ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 08, August 2026

Unified Runtime Monitoring, Explainable Risk Scoring and Adaptive Control for Secure LLM Agents

[1] Deepak Kumar Gour, [2] Dr. Sushma Agrawal

[1] [2] Jyoti Vidhyapeeth Woman's University, Jaipur, Rajasthan, India
Author Email: [1] deepakkumargour@gmail.com, [2] sushma.agrawal@jvwu.com

Abstract*— Large language model (LLM) applications are moving beyond passive text generation toward agentic systems that retrieve external knowledge, use tools, call APIs, and interact with enterprise data within the same request lifecycle. This shift introduces runtime security concerns that conventional infrastructure monitoring, offline safety testing, and standalone content moderation do not address well. AgentGuard AI is developed as a unified runtime governance framework for these environments. It brings together three mechanisms: the Secure Runtime Monitoring Protocol (SRMP), which records infrastructure telemetry, model behaviour, agent actions, retrieval events, security alerts, and audit information in a common event structure; Agentic and Accelerator-Aware Risk Scoring (AARS), which produces an interpretable composite risk score from six dimensions; and the Risk-Adaptive Scheduling Strategy (RASS), which translates the resulting risk profile into an operational response such as allowing the request, throttling it, routing it for human review, or blocking it. The prototype was evaluated on 300 controlled requests representing normal, attack, long-context, and tool-intensive workloads. Attack detection increased from 28.3% with the baseline logging configuration to 85.0% with AgentGuard AI, a relative improvement of about 200%. Audit coverage reached 100%. A lightweight threat classifier based on TF-IDF and Logistic Regression achieved 97.8% accuracy and a 97.8% F1 score under 5-fold cross-validation. Monitoring added approximately 8-12% runtime overhead. Taken together, the results show that combining event capture, multidimensional risk assessment, and deterministic runtime controls can strengthen the security and auditability of LLM-agent deployments without sacrificing interpretability or practical deployability.

Index Terms*— AgentGuard AI, AI agents, AI observability, GPU telemetry, human-in-the-loop governance, LLM security, Prompt injection, Retrieval-augmented generation, Risk scoring, Runtime monitoring.

I. INTRODUCTION

Large language models have become part of routine enterprise workflows, including customer support, document analysis, software development, research assistance, and decision support. At the same time, the way these systems are deployed is changing. Many applications now allow a model to retrieve documents, query databases, invoke external tools, write files, call APIs, and make several linked decisions before returning a result. Once a model can act on external systems, the security problem extends beyond the quality or safety of generated text.

Conventional observability platforms are effective at reporting CPU and memory utilisation, latency, service traces, and application logs. They are not, however, designed to interpret prompt injection, poisoned retrieval content, jailbreak attempts, unsafe tool calls, or efforts to extract hidden instructions. The opposite limitation is seen in many AI safety and red-teaming approaches: they can reveal model weaknesses during testing, but they do not necessarily provide a decision at the moment a live request is being executed. An agent can therefore appear healthy from an infrastructure perspective while still taking an unsafe action or exposing protected information.

AgentGuard AI addresses this gap by treating runtime governance as a problem of correlated evidence rather than isolated alerts. For each request, the framework collects signals from the infrastructure, the model, the agent, the retrieval layer, security detectors, and prior user behaviour. These signals are preserved in a common event record, converted into an explainable risk score, and then used to select a proportionate runtime action through SRMP, AARS, and RASS respectively.

The contribution of the work is fivefold. It defines a structured event protocol for monitoring AI-agent activity at runtime; introduces a weighted, request-level risk model whose component contributions remain visible; links the resulting risk profile to explicit control actions; implements the design in a working prototype; and evaluates the prototype against a conventional logging baseline using security, governance, and performance measures.

II. RELATED WORK AND RESEARCH GAP

Modern LLMs are rooted in the transformer architecture, which replaced recurrent sequence processing with attention-based modelling [1]. Larger pretrained models, instruction tuning, and retrieval-augmented generation subsequently extended their usefulness to broad, open-ended tasks [2]-[4]. The emergence of tool-using agents changes the security implications of these advances because model output may now determine which external action is selected and executed.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 08, August 2026

Research on LLM application security has documented prompt injection, indirect prompt injection, jailbreaks, data leakage, model denial of service, insecure plugins, and excessive agency as recurring threat classes [5]-[9]. Indirect prompt injection is particularly relevant to retrieval-augmented and agentic applications because the malicious instruction may arrive through retrieved content rather than the user's visible prompt [5]. The OWASP Top 10 for LLMs and GenAI Applications similarly identifies prompt injection and model denial of service among the principal application-level risks [9].

Work on LLM serving has largely concentrated on throughput and memory efficiency, including PagedAttention and improved management of the KV cache [10]. Infrastructure platforms such as NVIDIA DCGM expose detailed accelerator telemetry, but those measurements are normally interpreted as performance signals rather than evidence of application-level AI risk. In parallel, the NIST AI Risk Management Framework, its Generative AI Profile, and the EU AI Act emphasise risk management, traceability, transparency, documentation, and human oversight [11]-[13]. Meeting such requirements in operational systems depends on evidence that is produced while the system is running, not only on policies prepared before deployment.

The resulting tool landscape is fragmented. Infrastructure monitoring can explain resource pressure, LLM observability can reconstruct traces and cost, red-team scanners can expose weaknesses, and governance frameworks can define organisational obligations. What remains less developed is a runtime mechanism that connects these layers and uses their combined evidence to support a request-level decision. AgentGuard AI is positioned at this intersection by joining monitoring, scoring, and control within a single governance path.

Table I: Positioning of Agentguard AI Against Existing Monitoring and Governance Approaches

Approach	Typical Capability	Core Limitation	AgentGuard AI Position
Infrastructure observability	CPU, memory, GPU, latency, logs and traces	No awareness of prompt injection, unsafe tool-use or RAG poisoning	Adds AI-specific security and agent-risk context to runtime telemetry
LLM observability	Prompt traces, token usage, model calls, cost and feedback	Limited action-control logic and weak infrastructure/security correlation	Links model behaviour to risk bands and explicit runtime responses
AI red-teaming / scanners	Offline adversarial probes and vulnerability discovery	Not designed for request-by-request runtime enforcement	Applies threat detection and audit logging during live request execution
AI governance frameworks	Policy requirements for risk management, auditability and oversight	Generally not a technical implementation	Implements monitoring, scoring, human review, and auditable runtime evidence

III. PROBLEM DEFINITION AND THREAT MODEL

Consider an LLM-agent request q. During execution, the environment produces several classes of evidence: infrastructure telemetry T, model metrics M, agent tool-use metrics A, retrieval metrics G, security indicators S, and user-history indicators U. The task is to combine these heterogeneous signals into an interpretable total risk score R_total and an operational action a, while retaining enough information to reconstruct the decision later for audit or forensic analysis.

The threat model assumes that an adversary can influence user prompts, attempt jailbreaks, place malicious instructions in retrievable content, generate unusually long or high-volume requests, seek excessive tool privileges, or attempt data exfiltration. Compromise of the host operating system, model weights, GPU drivers, or the underlying cloud platform is outside the scope of this study. The framework therefore focuses specifically on security and governance at the LLM-application runtime layer.

Table II: Threat Categories Handled by the SEMP Security Detection Layer

Threat Category	Typical Trigger Pattern	Risk Rationale
Prompt injection	Ignore previous instructions; override policy; new system instruction	Attempts to override developer or system controls
Jailbreak attempt	DAN mode; unrestricted role play; bypass safety filters	Seeks policy evasion and harmful output generation
System-prompt	Reveal hidden instructions; show system	Targets confidential control logic and prompt

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 08, August 2026

Threat Category	Typical Trigger Pattern	Risk Rationale
extraction	prompt	assets
Data leakage attempt	Export user data; send records externally	Creates confidentiality and compliance risk
Unsafe tool instruction	Delete files; execute without approval; access admin API	May convert model output into real-world operational harm
Excessive agency	Grant admin access; bypass permissions; autonomous escalation	Indicates privilege misuse or runaway agent behaviour
Sensitive data exposure	API key, password, secret token or private record request	Raises credential and personal-data leakage risk
Suspicious tool request	Unusual tool sequence or repeated privileged invocation	May indicate reconnaissance, exploitation or denial-of-service behaviour

IV. PROPOSED AGENTGUARD AI FRAMEWORK

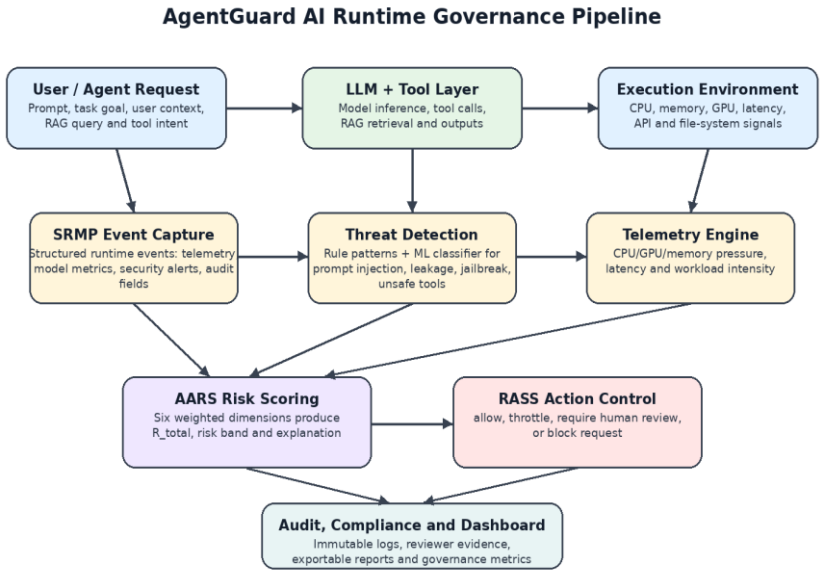

Figure 1. Unified monitoring, scoring and action-control pipeline for secure LLM agents.

A. Secure Runtime Monitoring Protocol (SRMP)

SRMP defines a common request-level event record for LLM and agent applications. Each record can be serialised to JSON or CSV and linked to the originating request through a persistent identifier. The schema captures prompt characteristics, model identity, user identity, infrastructure utilisation, token counts, context length, tool activity, retrieved sources, detected security conditions, the AARS score, the RASS decision, and the final audit status. Keeping the raw runtime evidence and the derived decision in the same record makes it possible to reconstruct why a request was allowed, escalated, throttled, or blocked.

Table III: Condensed SRMP Event Schema for Journal Presentation

Schema Group	Representative Fields	Purpose
Identity and traceability	event_id, request_id, timestamp, user_id, model_name	Links all runtime events to one request and one decision
Prompt and model behaviour	prompt_type, prompt_tokens, output_tokens, context_length, latency_ms	Captures model workload and abnormal context usage
Infrastructure telemetry	cpu_usage, memory_usage, gpu_usage, gpu_memory_mb	Detects resource pressure and possible denial-of-service patterns
Agent and RAG activity	tool_call_count, highest_tool_risk,	Quantifies agency and retrieval-channel risk

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 08, August 2026

Schema Group	Representative Fields	Purpose
	rag_source_count	
Security and governance	security_flags, risk_score, risk_band, recommended_action, explanation, status	Enables explainable action decisions and auditability

B. Agentic and Accelerator-Aware Risk Scoring (AARS)

AARS reduces the collected evidence to six normalised component scores and combines them into a value between 0 and 1. The six components represent infrastructure pressure, model-runtime behaviour, agent tool use, retrieval exposure, detected security severity, and user-history risk. They are kept separate before aggregation so that the final score can be explained in terms of the signals that contributed to it rather than treated as an opaque classification.

$$Rtotal = 0.15\,Rinfra + 0.15\,Rmodel + 0.20\,Ragent + 0.15\,Rrag + 0.25\,Rsecurity + 0.10\,Ruser$$

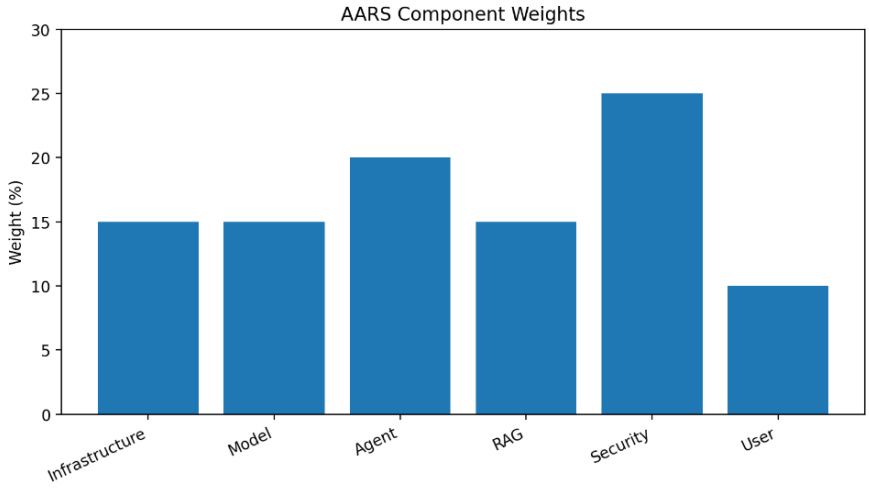

Figure 2. AARS component weights used in the prototype evaluation.

Table IV: AARS Component Definitions and Interpretability Rationale

Component	Weight	Primary Signals	Interpretation
R_infra	15%	CPU, memory, GPU usage, GPU memory, latency	Sustained system pressure can indicate overload, long-context stress, or a denial-of-service pattern
R_model	15%	Prompt tokens, output tokens, context length	Unusual context or output behaviour can indicate extraction, stuffing, or abnormal generation
R_agent	20%	Tool-call count, highest tool privilege	Frequent or privileged tool invocation increases the operational impact of a request
R_rag	15%	Number and trust level of retrieved sources	Greater exposure to untrusted retrieval sources increases indirect-injection risk
R_security	25%	Threat category and severity	Direct security detections provide the strongest evidence of malicious intent
R_user	10%	Historical high-risk request behaviour	Repeated high-risk behaviour increases the likelihood of a persistent adversarial pattern

C. Risk-Adaptive Scheduling Strategy (RASS)

RASS converts the AARS result into an operational response. Requests in the low-risk band are allowed and logged, while medium-risk requests are retained with fuller audit evidence. When infrastructure pressure is the dominant concern, the request can be throttled rather than rejected. High-risk model or agent behaviour can be routed for human review, and critical security events are blocked. The policy also includes component-level overrides; for example, maximum security severity triggers a block even if averaging with lower-risk components would otherwise reduce the total score.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 08, August 2026

Table V: Risk Bands and RASS Action Mapping

Risk Score	Risk Band	Default RASS Action	Operational Meaning
0.00-0.25	Low	allow	Process normally and log event
0.26-0.50	Medium	allow with full logging	Elevated concern; retain evidence for review
0.51-0.75	High	require_human_review / throttle	Pause, rate-limit or seek approval based on dominant component
0.76-1.00	Critical	block_request	Reject request, record full event and alert security
Any score with R_security = 1.0	Critical override	block_request	Security severity overrides averaged risk score

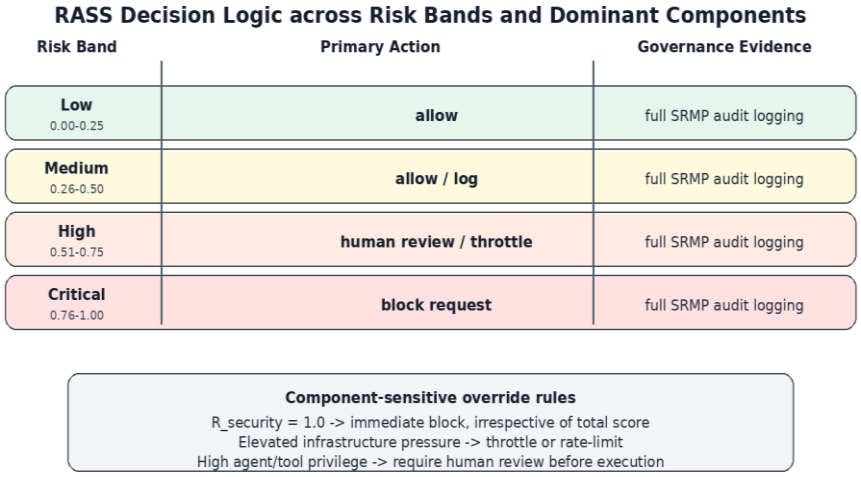

Figure 3. Component-sensitive RASS decision logic across risk bands.

D. Algorithmic Workflow

Algorithm : AgentGuard AI Runtime Governance
Input: request q, telemetry T, model metrics M, agent metrics A, retrieval metrics G, user history U
Output: SRMP event e, risk score R_total, action a
1. *Create SRMP event header with event_id, request_id, timestamp and model metadata.*
2. *Capture telemetry* $T = \{cpu, memory, gpu\_usage, gpu\_memory, latency\}$.
3. *Capture model metrics* $M = \{prompt\_tokens, output\_tokens, context\_length\}$.
4. *Capture agent/RAG metrics* $A, G = \{tool\_call\_count, highest\_tool\_risk, rag\_source\_count\}$.
5. *Detect threats S using rule − based patterns and optional ML threat classifier.*
6. *Compute component scores R_infra, R_model, R_agent, R_rag, R_security and R_user.*
7. *Compute R_total using the AARS weighted formula.*
8. *Assign risk band using threshold intervals.*
9. *Apply RASS rules and component − sensitive override conditions.*
10. *Write full SRMP audit record and return action a.*

V. EXPERIMENTAL DESIGN

Evaluation was carried out with a controlled set of 300 simulated LLM-agent requests covering four workload classes: normal activity, attacks, long-context requests, and tool-intensive activity. Every request produced SRMP-compatible runtime fields, an AARS score, a risk band, and a RASS action. The comparison baseline represented conventional runtime logging without integrated AI-specific risk scoring or action control.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 08, August 2026

Table VI: Experimental Design Summary

Element	Configuration
Sample size	300 monitored LLM/agent requests
Workload classes	Normal, attack, long-context and tool-intensive requests
Baseline	Conventional runtime logging without integrated SRMP, AARS, or RASS control
Proposed system	AgentGuard AI with SRMP event capture, AARS scoring, RASS decision logic, audit records, and dashboarding
Threat detector	Rule-based security detection with a TF-IDF + Logistic Regression classifier
Validation method	Baseline comparison and 5-fold cross-validation for ML threat detection
Primary metrics	Attack detection rate, audit coverage, risk-control action rate, classifier accuracy, F1 score and monitoring overhead

The experiment examined three hypotheses.

H1: Integrated runtime monitoring increases attack detection relative to conventional baseline logging.

H2: Structured risk scoring coupled with deterministic action control provides complete auditability and proportionate intervention.

H3: The additional runtime cost of monitoring remains acceptable for enterprise governance use cases when considered against the security and compliance benefit.

VI. RESULTS

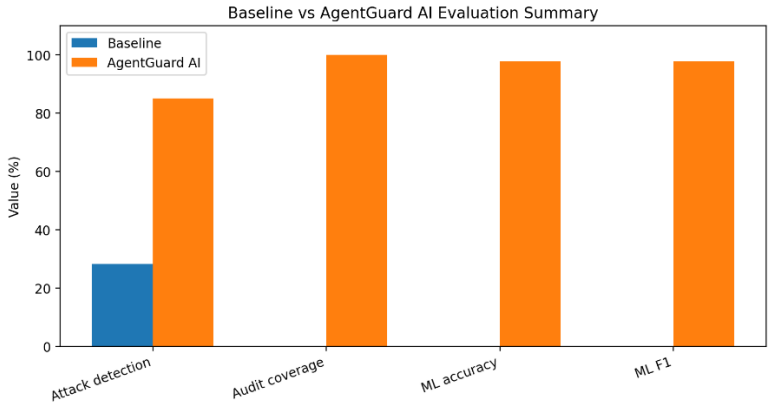

Figure 4. Baseline comparison and ML classifier performance from the controlled evaluation.

Table VII: Evaluation Results Summary

Metric	Baseline	AgentGuard AI	Interpretation
Attack detection rate	28.3%	85.0%	About 200% relative improvement in attack detection
Audit coverage	Not complete	100%	A structured audit record is produced for every processed request
Critical request blocking	Not available	29.7%	RASS adds active runtime intervention rather than passive observation alone
ML classifier accuracy	Not applicable	97.8%	Strong prompt-threat classification under cross-validation for the evaluated dataset
ML classifier F1 score	Not applicable	97.8%	Precision and recall remained balanced on the evaluated threat dataset
Monitoring overhead	0%	8-12%	The additional latency is measurable and must be weighed against governance and security requirements

A. Security Detection Improvement

Attack detection increased from 28.3% under the baseline to 85.0% with AgentGuard AI. The relative gain was (85.0 - 28.3)/28.3 x 100 = 200.4%. The improvement is important because the framework does not depend on a single indicator. Prompt content, security flags, tool behaviour, context

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 08, August 2026

length, and resource pressure are assessed together. Different attack classes leave different traces: prompt injection is often visible in linguistic patterns, model denial-of-service can surface through unusually long contexts and resource pressure, and unsafe agency is more likely to appear as high-risk or repeated tool use. Combining these signals therefore provides broader coverage than any one source of telemetry on its own.

B. Auditability and Governance Coverage

Complete audit coverage was obtained because SRMP creates a structured event for every processed request. The record contains request identity, runtime telemetry, model behaviour, security classification, risk score, selected action, and an explanation of the decision. This matters in practice because a blocked or escalated request may later need to be reviewed by a security team, internal auditor, governance committee, or regulator. The event record preserves the evidence needed for that review.

C. ML Threat Detection

The lightweight threat classifier used TF-IDF features with Logistic Regression. Under 5-fold cross-validation, it achieved 97.8% accuracy and a 97.8% F1 score. These results show that a conventional text classifier can add useful evidence for recognised prompt-threat patterns without making the runtime pipeline computationally heavy. The classifier should nevertheless be treated as one signal within AARS rather than as a complete defence, since previously unseen or deliberately obfuscated attacks may not resemble the examples on which it was trained.

D. Runtime Overhead

Monitoring introduced approximately 8-12% additional latency in the controlled experiment. That overhead may be significant for consumer-facing systems operating under strict throughput targets, but it can be a reasonable trade-off in enterprise settings where agents handle sensitive information, regulated processes, or privileged tools. A production implementation would still need optimisation through measures such as asynchronous event writing, batching, selective sampling, and accelerator-aware telemetry collection.

VII. DISCUSSION

The experiment suggests that runtime security for agentic LLM applications is difficult to achieve by looking at infrastructure health, model traces, or content safety separately. The improvement in detection came from bringing those signals together. SRMP supplies a consistent record of what occurred, AARS expresses the combined evidence as an interpretable risk profile, and RASS turns that profile into a bounded operational response.

Explainability is central to this design. AARS does not return only a safe/unsafe label. Instead, it retains separate contributions for infrastructure, model behaviour, agent activity, retrieval exposure, security events, and user history. This makes the decision easier to inspect and tune. Developers can see which component is driving a score, security teams can trace the source of an escalation, and auditors can review the basis on which a request was blocked or sent for human approval.

The architecture was also kept deliberately lightweight. The prototype can be integrated with common Python-based AI stacks and dashboard tools and can operate with simulated inputs, imported CSV data, or API-fed runtime events. That makes it useful both as a research test bed and as an internal proof of concept before more demanding production hardening is attempted.

The evidence should nevertheless be interpreted within the limits of the experiment. The evaluation used 300 controlled requests rather than sustained production traffic, and some hardware telemetry may be simulated or imported depending on the experiment mode. The results therefore establish the feasibility of the approach, not production certification. Broader validation requires larger and more diverse datasets, stronger adversarial testing, real GPU inference traces, ablation of individual AARS components, and comparison with specialised guardrail and observability systems.

VIII. THREATS TO VALIDITY AND LIMITATIONS

Table VIII: Current Limitations and Directions for Further Validation

Validity Area	Current Limitation	Further Validation
Dataset	300 controlled requests may not represent all enterprise workloads	Evaluate thousands of prompts across multiple domains and include multilingual threat examples
External	Simulated workloads may differ from production LLM traffic	Evaluate anonymised logs from real LLM APIs or local vLLM/Ollama deployments
Hardware	GPU metrics may be simulated or imported depending on experiment mode	Run the workload on real GPU servers and report utilisation, memory, throughput, and latency
Baseline	Baseline may be weaker than advanced commercial guardrail stacks	Benchmark against open-source guardrails, garak-style scanners, and LLM observability systems

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 08, August 2026

Validity Area	Current Limitation	Further Validation
Attack	Rule patterns and TF-IDF classifier may miss novel attacks	Add adversarial test suites, prompt mutation, and evaluation against previously unseen attacks
Policy	Fixed AARS weights may not fit every organisation	Test configurable weights and report sensitivity and ablation analysis

IX. PRACTICAL AND RESEARCH IMPLICATIONS

In an enterprise setting, AgentGuard AI provides a way to move AI governance from policy statements into runtime evidence and controls. Security teams can use SRMP to extend existing logging with AI-specific events, while AARS and RASS provide explicit mechanisms for assessing and responding to request-level risk. From a research perspective, the scoring and action layers create measurable abstractions that can be compared across workloads and policy settings. The same event trail is also relevant to audit and regulatory review because it records what the system observed, how risk was assessed, and why a particular action was taken.

The framework can also be extended to multi-agent and tool-oriented environments. As agent-to-agent communication, model context protocols, and automated workflow builders become more common, systems will need to account for the risk associated with chained actions, delegated privileges, and untrusted context. SRMP can accommodate these cases by adding protocol-specific event fields, while AARS can be adapted through policy-specific weights and additional risk components where required.

X. CONCLUSION

AgentGuard AI combines request-level monitoring, explainable risk assessment, and runtime control for LLM-agent applications. SRMP records the evidence generated during execution, AARS converts that evidence into a multidimensional risk score, and RASS selects a bounded response from allowing the request through to throttling, human review, or blocking. In the controlled 300-request evaluation, attack detection rose from 28.3% to 85.0%, audit coverage reached 100%, and the TF-IDF/Logistic Regression classifier achieved 97.8% accuracy and a 97.8% F1 score. The observed runtime overhead was 8-12%. These results support the feasibility of integrated runtime governance for agentic LLM systems, while also showing where further validation is needed. The next stage is to test the framework on larger adversarial datasets, production-like traffic, real GPU workloads, multilingual prompts, and organisation-specific policy configurations.

XI. DECLARATIONS

Data Availability: The anonymised experimental logs, threat-label definitions, and prototype source code are not currently deposited in a public repository.

Conflict of Interest: I hereby confirm that there will be conflict of interest for this work in any format.

Ethics Statement: The evaluation uses simulated and controlled AI-agent requests and does not require the collection of real personal, confidential, or institutional data.

Funding: This work is bootstrapped and no external funding or grant involved in the development.

REFERENCES

[1] A. Vaswani et al., "Attention is all you need," in Advances in Neural Information Processing Systems, 2017.

[2] T. B. Brown et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems, vol. 33, 2020.

[3] L. Ouyang et al., "Training language models to follow instructions with human feedback," in Advances in Neural Information Processing Systems, 2022.

[4] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Advances in Neural Information Processing Systems, vol. 33, 2020.

[5] K. Greshake et al., "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in Proc. ACM Workshop on Artificial Intelligence and Security (AISec), 2023.

[6] Y. Liu et al., "Prompt injection attack against LLM-integrated applications," arXiv:2306.05499, 2023.

[7] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," arXiv:2310.12815, 2023.

[8] Y. Ruan et al., "Identifying the risks of LM agents with an LM-emulated sandbox," in Proc. International Conference on Learning Representations, 2024.

[9] OWASP Foundation, "OWASP Top 10 for LLMs and GenAI Applications 2025," OWASP GenAI Security Project, 2025.

[10] W. Kwon et al., "Efficient memory management for large language model serving with PagedAttention," in Proc. ACM Symposium on Operating Systems Principles (SOSP), 2023.

[11] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, 2023.

[12] C. Autio et al., "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile," NIST AI 600-1, 2024.

[13] European Parliament and Council of the European Union, "Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence," Official Journal of the European Union, 2024.

[14] L. Derczynski et al., "garak: A framework for security probing large language models," arXiv:2406.11036, 2024.

[15] N. Carlini et al., "Extracting training data from large language models," in Proc. USENIX Security Symposium, 2021.

[16] A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson, "Universal

and transferable adversarial attacks on aligned language models," arXiv:2307.15043, 2023.

[17] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in Proc. International Conference on Learning Representations, 2023.

[18] N. Shinn et al., "Reflexion: Language agents with verbal reinforcement learning," in Advances in Neural Information Processing Systems, 2023.

[19] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond accuracy: Behavioral testing of NLP models with CheckList," in Proc. ACL, 2020.

[20] G. Pang, C. Shen, L. Cao, and A. van den Hengel, "Deep learning for anomaly detection: A review," ACM Computing Surveys, vol. 54, no. 2, 2021.

[21] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[22] NVIDIA Corporation, "NVIDIA Data Center GPU Manager User Guide," NVIDIA Developer Documentation, 2024.

[23] Cloud Native Computing Foundation, "OpenTelemetry Specification," CNCF OpenTelemetry Project, 2024.

[24] L. Weidinger et al., "Taxonomy of risks posed by language models," in Proc. ACM Conference on Fairness, Accountability, and Transparency, 2022.

[25] R. Bommasani et al., "On the opportunities and risks of foundation models," arXiv:2108.07258, 2021.