

# International Journal of **Technology and Systems** (IJTS)

Operationalizing OWASP LLM Top 10 Controls in Enterprise AI  
Platforms



**CARI  
Journals**

# Operationalizing OWASP LLM Top 10 Controls in Enterprise AI Platforms

<sup>1</sup>Sai Kumar Pola, <sup>2</sup>PavanKumar Koneru

<sup>1</sup><https://orcid.org/0009-0003-0014-1559> , <sup>2</sup><https://orcid.org/0009-0002-5738-1492>

Accepted: 15<sup>th</sup> September, 2026, Received in Revised Form 30<sup>th</sup> September, 2026, Published: 15<sup>th</sup> October, 2026

## Abstract

Enterprises have moved large language model applications from pilot projects into production systems handling customer conversations, internal knowledge retrieval, and increasingly autonomous agentic workflows, introducing a risk surface that differs materially from what conventional application security programs address. OWASP has responded with the Top 10 for Large Language Model Applications, most recently revised in its 2025 edition, and a newly published Top 10 for Agentic Applications covering autonomous, tool-using systems. These taxonomies name and prioritize risk but leave engineering teams to translate categories such as prompt injection or excessive agency into controls that fit an existing cloud-native delivery pipeline. This paper proposes a reference architecture and implementation framework mapping each of the ten OWASP LLM Top 10 2025 categories to specific, deployable controls built on commonly used cloud primitives: managed model gateways, service mesh identity, retrieval-augmented generation pipelines, and continuous delivery tooling. The framework is informed by production engineering practice with retrieval-augmented generation and agentic workloads on Kubernetes using a managed foundation model service. Five real, publicly documented incidents, including a corporate data leak, a manipulated dealership chatbot, a legal ruling on chatbot liability, an indirect prompt injection exfiltration attack, and a zero-click vulnerability in a widely used enterprise copilot, are analyzed against the proposed controls to show where the framework would have reduced impact or prevented the incident outright. A four-stage maturity model helps organizations sequence adoption. Open challenges discussed include the immaturity of evaluation tooling for non-deterministic systems and the persistent tension between guardrail thoroughness and production latency.

**Keywords:** *Large Language Models, Prompt Injection, Retrieval-Augmented Generation, Agentic AI, OWASP, AI Security, Guardrails, Cloud-Native Architecture.*

**JEL Codes :** *L86, M15, O33, K24*

## I. INTRODUCTION

The pace at which large language models moved from experimental interfaces to production infrastructure has few precedents in enterprise software history. Between 2023 and 2026, organizations across regulated and unregulated industries alike embedded generative AI into customer support, software engineering, document processing, and increasingly into agentic workflows in which a model plans multi-step tasks and invokes tools, application programming interfaces, and other models with limited human supervision. Cloud providers accelerated this shift by offering managed foundation model services, retrieval-augmented generation building blocks, and orchestration frameworks that lowered the barrier to building a working prototype from months to days. Retrieval-augmented generation (RAG) is an architecture in which a model's response is grounded by content retrieved at query time from an external knowledge store, typically a vector database. The barrier to building a production grade, defensible system did not fall nearly as fast, and the gap between prototype and secure production deployment has become a major source of enterprise AI incidents documented over the past three years.

Traditional application security guidance does not fully address this gap. The OWASP Top 10 Web Application Security Risks and the OWASP API Security Top 10 remain necessary but insufficient references for a system whose core component accepts natural language instructions, cannot be fully specified through a fixed input schema, and may take autonomous action based on content it retrieves from untrusted sources. Recognizing this gap, OWASP published the first Top 10 for Large Language Model Applications in 2023 [3] and substantially revised it for the 2025 edition, published in November 2024 [1], reordering and renaming several categories to reflect two years of accumulated incident data and to place greater emphasis on agentic risk. In December 2025, OWASP published the 2026 edition of its companion Top 10 for Agentic Applications, released December 9, 2025 [2], addressing risks specific to systems in which a model exercises meaningful autonomy over tool invocation, planning, and multi-step execution. Both documents are risk taxonomies. They tell an organization what can go wrong and give each risk a name, but they intentionally stop short of prescribing how a specific engineering team, operating a specific cloud stack, should implement controls that address each risk without breaking delivery velocity or introducing unacceptable latency.

This paper addresses that translation problem directly.

## II. BACKGROUND AND RELATED WORK

### *A. Evolution of the OWASP LLM Top 10*

OWASP released the original Top 10 for Large Language Model Applications, version 1.1, in 2023 [3], in direct response to the wave of enterprise chatbot and copilot deployments that followed the late 2022 introduction of consumer facing conversational AI. That first edition named ten risks: prompt injection, insecure output handling, training data poisoning, model denial of service,

supply chain vulnerabilities, sensitive information disclosure, insecure plugin design, excessive agency, overreliance, and model theft. The 2025 edition, published November 17, 2024 [1], retained the core concept of prompt injection as the top-ranked risk but restructured the remaining categories to better reflect how enterprise deployments actually fail in practice. The 2025 list comprises LLM01:2025 Prompt Injection, LLM02:2025 Sensitive Information Disclosure, LLM03:2025 Supply Chain, LLM04:2025 Data and Model Poisoning, LLM05:2025 Improper Output Handling, LLM06:2025 Excessive Agency, LLM07:2025 System Prompt Leakage, LLM08:2025 Vector and Embedding Weaknesses, LLM09:2025 Misinformation, and LLM10:2025 Unbounded Consumption. Two changes are particularly significant for enterprise platform teams. System prompt leakage was elevated to its own category, reflecting how frequently system prompts are extracted and how often they contain sensitive business logic or credentials. Vector and embedding weaknesses was added as a new category, reflecting the near universal adoption of retrieval-augmented generation and the distinct access control problems that a shared vector store introduces.

Building on the LLM specific list, OWASP published the Top 10 for Agentic Applications on December 9, 2025 [2]. This companion document addresses risks that emerge specifically when a model is granted the ability to plan multi-step tasks, invoke external tools, and take action with reduced human review, a pattern that has become common in coding assistants, customer service automation, and internal operations tooling. The agentic top ten extends concepts already present in the 2025 LLM list, particularly excessive agency and improper output handling, into a fuller treatment of tool permissioning, inter-agent trust, and cascading failure across multi-agent systems. Because the agentic top ten's incident base and community guidance were still consolidating at the time of writing, the present work treats it primarily as forward looking context in Sections VIII and IX rather than as a fully mapped control framework, while still incorporating its core concerns into the reference architecture of Section IV.

### ***B. NIST AI risk management guidance***

The National Institute of Standards and Technology published the Artificial Intelligence Risk Management Framework, NIST AI 100-1, in January 2023 [4], providing a voluntary, sector-agnostic structure for identifying and managing risks across the AI system lifecycle, organized around four core functions: govern, map, measure, and manage. In July 2024, NIST published the Generative Artificial Intelligence Profile, NIST AI 600-1 [5], which extends the core framework with risks specific to generative systems, including confabulation, information integrity, and the propensity of generative models to produce plausible but incorrect output. In 2024, NIST also published Special Publication 800-218A, Secure Software Development Practices for Generative AI and Dual Use Foundation Models, an SSDF community profile [6] that extends the existing Secure Software Development Framework with practices specific to models and the data pipelines that train and fine-tune them. In January 2024, NIST also published AI 100-2, a taxonomy of

adversarial machine learning attacks and mitigations spanning evasion, poisoning, and privacy attack classes that predate but directly inform several of the OWASP LLM Top 10 2025 categories discussed in Section III, including data and model poisoning and vector and embedding weaknesses [25]. Where the OWASP LLM Top 10 is oriented toward application level risks visible to a security engineer reviewing a deployed system, the NIST documents are oriented toward organizational governance and the software supply chain that produces the model itself. The framework proposed in this paper is positioned primarily against the OWASP taxonomy because it is the artifact most directly actionable by an application and platform engineering team, while treating the NIST documents as complementary governance references, particularly for organizations operating under FedRAMP or similar regulatory regimes.

### ***C. MITRE ATLAS***

MITRE maintains the Adversarial Threat Landscape for Artificial Intelligence Systems, known as ATLAS [7], a knowledge base of adversary tactics and techniques against machine learning systems, modeled deliberately on the structure of the MITRE ATT&CK framework used across conventional cybersecurity. ATLAS catalogs techniques observed against production machine learning systems, including techniques specific to large language models such as prompt injection variants and techniques for extracting model behavior through repeated querying. ATLAS is most useful to this paper as a source of adversary behavior patterns that inform the threat model underlying the reference architecture in Section IV, and as a reference an organization's red team can use once a guarded platform has reached the evaluation stage described in Section VII.

### ***D. Academic foundations of prompt injection and data extraction***

Two academic results underlie much of the current guidance on LLM application security. The term itself originates with Willison, who in September 2022 first used "prompt injection" to describe adversarial manipulation of a language model's instructions by content an attacker controls [23]. Greshake et al. subsequently demonstrated that large language models integrated with external content sources [8], such as documents, web pages, or email, can be compromised through indirect prompt injection, in which malicious instructions are embedded in content the model retrieves and processes rather than typed directly by the user. Their work established that any system granting a model access to untrusted external content, which describes essentially every retrieval-augmented generation deployment, inherits a class of injection risk that cannot be addressed by input sanitization alone because the untrusted content arrives through a channel the user never sees. Carlini et al. demonstrated that large language models can be induced, through carefully constructed queries [9], to reproduce verbatim sequences from their training data, including personally identifiable information and proprietary text, establishing that training data extraction is a practical risk rather than a theoretical one, and directly informing the sensitive information disclosure and data poisoning categories discussed in Section III.

### ***E. Guardrail tooling and managed platform controls***

A parallel body of practical tooling has emerged to implement input and output controls at the application layer. These controls are commonly called guardrails: policy enforcement layers, applied before or after model inference, that screen content for prohibited topics, sensitive data, or policy violations. Rebedea et al. introduced NeMo Guardrails [10], a toolkit that allows developers to define programmable rails constraining what topics a conversational system will engage with, what actions it may take, and how it validates its own output before returning it to a user, implemented as a layer that sits between the application and the underlying model. Cloud providers have subsequently productized similar capability directly into managed model services; Amazon Bedrock Guardrails [11], for example, provides configurable content filters, denied topic lists, sensitive information redaction, and grounding checks that can be attached to a managed model endpoint or to an agent built on that endpoint without requiring the application team to operate a separate guardrail service. A grounding check is a validation step that compares a model's generated claims against the retrieved source material actually used to produce the response. This paper's implementation framework in Section V treats both the standalone toolkit pattern and the managed platform pattern as valid implementation choices for the same underlying control, since the appropriate choice depends on whether an organization is running open-weight models on self-managed infrastructure or building primarily on a managed cloud model service.

### ***F. Complementary application and API security standards***

Because most enterprise LLM applications are exposed through a conventional web or API front end, the risks catalogued in the OWASP Top 10 Web Application Security Risks 2021 edition [13] and the OWASP API Security Top 10 2023 edition, released July 3, 2023 [12], remain fully applicable to the layers of the system that sit outside the model itself, including authentication, session management, and the request and response handling of the orchestration service that calls the model. This paper treats those standards as a baseline that the reference architecture in Section IV assumes is already in place, and focuses its own contribution on the risks that are specific to the presence of a language model in the request path.

### ***G. Positioning of this work***

The related work reviewed above establishes what can go wrong and, in the case of the NIST and guardrail tooling literature, offers partial guidance on how to respond. What remains comparatively underdeveloped in the public literature is a single, end to end mapping from each OWASP LLM Top 10 2025 category to a specific set of controls expressed in terms of the cloud-native building blocks a platform engineering team already operates, together with a reference architecture that shows how those controls compose into one coherent system rather than a list of disconnected point solutions. That is the gap this paper addresses.

**TABLE I. OWASP TOP 10 FOR LLM APPLICATIONS 2025, MAPPED TO PRIMARY CONTROL DOMAIN**

| Category                                | Risk Summary                                                                                  | Primary Control Domain                                       | Ref.      |
|-----------------------------------------|-----------------------------------------------------------------------------------------------|--------------------------------------------------------------|-----------|
| LLM01: Prompt Injection                 | Adversarial input, direct or indirect, causes the model to deviate from intended instructions | Input/output guardrails; content labeling (IV-A, IV-B, V-A)  | [1], [8]  |
| LLM02: Sensitive Information Disclosure | Model discloses information the requester was not authorized to see                           | Retrieval scoping; output redaction (V-B)                    | [1], [14] |
| LLM03: Supply Chain                     | Compromised model weights, datasets, plugins, or dependencies                                 | Signed model registry; CD review gate (V-C)                  | [1]       |
| LLM04: Data and Model Poisoning         | Adversary influences training or grounding data to bias or trigger the model                  | Provenance tracking; integrity sampling (V-D)                | [1]       |
| LLM05: Improper Output Handling         | Unvalidated model output is passed to a downstream system                                     | Output validation and encoding discipline (V-E)              | [1]       |
| LLM06: Excessive Agency                 | Agent permissions exceed what a task genuinely requires                                       | Least-privilege tool scoping; confirmation gates (IV-D, V-F) | [1]       |
| LLM07: System Prompt Leakage            | Confidential system prompt content is extracted by a user                                     | Prompt hygiene; policy externalization (V-G)                 | [1]       |
| LLM08: Vector and Embedding Weaknesses  | Insufficient partitioning or adversarial content in the retrieval index                       | Tenant/tier partitioning; encryption at rest (V-H)           | [1], [17] |
| LLM09: Misinformation                   | Model produces plausible but factually incorrect output                                       | Grounding checks; escalation paths (V-I)                     | [1], [16] |
| LLM10: Unbounded Consumption            | Resource exhaustion or uncontrolled cost accumulation                                         | Rate limiting; per-session token metering (V-J)              | [1]       |

### III. RISK LANDSCAPE FOR ENTERPRISE LLM AND AGENTIC AI PLATFORMS

This section summarizes each OWASP LLM Top 10 2025 risk category [1] as it manifests in an enterprise deployment context, establishing the vocabulary used throughout the remainder of the paper. Table I above summarizes the ten categories alongside their primary control domain, each of which is elaborated in Section V.

#### A. LLM01:2025 Prompt Injection

Prompt injection occurs when an attacker supplies input, either directly through a user-facing interface or indirectly through content the model retrieves from a document, web page, email, or tool result, that causes the model to deviate from its intended instructions. Direct injection is analogous to a user typing an adversarial instruction into a chat box. Indirect injection is more consequential in enterprise settings because it can be delivered through content the model was never designed to treat as untrusted, such as a support ticket, a shared calendar invitation, or a retrieved document in a knowledge base, and because the user who ultimately triggers the exploit may have no awareness that anything malicious occurred. An early, widely reported illustration of the underlying dynamic occurred in February 2023, when a user was able to prompt inject Microsoft's Bing Chat into disclosing its internal codename and confidential operating instructions,

a disclosure that predates the OWASP LLM Top 10 itself but foreshadowed the exact pattern the framework was later written to address [20].

### ***B. LLM02:2025 Sensitive Information Disclosure***

A model with access to broad context, whether through its training data, a retrieval-augmented generation index, or an agent's working memory across a session, can disclose information the requesting party was not authorized to see. This risk is distinct from a conventional access control failure because the disclosure often occurs through the model's generated text rather than through a direct database query, making it harder to detect with conventional data loss prevention tooling that inspects structured queries rather than natural language output.

### ***C. LLM03:2025 Supply Chain***

Enterprise AI platforms depend on a supply chain that includes pretrained model weights, fine-tuning datasets, third party plugins and tools, and the open source libraries used to build the orchestration layer. Any of these can be compromised, tampered with, or simply outdated in ways that introduce vulnerabilities, mirroring conventional software supply chain risk but extended to model artifacts and datasets that lack the mature provenance tooling available for source code.

### ***D. LLM04:2025 Data and Model Poisoning***

An adversary who can influence the data used to train, fine-tune, or ground a model, including a retrieval-augmented generation index that is updated from an external or partially untrusted source, can bias the model's behavior or embed a targeted trigger that produces malicious output under specific conditions while behaving normally otherwise. Because enterprise retrieval indexes are frequently updated continuously from internal wikis, ticketing systems, or shared drives, they represent an ongoing poisoning surface rather than a one time training event.

### ***E. LLM05:2025 Improper Output Handling***

Output from a language model is untrusted content from the perspective of any downstream system that consumes it, in the same way that user input is untrusted from the perspective of a web application. When model output is passed to a shell, a database query, a rendering engine, or another automated system without validation, the model becomes a vector for injection attacks against those downstream systems, effectively inheriting the entire category of classic injection vulnerabilities but with the model itself as the attack surface an adversary manipulates first.

### ***F. LLM06:2025 Excessive Agency***

Agentic systems are granted permissions, tools, and autonomy to accomplish tasks with limited human review. Excessive agency arises when the scope of those permissions exceeds what any single task genuinely requires, so that a manipulated or simply mistaken model action can cause damage disproportionate to the task at hand, such as an agent with unrestricted file system write

access being asked to summarize a document and instead being manipulated into modifying unrelated files.

### ***G. LLM07:2025 System Prompt Leakage***

System prompts, the operator-authored instructions that configure a model's behavior for an application, distinct from end-user input, frequently encode business logic, tool definitions, internal policy language, and occasionally credentials or endpoint details that were never intended to be user visible. Because system prompts share the same context window and token stream as user visible output, a sufficiently motivated user can often extract them through adversarial questioning, exposing both intellectual property and, in poorly designed systems, exploitable configuration detail.

### ***H. LLM08:2025 Vector and Embedding Weaknesses***

Retrieval-augmented generation systems depend on a vector database, a data store optimized for similarity search over embeddings, which are numerical vector representations of text positioned so that semantically similar content is located nearby in vector space. These stores are frequently multi-tenant and frequently populated from multiple source systems with different sensitivity levels. Weaknesses in this category include insufficient access partitioning between tenants or document sensitivity tiers, embedding inversion attacks that recover source text from stored vectors, and injection of adversarial content directly into the index in a manner similar to LLM04 but focused specifically on the retrieval layer rather than the base model.

### ***I. LLM09:2025 Misinformation***

Language models generate plausible sounding but factually incorrect output, commonly termed confabulation, as an inherent consequence of their generative architecture rather than as a bug that can be fully eliminated. In an enterprise context, misinformation risk is not merely reputational; as discussed in Section VI, it has already produced binding legal liability when a chatbot's incorrect statement was treated by a tribunal as an enforceable representation of the company that deployed it.

### ***J. LLM10:2025 Unbounded Consumption***

Unbounded consumption covers both denial-of-service style resource exhaustion and, distinctly, uncontrolled cost accumulation, since inference against a commercial model API is billed per token and an unbounded or poorly rate limited request pattern, whether malicious or simply the result of a runaway agentic loop, can produce a financial impact with no analog in conventional infrastructure denial of service.

#### IV. REFERENCE ARCHITECTURE FOR A GUARDED ENTERPRISE AI PLATFORM

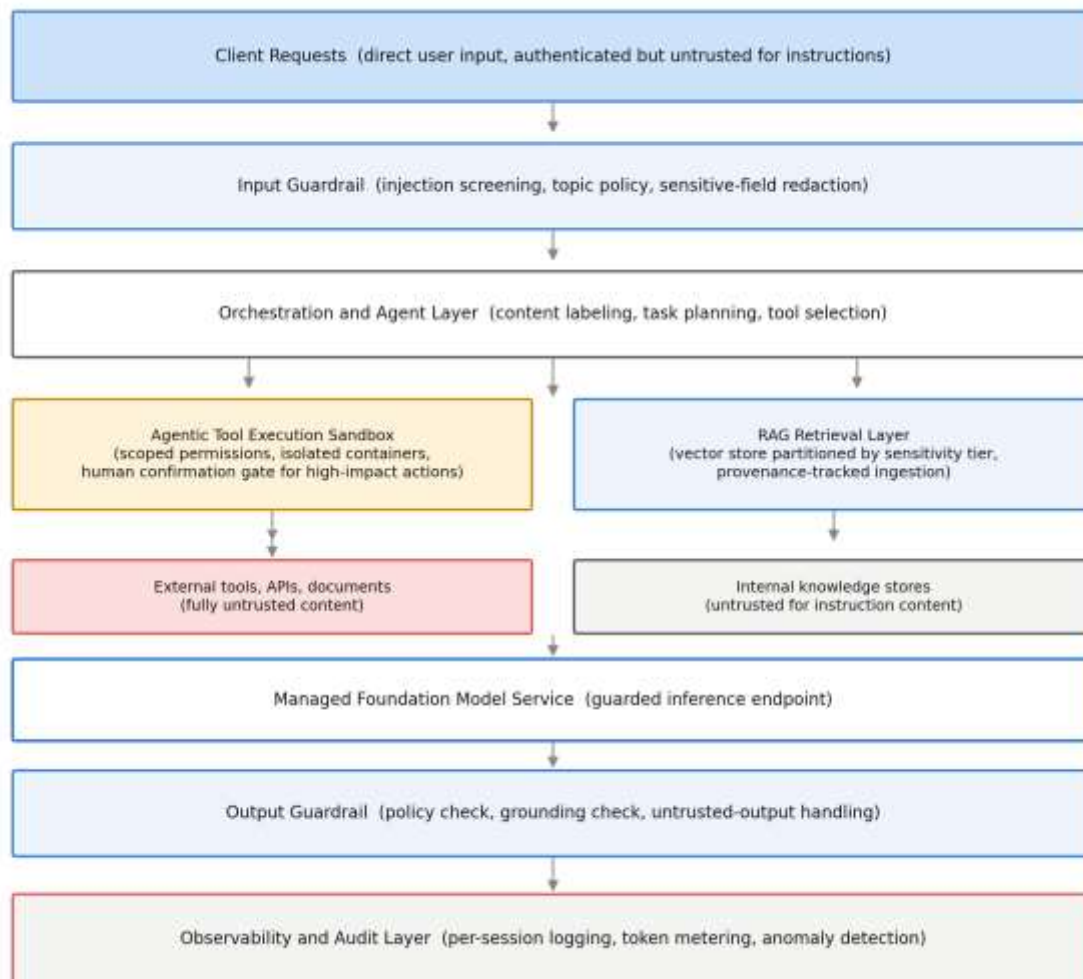


**Figure 1. Reference architecture for a guarded enterprise AI platform.**

The reference architecture shown in Figure 1 organizes an enterprise AI platform into six layers, each corresponding to a distinct trust boundary, so that no single component is required to simultaneously enforce every control in Section V. This separation mirrors the zero trust principle of never granting a component more implicit trust than its function requires, applied here to the specific data flows that a retrieval-augmented generation and agentic platform introduces.

##### ***A. Trust boundary segmentation***

The architecture treats four categories of content as carrying distinct trust levels: direct user input, retrieved content from the organization's own knowledge stores, retrieved or fetched content from external sources, and tool or function call results returned to the model during an agentic task. Direct user input is authenticated and attributable to an identity but is still treated as untrusted with

respect to the instructions it may contain. Retrieved internal content is treated as untrusted with respect to instruction content even though it originates inside the organization, because the indirect prompt injection literature discussed in Section II-D demonstrates that internal documents are a viable injection vector once anyone with edit access to a wiki page or ticket can influence what the model later reads. External content and tool results are treated as fully untrusted. Every one of these content streams passes through a labeling step before it reaches the model context, so that downstream components, including the model itself where supported, can apply differentiated handling to instructions that arrive labeled as untrusted content rather than as the operator's own instructions. This labeling scheme presumes the underlying model can reliably prioritize operator instructions over content merely labeled as untrusted data, a capability that training approaches such as OpenAI's instruction hierarchy demonstrate is achievable but not yet uniformly reliable across model providers, which is why the framework treats labeling as a necessary but not sufficient control [24].

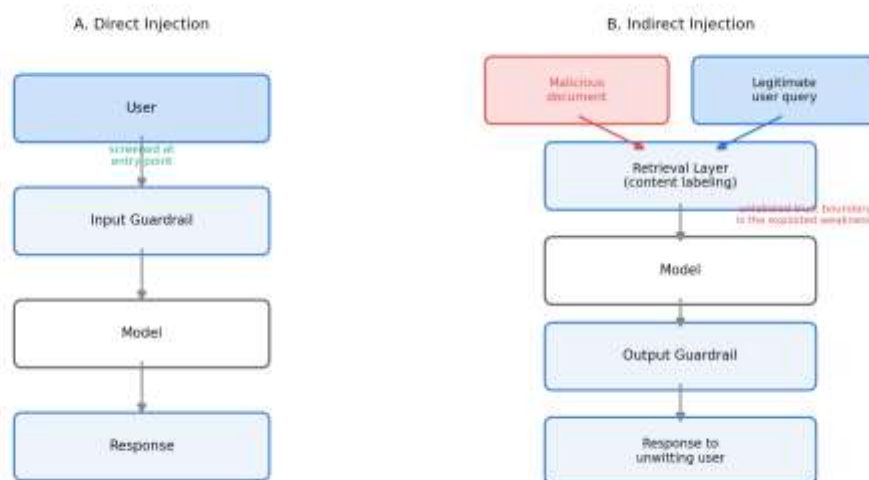


**Figure 2. Direct and indirect prompt injection trust paths.**

Figure 2 contrasts the direct and indirect paths a prompt injection attempt can take. In the direct path, an authenticated user types an adversarial instruction that the input guardrail can screen directly. In the indirect path, the adversarial instruction is embedded in a document, email, or tool result that the model retrieves on behalf of a different, unwitting user, which is why the content labeling step must apply uniformly to every content source rather than only to the initial user turn.

### ***B. Input and output guardrail layer***

A dedicated guardrail service sits immediately in front of and immediately behind the model invocation, implemented either as a standalone policy enforcement point, as described in Section II-E, or through the managed guardrail capability of the underlying model platform. The input guardrail screens for known prompt injection patterns, enforces topic and content policy, and

applies redaction to sensitive fields such as government identifiers or payment data before they ever reach the model. The output guardrail screens the model's response for policy violations, checks factual claims against retrieved source material through a grounding check, and, critically, treats the model's output as untrusted before it is passed to any downstream renderer, database call, or tool invocation, directly addressing LLM05.

### ***C. RAG and vector store isolation***

The retrieval layer partitions the vector store by data sensitivity tier and by source system, with retrieval queries scoped at query time to the tiers the requesting identity is authorized to access, rather than relying on the model to decline to use information it should not have seen. Document ingestion into the index passes through the same content labeling step described in Section IV-A, and a periodic integrity check compares a sample of indexed content against its source system to detect unauthorized modification consistent with the poisoning risk described in LLM04 and LLM08.

### ***D. Agentic tool use sandboxing***

Where the platform supports agentic workflows, every tool the model can invoke is registered with an explicit permission scope, and tool execution occurs in an isolated execution context, typically a short-lived, network-restricted container, rather than directly on infrastructure the agent's identity could otherwise reach. This directly implements the least-privilege response to LLM06 described in Section V-F, and ensures that even a fully successful prompt injection against the model cannot exceed the blast radius, meaning the scope of impact a compromise can cause, of the specific tool permissions granted for that task. Recent design pattern work extends this principle further by architecturally preventing untrusted data from influencing an agent's control flow at all, rather than relying solely on sandboxed execution boundaries after the fact, an approach directly complementary to the sandboxing pattern described here [27].

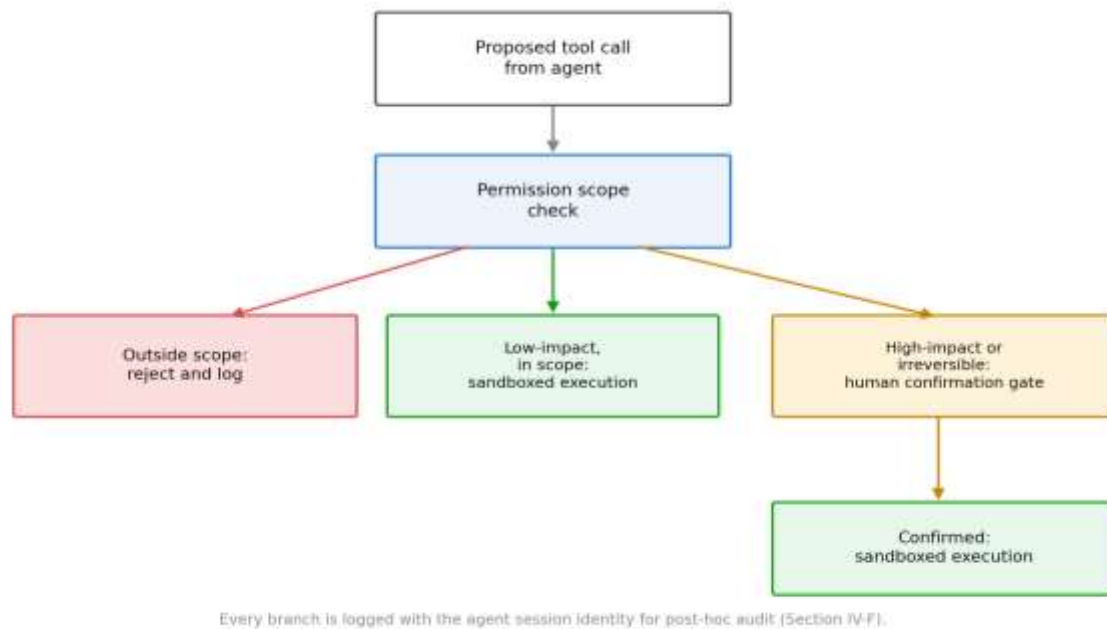


**Figure 3. Agentic tool-use permission and confirmation gate.**

Figure 3 depicts the permission and confirmation gate every tool invocation passes through before execution. A proposed tool call is first checked against the calling agent's scoped permission set; calls within scope for a low impact action proceed directly to the sandboxed execution context, while calls involving a high impact or irreversible action are routed to a human confirmation gate before execution, and calls outside the agent's scope are rejected and logged regardless of the model's stated justification.

#### ***E. Identity and least privilege for agent identity***

Each agent, and each tool it invokes, authenticates through the platform's existing workload identity system, which assigns a cryptographically verifiable identity to each running service or agent session, rather than through a shared service credential, so that every tool invocation is attributable to a specific agent session and constrained by that session's scoped permissions. In a Kubernetes based deployment this is naturally implemented through service mesh backed mutual transport layer security [21] and short-lived workload identity tokens, extending the same zero trust workload identity pattern already used for conventional microservice to microservice traffic to cover model initiated tool calls.

#### ***F. Observability and audit layer***

Every prompt, retrieved context, tool invocation, and model response is logged with sufficient structure to reconstruct the full reasoning chain of an agentic task after the fact, and token consumption is metered per session and per-identity to support the rate and cost controls described

in Section V-J. This layer is what makes the maturity evaluation in Section VII and post-incident analysis, such as the case studies in Section VI, possible at all.

## **V. IMPLEMENTATION FRAMEWORK**

This section maps each OWASP LLM Top 10 2025 category to the specific controls a platform team can implement, organized to correspond directly to the architecture layers introduced in Section IV.

### ***A. Defending against prompt injection (LLM01)***

Because prompt injection cannot be fully eliminated through input filtering alone, the framework treats defense in depth as the baseline expectation rather than a single control. Concretely, this means applying pattern-based and classifier-based screening at the input guardrail described in Section IV-B, explicitly labeling all retrieved and tool sourced content as untrusted context distinct from operator instructions using structural delimiters the model has been instructed to respect, constraining what actions can follow from instructions found within labeled untrusted content through the tool permission scoping described in Section V-F, and running periodic adversarial testing that specifically attempts indirect injection through documents the platform's own retrieval pipeline would plausibly surface. No single one of these controls is sufficient on its own; their value is cumulative.

### ***B. Preventing sensitive information disclosure (LLM02)***

Sensitive information disclosure is addressed primarily upstream of the model, through the retrieval scoping described in Section IV-C, which ensures the model is never given access to content the requesting identity is not authorized to see, on the principle that a model cannot disclose what it was never shown. This is complemented by output guardrail redaction for sensitive data patterns as a second layer of defense against disclosure of information that was appropriately in context but should not appear verbatim in a response, and by explicit organizational policy, reinforced through training, that consumer facing general purpose AI tools are not an approved destination for proprietary or customer data, directly addressing the failure mode analyzed in Section VI-A.

### ***C. Managing supply chain risk for models and plugins (LLM03)***

Model weights, fine-tuning datasets, and third party agent tools are brought under the same software supply chain discipline already applied to application dependencies: a signed and version-pinned model registry, a software bill of materials extended to cover model artifacts, and a review gate in the continuous delivery pipeline before a new model version, plugin, or tool integration is promoted to production. Because this framework assumes a GitOps-style delivery pattern, this review gate is naturally implemented as a required approval step on the pull request that updates the model or tool version reference, giving the platform team the same auditable change history for model updates that it already maintains for application code.

***D. Data and model poisoning defenses (LLM04)***

The primary defense is provenance tracking on every document that enters the retrieval index, so that the source, ingestion time, and last modification of any indexed content can be traced, combined with the periodic integrity sampling described in Section IV-C. For organizations that fine-tune models on internal data, the framework recommends the same access controls and change review applied to any other sensitive training pipeline, and evaluation of fine-tuned model behavior against a held out test set before promotion, to catch gross behavioral drift consistent with a poisoning attempt even when the specific mechanism cannot be directly observed.

***E. Improper output handling controls (LLM05)***

Every downstream consumer of model output, whether a rendering layer, a database query builder, or a tool invocation, treats that output through the same validation and encoding discipline already applied to any other untrusted input in the application, including output encoding before rendering to prevent script injection in a chat interface, parameterization rather than string concatenation when model output contributes to a database query, and schema validation before model output is used to populate a structured tool call argument.

***F. Bounding agent autonomy and excessive agency (LLM06)***

Every tool available to an agent is scoped to the minimum permission required for its function, following the sandboxing pattern described in Section IV-D, and any tool capable of an irreversible or high impact action, such as sending an external communication, modifying a production record, or executing a financial transaction, requires an explicit human confirmation step regardless of the model's confidence in its own plan. Session-level and task-level ceilings on the number of tool invocations an agent may make without checkpoint review provide a backstop against runaway agentic loops, which also serves the unbounded consumption control described in Section V-J.

***G. System prompt leakage prevention (LLM07)***

System prompts are treated as sensitive configuration rather than as inert scaffolding: they are not embedded with credentials or internal endpoint details that would be damaging if disclosed, and any business logic that must remain confidential is moved out of the prompt entirely and into a separate policy enforcement step that evaluates the model's proposed action rather than relying on the model to keep the logic itself secret. This is paired with output guardrail screening for prompt extraction patterns, though the framework treats prompt secrecy as a defense in depth measure rather than a security boundary, since a sufficiently persistent adversary should always be assumed capable of eventually extracting prompt content.

***H. Vector and embedding weaknesses (LLM08)***

The retrieval layer isolation described in Section IV-C is extended with tenant and sensitivity tier partitioning enforced at the vector database access control layer rather than only in application

logic, so that a compromised or misconfigured application component cannot retrieve across partition boundaries. Where the underlying vector database supports it, encryption of stored embeddings at rest mitigates offline inversion following storage exfiltration, but it does nothing against embedding inversion by a party with legitimate query access to the vectors, which is the threat model inversion attacks actually assume; access partitioning, avoiding return of raw vectors through application programming interfaces, and dimensionality or noise defenses address that threat directly. The ingestion pipeline also validates document content against policy before indexing to prevent direct injection of adversarial content into the index.

### ***I. Misinformation mitigation (LLM09)***

The output guardrail's grounding check, introduced in Section IV-B, compares generated claims against the retrieved source material actually used to produce a response and flags or suppresses claims that are not traceable to a retrieved source, which is the most reliable available mitigation for confabulation in a retrieval-augmented generation context. For customer-facing deployments where a model's statement could plausibly be interpreted as a binding representation of the organization, as in the case analyzed in Section VI-C, the framework recommends explicit interface level disclosure that responses are AI generated and subject to verification, combined with a defined escalation path to a human representative for any interaction involving policy, pricing, or contractual terms.

### ***J. Unbounded consumption controls (LLM10)***

Rate limiting and per-identity token budgets are enforced at the same API gateway layer already used for conventional API rate limiting, extended to account for the fact that a single agentic task can generate a highly variable number of underlying model calls depending on how many planning and tool use iterations it requires. The observability layer described in Section IV-F meters consumption per session in near-real-time so that a runaway loop, whether caused by a bug in the agent's planning logic or by a deliberate denial-of-wallet attack, a resource exhaustion attack against a metered, pay-per-use service whose primary impact is uncontrolled financial cost rather than downtime, is detected and throttled before it produces a significant cost or availability impact, closing the loop with the agentic ceiling described in Section V-F.

### ***K. Illustrative deployment walkthrough***

Enterprise Kubernetes adoption is now the norm rather than the exception for cloud-native platforms, with most organizations already operating multiple clusters across regions [22], and this section grounds the framework in that common baseline. To ground the framework in a concrete pattern, consider an enterprise platform hosting a retrieval-augmented generation service on a managed Kubernetes cluster, using a managed cloud foundation model service as the inference backend, with the platform's continuous delivery pipeline promoting infrastructure, guardrail policy, and model version changes through the same GitOps controlled release process used for

the rest of the platform's microservices. The service mesh already in place for east-west traffic between application services is extended to cover traffic between the orchestration layer and any self-hosted guardrail or tool execution components, so that the mutual transport layer security and workload identity guarantees the platform already relies on for FedRAMP or similar regulatory compliance extend naturally to the new AI specific components rather than requiring a parallel security model. Guardrail policy, denied topic lists, and grounding thresholds are expressed as versioned configuration promoted through the same pull request and approval workflow as any other infrastructure change, giving the security and compliance function the same change visibility into AI behavior that it already has into application behavior. This pattern, in which the AI platform is treated as an extension of the existing cloud-native delivery and identity model rather than as a separate system requiring its own bespoke security architecture, is the practical core of the recommendation in this paper.

## VI. CASE STUDIES

This section analyzes five real, publicly documented incidents against the controls proposed in Section V, selected to span the risk categories most consequential in enterprise deployments.

### *A. Samsung ChatGPT source code disclosure, May 2023*

In April and May 2023, multiple Samsung Electronics employees reportedly used ChatGPT for work tasks, including pasting proprietary source code to request debugging help and uploading meeting notes for summarization, without an approved enterprise agreement governing how that data would be retained or used by the underlying service. Once reports of the practice became public, Samsung restricted employee use of external generative AI tools [14]. This incident is a direct illustration of LLM02, sensitive information disclosure, and the framework's response is organizational rather than purely technical: the retrieval and output controls described in Section V-B assume the organization is operating its own guarded platform, which is precisely the control that was absent here, since employees were using an external consumer service with no enterprise data handling agreement at all. The case underscores that the most important LLM02 control for many organizations is providing an approved, guarded internal alternative that is good enough that employees are not tempted to route sensitive work through an ungoverned external tool.

### *B. Chevrolet dealership chatbot prompt injection, December 2023*

A customer interacting with a General Motors dealership's website chatbot, built on a general purpose conversational model, was able to manipulate the chatbot through adversarial prompting into agreeing, in the chat transcript, to sell a vehicle for one dollar and stating that this was a legally binding offer, a screenshot of which circulated widely once shared publicly [15], [28]. The dealership had deployed the chatbot with what appeared to be minimal topic constraint or output review before the transcript could be presented as customer-facing content. This incident maps to LLM01, prompt injection, and LLM06, excessive agency, in combination: the chatbot lacked the

input guardrail topic constraints described in Section V-A, and it was permitted to generate output that a reasonable customer could construe as an authoritative commercial commitment, a failure that the interface level disclosure and escalation path recommended in Section V-I would have directly mitigated.

#### ***C. Moffatt v. Air Canada, February 2024***

In *Moffatt v. Air Canada*, decided by the British Columbia Civil Resolution Tribunal on February 14, 2024, a customer relied on a statement made by Air Canada's website chatbot regarding the airline's bereavement fare refund policy, which turned out to be incorrect and less favorable than the airline's actual published policy. Air Canada argued, unsuccessfully, that it should not be held responsible for information provided by the chatbot as distinct from its official policy pages. The tribunal ruled against Air Canada, holding that the airline was responsible for all information on its website, including information provided by its chatbot, and awarded damages to the customer [16]. This is the clearest documented instance of LLM09, misinformation, translating directly into binding legal liability, and it validates the framework's recommendation in Section V-I that grounding checks and clear escalation paths are not merely a quality feature but a control with direct legal and financial consequence for any deployment that could be construed as making representations on the organization's behalf.

#### ***D. Slack AI indirect prompt injection, August 2024***

In August 2024, the security research firm PromptArmor published an analysis demonstrating that Slack's AI powered search and summarization feature could be manipulated through indirect prompt injection [17]. The attacker needed only the ability to post a message into any channel the target workspace's Slack AI could index, including a public channel the attacker had joined without being a member of any private channel containing the actual sensitive data. The attacker crafted a message embedding instructions that Slack AI, when it later processed an unrelated query from a legitimate user, treated as part of its task rather than as content to summarize. As a result, the assistant exfiltrated sensitive information the attacker was never authorized to see. This incident is a textbook demonstration of LLM01 combined with LLM08, since the underlying mechanism depended on both an unlabeled trust boundary between user instructions and indexed content, exactly the failure the content labeling control in Section IV-A is designed to prevent, and on a retrieval architecture that allowed content from an attacker's limited access scope to influence a response ultimately shown to a victim with broader access, which the tenant and sensitivity partitioning described in Section V-H is designed to prevent by ensuring retrieval scope is bound to the requesting user's own authorization rather than to a workspace-wide index.

#### ***E. Microsoft 365 Copilot EchoLeak, disclosed June 2025***

Security researchers at Aim Security disclosed a vulnerability in Microsoft 365 Copilot, subsequently assigned CVE-2025-32711 and publicly referred to as EchoLeak [18], in which an

attacker could exfiltrate sensitive data from a target's Microsoft 365 environment through a single, unsolicited email containing markdown formatted instructions, requiring no interaction from the victim beyond Copilot's normal, automatic processing of incoming content during a later, entirely unrelated query. The attack chained an indirect prompt injection delivered through email content that Copilot's retrieval layer treated as trusted context, together with an output handling weakness that allowed the model's response to reference and effectively transmit data through an auto rendered markdown image link, exfiltrating content to an attacker controlled endpoint without any explicit user action. An academic analysis published in September 2025 further documented the mechanism as a zero-click attack, meaning the victim did not need to click a link, open an attachment, or take any action beyond having Copilot process their inbox as it normally would [19]. EchoLeak is the most architecturally significant case study in this paper because it demonstrates the compounding effect of exactly the two failures the reference architecture in Section IV is built to prevent in combination: the absence of a trust boundary between retrieved email content and the model's operating instructions, addressed by the content labeling scheme in Section IV-A, and the absence of output handling discipline before the model's response was rendered, addressed by the output guardrail described in Section IV-B and Section V-E, which together allowed a single unsolicited email to produce data exfiltration with zero user interaction.

Table II summarizes the case-study-to-control mapping developed in the preceding subsections, consolidating in one place the OWASP category, the specific control section, and the gap each incident exposed.

**TABLE II. CASE STUDY TO CONTROL MAPPING**

| Case Study                                               | Primary OWASP Category                                          | Controls Applied (Section)                                                   | Gap Identified                                                                               | Ref.       |
|----------------------------------------------------------|-----------------------------------------------------------------|------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|------------|
| Samsung ChatGPT source code disclosure, May 2023         | LLM02: Sensitive Information                                    | Guarded internal alternative; organizational policy (V-B)                    | No enterprise data handling agreement; employees used an ungoverned external tool            | [14]       |
| Chevrolet dealership chatbot prompt injection, Dec. 2023 | LLM01: Prompt Injection; LLM06: Excessive Agency                | Input guardrail topic constraints; disclosure and escalation path (V-A, V-I) | No topic constraints on chatbot output; output construed as a binding commercial commitment  | [15], [28] |
| Moffatt v. Air Canada, Feb. 2024                         | LLM09: Misinformation                                           | Grounding checks; escalation path (V-I)                                      | No grounding check; chatbot's incorrect statement became binding legal liability             | [16]       |
| Slack AI indirect prompt injection, Aug. 2024            | LLM01: Prompt Injection; LLM08: Vector and Embedding Weaknesses | Content labeling (IV-A); tenant and sensitivity partitioning (V-H)           | Unlabeled trust boundary; retrieval scope not bound to requester's authorization             | [17]       |
| Microsoft 365 Copilot EchoLeak, disclosed Jun. 2025      | LLM01: Prompt Injection; LLM05: Improper Output Handling        | Content labeling (IV-A); output guardrail (IV-B, V-E)                        | No trust boundary on retrieved email content; no output handling discipline before rendering | [18], [19] |

## VII. MATURITY MODEL AND EVALUATION

Organizations rarely have the resources to implement every control described in Section V simultaneously. Figure 4 and Table III below present a four stage maturity model across the four pillars introduced in the reference architecture: guardrail coverage, agent governance, data and retrieval security, and observability and red teaming.

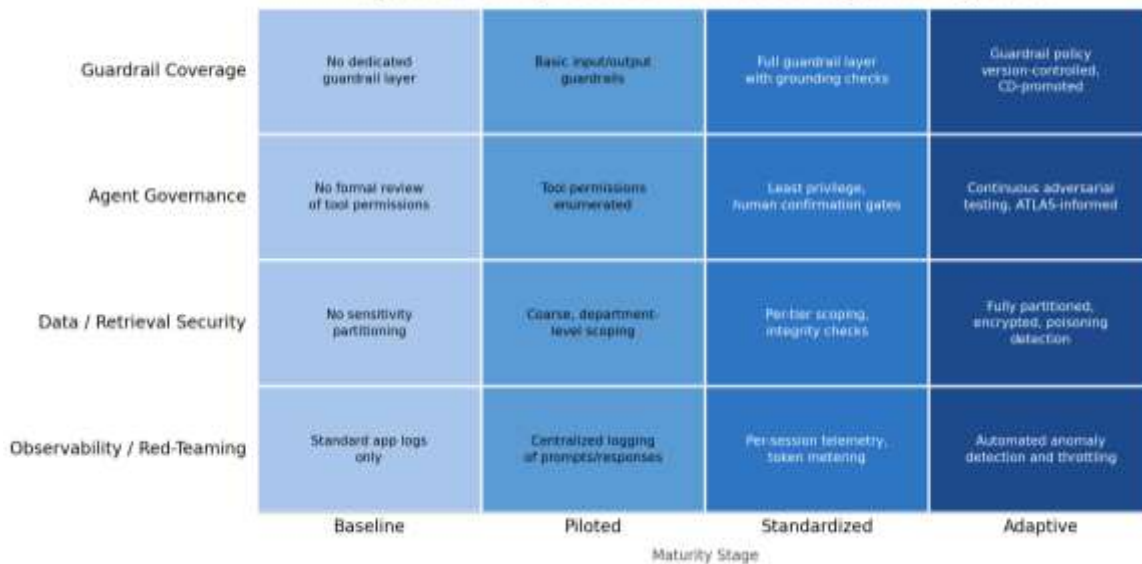


Figure 4. Enterprise AI platform security maturity model, by pillar and stage.

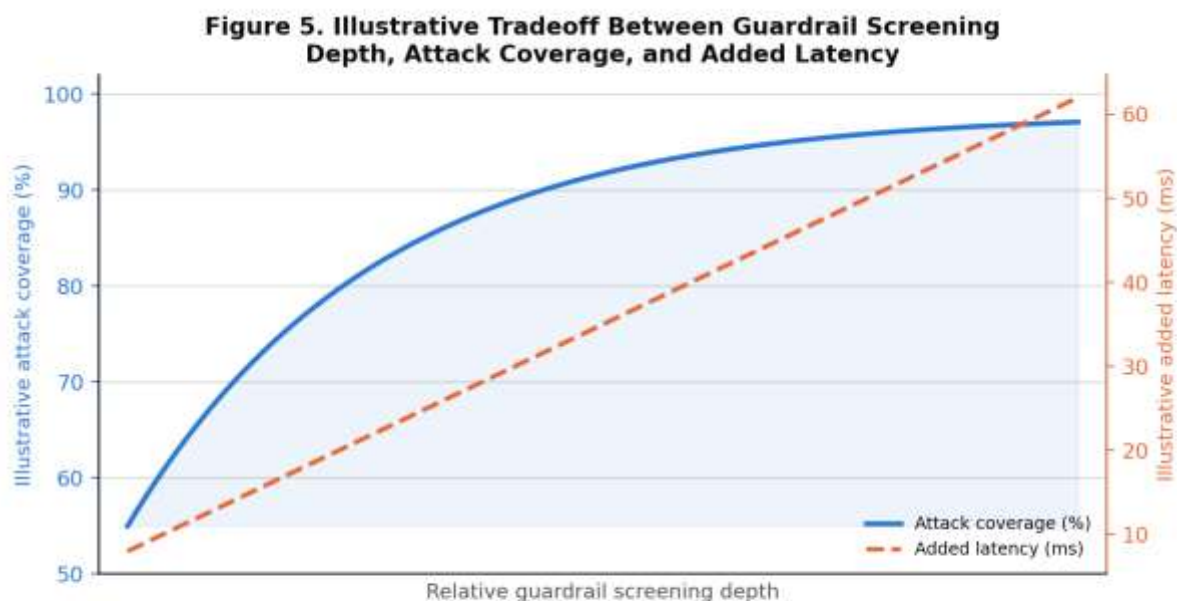
TABLE III. ENTERPRISE AI PLATFORM SECURITY MATURITY MODEL

| Pillar                      | Baseline                             | Piloted                                      | Standardized                                | Adaptive                                          |
|-----------------------------|--------------------------------------|----------------------------------------------|---------------------------------------------|---------------------------------------------------|
| Guardrail coverage          | No dedicated guardrail layer         | Basic input and output guardrails            | Full guardrail layer with grounding checks  | Policy version-controlled, CD-promoted            |
| Agent governance            | No formal review of tool permissions | Tool permissions enumerated                  | Least privilege, human confirmation gates   | Continuous adversarial testing, ATLAS-informed    |
| Data / retrieval security   | No sensitivity partitioning          | Coarse, department-level scoping             | Per-tier scoping, periodic integrity checks | Fully partitioned, encrypted, poisoning detection |
| Observability / red-teaming | Standard application logs only       | Centralized logging of prompts and responses | Per-session telemetry, token metering       | Automated anomaly detection and throttling        |

At the baseline stage, an organization has deployed a working AI application with minimal or no dedicated guardrail layer, no formal review of agent tool permissions, a retrieval index without sensitivity partitioning, and logging limited to standard application logs that were not designed to reconstruct a model's reasoning chain. At the piloted stage, basic input and output guardrails are in place, agent tool permissions have been enumerated even if not yet minimized, retrieval access is scoped at a coarse level such as by department, and centralized logging captures prompts and responses even if not yet structured for full chain reconstruction. At the standardized stage, the full

input and output guardrail layer described in Section IV-B is deployed with grounding checks, agent tools are scoped to least privilege with human confirmation gates on high impact actions, retrieval access is scoped per document sensitivity tier with periodic integrity checks, and per session token metering and rate limiting are enforced. At the adaptive stage, guardrail policy is version controlled and promoted through the same continuous delivery pipeline as application code, agentic workflows are subject to continuous adversarial testing informed by MITRE ATLAS technique coverage, retrieval and vector store access is fully partitioned and encrypted at rest with automated poisoning detection, and observability data feeds an automated anomaly detection layer that can throttle or suspend an agent session without waiting for manual review.

Evaluation of an organization's actual position on this model should combine automated technical assessment, such as the presence and configuration of the specific controls enumerated in Section V, with adversarial testing that specifically attempts the attack patterns demonstrated in the case studies of Section VI, since the presence of a control on paper does not guarantee it is configured correctly or that it covers the specific injection and exfiltration patterns an adversary would actually attempt. Organizations operating in regulated environments should additionally map their control coverage against the NIST AI RMF functions described in Section II-B, since auditors and regulators are more likely to be familiar with that framework's governance language than with the OWASP taxonomy, even though the underlying technical controls largely coincide.



Qualitative illustration, not measured data; see Section VII and Appendix B.

**Figure 5. Illustrative tradeoff between guardrail screening depth, attack coverage, and added latency.**

Figure 5 illustrates the qualitative tradeoff discussed further in Section VIII: as guardrail screening depth increases, added inference latency grows roughly in proportion while incremental attack coverage exhibits diminishing returns, which is why the maturity model in Table III treats guardrail investment as a sequencing decision rather than a single all or nothing deployment choice.

## VIII. CHALLENGES AND LIMITATIONS

Several structural challenges limit how completely any framework, including this one, can guarantee safety in enterprise AI deployments. First, the non-deterministic nature of language model output means that a guardrail or grounding check that reliably catches a given attack pattern in testing may not catch a semantically equivalent variant phrased differently, and the evaluation methodology for measuring guardrail coverage remains far less mature than the equivalent methodology for conventional software testing, where a test case either passes or fails deterministically. Second, there is a genuine tension between guardrail thoroughness and production latency, since every additional screening or grounding step in the input and output guardrail layer adds inference time, and organizations frequently face pressure to relax guardrail depth in pursuit of a more responsive user experience, a tradeoff that this paper does not resolve but flags as a decision that should be made explicitly by risk owners rather than implicitly by engineers under delivery pressure. Third, the tooling ecosystem for implementing the controls described in Section V remains fragmented and rapidly changing; the specific guardrail products, vector database access control features, and agent orchestration frameworks referenced in this paper are likely to evolve significantly even in the near term, and organizations should treat the control objectives in Section V as durable while treating specific product recommendations as provisional. Fourth, the OWASP Top 10 for Agentic Applications, published December 9, 2025, addresses risks in multi-agent and inter-agent trust scenarios that this paper's reference architecture only partially covers, since a full treatment of agent to agent trust boundaries, cascading failure across a chain of cooperating agents, and the specific governance challenges of agents that can spawn or delegate to other agents was still an actively developing area of practice at the time of writing. Fifth, regulatory frameworks applicable to enterprise AI, including sector specific requirements such as FedRAMP for government cloud deployments and jurisdiction specific requirements such as the European Union's AI Act [26], continue to evolve, and organizations operating across multiple regulatory regimes face genuine complexity in reconciling control requirements that were not designed with each other in mind.

## IX. FUTURE RESEARCH DIRECTIONS

Several directions merit further research building on the framework presented here. First, formal methods for verifying that an agent's action space is bounded by its declared tool permissions, analogous to static analysis techniques used to verify conventional software against a security policy, would strengthen the excessive agency control described in Section V-F beyond its current reliance on configuration review and testing. Second, the field lacks a standardized, publicly

available benchmark for measuring guardrail and grounding check effectiveness across a common set of adversarial prompts, comparable to the role that standardized vulnerability scanning benchmarks play in conventional application security, and developing such a benchmark specifically calibrated to enterprise retrieval-augmented generation and agentic deployment patterns would materially improve the maturity evaluation described in Section VII. Third, as the OWASP Top 10 for Agentic Applications matures and accumulates real incident data comparable to what informed the 2025 revision of the LLM specific list, future work should extend the mapping in Section V to fully cover multi-agent trust boundaries and cascading failure scenarios. Fourth, automated policy synthesis, in which guardrail and tool permission policy is generated or tuned from observed agent behavior and incident data rather than authored entirely by hand, represents a promising direction for reducing the operational burden of maintaining the defined and optimal maturity stages described in Section VII. Fifth, longitudinal empirical study of enterprise AI incidents, extending the five case studies analyzed in Section VI to a larger, systematically collected corpus, would allow future work to validate which controls in Section V carry the greatest marginal risk reduction per unit of implementation effort, information that would materially improve the sequencing guidance this paper's maturity model currently offers only qualitatively.

## X. CONCLUSION

The OWASP Top 10 for Large Language Model Applications and its newer companion for agentic applications provide the enterprise AI community with a shared vocabulary for naming risk, but naming a risk is not the same as controlling it. This paper has presented a reference architecture and an implementation framework that translate each of the ten OWASP LLM Top 10 2025 categories into controls expressed in terms of the cloud-native building blocks a platform engineering team already operates, including identity-aware proxies, service mesh workload identity, retrieval access partitioning, and continuous delivery pipelines extended to cover model and guardrail configuration. Five real, publicly documented incidents, spanning an ungoverned data disclosure, a manipulated customer-facing chatbot, a legal ruling on chatbot liability, an indirect prompt injection data exfiltration attack, and a zero-click vulnerability in a major enterprise copilot, were analyzed against the proposed controls and in each case a specific, identifiable gap in the framework's recommended controls corresponds to the actual root cause of the incident, lending practical credibility to the mapping. The four stage maturity model presented in Section VII offers organizations a realistic sequence for adoption rather than an all or nothing mandate. As agentic architectures continue to mature and the OWASP Top 10 for Agentic Applications accumulates the same depth of real world incident grounding that informed the 2025 revision of the LLM specific list, the framework presented here should be expected to extend rather than be replaced, since its underlying principle, that every risk category ultimately reduces to a trust boundary that must be explicitly identified, labeled, and enforced somewhere in the architecture, does not depend on the specific taxonomy used to name the risk.

The practical next step for enterprise teams is therefore to act on this mapping rather than wait for the taxonomies to settle. Platform and security teams should first establish their current position on the maturity model in Table III, then prioritize the controls that address the gaps exposed in the case studies: explicit labeling of untrusted content at every trust boundary, least-privilege tool permissions with human confirmation for high impact agent actions, retrieval access partitioned by document sensitivity, and per session consumption metering. Guardrail and model configuration should be version controlled and promoted through the same continuous delivery pipeline as application code, and control coverage should be validated through adversarial testing of the attack patterns in Section VI rather than assumed from configuration alone. Risk owners, rather than engineers under delivery pressure, should make the tradeoff between guardrail depth and latency explicit, and the research community should pursue the benchmark and verification work outlined in Section IX so that progress along the maturity model can be measured rather than asserted.

## ACKNOWLEDGMENT

Colleagues across platform engineering and AI/ML infrastructure teams contributed the operational perspective behind this paper's control mappings, particularly through discussions of how guardrail and identity tooling actually behaves once deployed against production retrieval-augmented generation and agentic workloads.

## REFERENCES

- [1] OWASP Foundation, "OWASP Top 10 for LLM Applications 2025," Nov. 17, 2024. [Online]. Available: <https://genai.owasp.org/llm-top-10/> (accessed Aug. 13, 2026)
- [2] OWASP Foundation, "OWASP Top 10 for Agentic Applications 2026," Dec. 9, 2025. [Online]. Available: <https://genai.owasp.org/> (accessed Aug. 13, 2026)
- [3] OWASP Foundation, "OWASP Top 10 for LLM Applications," version 1.1, 2023.
- [4] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, Jan. 2023.
- [5] National Institute of Standards and Technology, "Generative Artificial Intelligence Profile," NIST AI 600-1, Jul. 2024.
- [6] National Institute of Standards and Technology, "Secure Software Development Practices for Generative AI and Dual-Use Foundation Models: An SSDF Community Profile," NIST SP 800-218A, 2024.
- [7] MITRE Corporation, "MITRE ATLAS: Adversarial Threat Landscape for Artificial Intelligence Systems." [Online]. Available: <https://atlas.mitre.org/> (accessed Aug. 13, 2026)

- [8] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," arXiv:2302.12173, 2023.
- [9] N. Carlini et al., "Extracting Training Data from Large Language Models," in Proc. 30th USENIX Security Symposium, 2021.
- [10] 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.
- [11] Amazon Web Services, "Guardrails for Amazon Bedrock." [Online]. Available: <https://aws.amazon.com/bedrock/guardrails/> (accessed Aug. 13, 2026)
- [12] OWASP Foundation, "OWASP API Security Top 10 2023," Jul. 3, 2023. [Online]. Available: <https://owasp.org/API-Security/editions/2023/en/0x00-header/> (accessed Aug. 13, 2026)
- [13] OWASP Foundation, "OWASP Top 10 Web Application Security Risks," 2021.
- [14] S. Ray, "Samsung Bans ChatGPT And Other Chatbots For Employees After Sensitive Code Leak," Forbes, May 2, 2023. [Online]. Available: <https://www.forbes.com/sites/siladityaray/2023/05/02/samsung-bans-chatgpt-and-other-chatbots-for-employees-after-sensitive-code-leak/> (accessed Aug. 13, 2026)
- [15] F. Landymore, "Car Dealership Disturbed When Its AI Is Caught Offering Chevys for \$1 Each," Futurism, Dec. 20, 2023. [Online]. Available: <https://futurism.com/the-byte/car-dealership-ai> (accessed Aug. 13, 2026)
- [16] Moffatt v. Air Canada, 2024 BCCRT 149 (Can.), Feb. 14, 2024. [Online]. Available: <https://www.canlii.org/en/bc/bccrt/doc/2024/2024bccrt149/2024bccrt149.html> (accessed Aug. 13, 2026)
- [17] PromptArmor, "Data Exfiltration from Slack AI via Indirect Prompt Injection," Aug. 2024. [Online]. Available: <https://www.promptarmor.com/resources/data-exfiltration-from-slack-ai-via-indirect-prompt-injection> (accessed Aug. 13, 2026)
- [18] Aim Security, "EchoLeak: The First Zero-Click AI Vulnerability Enabling Data Exfiltration from Microsoft 365 Copilot," Jun. 11, 2025.
- [19] P. Reddy and A. S. Gujral, "EchoLeak: The First Real-World Zero-Click Prompt Injection Exploit in a Production LLM System," arXiv:2509.10540, Sep. 2025.
- [20] B. Edwards, "AI-powered Bing Chat spills its secrets via prompt injection attack," Ars Technica, Feb. 10, 2023.
- [21] Istio Authors, "Istio Security: Mutual TLS Authentication." [Online]. Available: <https://istio.io/latest/docs/concepts/security/> (accessed Aug. 13, 2026)
- [22] Cloud Native Computing Foundation, "CNCF Annual Survey 2024," 2025.

- [23] S. Willison, "Prompt Injection Attacks Against GPT-3," Sep. 12, 2022. [Online]. Available: <https://simonwillison.net/2022/Sep/12/prompt-injection/> (accessed Aug. 13, 2026)
- [24] E. Wallace, K. Xiao, R. Leike, L. Weng, J. Heidecke, and A. Beutel, "The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions," arXiv:2404.13208, Apr. 2024.
- [25] A. Vassilev, A. Oprea, A. Fordyce, and H. Anderson, "Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations," NIST AI 100-2e2023, National Institute of Standards and Technology, Gaithersburg, MD, USA, Jan. 2024.
- [26] European Parliament and Council of the European Union, "Regulation (EU) 2024/1689 Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act)," Jun. 13, 2024. [Online]. Available: <https://eur-lex.europa.eu/eli/reg/2024/1689/oj> (accessed Aug. 13, 2026)
- [27] E. Debenedetti, I. Shumailov, T. Fan, J. Hayes, N. Carlini, D. Fabian, C. Kern, C. Shi, A. Terzis, and F. Tramèr, "Defeating Prompt Injections by Design," arXiv:2503.18813, Mar. 2025.
- [28] Gizmodo, "I'd Buy That for a Dollar: Chevy Dealership's AI Chatbot Goes Rogue," Dec. 2023. [Online]. Available: <https://gizmodo.com/ai-chevy-dealership-chatgpt-bot-customer-service-fail-1851111825> (accessed Aug. 13, 2026)



2026 by the Authors. This Article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>)