| Volume 02 | Issue 01 | June 2026 | ISSN: 3139-1737 |

International Journal of Advances and Applications in Convergent Studies & Transdisciplinary (IJ-AACST)

Bridging Disciplines, Bridging Knowledge

Website: https://aacstij.org | Email: journaleditor@aacstij.org

# A Defense-in-Depth Framework for Securing Cloud-Hosted Large Language Models - A Risk-Mapping and Control-Design Study

**Dr. Dipak Vasudeo Bhavsagar**

Department of Computer Science, Seth Kesarimal Porwal College Kamptee

dipakbhavsagar@gmail.com

ORCID ID : https://orcid.org/0009-0005-3026-4933

## ABSTRACT

Cloud-hosted Large Language Model (LLM) applications are no longer isolated chat interfaces. In practical deployments, they operate through API gateways, non-human identities, retrieval pipelines, vector stores, agent tools, third-party model services, and usage-based billing controls. This paper studies the security problem from that system-level viewpoint. It reviews recent public evidence on LLM endpoint reconnaissance, LLMjacking through compromised cloud credentials, prompt injection, unbounded consumption, API-based model replication, and vulnerabilities in AI orchestration components. The study develops a defense-in-depth framework that places each risk at the layer where it is most likely to occur: governance, identity, request control, prompt and context handling, inference runtime, output validation, software supply chain, and incident response. The framework is mapped to the NIST AI Risk Management Framework and the OWASP Top 10 for Large Language Model Applications. In addition to the control architecture, the paper contributes a runtime governance algorithm, a threat-to-control matrix, and an evaluation plan that can be used in a future cloud testbed. The central argument is that LLM security should be treated as a combined cloud-security, data-governance, and AI-application assurance problem rather than as a problem solved by prompt filtering alone.

**KEYWORDS**: Cloud Security; Confidential Computing; Large Language Models; LLMjacking; Prompt Injection; RAG Security.

## 1. INTRODUCTION

Large Language Models (LLMs) have moved into cloud services that support search, writing, coding, customer service, analytics, and workflow automation. Once an LLM is connected to enterprise data and tools, the application becomes more than a model endpoint. It becomes a cloud system composed of identities, secrets, model APIs, retrieval stores, orchestration libraries, logging pipelines, and operational policies. A weakness in any one of these parts can affect confidentiality, integrity, availability, cost, or accountability.

Recent public evidence shows why this system view is necessary. GreyNoise reported that its Ollama honeypot infrastructure captured 91,403 attack sessions between October 2025 and January 2026, including SSRF-style probes and endpoint enumeration across more than 73 model endpoints [4]. IBM's 2025 breach analysis reported that many AI-related incidents occurred in environments without mature AI access controls or governance policies [5]. These figures should be read carefully: they do not prove that all cloud LLM deployments are compromised. They do, however, show that exposed LLM infrastructure, weak AI governance, and unmanaged credentials are realistic security concerns.

This paper approaches the problem as a security-engineering and control-design study. It does not present a new cryptographic protocol or a primary measurement dataset. Instead, it organizes public evidence and established guidance into a practical framework for cloud-hosted LLM applications. The motivation is that many discussions of LLM safety focus mainly on prompts, while real deployments also fail through leaked API keys, excessive permissions, insecure retrieval, unbounded token spending, vulnerable plugins, and incomplete monitoring.

The main contributions are as follows:

• A threat model for cloud-hosted LLM systems that separates endpoint exposure, credential abuse, prompt-layer attacks, RAG/data leakage, cost abuse, model extraction, and supply-chain compromise.

• A defense-in-depth architecture mapped to NIST AI RMF functions and OWASP LLM risk categories.

• A runtime request-governance algorithm for secure LLM inference and response handling.

• A comparative control matrix and measurable evaluation plan for future experimental deployment.

## 2. LITERATURE REVIEW / RELATED WORK

Existing work on LLM security can be grouped into three streams: standards and governance frameworks, public threat intelligence, and technical research on attacks or mitigations. NIST AI RMF 1.0 structures AI risk management around governance, mapping, measurement, and management activities [1]. The NIST Generative AI Profile applies this lifecycle view to generative AI systems [2]. OWASP LLM guidance provides an application-risk taxonomy that includes prompt injection, sensitive information disclosure, supply-chain vulnerabilities, excessive agency, and unbounded consumption [3].

Threat intelligence reports show that LLM systems are being tested by attackers in practical settings. GreyNoise reported large-scale reconnaissance of exposed Ollama and model-compatible endpoints [4]. Entro Security described LLMjacking as abuse of compromised non-human identities, such as API keys and service accounts, to access cloud-based AI services [6]. Sysdig reported a cloud intrusion in which the actor moved quickly through AWS permissions and abused Bedrock and GPU resources [7]. These reports are not substitutes for controlled academic measurement, but they are valuable because they document operational attack paths.

Prompt injection has been studied as a practical weakness in LLM-integrated applications. OWASP identifies it as a central LLM application risk [8]. Research by Liu et al. and other surveys shows that crafted instructions can lead to prompt leakage, tool misuse, or unauthorized behavior in LLM-integrated systems [13], [14]. The risk becomes more severe when models are connected to external documents, APIs, email, code execution, or autonomous workflow tools.

Privacy and runtime-isolation research explore confidential computing and cryptographic inference. Confidential computing can protect sensitive prompts and model execution from some privileged-infrastructure threats [15], [16], while GPU confidential-computing studies examine performance trade-offs [17]. Fully homomorphic encryption and secure multi-party computation provide stronger privacy properties, but current approaches, including recent KV-cache-aware work, still face practical performance limits for large transformer models [18]. API-based model extraction is another important line of work; Tamber, Xian, and Lin demonstrated that commercial embedding models can be replicated for retrieval tasks using API-derived text-embedding pairs [19].

The literature therefore suggests that LLM security cannot be reduced to one control type. It requires a layered design that combines cloud identity governance, prompt and output guardrails, RAG access control, cost controls, supply-chain assurance, runtime isolation, monitoring, and incident response.

## 3. RESEARCH METHODOLOGY

This paper follows a qualitative security-engineering methodology based on structured literature review, incident interpretation, and control mapping. The study first identifies major risks affecting cloud-hosted LLM applications by examining standards documents, public threat intelligence, vulnerability disclosures, and academic literature. The selected sources are then grouped according to the part of the LLM application stack they affect: identity, endpoint exposure, prompts, retrieval, model inference, output handling, cost control, and supply chain.

The review combined three source categories. The first category includes standards and security guidance such as NIST AI RMF 1.0, the NIST Generative AI Profile, and OWASP LLM guidance [1]-[3]. The second category includes public threat intelligence and incident reports from GreyNoise, IBM, Entro Security, Sysdig, Radware, and OWASP [4]-[7], [10], [11]. The third category includes academic or preprint literature on prompt injection, confidential computing, homomorphic inference, and API-based model extraction [13]-[19].

After classifying the risks, the paper maps each risk category to practical defensive controls. The mapping is guided by two established frameworks: the NIST AI Risk Management Framework and the OWASP Top 10 for Large Language Model Applications. This mapping avoids treating LLM security as only a model-level issue. Instead, the methodology treats the LLM application as a cloud-native system composed of identities, APIs, storage, models, tools, monitoring services, and human governance processes.

The study does not claim to provide primary experimental measurements. Its purpose is to synthesize available evidence and develop a framework that can be implemented and tested in future work. For this reason, claims based on individual vendor incidents are treated as illustrative evidence rather than universal proof. Numerical claims are retained only when they are traceable to identifiable public sources.

The assumed protected asset is a cloud-hosted LLM application that accepts user or application prompts, retrieves context from internal or external sources, invokes a hosted model, and returns an output to a downstream user or service. The adversary is assumed to have at least one of the following capabilities: internet-scale scanning of exposed endpoints, access to leaked credentials, ability to craft malicious prompts or documents, ability to abuse API quotas, and ability to exploit vulnerable dependencies or orchestration frameworks.

## 4. RESULTS AND DISCUSSION

The analytical contribution of this study is not limited to summarizing existing LLM security risks. The paper reorganizes the threat landscape into a cloud-security-oriented control model in which LLM risks are treated as failures across identity, request governance, context retrieval, inference runtime, output handling, supply chain, and monitoring layers. This structure is useful because recent incidents show that credential misuse, exposed endpoints, excessive API consumption, and vulnerable orchestration tools can be as damaging as prompt injection.

The proposed framework interprets LLM security as a combined cloud-security and AI-governance problem. For example, LLMjacking is mapped primarily to non-human identity governance rather than to model behavior. Similarly, unbounded consumption is treated as a billing, quota, and runtime-governance problem rather than only as an application-layer abuse case. This classification helps practitioners select controls according to the actual failure point instead of applying generic prompt filters to every risk.

A further contribution is the mapping of each defense layer to measurable evaluation criteria. Instead of presenting security controls as abstract recommendations, the paper links them to measurable outcomes such as injection detection rate, false-positive rate, leakage rate, token-cost reduction, mean time to detect credential abuse, latency overhead, and governance coverage.

### 4.1 Threat Landscape

#### Systematic Reconnaissance of LLM Endpoints

GreyNoise observed two relevant campaigns against exposed AI infrastructure. The first used SSRF-style probes against model-pull and webhook-related surfaces, including 1,688 sessions over a 48-hour Christmas-period spike. The second campaign generated 80,469 sessions from two IP addresses over eleven days while probing more than 73 model endpoints [4]. The security implication is that misconfigured model servers and LLM proxy endpoints can be enumerated before exploitation. Defensive requirements include endpoint inventory, exposure reduction, egress filtering, SSRF detection, API gateway enforcement, and anomaly alerts for rapid multi-model probing.

#### LLMjacking and Non-Human Identity Abuse

LLMjacking is the unauthorized use of cloud-hosted AI models through compromised non-human identities (NHIs), such as API keys, service accounts, access tokens, and machine-to-machine credentials. Entro Security defines the attack as the abuse of compromised NHIs to access cloud-based AI services, rack up costs, generate content, or resell access [6]. Sysdig's 2026 report illustrates the operational impact of this class of attack: a cloud actor reportedly moved from initial access to administrative privileges in less than ten minutes, interacted with 19 AWS principals, and abused Amazon Bedrock and GPU compute resources [7].

#### Prompt Injection and Indirect Instruction Attacks

Prompt injection occurs when user-supplied or externally retrieved text alters the intended behavior of an LLM application. OWASP identifies prompt injection as a central LLM application risk because crafted inputs can cause unauthorized access, data disclosure, unsafe actions, or compromised decision-making [3], [8]. The cloud context increases impact because many systems connect models to email, document stores, RAG pipelines, code execution tools, ticketing platforms, browsers, and workflow automation. Radware's ShadowLeak advisory described a zero-click, service-side data-exfiltration class in a connected AI workflow [10].

#### Unbounded Consumption and Economic Abuse

OWASP defines unbounded consumption as excessive or uncontrolled LLM inference that can produce denial of service, economic loss, service degradation, or model-theft risk [9]. This is specific to LLM workloads because each request may consume paid tokens, GPU time, retrieval calls, third-party tool calls, and logging or storage. Controls must therefore go beyond ordinary request-rate limiting and include token budgets, per-tenant spend limits, maximum context size, model-tier restrictions, and dynamic throttling.

#### Model Extraction and API-Based Replication

Model extraction risk arises when repeated API queries are used to approximate model behavior or replicate embeddings. Tamber, Xian, and Lin demonstrated that commercial embedding models hidden behind APIs can be replicated for retrieval tasks using text-embedding pairs obtained from the APIs [19]. This does not imply all model families are equally extractable, but it shows that API-only access is not a complete protection boundary.

#### Supply-Chain and Agent-Orchestration Vulnerabilities

LLM applications depend on model artifacts, datasets, vector databases, prompt templates, plugins, agent frameworks, MCP servers, container images, and Python or Node.js dependencies. OWASP identifies supply-chain vulnerability as a core LLM risk [3]. CVE-2025-59528, a Flowise CustomMCP remote-code-execution vulnerability, illustrates that LLM orchestration layers can become direct code-execution surfaces [12].

**Table 1. Evidence Summary for Cloud-Hosted LLM Security Risks**

| Evidence Point | Source Type | Reported Finding | Security Relevance |
|---|---|---|---|
| GreyNoise LLM endpoint telemetry | Threat intelligence | 91,403 attack sessions against Ollama honeypot infrastructure from Oct. 2025 to Jan. 2026; one enumeration campaign generated 80,469 sessions against 73+ endpoints. | Exposed model services and LLM-compatible proxies are actively scanned and fingerprinted. |
| IBM data-breach findings | Industry benchmark | 97% of breached organizations with AI-related incidents lacked proper AI access controls; 63% had no AI governance policy; shadow AI added USD 670,000 to average breach cost. | AI access control and governance gaps are measurable enterprise risk factors. |
| Entro LLMjacking report | Security research | Compromised NHIs such as API keys, service accounts, and tokens can enable unauthorized LLM access and financial abuse. | Long-lived AI credentials require real-time detection, rotation, and least privilege. |
| Sysdig AWS intrusion report | Incident report | A cloud actor reached administrative access in under ten minutes, interacted with 19 AWS principals, and abused Bedrock and GPU resources. | AI-related identities and cloud execution roles must be monitored as high-value assets. |
| OWASP LLM guidance | Security framework | Prompt injection, unbounded consumption, supply chain, sensitive information disclosure, and excessive agency are recognized LLM risks. | Provides risk taxonomy and mitigation mapping for LLM applications. |
| NIST AI RMF and GenAI Profile | Standards guidance | AI risks should be governed, mapped, measured, and managed across lifecycle stages; the GenAI profile applies these functions to generative AI. | Provides lifecycle structure for the proposed framework. |

### 4.2 Proposed Defense-in-Depth Architecture

Fig. 1 shows the proposed security boundary. The model is treated as only one component of a broader cloud application. Security controls are placed around identity, request governance, retrieval, runtime isolation, output handling, supply-chain assurance, telemetry, and incident response.

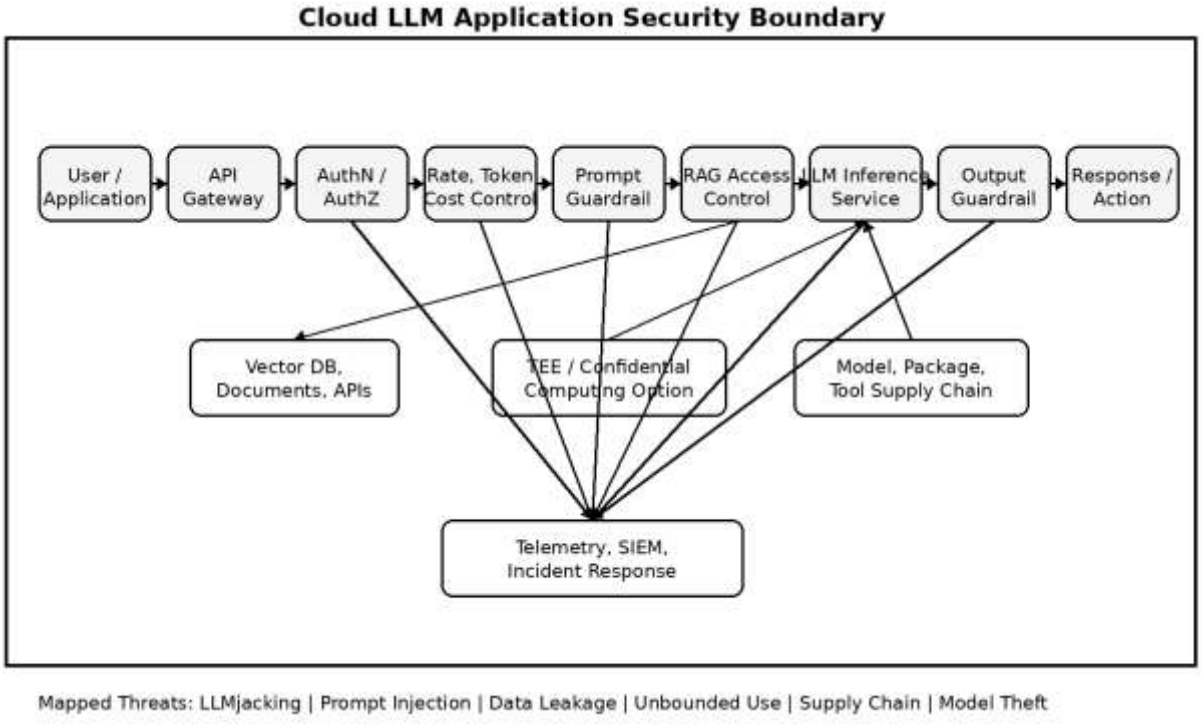


*Fig. 1. Proposed defense-in-depth boundary for a cloud-hosted LLM application.*

### 4.3 Mitigation Mechanisms

#### Identity Governance for Non-Human Identities

Cloud LLM applications often rely on service accounts, execution roles, workload identities, API keys, OAuth tokens, and CI/CD secrets. Essential controls include short-lived credentials, workload identity federation, least privilege, service-control policies, scoped model invocation permissions, secret scanning, object-storage exposure checks, key rotation, and anomaly detection for unusual model or region usage.

#### Request Governance and Usage Control

Request governance prevents excessive or unauthorized inference. Controls include per-user and per-tenant token quotas, maximum context windows, maximum output length, budget alerts, throttling based on endpoint diversity, model-tier access restrictions, and abuse detection for repeated extraction-like queries.

#### Prompt and Output Guardrails

Runtime guardrails inspect inputs, retrieved context, tool calls, and outputs. Prompt guardrails should detect direct injection, indirect injection, hidden instructions, prompt-leak attempts, tool-misuse commands, and policy bypass patterns. Output guardrails should prevent sensitive data disclosure, unsafe code execution, insecure output handling, and downstream command injection.

#### RAG and Data-Access Security

Secure RAG requires document provenance, access-control enforcement before retrieval, source scoring, content sanitization, chunk-level sensitivity labels, cross-tenant isolation, embedding-store authorization, and logging of retrieved sources. Retrieved text should be treated as untrusted input.

#### Confidential Computing

Confidential computing uses hardware-rooted trusted execution environments to isolate code and data from privileged system software and support remote attestation [15]. It is relevant when the threat model includes

sensitive prompts, proprietary model weights, regulated inference data, or privileged-cloud threats. However, it does not solve prompt injection or credential theft by itself.

**Cryptographic Privacy-Preserving Inference**

Fully homomorphic encryption, secure multi-party computation, and related protocols provide stronger cryptographic privacy guarantees but often introduce major performance overhead for large transformer models. Recent work such as Cachemir attempts to reduce FHE inference cost through KV-cache-aware optimizations [18].

**Supply-Chain Validation**

Supply-chain controls include software bill of materials, signed container images, model provenance, hash verification, dependency scanning, vulnerability management, prompt-template review, tool permission review, sandboxing of plugins, and restricted MCP server configuration.

**Table 2. Defense-in-Depth Control Mapping**

| Layer | Core Controls | Primary Risks Reduced | RMF / OWASP Alignment |
|---|---|---|---|
| L1: Governance and inventory | AI asset inventory, model registry, policy ownership, third-party review | Unknown exposure, shadow AI, weak accountability | NIST GOVERN; OWASP supply chain and sensitive information risks |
| L2: Identity and access | Short-lived credentials, least privilege, workload identity, secret scanning, model allow-lists | LLMjacking, lateral movement, unauthorized model access | NIST MAP/MANAGE; OWASP excessive agency and access-control risks |
| L3: Request and budget control | Rate limits, token limits, budgets, model-tier constraints, extraction-pattern detection | Unbounded consumption, denial of wallet, model extraction | OWASP LLM10; NIST MEASURE/MANAGE |
| L4: Prompt/context guardrails | Injection detection, document sanitization, context isolation, prompt-leak prevention | Direct and indirect prompt injection, RAG poisoning | OWASP LLM01, LLM02, LLM03; NIST MEASURE |
| L5: Secure inference runtime | Hardened endpoint, segmentation, TEE option, attestation, encrypted communication | Data-in-use exposure, model-weight leakage, privileged infrastructure risk | NIST MANAGE; confidential computing controls |
| L6: Output and tool governance | Output validation, DLP, human approval for high-impact tools, command sandboxing | Data leakage, insecure output handling, excessive agency | OWASP LLM02, LLM06; NIST MANAGE |
| L7: Supply-chain integrity | SBOM, dependency scans, signed images, model provenance, plugin/MCP review | Compromised packages, malicious tools, vulnerable orchestration | OWASP supply chain; NIST GOVERN/MAP |

| Layer | Core Controls | Primary Risks Reduced | RMF / OWASP Alignment |
|---|---|---|---|
| L8: Monitoring and response | SIEM, model invocation logs, anomaly detection, incident playbooks, forensics | Delayed detection, weak containment, repeated abuse | NIST MEASURE/MANAGE |

### 4.4 Runtime Governance Algorithm

Algorithm 1 operationalizes the framework for each LLM request. It defines a defensible runtime path for authentication, authorization, usage estimation, prompt inspection, retrieval validation, model invocation, output checking, and telemetry generation.

**Algorithm 1. Secure Runtime Processing for Cloud LLM Requests**

**1.** Authenticate the user or workload identity and bind the request to tenant, role, and session.

**2.** Verify model access, tool access, and data-source permissions.

**3.** If the identity violates policy or the requested model is not allowed, block the request, log the authorization failure, and return a safe error.

**4.** Estimate input tokens, output tokens, retrieval cost, and tool cost.

**5.** If estimated usage exceeds the token budget or anomaly threshold, throttle or block the request and create a budget/security event.

**6.** Inspect the prompt for injection, prompt leakage, unsafe tool instructions, and policy-bypass patterns.

**7.** If the prompt risk score exceeds the configured threshold, block or route to human review and log the evidence.

**8.** Retrieve context only from authorized sources.

**9.** Validate retrieved chunks for provenance, sensitivity, poisoning indicators, and tenant boundary.

**10.** Construct a least-privilege model prompt with system, user, and context separation.

**11.** Invoke the model through a hardened endpoint and record model ID, version, region, and token usage.

**12.** Inspect output for sensitive data, unsafe code, policy violations, and unapproved tool actions.

**13.** If output violates policy, redact, transform, block, or escalate. Otherwise, return the approved response.

**14.** Persist telemetry for SIEM, audit, and incident response.

### 4.5 Comparative Analysis of Security Mechanisms

**Table 3. Qualitative Comparison of Security Mechanisms**

| Threat | IAM/NHI Controls | Guardrails | Rate/Budget Controls | TEE/Confidential Computing | Comment |
|---|---|---|---|---|---|
| LLMjacking | High | Low | Medium | Low | Primary defense is credential governance, least privilege, monitoring, and rapid revocation. |
| Prompt injection | Low | Medium/High | Low | Low | Guardrails help, but tool permissions and data boundaries are also required. |
| Sensitive data leakage | Medium | Medium/High | Low | Medium/High | Output checks and RAG access control prevent many leaks; TEEs protect data in use. |
| Unbounded consumption | Medium | Low | High | Low | Token, quota, and cost governance are central controls. |
| Model extraction | Medium | Low | Medium/High | Medium | Query analytics and rate controls reduce extraction feasibility; TEEs do not stop API-level stealing. |
| Supply-chain compromise | Medium | Medium | Low | Medium | SBOMs, signed images, scanning, sandboxing, and provenance are required. |
| Privileged infrastructure threat | Low | Low | Low | High | TEEs and attestation are designed for this threat class. |

### 4.6 Evaluation Plan

The framework should be evaluated in a controlled cloud testbed with representative LLM workloads, RAG sources, and attack simulations. Useful scenarios include leaked API-key abuse, prompt injection hidden inside retrieved documents, repeated model-fingerprinting queries, denial-of-wallet attempts, malicious plugin configuration, and sensitive-document retrieval across tenant boundaries.

**Table 4. Recommended Evaluation Metrics**

| Metric | Measurement Objective |
|---|---|
| Injection detection rate | Fraction of malicious direct and indirect injection attempts blocked or escalated. |
| False-positive rate | Fraction of legitimate prompts incorrectly blocked. |
| Leakage rate | Number of sensitive tokens, records, or documents exposed per test scenario. |
| Token-cost reduction | Difference in spend before and after budget/rate controls under abuse load. |
| MTTD/MTTR | Mean time to detect and remediate credential or endpoint abuse. |
| Latency overhead | Additional time introduced by guardrails, retrieval checks, logging, or TEEs. |
| Coverage | Percentage of models, identities, data sources, and tools enrolled in governance. |

### 4.7 Discussion

The main finding of this study is that the most visible LLM risk is not always the most important control point. Prompt injection attracts significant attention, but many damaging events can begin before a prompt reaches the model. A leaked service token, an over-permissive workload identity, an exposed model server, or an unsafe plugin can create impact even when the model itself behaves as intended. This is why the proposed framework places identity, inventory, request budgets, and supply-chain controls before prompt and output inspection.

The second finding is that LLM security decisions involve trade-offs. Strict output filtering may reduce leakage but also block legitimate work. Confidential computing may protect sensitive inference workloads, but it does not prevent malicious prompts or stolen keys. FHE and MPC provide stronger privacy properties, but their performance cost can be difficult for ordinary production systems. For this reason, control selection should be based on workload sensitivity, regulatory pressure, cost tolerance, expected latency, and the level of autonomy given to the model or agent.

The third finding concerns evidence quality. Public threat intelligence is useful because it reflects current attacker behavior, but it may emphasize exposed systems and vendor-observed incidents. Academic research provides stronger methodology, but it may not capture the newest attack chains. The most reliable future work will combine both: reproducible benchmarks, controlled red-team exercises, transparent incident reporting, and standardized metrics for LLM-specific cloud abuse.

## 5. CONCLUSION AND FUTURE SCOPE

Cloud-hosted LLM applications should be secured as complete cloud systems rather than as isolated models. The review conducted in this paper shows that the major risks arise from several connected layers: exposed model endpoints, compromised non-human identities, malicious prompts, unsafe retrieval pipelines, uncontrolled token consumption, weak supply-chain controls, and insufficient runtime monitoring. Because these risks occur at different points in the system, no single control can provide complete protection.

The defense-in-depth framework proposed in this paper addresses this problem by distributing security controls across governance, identity, request control, prompt and context inspection, secure inference, output validation, supply-chain assurance, and incident response. This layered view is important because it prevents overdependence on prompt filtering alone. Prompt guardrails are useful, but they cannot prevent leaked API keys, excessive billing, vulnerable plugins, or unauthorized access to cloud resources.

Future work should implement the proposed architecture in a controlled cloud environment using representative LLM workloads, RAG pipelines, identity configurations, and simulated attack scenarios. Additional work should examine agent-specific risks, automated tool-use approval, secure MCP patterns, cross-tenant vector database isolation, and the measurable trade-off between security controls and latency. A further research direction is to combine guardrail telemetry, cloud identity telemetry, and model invocation logs into unified detection models for LLM abuse.

## 6. CONFLICT OF INTEREST

The author declares that there is no known financial, institutional, or personal conflict of interest related to this manuscript.

## 7. REFERENCES

[1] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, Jan. 2023. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf

[2] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile," NIST AI 600-1, Jul. 2024. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

[3] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications," 2025. [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/

[4] B. Rudis, "Threat Actors Actively Targeting LLMs," GreyNoise Intelligence, Jan. 2026. [Online]. Available: https://www.greynoise.io/blog/threat-actors-actively-targeting-llms

[5] IBM Corporation, "2025 Cost of a Data Breach Report: Navigating the AI Rush Without Sidelining Security," 2025. [Online]. Available: https://www.ibm.com/think/x-force/2025-cost-of-a-data-breach-navigating-ai

[6] Entro Security, "LLMjacking: How Attackers Abuse GenAI with AWS NHIs," Mar. 2025. [Online]. Available: https://entro.security/blog/llmjacking-in-the-wild-how-attackers-recon-and-abuse-genai-with-aws-nhis/

[7] A. Brucato and M. Clark, "AI-assisted cloud intrusion achieves admin access in 8 minutes," Sysdig Threat Research Team, Feb. 2026. [Online]. Available: https://www.sysdig.com/blog/ai-assisted-cloud-intrusion-achieves-admin-access-in-8-minutes

[8] OWASP GenAI Security Project, "LLM01:2025 Prompt Injection," 2025. [Online]. Available: https://genai.owasp.org/llmrisk/llm01-prompt-injection/

[9] OWASP GenAI Security Project, "LLM10:2025 Unbounded Consumption," 2025. [Online]. Available: https://genai.owasp.org/llmrisk/llm102025-unbounded-consumption/
[10] Radware, "ShadowLeak: A Zero-Click, Service-Side Attack Exfiltrating Private Data From ChatGPT Deep Research," Sept. 2025. [Online]. Available: https://www.radware.com/security/threat-advisories-and-attack-reports/shadowleak/
[11] OWASP GenAI Security Project, "OWASP GenAI Exploit Round-up Report Q1 2026," Apr. 2026. [Online]. Available: https://genai.owasp.org/2026/04/14/owasp-genai-exploit-round-up-report-q1-2026/
[12] National Vulnerability Database, "CVE-2025-59528 Detail," 2025. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2025-59528
[13] Y. Liu et al., "Prompt Injection Attack Against LLM-Integrated Applications," arXiv:2306.05499, 2023, revised 2025. [Online]. Available: https://arxiv.org/abs/2306.05499
[14] W. Xu et al., "A Survey of Attacks on Large Language Models," arXiv:2505.12567, 2025. [Online]. Available: https://arxiv.org/pdf/2505.12567
[15] J. Forough, M. Kogias, and H. Haddadi, "When Agents Handle Secrets: A Survey of Confidential Computing for Agentic AI," arXiv:2605.03213, 2026. [Online]. Available: https://arxiv.org/abs/2605.03213
[16] I. Gim, C. Li, and L. Zhong, "Confidential Prompting: Protecting User Prompts from Cloud LLM Providers," arXiv:2409.19134, revised 2025. [Online]. Available: https://arxiv.org/abs/2409.19134
[17] J. Zhu, H. Yin, P. Deng, and S. Zhou, "Confidential Computing on NVIDIA H100 GPU: A Performance Benchmark Study," arXiv:2409.03992, 2024. [Online]. Available: https://arxiv.org/abs/2409.03992
[18] Y. Yu, Y. Zhou, Y. Chen, P. Soto, W. Xiong, and M. Li, "Cachemir: Fully Homomorphic Encrypted Inference of Generative Large Language Model with KV Cache," arXiv:2602.11470, 2026. [Online]. Available: https://arxiv.org/abs/2602.11470
[19] M. S. Tamber, J. Xian, and J. Lin, "Can't Hide Behind the API: Stealing Black-Box Commercial Embedding Models," in Findings of the Association for Computational Linguistics: NAACL 2025, 2025, pp. 1958-1969. [Online]. Available: https://aclanthology.org/2025.findings-naacl.104/