https://doi.org/10.70844/ijas.2026.3.50

Innovative Journal of Applied Science

Review Article ISSN: 3049-3072

IJAS-26-137

Bounded Autonomy: Behavioral Specification Languages and Runtime Enforcement Architectures for Trustworthy Agentic AI Systems

Harper Gough*

Independent Researcher

Corresponding Author: Harper Gough, Independent Researcher, E-mail: gharper0299@gmail.com

Received date: 11 April, 2026, **Accepted date:** 25 April, 2026, **Published date:** 30 April, 2026

Citation: Gough H (2026) Bounded Autonomy: Behavioral Specification Languages and Runtime Enforcement Architectures for Trustworthy Agentic AI Systems. Innov J Appl Sci 3(2): 50.

Abstract

Agentic AI systems autonomously plan and execute multi-step workflows using tools, external services and mutable environments. This shifts the engineering risk from single-call model correctness to end-to-end behavioral predictability across time, resources, dependencies and adversarial conditions. Current agent research largely optimizes capability, whereas the reliability and trustworthiness of agent behavior remain weakly specified, monitored and enforced at runtime. Existing AI safety work provides limited guidance for engineering trustworthy agents at current capability levels and classical software specifications and runtime verification methods do not transfer directly because agent behavior emerges from learned models interacting with tools and context rather than explicit code paths. This study frames the central deployment problem as bounded autonomy: preserving beneficial agent initiative while enforcing a behavioral envelope that is auditable, resilient and secure. This study proposes a six-dimensional human-centered AI framework adapted to agentic systems and uses it to identify gaps in behavioral specification languages, runtime enforcement architectures, competence-boundary handling and graceful degradation. It then defines evaluation metrics and methodology grounded in fault injection, resilience testing, adversarial ML and production-readiness practices and provides case analyses in safety investigation operations, infrastructure management and cyber-physical automation.

Keyword: Bounded autonomy, Agentic AI systems, Behavioral specification languages, Runtime policy enforcement, Competence boundaries, Fault injection, Machine learning

Introduction

Motivation: Emergent behavior as the new failure surface

Agentic AI systems differ from request-response model serving because they select actions over time, compose tools, create and modify artifacts and adapt plans based on intermediate outcomes. Their behavior is emergent, arising from the interaction of model capabilities, tool affordances, environmental state and latent context. In practice, this creates reliability hazards that resemble cascading dependency failures in model-as-a-service ecosystems, except that the dependency graph is now traversed by an autonomous planner rather than a static caller [1]. When an agent is connected to external APIs, code execution, ticketing systems, knowledge bases and operational control planes, "what the system can do" becomes a moving target shaped by both the available tools and environmental permissions. Capability-centric research on agents emphasizes planning, reasoning, tool use and task completion rates, as reflected in recent surveys of Large Language Model (LLM)-based and autonomous agents [2,3]. However, deployment experience in ML systems shows that the dominant risk is not average-case capability but out-of-distribution behavior, silent failure and operational fragility, especially when systems interact with complex dependencies and non-stationary environments [4]. AIOps and failure management research further demonstrate that operational failure is a lifecycle phenomenon involving monitoring, diagnosis, remediation and rollback and not a one-time model selection decision [5]. For agentic systems, the challenge sharpens because the system's "code path" is not predefined; it is synthesized at the runtime. Therefore, trustworthy deployment requires engineering bounded autonomy. Bounded autonomy means that the agent retains useful initiative inside an explicitly defined behavioral envelope, whereas runtime enforcement ensures that deviations are detected and controlled before they become incidents. This is a systems engineering problem rather than a prompt engineering problem.

Research questions and contributions

This study addresses four engineering questions. First, how should agent behavior be specified in a way that is both expressive enough to capture temporal and contextual constraints and practical enough to implement across heterogeneous toolchains? Second, what runtime enforcement architecture can monitor and constrain agent behavior with acceptable latency and without collapsing capability. Third, how can competence boundaries be operationalized for agents so that uncertainty and drift in behavior trigger safe escalation and graceful degradation rather than unbounded exploration? Fourth, how should these mechanisms be evaluated using methods that reflect real

operational conditions, including fault injection, adversarial manipulation and dependency failures? This study contributes a structured framework rather than a single algorithm. It defines a six-dimensional human-centered AI framework specialized for agentic systems, connecting specification, monitoring, enforcement and oversight to operational resilience and security. It provides a systematic gap analysis grounded in AI failure analysis and resilience literature resilient ML perspectives, production readiness guidance and deployment case study synthesis [4,6-9]. It then defines evaluation metrics and methodology by integrating fault injection, AIOps practices, adversarial ML threat models and zero-trust architecture principles [5,10,11]. Finally, it grounds the discussion in case analyses, including safety investigation operations with agentic orchestration, cyber-physical automation contexts that motivate bounded action and infrastructure management scenarios that inherit non-stationarity and cascading dependencies [12-15,1].

Background

Agentic systems and orchestration in practice

Agentic systems can be modeled as closed-loop controllers in a tool-rich environment. At each step, an agent observes the state, selects a tool invocation or internal reasoning step, receives the results, updates the state and continues until termination. Surveys of LLM-based and autonomous agents describe common design patterns, such as planning modules, memory, tool interfaces, reflection loops and multi-agent coordination [2,3]. These patterns improve task completion but also increase the state space in which failures can occur in the task. Each additional tool expands the action space, each memory mechanism expands the persistence of errors and each reflection loop can amplify incorrect assumptions. The operational significance is that reliability is now a behavioral phenomenon. The question is not whether a model produces a correct answer, but whether an agent's sequence of actions remains within acceptable bounds across time, under partial observability and under adversarial or unexpected conditions. This shifts the focus toward behavioral specification and runtime enforcement.

Reliability, resilience and failure analysis for AI systems

AI system resilience research provides taxonomies and models for understanding failures across the ML lifecycle and the operational systems in which ML components are executed [7]. Survey work on failure analysis and fault injection in AI systems emphasizes that failures are multi-layered, including data, model, infrastructure, integration and human-in-the-loop faults and argues for systematic fault injection to test resilience claims rather than relying on offline accuracy alone [6]. Resilient ML surveys reinforce the need for robustness under distribution shifts, noise and adversarial conditions, while acknowledging that resilience is not a single property but an ensemble of mechanisms, including detection, tolerance, recovery and adaptation [8]. AIOps research complements this by focusing on the operational cycle: monitoring signals, detecting anomalies, diagnosing root causes and executing remediation actions at scale [5]. AIOps is not optional for agentic systems. Agents operate in the same operational environments that AIOps already struggles to stabilize and they can become additional sources of change and load. Adepoju's analysis of cascading failure modes in model-as-a-service architectures frames a specific risk pattern: dependency chains can fail in non-obvious ways and failures propagate through systems that treat ML outputs as reliable dependencies [1]. Agents amplify this risk because they dynamically discover and traverse dependency graphs at runtime, potentially creating new cascades through repeated retries, misinterpretation of errors, or unbounded tool usage.

Trustworthy and secure ML foundations that agents inherit

The security and trustworthiness of agents inherit and intensify the classical ML security risks. Adversarial ML formalizes how attackers can manipulate training or inference to induce harmful behavior, highlighting the mismatch between attacker incentives and model assumptions [10]. Security and privacy surveys for ML broaden this to include data leakage, model extraction, inference attacks and system-level vulnerabilities that arise from the deployment of ML [16]. Safe and reliable ML perspectives emphasize that reliability is context-dependent and that safe deployment requires explicit problem formulation, monitoring and safeguards rather than assuming generalization [17]. Production readiness guidance, such as the ML Test Score, frames operational quality as a set of tests and processes that reduce technical debt in ML systems [9]. Deployment case study surveys have shown recurring failure themes, including hidden feedback loops, monitoring gaps and organizational mismatches that undermine reliability [4]. Agents inherit these risks and add new ones to the system. Tool interfaces become attack surfaces, memory becomes a persistence mechanism for injected content and autonomous planning can turn small manipulations into large sequences of harmful actions. Zero trust architecture provides a system design principle that applies directly: no component, identity, or request should be trusted by default; verification and least privilege should be continuous, contextual and explicit [11]. For agents, this implies that tool access, data access and action execution must be mediated by a policy enforcement point rather than being embedded implicitly inside the agent's runtime.

Human and organizational factors shaping agent reliability

Agentic systems are deployed within socio-technical organizations. Human cognitive biases influence how operators interpret agent outputs, escalate incidents and respond to uncertainty. Research on cognitive biases and fair AI system design argues that biases interact with AI outputs and interface design, shaping trust and decision quality [18]. A parallel lesson appears in Adepoju's review of GitHub Copilot: productivity tools can induce over-reliance and reduce verification discipline unless workflows and safeguards enforce appropriate validation [19]. The same pattern is expected for agents: when autonomy increases, verification effort often decreases unless enforced by processes and tools. The organizational context also evolves. Adepoju's mapping of IT roles and skills in the age of AI implies that successful deployment requires shifting skills toward orchestration, monitoring, governance and risk management, not only model building [20]. Therefore, agentic systems require an engineering discipline that merges formal specifications, runtime enforcement and operational governance.

https://doi.org/10.70844/ijas.2026.3.50

Innovative Journal of Applied Science

Framework: Six Dimensions of Human-Centered AI for Bounded Autonomy

Framework objective: From capability to bounded behavior

Human-centered AI is commonly discussed in terms of usability, fairness and transparency. For agentic systems, human-centered AI must be reframed as bounded autonomy: A set of engineered constraints and oversight mechanisms that keep autonomous behavior predictable, accountable and recoverable, while preserving utility. The framework below defines six dimensions that jointly determine whether an agent can be trusted in an operational context.

Six dimensions and their implications for specification and enforcement

The first dimension is the behavioral intent and scope alignment. The system must encode the objectives that the agent is authorized to pursue and those that are explicitly out of scope. In agentic settings, misalignment is often practical rather than philosophical; the agent pursues a proxy objective, optimizes for a local metric, or expands its own scope by discovering additional tools. This dimension implies specification constructs for goal scoping, task boundaries and explicit disallowed intents, as well as runtime checks that prevent scope expansion through tool discovery and chaining. The second dimension is the policy's legibility and auditability. Operators need an auditable record of what the agent did, why it did it, what it observed and what policy allowed it to do so. Traditional systems achieve this through code reviews and logs. Agents require a different artifact: a structured action trace that binds tool calls, intermediate artifacts, permissions and policy decisions to a single causal narrative. This aligns with production readiness practices that emphasize testability and traceability for ML systems and with deployment case studies that show monitoring gaps as the dominant failure drivers [4,9]. Specification languages must support trace obligations and enforcement architectures must generate tamper-evident logs. The third dimension is uncertainty and competence-boundary management. Agents operate under partial observability and uncertain reasoning models. Safe and reliable ML emphasizes that reliability must be assessed in the context and monitored over time [6]. For agents, competence boundaries must be specified as the conditions under which the agent must stop, defer, or escalate. This dimension implies specification constructs for uncertainty thresholds, ambiguity detection and required escalation paths and it implies runtime enforcement that can gate actions when uncertainty increases or when anomalous states appear. The fourth dimension is the least privilege and contextual authorization. Zero trust architecture emphasizes continuous verification and least-privilege access [11]. Agents require contextual authorization because the same tool call can be safe or unsafe depending on the environment, data sensitivity and current state. This dimension implies specification constructs for contextual permissions, information flow constraints and resource access rules and it implies an enforcement point that mediates every tool invocation based on the identity, context and policy. The fifth dimension is resilience, fault tolerance and graceful degradation of the system. Resilience taxonomies for AI systems frame resilience as detection, tolerance, recovery and adaptation across layers [7]. Fault injection surveys emphasize the systematic testing of failure modes rather than assuming robustness. For agents, resilience means that failures in tools, dependencies, or model reasoning trigger bounded behaviors, such as retry budgets, fallback strategies and safe termination, rather than uncontrolled loops. This dimension implies specification constructs for temporal properties, retry limits, circuit breakers and fallback policies, as well as runtime enforcement mechanisms that can intervene when failure patterns emerge. The sixth dimension is human oversight and accountability. Human-in-the-loop is not a UI feature; rather, it is an enforcement and governance pathway. AIOps methods emphasize failure management workflows, including escalation and remediation [5]. For agents, oversight requires defined escalation channels, approval gates for high-impact actions and clear responsibility and accountability boundaries. Cognitive bias research further implies that oversight interfaces must be designed to reduce misinterpretation and over-reliance by making uncertainty and policy constraints explicit [18]. This dimension implies specification constructs for approval requirements and escalation semantics and implies enforcement components that can pause, request approval, or hand off control to a human.

From dimensions to engineering artifacts

These dimensions are translated into three concrete artifact classes. The first class is a behavioral specification that defines the policy envelope. The second class is an enforcement architecture that mediates actions and records the evidence of compliance. The third class is an operational governance layer that defines monitoring, escalation, incident response and management of changes. The remainder of this paper uses this framework to identify gaps and define evaluation methods.

Research Gaps: Systematic Gap Identification

Gap identification method

The gap analysis treats agentic systems as layered socio-technical stacks. At the base are models and prompts; above them are tool interfaces and orchestration; above them are runtime monitoring and enforcement; and above them are organizational workflows and governance. Each layer was evaluated against the six framework dimensions. The cited literature provides strong coverage of capability development in agents and strong coverage of general AI reliability and resilience concepts, but does not provide deployable mechanisms for bounded autonomy [2,3,6,8,17,18].

Behavioral specification languages for agents

The first gap is the absence of practical behavioral specification languages designed for such agentic actions. Existing safety controls for agents often reduce to action whitelists and simple permission toggles. This is insufficient because harmful behavior is often temporal and contextual. A safe action in isolation becomes unsafe when repeated, combined with other actions, or executed in a different context. Agentic behavior must be constrained by temporal properties such as "never execute irreversible actions without approval," "never perform more than N retries in T time," and "never exfiltrate sensitive data across a trust boundary." It must also be

constrained by resource bounds, including cost budgets, compute limits and rate limits, because agents can create denial-of-service patterns through unbounded tool invocation, echoing cascading failure risks in dependency-driven architectures [1]. Classical formal methods can express temporal properties, resource bounds and information flow restrictions; however, they assume a program structure that is explicitly coded. Agents synthesize behavior at runtime from learned representations and the action set can be open-ended through the discovery of tools. Therefore, the gap is not expressivity in principle; it is an engineering language that maps to agent actions, tool semantics and organizational permissions without requiring full formal modeling of the underlying model.

Runtime enforcement architectures that preserve capability

The second gap is runtime enforcement, which can monitor agent behavior against specifications and intervene appropriately without unacceptable latency or capability collapse. AIOps research provides methods for failure management but typically assumes deterministic services and well-defined telemetry [5]. Fault injection surveys have emphasized that AI systems fail across layers and that resilience requires systematic testing [6]. Agentic systems add a new difficulty: enforcement must operate on partially observed intent and tool calls whose semantics may be complex, stateful and external. Enforcement actions in agentic systems require a controlled repertoire that includes blocking unsafe actions, modifying actions into safe variants, inserting guard steps such as confirmation or sanitization and escalation to humans. These interventions must be bounded and auditable. Without careful design, enforcement can either be too permissive, allowing harmful sequences, or too strict, preventing useful autonomy and driving operators to bypass the controls. The missing element is an architecture that implements policy decision points and policy enforcement points for every tool invocation, integrates the least privilege from zero trust principles and maintains low latency through caching, pre-authorization and incremental policy evaluation [11].

Formal reasoning about behavioral envelopes and competence boundaries

The third gap is the formal reasoning about whether a given specification is sufficient to bound the behavior under adversarial or unexpected conditions. Adversarial ML studies show that attackers adapt to the system's assumptions and security and privacy surveys show broad attack surfaces at the system level [10,16]. For agents, this means that even if a policy restricts explicit tool calls, an attacker may induce the agent to create indirect pathways, such as writing scripts that are later executed, embedding instructions in artifacts that are re-consumed, or exploiting tool side effects. Resilient AI taxonomies emphasize systematic methods and models for resilience and resilient ML surveys emphasize robustness and tolerance mechanisms; however, they do not provide a behavioral envelope calculus for agentic action sequences [7,8]. The missing element is a method to reason about the compositions of tool semantics, policy constraints and model behavior uncertainty to establish that certain classes of unsafe outcomes are unreachable or that the risk is bounded within acceptable limits. Competence boundaries are specific instances. Safe and reliable ML emphasizes context-dependent reliability [17]. Agents require competence boundary specifications that define when the system must stop acting autonomously because the state is ambiguous, the environment is adversarial, the action impact is high, or the model's uncertainty signals indicate unreliability. This boundary must be enforced at runtime, not only documented.

Graceful degradation for agentic systems

The fourth gap is the graceful degradation strategies for agents. Graceful degradation defines the behavior of a system when it approaches policy limits, resource budgets, uncertainty thresholds, or competence boundaries. This includes how the agent should simplify plans, switch to lower-risk tools, request approvals, or hand off to humans. In production ML, readiness and technical debt guidance emphasize testing, monitoring and rollback processes and deployment case studies highlight the need for fallback behavior when models behave incorrectly [4,9]. In AI failure analysis, fault injection emphasizes that systems must be evaluated under faults to validate resilience claims [6]. Agentic systems still lack standardized degradation patterns, such as circuit breakers, safe-mode tool subsets and escalation protocols that preserve continuity while preventing uncontrolled behavior. This gap is particularly important in high-stakes workflows, such as safety investigation operations, where agentic orchestration can accelerate triage and evidence linking but must not contaminate evidence, leak sensitive data, or take unauthorized actions [12]. Without graceful degradation, agents either remain overly constrained and unusable or become operationally liable.

Evaluation Metrics and Methodology

Metrics for bounded autonomy

The evaluation must measure both capability and constraint compliance under realistic conditions. Capability metrics remain necessary, including task completion, time-to-completion and quality of the produced artifacts, consistent with agent capability research [2,3]. Bounded autonomy adds compliance metrics that measure whether action sequences satisfy behavioral specifications. The core compliance measure is the violation rate of specified temporal, resource and information flow properties. Because enforcement operates under uncertainty, evaluation must also measure the false intervention rate, where safe actions are blocked or modified and the missed violation rate, where unsafe behavior passes through. Runtime enforcement imposes an overhead. The latency overhead must be measured per tool invocation and end-to-end and the distribution is important because tail latency can break operational workflows. Resource overhead must be measured as additional computation, token usage and external API calls induced by monitoring and policy evaluation. Resilience metrics must reflect fault tolerance and recovery capabilities. Failure analysis and fault injection research suggest evaluating system behavior under injected faults across the data, model and infrastructure layers [6]. For agents, fault injections include tool timeouts, malformed tool outputs, dependency outages, rate-limit errors, permission denials and adversarial content inserted into tool responses. The metrics include the rate of safe termination versus unsafe looping, number of retries before fallback and stability

of behavior under repeated faults. Resilience taxonomies suggest measuring detection, tolerance, recovery time and adaptation quality [7]. AIOps perspectives suggest measuring the time to detect, diagnose and mitigate anomalies within operational workflows [5]. Security metrics must reflect the adversarial pressure. Adversarial ML threat models imply testing against prompt injection, data poisoning in the retrieved context and exploitation of model blind spots [10]. Security and privacy surveys imply testing for data leakage and unintended information flows across trust boundaries [7]. Zero trust principles imply measuring the correctness of least privilege, including whether the agent can access only the minimum required resources and whether contextual authorization decisions are consistent [2]. The evaluation must also measure audit completeness, including whether action traces are sufficient to reconstruct why an action was permitted or blocked, aligned with production readiness practices emphasizing testability and traceability [9].

Methodology: Specification-driven evaluation under faults and attacks

The evaluation methodology was specification-driven. The starting point is a behavioral specification suite composed of temporal safety, resource and information flow constraints. The system under test is a full agent stack, including the model, orchestration, tool interfaces, monitoring, enforcement and logging. The evaluation proceeds in three layers. The first layer is an offline trace check. Recorded agent action traces were validated against specifications to measure baseline violation rates and diagnose where violations occurred. This layer also supports regression testing: when model versions, prompts, or tools change, the same trace suite can be replayed or simulated to assess behavioral drift, consistent with production readiness emphasizing tests that reduce technical debt [9]. The second layer involves online fault injection and adversarial testing. Fault injection is applied to tool calls and dependencies to validate resilience claims, following the AI system fault injection guidance [6]. Adversarial testing is applied to the retrieved context, tool outputs and user inputs, following adversarial ML threat models and security and privacy considerations [10,16]. The objective was not only to measure task completion but also to determine whether the enforcement architecture prevents specification violations under realistic stress. The third layer is the operational workflow evaluation. Because bounded autonomy aims for trustworthy deployment, evaluation must include human oversight behavior and incident response. AIOps methods imply that monitoring, triage and remediation must be evaluated as system properties and not only as human performance [5]. This includes measuring whether escalations occur at the right time, approvals are triggered appropriately and logs support rapid diagnosis. Cognitive bias considerations imply that operator-facing artifacts must reduce misinterpretation and overreliance by presenting the policy state, uncertainty signals and reasons for enforcement actions in a structured form [18].

Baselines and comparators

Comparators are necessary to isolate the effects of bounded autonomy mechanisms. A natural baseline is an unconstrained agent that directly uses tools. The second baseline is a permission-only agent that uses static allowlists. The bounded autonomy system adds behavioral specifications, runtime enforcement and graceful degradation. The methodological objective is to quantify the trade-off between capability and predictability and to show that enforcement improves safety and resilience under faults and attacks without unacceptable capability degradation.

Case Analysis: Illustrative Applications

Safety investigation operations with agentic orchestration

Safety investigation operations involve gathering heterogeneous evidence, correlating incidents, preserving the chain of custody and producing structured case narratives. Adepoju proposed a reference architecture and benchmark direction for safety investigation operations using agentic orchestration, framing the domain as one where autonomy can improve throughput, but errors have high consequences [12]. Bounded autonomy is essential in this domain because the agent must not contaminate evidence, fabricate claims, access unauthorized sources, or disclose sensitive information outside permitted channels. A behavioral specification language for this case must express temporal properties, such as requiring human approval before actions that alter official records, enforcing immutability constraints on evidence artifacts and requiring that every claim in a report be linked to source artifacts. Runtime enforcement must mediate tool calls to evidence stores and ticketing systems, enforce the least privilege per case and maintain tamper-evident action logs. Fault injection is necessary because investigation environments involve missing data, inconsistent records and intermittent access and resilience requires graceful degradation that escalates uncertainty rather than improvisation.

Infrastructure management and smart water grid operations

Infrastructure management combines sensor-driven detection and operational interventions. Adepoju's targeted review of deep learning for leak detection in smart water grids illustrated an operational setting in which models drive alerts and decisions in complex, distributed systems [15]. In this context, agents can autonomously correlate leak signals, open work orders, schedule inspections and coordinate field teams. The risk is that distribution shifts, sensor drift and dependency outages can induce false interventions or missed critical events. Bounded autonomy requires resource and impact constraints. Specifications must bound actions, such as valve control, dispatch scheduling and customer notifications. Runtime enforcement must incorporate contextual permissions derived from the operational state and safety constraints, consistent with the zero-trust principles applied to operational control planes [11]. Fault injection should include sensor drift, missing telemetry and dependency outages, which are aligned with AI fault injection guidance [6]. Graceful degradation should restrict the agent to advisory mode when uncertainty rises or when telemetry integrity is compromised, aligning with safe and reliable ML principles that emphasize context-dependent reliability [17].

Cyber-physical automation analogs from smart building control

Although building automation is not identical to agentic orchestration, it provides a concrete analogy for bounded action under non-stationarity. Adepoju's work on the impact of occupancy prediction on HVAC control and API frameworks for real-time occupancy-based HVAC integration highlights how AI-driven automation operates under distribution shift, sensor drift and equipment degradation [13]. In such settings, safe performance requires uncertainty signaling, drift detection and a graceful fallback to conservative control and failures can propagate through integration layers. Agents operating in cyber-physical environments inherit similar constraints. Tool calls become actuations and planned errors become physical outcomes. Behavioral specifications must include temporal and safety properties that prevent unsafe actuation sequences and enforce resource constraints that prevent oscillatory behaviors. Runtime enforcement must integrate competence boundary handling so that agents degrade to advisory roles when the models drift or when the actuation risk is high. The architectural lessons from API integration emphasize that enforcement must be at the mediation layer where actions are executed, not only inside the agent's internal logic [14].

Software delivery and model-as-a-service ecosystems

Agentic systems are increasingly deployed in software delivery workflows, including code generation, dependency updates, incident triage and remediation. Adepoju's review of GitHub Copilot highlighted productivity gains and the parallel need for verification discipline and oversight to prevent over-reliance [19]. Cascading failure analysis in MaaS architectures highlights how dependency chains fail and propagate when systems treat model outputs as stable dependencies [1]. Agents combine both: They generate artifacts that can enter production and traverse dependency chains autonomously. Bounded autonomy in this context requires information flow constraints that prevent credential leakage, temporal constraints that require approval before merging or deploying changes and resource constraints that limit repeated attempts against failing services. Runtime enforcement must be integrated with CI/CD control points and ticketing systems, apply policy decisions consistently and generate audit trails aligned with production readiness expectations [9]. Operational evaluation must use fault injection and deployment failure scenarios consistent with AI system failure analysis and AIOps practices [6,5].

Discussion

Implications: Toward an engineering discipline for agentic systems

Bounded autonomy reframes the deployment of agents. The central design object is the behavioral envelope, not the prompt. Capability research remains necessary, but deployment requires specification, enforcement and resilience engineering, grounded in operational reality. AI system failure analysis shows that without fault injection and systematic testing, reliability claims are fragile [6]. Resilience taxonomies show that resilience must be designed across layers and not assumed from model robustness alone [7]. Deployment case studies have shown that monitoring and organizational workflows often dominate outcomes [4]. For agents, these lessons imply that trustworthy behavior must be achieved through runtime mediation, continuous verification and safe degradation. Zero trust architecture provides a structural template for runtime enforcement, including continuous verification, least privilege and explicit policy decision points [11]. Security and privacy surveys for ML show that threats span the data, model and deployment layers [16]. For agents, tool interfaces are the locus of enforcement because they are where intent becomes an action. This implies that the enforcement architecture must be designed as a first-class control plane rather than as a wrapper around a single model. Human-centered framing is therefore necessary. Cognitive bias research indicates that operator behavior can be predictably distorted by interface design and the perceived authority of AI outputs [18]. The Copilot experience indicates that productivity tools shift verification behavior unless processes and safeguards enforce appropriate checks [19]. For agents, this implies that audit artifacts, uncertainty signals and enforcement decisions must be legible and structured to support appropriate oversight.

Limitations and open technical constraints

Formal methods face a foundational constraint: the learned models resist full behavioral proof because the behavior is not explicitly coded. This renders complete formal verification unrealistic in the near term. The practical target is partial specification with runtime enforcement and empirical validation under stress conditions. Fault injection provides empirical evidence but cannot exhaust the state space [6]. Adversarial ML shows that attackers adapt, which implies a moving target for threat modeling [10]. Zero trust reduces the blast radius but does not eliminate misconfigurations and insider risks [11]. Enforcement also creates trade-offs. Aggressive constraints can degrade capabilities and drive bypass behavior, whereas permissive constraints can leave residual risks. Achieving acceptable trade-offs requires calibrated policies, well-designed escalation and graceful degradation patterns that preserve operational usefulness. Latency and cost overhead are also binding constraints in production environments, especially when enforcement mediates each tool call. The final constraint is ecosystem standardization. Agent tool semantics are heterogeneous and without standard representations of tool effects, contextual permissions and information flow, specification languages and enforcement architectures remain fragmented. The most plausible path is incremental standardization around action schemas, policy decision interfaces and audit log formats, aligned with production readiness practices that emphasize testability and monitoring and AIOps practices that emphasize operational integration [5,9].

Conclusion

This study frames trustworthy agentic AI deployment as a bounded autonomy problem: Maintaining useful autonomy while enforcing a behavioral envelope that constrains temporal sequences, resources, information flows and contextual permissions. Capability-focused agent research describes architectures and patterns for planning and tool use but trustworthy deployment requires behavioral specification languages and runtime enforcement architectures that

are currently lacking. By grounding the problem in AI failure analysis and fault injection resilience models resilient ML perspectives safe and reliable ML principles production readiness practices and deployment case study evidence this study identifies concrete gaps in specification expressivity, runtime enforcement design, competence boundary handling and graceful degradation. It defines evaluation metrics and methodology that treat compliance, resilience under faults, security under adversarial pressure and operational oversight as measurable system properties. The case analyses show that bounded autonomy is the enabling condition for deploying agents in high-stakes domains such as safety investigation operations infrastructure management and cyber-physical automation analogs where uncontrolled emergent behavior is unacceptable.

Conflict of interest

The author declares no conflict of interest.

References

1. Adepoju S (2023) Cascading failure modes in model-as-a-service architectures: When your dependencies think. International Journal of Scientific Research in Civil Engineering 7(6): 109-120. [Crossref] [GoogleScholar]
2. Xi Z, Chen W, Guo X, He W, Ding Y, et al. (2023) The rise and potential of large language model-based agents: A survey. arXiv preprint. [Crossref] [GoogleScholar]
3. Wang L, Ma C, Feng X, Zhang Z, Yang H, et al. (2024) A survey on large language model based autonomous agents. Frontiers of Computer Science 18(6) 186345. [Crossref] [GoogleScholar]
4. Paleyes A, Urma RG, Lawrence ND (2022) Challenges in deploying machine learning: A survey of case studies. ACM Computing Surveys 55(6): 1-29. [Crossref] [GoogleScholar]
5. Notaro P, Cardoso J, Gerndt M (2021) A survey of AIOps methods for failure management. ACM Transactions on Intelligent Systems and Technology 12(6): 1-45. [Crossref] [GoogleScholar]
6. Cheng P, Xu Z, Wang Z, Zhao Y, Li T, et al. (2024) A survey on failure analysis and fault injection in AI Systems. ACM Transactions on Software Engineering and Methodology 35(1): 1-42. [Crossref] [GoogleScholar]
7. Kharchenko V, Illiashenko O, Morozova O, Sokolov S (2023) Resilience and resilient systems of artificial intelligence: Taxonomy, models and methods. Algorithms 16(3): 165. [Crossref] [GoogleScholar]
8. Kumar A, Goyal N (2017) Survey on resilient machine learning. arXiv preprint. [Crossref] [GoogleScholar]
9. Breck E, Cai S, Nielsen E, Salib M, Sculley D (2017) The ML test score: A rubric for ML production readiness and technical debt reduction. In Proc IEEE Big Data 1123-1132. [Crossref] [GoogleScholar]
10. Huang L, Joseph AD, Nelson B, Rubinstein BIP, Tygar JD (2011) Adversarial machine learning. In Proc. 4th ACM Workshop on Security and Artificial Intelligence 43-58. [Crossref] [GoogleScholar]
11. Rose S, Borchert O, Mitchell S, Connelly S (2020) Zero trust architecture. NIST Special Publication 800-207. [Crossref] [GoogleScholar]
12. Adepoju SA, Adepoju MA (2026) From portals to case graphs: A reference architecture and benchmark for safety investigation operations with agentic orchestration. Journal of Information Systems Engineering and Management 11(1) 1-15. [GoogleScholar]
13. Adepoju SA (2025) How machine learning can revolutionize building comfort: Accessing the impact of occupancy prediction models on HVAC control system. World Journal of Advanced Research and Reviews 25(1) 2315-2327. [Crossref] [GoogleScholar]
14. Adepoju S, David S (2025) An intelligent API framework for real-time occupancy-based HVAC integration in smart building management systems. Journal of Knowledge Learning and Science Technology 4(1): 61-70. [Crossref] [GoogleScholar]
15. Adepoju S (2024) Deep learning for smart water grids: A targeted review of leak detection technologies. International Journal of Scientific Research in Computer Science, Engineering and Information Technology 10(1) 344-355. [Crossref] [GoogleScholar]
16. Papernot N, McDaniel P, Sinha A, Wellman MP (2018) SoK: Security and privacy in machine learning. In Proc. IEEE European Symposium on Security and Privacy (EuroS&P) 399-414. [Crossref] [GoogleScholar]
17. Saria S, Subbaswamy A (2019) Tutorial: Safe and reliable machine learning. arXiv preprint. [Crossref] [GoogleScholar]
18. Adepoju S, Adepoju M (2025) Cognitive biases: Understanding and designing fair AI systems for software development. Journal of Knowledge Learning and Science Technology 4(2) 44-54. [Crossref] [GoogleScholar]
19. Adepoju S (2023) Github copilot's impact on developer productivity: A review of early evidence. International Journal of Scientific Research in Science and Technology 10(4) 814-822. [Crossref] [GoogleScholar]
20. Adepoju MAO, Adepoju SA (2025) Mapping the evolution: IT roles and skills requirements in the age of AI. World Journal of Advanced Research and Reviews 25(2): 1-11. [Crossref] [GoogleScholar]