe/concepts/ (accessed Sep. 22, 2026).

[15] M. Du, F. Li, G. Zheng and V. Srikumar, DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning, Proc. ACM CCS, pp. 1285-1298, 2017. doi: 10.1145/3133956.3134015.

[16] H. Guo, S. Yuan and X. Wu, LogBERT: Log Anomaly Detection via BERT, Proc. IJCNN, 2021. doi: 10.1109/IJCNN52387.2021.9534113.

[17] V.-H. Le and H. Zhang, Log-based Anomaly Detection with Deep Learning: How Far Are We?, Proc. ICSE, 2022. arXiv:2202.04301.

[18] Z. Xie et al., Unsupervised Anomaly Detection on Microservice Traces through Graph VAE, Proc. The Web Conference, 2023. doi: 10.1145/3543507.3583215.

[19] Cloud Security Alliance, Securing Non-Human Identities in the Age of AI Agents, Apr. 2025. Available: https://cloudsecurityalliance.org/artifacts/securing-non-human-identities-in-the-age-of-ai-agents-rsac-2025 (accessed Sep. 22, 2026).

[20] Amazon Web Services, Pass Session Tags in AWS STS, 2026. Available: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_session-tags.html (accessed Sep. 22, 2026).

[21] Google Cloud, Example Logs for Workload Identity Federation, 2026. Available: https://cloud.google.com/iam/docs/audit-logging/examples-workload-identity (accessed Sep. 22, 2026).

[22] Q. Zhan, Z. Liang, Z. Ying and D. Kang, "InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents," Findings of the Association for Computational Linguistics: ACL 2024, 2024, doi: 10.18653/v1/2024.findings-acl.624.

[23] Y. Ruan et al., Identifying the Risks of LM Agents with an LM-Emulated Sandbox, Proc. ICLR, 2024.

[24] F. Yan et al., Berkeley Function Calling Leaderboard, 2024. Available: https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html.

[25] K. Greshake et al., Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection, Proc. ACM Workshop on AI and Security (CCS), 2023. doi: 10.1145/3605764.3623985.

[26] K. Hines et al., Defending Against Indirect Prompt Injection Attacks With Spotlighting, 2024. arXiv:2403.14720.

[27] E. Wallace et al., The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions, 2024. arXiv:2404.13208.

[28] S. Willison, The Dual LLM pattern for building AI assistants that can resist prompt injection, 2023. Available: https://simonwillison.net/2023/Apr/25/dual-llm-pattern/.

[29] Y. Wu et al., "IsolateGPT: An Execution Isolation Architecture for LLM-Based Agentic Systems," arXiv:2403.04960, 2024. Verify the final author list and publication version against the cited source before submission.

[30] F. Tramer, N. Carlini, W. Brendel and A. Madry, On Adaptive Attacks to Adversarial Example Defenses, Proc. NeurIPS, 2020.