JSL, 10(3), March 2025: 01-15

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

JOURNAL OF SCIENTIFIC LETTERS
www.jslsci.com

RUNTIME MONITORING AND SECURITY IN LARGE LANGUAGE MODEL AND AGENTIC AI SYSTEMS: A STRUCTURED REVIEW

Deepak Kumar Gour

Research Scholar, JyotiVidyapeeth,Woman's University, Jaipur, Rajasthan, India

Dr. Sushma Agrawal

Research Supervisor, Jyoti Vidyapeeth Woman's University, Jaipur, Rajasthan, India

ABSTRACT

Large language models are increasingly deployed as agents that retrieve external information, maintain memory, invoke tools, and execute multi-step actions. This shift expands the security boundary beyond harmful text generation to the runtime interaction among prompts, retrieval sources, tools, memory, policies, and infrastructure. This paper presents a structured review of runtime security for LLM and agentic AI systems, with emphasis on inference-time threats that can be observed without access to model internals. The literature is synthesised across prompt injection, jailbreaking, data leakage, retrieval-augmented generation, tool use, agent benchmarks, guard models, application guardrails, runtime monitoring, and adaptive control. A layered defence view is used to distinguish model-level safeguards, semantic guards, application controls, runtime observation, and operational response. Recent benchmarks such as AgentDojo, InjecAgent, and Agent Security Bench show that tool-enabled agents remain vulnerable across prompt, tool, and memory channels, while newer defences increasingly combine isolation, policy enforcement, and adaptive validation. Five gaps remain especially relevant: integrated model-agnostic runtime monitoring, multidimensional risk assessment, proportionate

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

adaptive response, resource-constrained operation, and reproducible runtime evidence. The review argues that runtime monitoring should be treated as one layer of defence in depth, complementing rather than replacing model alignment, least privilege, application security, and human oversight.

Keywords **-** Large language models; agentic AI; runtime security; prompt injection; LLM agents; AI guardrails; risk scoring; adaptive control.

I. INTRODUCTION

Large language models (LLMs) have moved from stand-alone text generators to components of software systems that retrieve information, maintain state, call tools, and act on behalf of users. Transformer architectures established the technical basis for modern LLMs [1], while large-scale pre-training and retrieval-augmented generation broadened their use beyond conventional natural-language processing [2], [3]. Agent-oriented approaches such as ReAct further connected language-model reasoning with external actions [4]. The resulting systems are more useful, but the security problem is no longer confined to whether a model produces an unsafe response.

A tool-enabled agent processes information from several trust domains at once. User instructions may be combined with system prompts, retrieved documents, memory, API responses, and outputs from other tools or agents. If untrusted natural-language data are interpreted as executable instructions, the agent can be redirected from its intended task. Early prompt-injection work demonstrated goal hijacking and system-prompt leakage [5], and indirect prompt injection later showed that malicious instructions can be hidden inside webpages, documents, e-mails, or other content consumed by an application [6]. Once agents are allowed to invoke tools, the consequence can extend from a bad answer to data disclosure, unauthorised actions, or persistent corruption of state.

The research literature has responded with model alignment, safety classifiers, prompt isolation, programmable guardrails, tool policies, and benchmark

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

environments. These approaches are important but operate at different points in the execution path. A model-level safeguard may refuse an unsafe request yet remain vulnerable to indirect context manipulation. A semantic guard may classify a prompt correctly but does not necessarily constrain a privileged tool. An application policy may enforce an allow-list without maintaining a cumulative picture of session risk. This motivates a runtime perspective in which security-relevant events are observed throughout execution and translated into evidence that can support a proportionate response.

This review is aligned with the research theme of secure runtime monitoring and risk-adaptive control for LLM agents in resource-constrained environments. It does not evaluate the AgentGuard AI prototype or reuse its experimental results. Instead, it asks what the current literature establishes about runtime threats, what protection layers are available, and which gaps remain when the desired security layer must be model-agnostic, explainable, lightweight, and reproducible.

II. REVIEW SCOPE AND METHOD

The paper adopts a structured thematic review rather than a formal meta-analysis. The starting corpus is the reference base developed for the associated doctoral research and updated through 2026 with primary publications, benchmark papers, security venues, and authoritative standards. The scope is deliberately restricted to inference-time and runtime risks. Training-time poisoning, model theft, and alignment research are included only where they materially affect deployment-time security.

The literature is organised around four questions: (1) which runtime attack surfaces are introduced by retrieval, tools, memory, and multi-agent interaction; (2) how current systems detect or prevent these threats; (3) how security evidence is converted into operational decisions; and (4) which assumptions current defences make about model access and computational resources.

E-mail: editor.jslsci888@gmail.com

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

Because the surveyed studies use different agents, models, datasets, attack definitions, and metrics, reported numerical results are interpreted only within their original experimental context. Cross-paper values are not treated as direct head-to-head comparisons.

III. RUNTIME THREAT SURFACE AND SECURITY CHALLENGES

A. Prompt Injection and Jailbreaking

Prompt injection exploits the model's inability to enforce a hard architectural boundary between trusted instructions and untrusted natural-language data. Perez and Ribeiro described direct attacks that redirect an application or expose its hidden prompt [5]. Greshake et al. extended the problem to indirect delivery channels, where attacker-controlled text is embedded in retrieved content rather than entered directly by the user [6]. This distinction is critical for agents because an apparently legitimate tool or document can become the carrier of the attack.

Jailbreaking addresses a related but different objective: bypassing safety or policy restrictions. Universal adversarial suffixes showed that carefully constructed strings can transfer across aligned models [7]. For runtime monitoring, the central lesson is that lexical signatures alone are insufficient. Adversarial phrasing, role-play, long-context demonstrations, and gradual multi-turn escalation can preserve malicious intent while substantially changing surface form.

B. Data Leakage, RAG, and External Context

Agentic systems routinely place sensitive operational data in model context. Leakage may involve system prompts, retrieved documents, personal information, credentials, or tool outputs. Training-data extraction demonstrates the broader confidentiality risk associated with large language models [8], but deployed agents create more immediate pathways because they often connect directly to files, databases, e-mail, and enterprise applications.

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

Retrieval-augmented generation adds a second input channel that is frequently less trusted than the user prompt. RAG improves factual grounding [3], yet retrieved text can contain attacker-controlled instructions. The problem is therefore not simply whether a document is relevant; it is whether content retrieved as data can acquire control authority once placed in the model context. Runtime security requires provenance information about what was retrieved, where it came from, and what action followed from it.

C. Tool Use, Excessive Agency, and Memory

Tool use converts model decisions into external effects. ToolEmu demonstrated that LLM agents can exhibit risky behaviour in realistic tool environments even when the evaluation itself is conducted safely [9]. AgentDojo later introduced an extensible environment with realistic tasks and security test cases for prompt injection against tool-using agents [10]. InjecAgent broadened this setting with 1,054 cases spanning 17 user tools and 62 attacker tools, including both direct harm and private-data exfiltration objectives [11].

Agent Security Bench (ASB) extends evaluation across system prompts, user prompts, tools, memory, and mixed attacks [12]. The importance of memory is easily underestimated: malicious or misleading content that enters persistent memory can influence later decisions after the original attack context has disappeared. Tool privileges create a parallel concern. Even a benign request becomes high impact when an agent possesses unnecessary authority or when a compromised instruction is allowed to call sensitive functions.

IV. DEFENCE LAYERS AND RUNTIME MONITORING

A. Model-Level and Semantic Safeguards

Model alignment and refusal training remain the first protection layer, but they are not a complete application-security boundary. The model may still receive malicious context from a trusted application component or operate with

E-mail: editor.jslsci888@gmail.com

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

permissions that exceed the user's task. Semantic guard models address part of this problem by classifying prompts and responses. Llama Guard introduced an LLM-based safeguard organised around an explicit safety taxonomy [14]. More recent compact variants show that semantic safeguards can be reduced enough to operate on commodity CPU-class devices, although they still require model inference [15].

These guards are valuable because they generalise beyond literal string matching. Their limitations are equally important: they add inference cost, require calibration, and remain exposed to distribution shift and adversarial adaptation. For resource-constrained systems, the operational question is therefore not whether semantic models are useful, but whether every request can afford an additional model invocation.

B. Application Guardrails and Structured Controls

Application guardrails move security logic outside the model. NeMo Guardrails provides programmable rails for conversation and application behaviour and is independent of a single underlying model provider [16]. Structured-query approaches go further by separating trusted instructions from untrusted user data. StruQ formalises this separation through a secure front-end and a specially trained model that is instructed to follow only the trusted instruction channel [13].

These approaches restore a principle familiar from conventional security engineering: untrusted data should not automatically acquire control authority. Their main limitation relative to runtime monitoring is scope. A strong input boundary can reduce prompt injection risk, but an agent may still make unsafe decisions because of compromised memory, overly broad privileges, anomalous tool sequences, or contextual risk that emerges only after execution begins.

C. Runtime Monitoring and Adaptive Enforcement

Runtime monitoring treats the interaction lifecycle as a stream of security-

E-mail: editor.jslsci888@gmail.com

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

relevant events rather than a single prompt. Useful evidence includes prompts, retrieved content, tool calls, memory updates, model responses, user context, policy decisions, and infrastructure telemetry. This creates the possibility of correlating weak signals. A suspicious phrase may be low risk by itself; the same phrase followed by a privileged tool request and an unusual destination is materially different.

Recent work increasingly moves toward dynamic enforcement. DRIFT combines planning constraints, dynamic validation, and isolation of injected instructions in the memory stream [17]. Spotlight-Guard uses a layered design that combines isolation of untrusted content, semantic detection, integrity controls, and adaptive evaluation, illustrating a broader shift from single filters toward defence in depth [18]. The common theme is that security must account for what the agent is attempting to do, not only what the current text appears to mean.

Operational response is also becoming more graduated. Binary allow/block logic is easy to specify but can impose unacceptable false-intervention cost. Runtime systems may instead allow low-risk requests, increase logging for uncertain activity, restrict sensitive tools, require human approval, isolate execution, or block only the highest-risk cases. Such policies require an explicit risk representation and a stable mapping from evidence to action.

V. EVALUATION, GOVERNANCE, AND REPRODUCIBILITY

A. Benchmarking Beyond a Single Detection Score

Evaluation practice is now one of the most important issues in agent security. AgentDojo, InjecAgent, and ASB have improved the field by moving beyond isolated prompt examples toward tool-enabled tasks, attack scenarios, and explicit security criteria [10]-[12]. Even so, the benchmarks answer different questions. AgentDojo emphasises security and task utility in realistic workflows; InjecAgent focuses on indirect prompt injection across a broad tool

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

ecosystem; ASB expands coverage to multiple attack stages, including tools and memory. Their metrics are therefore not interchangeable, and numerical values from one benchmark should not be used as evidence that a defence would achieve the same result on another.

A useful runtime evaluation should report both security and operational cost. Precision, recall, F1-score, false-positive rate, and attack success rate describe different aspects of performance, while benign-task success, intervention frequency, latency, throughput, and memory use describe whether the control is usable. Spotlight-Guard is representative of the newer evaluation direction because it reports attack resistance together with benign-task success and computational overhead rather than treating security as the only objective [18]. This is particularly important for agent systems, where excessive blocking can make a technically secure system operationally ineffective.

Adaptive testing is equally important. A detector can appear strong on a fixed corpus because the attack distribution is familiar or because the benchmark contains repeated lexical structures. Runtime security should therefore be tested against paraphrase, obfuscation, indirect delivery, multi-turn escalation, and defence-aware attacks. For model-agnostic monitoring, this also means that labels should be independent of the detector being evaluated and that threshold selection should be separated from final test evaluation. These controls reduce the risk of circular validation and benchmark-specific overfitting.

B. Governance and Accountability

Governance frameworks reinforce the need for observable and reviewable runtime behaviour. The NIST AI Risk Management Framework structures organisational AI risk around governance, mapping, measurement, and management [19], while the Generative AI Profile extends that guidance to risks specific to generative systems [20]. OWASP's current guidance identifies prompt injection, sensitive information disclosure, excessive agency, system-

E-mail: editor.jslsci888@gmail.com

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

prompt leakage, vector and embedding weaknesses, and unbounded consumption among the major risks facing LLM applications [21]. MITRE ATLAS provides a complementary adversary-oriented view of tactics and techniques [22], and ISO/IEC 42001 establishes management-system requirements for AI governance and continuous improvement [23].

These documents do not prescribe one technical runtime architecture. Their relevance lies in the operational expectations they create. An organisation should be able to determine what happened, which component was involved, what risk was identified, what action followed, and whether a human override occurred. For LLM agents, that requires more than traditional application logs. Useful evidence may include the source of retrieved content, the exact tool selected, policy checks applied to its arguments, the risk state before and after the action, and the provenance of any persistent memory update.

The European Union AI Act adds a regulatory context in which logging, risk management, human oversight, and technical documentation become particularly important for certain classes of AI systems [24]. The implication for runtime-security research is not that a monitoring framework automatically creates compliance. Rather, a well-designed evidence layer can make later assurance, investigation, and governance more practical by preserving the information needed to reconstruct consequential decisions.

C. Reproducibility as a Security Property

Reproducibility is often treated as a general research-quality requirement, but in runtime security it is also part of the assurance problem. A security result is difficult to interpret when the evaluated rule set, model version, threshold, dataset revision, or random seed is unknown. The problem becomes more serious when a system adapts its response over time, because identical prompts may legitimately lead to different actions if the surrounding session state has changed.

E-mail: editor.jslsci888@gmail.com

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

A reproducible runtime experiment should therefore preserve the configuration that produced each decision. At minimum, this includes the dataset and split definition, model or rule-library version, threshold configuration, risk-weight settings, execution environment, random seed, and event-level prediction. Where performance claims include latency or resource use, the measurement method and hardware environment should also be reported. These records make it possible to distinguish an algorithmic improvement from an artefact caused by changed data, changed configuration, or different infrastructure.

The same principle applies to operational systems. Audit trails should be append-oriented, attributable, and sufficient to trace a high-impact decision from input evidence through risk assessment to final action. Runtime monitoring is therefore not only a detection problem; it is also an evidence-management problem.

VI. RESEARCH GAPS AND DIRECTIONS

A. Integrated, Model-Agnostic Monitoring

The literature is rich in specialised defences but remains fragmented across layers. Prompt classifiers, safety models, policy engines, tool validators, and secure execution schemes address important parts of the problem, yet relatively few frameworks maintain a unified runtime view that links evidence, assessed risk, response, and audit history. A model-agnostic monitor is particularly valuable where operators cannot inspect model internals or alter provider weights.

B. Multidimensional Risk Scoring

Most security mechanisms eventually reduce their output to a category, confidence score, or policy decision. Agent risk is broader. A suspicious prompt may be low impact when the agent has no privileged tools, while a subtle instruction can be high impact when combined with confidential context or

E-mail: editor.jslsci888@gmail.com

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

sensitive capabilities. A useful runtime score should therefore distinguish security evidence from operational severity and should combine prompt, agent, retrieval, infrastructure, user, and model-related signals without treating a heuristic score as a calibrated probability.

C. Adaptive Response

Detection alone does not determine the correct operational action. High false-positive rates can make a secure system unusable, while overly permissive controls leave the agent exposed. Research is still needed on graduated policies that connect risk bands with observation, throttling, sandboxing, tool restriction, human review, and blocking. Such policies should be tested for stability as well as security, particularly where risk fluctuates around a threshold.

D. Resource-Constrained Security

The growing availability of compact guard models reduces deployment cost, but many state-of-the-art defences still assume additional LLM inference, GPU resources, or external services. This leaves an important design space for lightweight controls based on observable runtime data, deterministic rules, conventional statistical models, and selective use of semantic guards. The objective is not to replace model-based defences, but to determine which protection remains feasible when compute, privacy, or connectivity constraints are strict.

E. Reproducible Runtime Evidence

Agent benchmarks have improved reproducibility, but runtime-system evidence requires more than a benchmark label. Reproducible evaluation should retain dataset versions, model and rule versions, configuration, thresholds, random seeds, event-level predictions, action decisions, timing methodology, and environment information. Without these records, it is difficult to distinguish a genuine security improvement from a benchmark artefact or implementation-specific effect.

E-mail: editor.jslsci888@gmail.com

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

F. Research Direction

The strongest direction suggested by the literature is a hybrid runtime architecture. Deterministic checks remain useful for transparent, high-confidence patterns; semantic guards are better suited to paraphrase and contextual ambiguity; application policies constrain authority; and runtime monitoring links events over time. Future evaluation should include adaptive attacks, independently labelled test data, security-utility trade-offs, measured runtime overhead, cross-domain transfer, and external replication. These requirements follow directly from the weaknesses exposed by recent agent benchmarks and layered defences rather than from a need to maximise one detection metric.

VII. LIMITATIONS OF THE CURRENT EVIDENCE BASE

A. Heterogeneous Threat Models

The current literature should be interpreted with caution because studies often define the protected system differently. Some evaluations examine only a prompt and a model response; others include retrieved documents, tool calls, memory, or an entire agent workflow. A defence that performs well at one layer may therefore address only a subset of the attack surface considered by another study. The same issue affects labels such as prompt injection, jailbreak, indirect injection, and excessive agency, which are sometimes used at different levels of abstraction. For this reason, the review treats published results as evidence about the experimental setting in which they were obtained rather than as universal performance estimates.

B. Benchmark and Model Dependence

Benchmark choice strongly influences reported security performance. AgentDojo, InjecAgent, and ASB differ in task design, attack construction, available tools, model backbones, and evaluation metrics [10]-[12]. A defence may also depend on the behaviour of a particular underlying LLM, especially

E-mail: editor.jslsci888@gmail.com

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

when the defence itself uses an LLM classifier or requires a modified instruction-following model. Consequently, transfer across models, domains, and tool ecosystems remains an open question. Independent replication on multiple backbones and attack distributions is more informative than a single benchmark score.

C. Limited Operational Validation

A second limitation is the distance between controlled security benchmarks and production operation. Benchmarks are necessary because they provide repeatability and clear ground truth, but real deployments contain long-running sessions, changing permissions, noisy telemetry, organisation-specific policies, and unpredictable user behaviour. They also introduce operational questions that are rarely captured by attack success alone: how many benign requests are delayed, how often human review is required, what runtime overhead is added, and whether security controls remain stable under sustained load. The most credible future studies will combine controlled benchmarks with measured runtime overhead, red-team testing, and carefully governed field evaluation.

D. Scope of This Review

This paper intentionally concentrates on inference-time and runtime security. It does not attempt to cover every training-time threat, model-supply-chain vulnerability, watermarking method, or general AI-safety problem. The review also does not claim that the five identified gaps are the only unresolved questions in LLM security. They are the gaps most directly connected to the research theme of model-agnostic runtime monitoring, multidimensional risk assessment, adaptive control, resource-constrained execution, and reproducible evidence. As the field develops, particularly around multi-agent systems and autonomous tool use, the taxonomy and defence layers described here will require continued revision.

E-mail: editor.jslsci888@gmail.com

Online ISSN: 2456-1495
CODEN (USA): JSLOBJ

VIII. CONCLUSION

LLM security has evolved from a model-safety problem into a systems-security problem. Prompt injection, indirect context manipulation, data leakage, unsafe tool use, memory abuse, and excessive agency arise from interactions among components that were not present in conventional chat interfaces. Recent benchmarks confirm that these vulnerabilities persist across realistic agent workflows, while recent defences show that meaningful improvement often requires more than a single classifier.

The literature therefore supports a defence-in-depth interpretation. Model safeguards, semantic guards, application policies, runtime monitoring, and operational controls serve different purposes and should be evaluated as complementary layers. Five gaps remain especially important for the research theme considered here: integrated model-agnostic monitoring, multidimensional risk scoring, adaptive response, resource-constrained operation, and reproducible runtime evidence. Addressing these gaps requires careful separation between observed evidence, inferred risk, and enforced action.

Runtime monitoring should not be presented as a substitute for secure model development, least privilege, software security, or human oversight. Its value lies in observing what happens after deployment and providing an accountable basis for proportionate control when preventive safeguards are insufficient.

REFERENCES

[1] A. Vaswani et al., “Attention Is All You Need,” in Advances in Neural Information Processing Systems, vol. 30, 2017.

[2] T. B. Brown et al., “Language Models are Few-Shot Learners,” in Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901, 2020.

[3] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in Advances in Neural Information Processing Systems, vol. 33, 2020.

[4] S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models," in Proc. Int. Conf. Learn. Representations (ICLR), 2023.

[5] F. Perez and I. Ribeiro, "Ignore Previous Prompt: Attack Techniques for Language Models," arXiv:2211.09527, 2022.

[6] K. Greshake et al., "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," arXiv:2302.12173, 2023.

[7] A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson, "Universal and Transferable Adversarial Attacks on Aligned Language Models," arXiv:2307.15043, 2023.

[8] N. Carlini et al., "Extracting Training Data from Large Language Models," in Proc. 30th USENIX Security Symp., pp. 2633–2650, 2021.

[9] Y. Ruan et al., "Identifying the Risks of LM Agents with an LM-Emulated Sandbox," in Proc. Int. Conf. Learn. Representations (ICLR), 2024.

[10] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," in Advances in Neural Information Processing Systems, vol. 37, 2024.

[11] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, "InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents," in Findings of ACL 2024, pp. 10471–10506, 2024, doi: 10.18653/v1/2024.findings-acl.624.

[12] H. Zhang et al., "Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-Based Agents," in Proc. Int. Conf. Learn. Representations (ICLR), 2025.

[13] S. Chen, J. Piet, C. Sitawarin, and D. Wagner, "StruQ: Defending Against Prompt Injection with Structured Queries," in Proc. 34th USENIX Security Symp., pp. 2383–2400, 2025.

[14] H. Inan et al., "Llama Guard: LLM-Based Input-Output Safeguard for Human-AI Conversations," arXiv:2312.06674, 2023.

[15] I. Fedorov et al., "Llama Guard 3-1B-INT4: Compact and Efficient Safeguard for Human-AI Conversations," arXiv:2411.17713, 2024.

[16] T. Rebedea, R. Dinu, M. Sreedhar, C. Parisien, and J. Cohen, "NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails," arXiv:2310.10501, 2023.

[17] H. Li, X. Liu, H.-C. Chiu, D. Li, N. Zhang, and C. Xiao, "DRIFT: Dynamic Rule-Based Defense with Injection Isolation for Securing LLM Agents," arXiv:2506.12104, 2025.

[18] D. Demirol and M. Aydogan, "Balancing Security and Performance in LLM Agents: Spotlight-Guard, a Layered Defense Against Indirect Prompt Injection," Applied Sciences, vol. 16, no. 15, Art. no. 7662, 2026, doi: 10.3390/app16157662.

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

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

[21] OWASP Foundation, OWASP Top 10 for LLM Applications 2025, OWASP GenAI Security Project, 2025.

[22] MITRE, "Adversarial Threat Landscape for Artificial-Intelligence Systems (ATLAS)," MITRE Corporation, accessed 2026.

[23] ISO/IEC 42001:2023, Information Technology—Artificial Intelligence—Management System, International Organization for Standardization, 2023.

[24] 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.