Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

Securing Enterprise Data for LLM-Powered Applications: A Reference Architecture for Inference-Time Data Protection

Sathiesh Veera
IEEE Senior member, Phoenix, AZ, USA
Corresponding Author: Sathiesh Veera, **E-mail**: sathyvsk@gmail.com

| ABSTRACT

Enterprises are rapidly adopting large language models (LLMs) by connecting them to internal data sources through approaches like retrieval-augmented generation (RAG), natural language database querying, and API based tool calling. In most enterprise deployments, the LLM itself is not hosted with in the company's cloud tenant, but hosted by a third-party provider, which means the company data including customer records, financial information and proprietary documents must leave the organization's, controlled environment every time a query is processed. This applies to the adoption of Saas products as well that leverage GenAI capabilities, where often times the data leaves the organization, the Saas platforms and are sent to the LLM vendors. Contractual agreements with the LLM providers help, but they cannot fully address risks like prompt injection, unintended data exposure, or regulatory non-compliance. This paper presents a practical reference architecture for securing enterprise data at inference time, organized around three main layers. The first layer addresses the data sources including unstructured content accessed through RAG, structured databases queried through SQL or graph languages, semi-structured records from SaaS APIs, and real-time event streams. The second layer covers the access mechanisms that connect these sources to LLMs, including function and tool calling, the Model Context Protocol, and code execution, along with agentic orchestration that combines multiple mechanisms autonomously. The third layer provides the governance controls such as data classification, role-based and context-aware access control, monitoring, audit trails, and regulatory compliance. The architecture draws on established industry standards including the NIST AI Risk Management Framework and the OWASP Top 10 for LLM Applications. This paper consolidates proven industry security practices into a coherent, practical, layered model that any organization can adapt when deploying LLM-powered applications against their enterprise data.

| KEYWORDS

Enterprise data protection, LLM security, Retrieval-augmented generation (RAG), Large Language Models (LLMs), Inference-time security, MCP, Prompt attacks, Tools calling, Governance, Cybersecurity.

| ARTICLE INFORMATION

ACCEPTED: 20 February 2026 **PUBLISHED:** 25 March 2026 **DOI:** 10.32996/jcsts.2026.8.5.9

1. Introduction

Organizations across industries are connecting their internal data systems to large language models. A customer service chatbot pulls answers from a company's knowledge base. A business analyst queries a data warehouse by typing a question in plain English. An AI agent looks up customer records, checks inventory, and drafts responses without human intervention. In each of these cases, enterprise data leaves the organization's, controlled environment and enters a large language model that is, more often than not, operated by a third-party provider.

The scale of adoption is significant. The global retrieval-augmented generation market is estimated at 1.94 billion USD in 2025 and projected to reach 9.86 billion USD by 2030 (MarketsandMarkets, 2025). At the same time, research suggests that roughly one in twelve employee prompts to generative AI tools includes confidential organizational information (Lasso Security, 2025). This creates a growing attack surface where sensitive customer data, intellectual property, and regulated records flow through inference pipelines to external model providers, frequently without adequate security controls in place.

While contracts with LLM providers typically state that enterprise data will not be used for training or any type of auditing, contracts alone are not a security strategy. The OWASP Top 10 for LLM Applications, updated for 2025, ranks prompt injection as the leading risk, followed by sensitive information disclosure and vector and embedding weaknesses (OWASP, 2025). These threats are particularly relevant in enterprise settings where LLMs connect to knowledge bases containing customer personally identifiable information, financial records, health data, un-released products, trade secrets etc. Prior work has begun addressing parts of this problem. (Siddharth Nandagopal, 2025) proposed a framework for securing RAG pipelines specifically, combining encryption, zero-trust architecture, and guardrail mechanisms like immutable system prompts. That work provides a useful foundation for RAG-specific security, but organizations today rarely use RAG alone. A single enterprise application might combine document retrieval, database querying, API calls, and autonomous agent actions within a single workflow. Security frameworks that address only one of these patterns leave the others exposed.

This paper takes a broader view. It presents a reference architecture for securing all the ways enterprise data reaches an LLM at inference time. The architecture is organized around three practical layers: the data sources being accessed, the mechanisms used to access them, and the governance controls that should wrap around both. It draws on industry standards including the NIST AI Risk Management Framework (NIST, 2023) and the OWASP Top 10 for LLM Applications (OWASP, 2025), and consolidates proven practices into a model that organizations of any size can adapt for their own deployments.

2. Enterprise Data sources exposed to LLMs

The first question any organization should ask is: what types of data are we sending to the LLM? The answer usually falls into four categories, each with its own security profile.

2.1 Unstructured Data (RAG)

Retrieval-augmented generation is the most widely adopted pattern for grounding LLM responses in enterprise knowledge. Documents, wikis, emails, PDFs, and other free-form content are converted into vector embeddings and stored in a vector database. At query time, the system performs a semantic search to find the most relevant chunks, ranks them, and passes them to the LLM as context (Gao et al., 2024). The security concerns here are well-documented: sensitive information embedded in retrieved documents gets transmitted to the LLM endpoint, retrieval may return documents the user should not have access to, and the vector database itself can be poisoned with content designed to manipulate LLM behavior (Zeng et al., 2024). The security risk is huge here, because most of these data are company internal, crowd sourced and some of them like wikis, are not even regularly audited.

2.2 Structured Data (Databases and Knowledge Graphs)

When users query enterprise databases through natural language, or when an Agentic AI workflow needs to get some data from databases, the LLM generates SQL, SPARQL, or Cypher queries that run against databases, data warehouses, or knowledge graphs. This pattern is powerful because it gives non-technical users direct access to enterprise data. It is also risky because the LLM-generated query may access columns or rows the user is not authorized to see, or it may generate overly broad queries that scan entire tables when only an aggregate was intended. The database schema metadata that gets shared with the LLM to help it write accurate queries can itself leak information about the organization's data model. Further, any additional sensitive information passed to an LLM, can also influence the reasoning, and persist decisions in memory, carrying the risk to further decisions and steps through the workflows.

2.3 Semi-Structured Data (APIs and SaaS Records)

Many enterprise Saas systems such as Salesforce, ServiceNow, Jira, SAP and even internal systems such as microservices expose data through REST or GraphQL APIs that return JSON or XML payloads. These are often made available to Agentic AI workflows for customer facing automations. When an LLM calls these APIs through tool-calling mechanisms, the response may include fields the user does not need to see or should not have access to. Unlike database queries where security policies can be enforced at the query level, API responses often return complete records, leaving it to the calling application to filter out sensitive fields. This exposes a big risk, when the APIs can accidentally expose some internal flags, or details about the service, etc. to the LLMs, which might even use the information to cite and quote responses.

2.4 Real-Time and Streaming Data

Some LLM applications consume live data feeds such transaction streams, monitoring events, IoT telemetry, or log data from systems like Kafka or Kinesis. The security challenge here is compounded by volume and velocity. Data arrives continuously, and the window for applying security controls before it enters an LLM context is narrow. Sensitive values in event payloads may be harder to detect in real time than in static documents or database records. If the organizations do not apply the necessary security measures on the stream itself, the LLM can access critical secure information in any single query, use that transient data for reasoning, and inadvertently log them in the LLM transactional logs.

3. Access Mechanisms: How LLMs Retrieve Enterprise Data

The data sources described above do not connect to LLMs directly. There is always an intermediary mechanism that fetches data from enterprise systems and delivers it to the model. Understanding and securing these mechanisms is the second layer of the architecture.

3.1 Function and Tool Calling

Most modern LLMs support function calling, where the model outputs a structured request to invoke a predefined function with specific parameters. The application then executes the function, retrieves the result, and passes it back to the model. This is the basic building block for all data retrieval: a RAG lookup, a database query, and an API call are all implemented as tool calls under the hood. The security risk is that prompt injection can manipulate which function the model calls and with what parameters. An attacker who controls part of the input can trick the model into calling a function that retrieves data beyond the intended scope (OWASP, 2025).

3.2 Model Context Protocol (MCP)

The Model Context Protocol, introduced by Anthropic in November 2024, standardizes how LLMs connect to external tools and data sources. It has been adopted rapidly, with integrations across major platforms including Claude, OpenAI Agents, GitHub Copilot, and enterprise tools like Slack and Salesforce (Radosevich & Halloran, 2025). MCP provides a client-server architecture where MCP servers expose capabilities and MCP clients mediate between the user, the LLM, and the servers.

The security implications are significant. Each MCP server effectively gives the LLM a set of permissions to interact with an external system. If an MCP server is misconfigured or compromised, the LLM gains access to resources it should not reach. Recent security research has demonstrated that LLMs can be coerced into using MCP tools to execute malicious code, steal credentials, and exfiltrate data (Radosevich & Halloran, 2025). Tool poisoning attacks can embed malicious logic in MCP server descriptions that influence how the LLM uses the tool (Hou et al., 2025). The Cloud Security Alliance has described MCP servers as a form of shadow IT, noting that they often act as a direct line from a web-hosted LLM into an organization's internal applications (CSA, 2025).

3.3 Code Generation and Execution

Some LLM applications generate and execute code like Python, R, or SQL scripts against enterprise data environments. Data analysis copilots, for example, write code that runs in a notebook or sandbox and returns results to the LLM. This introduces the risk that generated code may access data beyond the user's authorization, produce side effects like writing or deleting data, or contain logic errors that expose sensitive information. If the code is not restricted or if not run in a sandboxed execution environment, it might have additional access at the filesystem and network layer. If the LLM's generated code is compromised to download and install vulnerable libraries in an environment that has access to secure data, the security risk extends beyond data exfiltration to full system compromise.

3.4 Agentic Orchestration

Agentic AI workflows combine multiple access mechanisms into autonomous, multi-step operations. An agent might retrieve a document through RAG, look up related records through an API, query a database for some details, and compose a summary, all without any human intervention between steps. The security concern is not just any single step, but the accumulation of data access across steps. Individually authorized actions can combine to create an exposure that no single action would cause on its own. Research has identified this as the lethal trifecta: a system with access to untrusted input, access to sensitive data, and the ability to act externally (Gulyamov et al., 2026). Organizations deploying agentic systems might already be exposed to a significant security gap because of the cumulatively data sensitivity across an agent's session.

4. Threats and Attack Vectors

The threats to enterprise data at inference time come from multiple directions. Prompt injection remains the most prominent: attackers craft inputs that cause the LLM to execute unintended actions, bypass access controls, or leak sensitive information. OWASP classifies this as the number one risk for LLM applications, noting that both direct injection through user input and indirect injection through retrieved content pose serious dangers (OWASP, 2025).

RAG-specific attacks include poisoning the knowledge base with documents that contain hidden instructions, and membership inference attacks where adversaries determine whether specific documents exist in the retrieval corpus (Zeng et al., 2024; Arzanipour et al., 2025). For structured data, the risk includes SQL injection through generated queries and schema inference through repeated interactions. For API-based access, excessive data exposure through full record returns and unauthorized field access are common concerns.

MCP introduces additional attack surfaces. Tool poisoning embeds malicious logic in MCP server descriptions, causing the LLM to misuse tools (Hou et al., 2025). Namespace typosquatting tricks users into connecting to malicious MCP servers with names similar to legitimate ones. A real-world example surfaced in 2025 when Asana's MCP implementation was found to expose data

across organizations due to a tenant isolation flaw (Pomerium, 2025). The enterprise implication is clear: every mechanism that connects enterprise data to an LLM is a potential exfiltration path, and security must be applied at each one.

5. Security Controls and Governance

With the data sources, access mechanisms and the threat vectors identified, security measures should be applied to each of this in a three-layered architecture.

5.1 Controlling the Data Layer

Before any enterprise data reaches an LLM, organizations need to know what sensitivity level it carries and make deliberate decisions about what should be accessible to LLM applications in the first place. This is not a one-time exercise. It should be a continuous process integrated with the organization's broader data governance program. The NIST AI Risk Management Framework emphasizes that AI risks should be integrated into enterprise risk management alongside cybersecurity and privacy (NIST, 2023). In practice, this means maintaining a registry of every data source connected to an LLM pipeline, documenting its sensitivity classification, regulatory constraints, and approved use cases.

5.1.1 Data Preparation

The most effective pre-transmission controls are not applied as a last-minute filter before data reaches the LLM. They are built into each layer of the data access path, starting at the source itself. At the data source level, organizations should adopt the principle of building purpose-specific knowledge domains for LLM use, rather than exposing all available data and hoping security controls downstream will handle it. For unstructured data, this means creating a dedicated RAG corpus that contains only the content intended for LLM consumption, with sensitive information removed or redacted before embeddings are ever generated. If a document contains customer social security numbers but the LLM application only needs product information from that document, the sensitive fields should be stripped at ingestion time, not filtered after retrieval. For structured data, database administrators should define LLM-specific roles or views that restrict which tables, columns, and rows are accessible when queries originate from an LLM application. A database view that excludes personally identifiable columns is more reliable than trusting a downstream filter to catch every query that touches those columns. For APIs, services should recognize when a request comes from an LLM integration, based on authentication headers or dedicated service credentials, and return only the fields relevant to the LLM use case rather than the full record. An API that returns a complete customer profile to a human user might return only the customer's name and account status when the caller is an LLM tool.

5.1.2 Pre-Transmission Controls

At the retrieval level, additional controls act as a second line of defense. Even with a well-scoped knowledge corpus, retrieval may surface content that was missed during ingestion-time filtering, or data sensitivity may change over time. Scanning retrieved content for PII, financial data and other sensitive categories before it enters the LLM context provides a safety net. For database query results, this means validating that returned data does not exceed what the user and the use case require. For API responses, it means stripping any fields that passed through the source-level filter but are not needed for the specific query being answered. For streaming data, it means applying inline masking on the event payload before it enters the LLM context window.

The underlying principle is straightforward: do not make data available to LLM systems unless you have a specific reason to. Organizations that connect their entire document repository or full database to an LLM application because it might be useful someday are accepting unnecessary risk. A smaller, well-curated, sensitivity-aware knowledge domain is easier to secure, easier to audit, and produces better LLM responses than a sprawling corpus where security controls are bolted on after the fact.

5.2 Securing the Access Mechanism Layer

The access mechanism layer sits between enterprise data sources and the LLM. It is where tool calls are constructed, queries are generated, and code is written on behalf of the user. Securing this layer requires more than traditional role-based access control. Three distinct concerns need to be addressed: authentication and transport security between components, validation of generated actions before they execute, and guardrails against injection attacks that manipulate what the LLM decides to do.

5.2.1 Authentication and Transport Security

Every connection between an LLM client and a data-access service should be authenticated and encrypted. This sounds obvious, but in practice it is often missing. The Model Context Protocol, for example, makes OAuth optional rather than mandatory. Many MCP server implementations deployed today lack authentication entirely, meaning any client that can reach the server can invoke its tools (Red Hat, 2025). Organizations deploying MCP should maintain a registry of approved MCP servers, run local servers in sandboxed environments, require mutual authentication between MCP clients and servers, verify server identity before allowing tool registration, and encrypt all communication channels. For direct tool calling and API integrations, the same

principles apply: the LLM application should authenticate to backend services using scoped credentials with the minimum permissions needed for the task, not broad service accounts that grant access to everything.

5.2.2 Pre-Execution Validation

When an LLM generates a SQL query, writes code, or constructs a tool call, the output should be reviewed before it runs against enterprise systems. A practical approach is to use a secondary model or a rule-based validator as a judge that inspects the generated action and checks whether it stays within the user's authorized scope. For SQL queries, this means verifying that the query only touches tables and columns the user is permitted to access, that it does not perform full table scans when only aggregates were requested, and that it does not contain patterns associated with data exfiltration such as UNION-based appending or subqueries against unrelated tables. For code execution, it means checking that generated scripts do not access the filesystem beyond permitted directories, do not make network calls to external endpoints, and do not import restricted libraries. For tool calls, it means validating that the function being called and the parameters being passed match what the user's request actually requires, rather than what an injected instruction might have steered the LLM toward. This validation step adds some latency, but it catches a category of risks that post-execution monitoring cannot, because by the time monitoring detects the problem, the data has already been accessed.

5.2.3 Injection Guardrails at the Mechanism Boundary

Prompt injection attacks do not just affect the LLM's text output. They can manipulate which tool the LLM selects, what parameters it passes, and how it interprets results. An attacker who controls part of the input, whether through a user prompt or through poisoned content retrieved by RAG, can steer the LLM into calling a different function than intended, passing parameters that expand the scope of data access, or ignoring constraints defined in the system prompt (OWASP, 2025). Defending against this requires multiple measures working together. Tool call schemas should be strict, with well-defined parameter types and value ranges, so that unexpected values are rejected before execution. The set of tools available to the LLM should be limited to the minimum required for the current task and user role, reducing the attack surface if injection succeeds. Input sanitization should be applied to user prompts before they enter the tool-calling pipeline, filtering known injection patterns. For MCP specifically, tool descriptions provided by MCP servers should be treated as untrusted input, since tool poisoning attacks embed malicious instructions in these descriptions to influence how the LLM uses the tool (Hou et al., 2025). Organizations should review and approve MCP server tool descriptions as part of the server registration process rather than accepting them at face value.

5.3 Monitoring, Audit, and Compliance

Every interaction between enterprise data and an LLM should be logged. This includes the user identity, the query or prompt submitted, the data sources accessed, the mechanism used, the data retrieved and any masking applied, the context sent to the LLM, and the response received. These logs serve multiple purposes: forensic investigation of security incidents, regulatory compliance documentation, anomaly detection for ongoing threats, and continuous improvement of security controls.

The European Data Protection Supervisor has noted that RAG systems involving personal data transfer to external parties present particular GDPR compliance challenges (EDPS, 2025). The NIST AI RMF's Measure function calls for continuous monitoring and risk assessment throughout the AI lifecycle (NIST, 2023). In practical terms, organizations should implement real-time monitoring for anomalous data access patterns, such as a single user suddenly retrieving an unusual volume of records, or an agent making tool calls to systems it does not normally access and maintain audit trails sufficient to reconstruct any data access event for regulatory review.

6. Reference Architecture

Fig. 1 presents the complete reference architecture, organized as a top-to-bottom pipeline with supporting security and governance pillars. The core pipeline illustrates the data path, beginning at the top with the User/Application and Orchestrator layer, crossing a trust boundary to the Third-Party LLM Provider, and then flowing downward through Pre-Execution Validation, Access Mechanisms, and Pre-Transmission Filters to reach the Purpose-Built LLM Data Sources and the underlying Raw Enterprise Data. This vertical flow is flanked by continuous security pillars: Authentication & Transport and Injection Guardrails on the left, and Output Monitoring and Governance & Compliance on the right.

The architecture is intentionally technology-agnostic. An organization using OpenAI, Anthropic, Google, or an open-source model can apply the same layered structure. The specific tools used for data masking, access control, or monitoring will vary, but the pipeline stages and the principles behind them remain consistent.

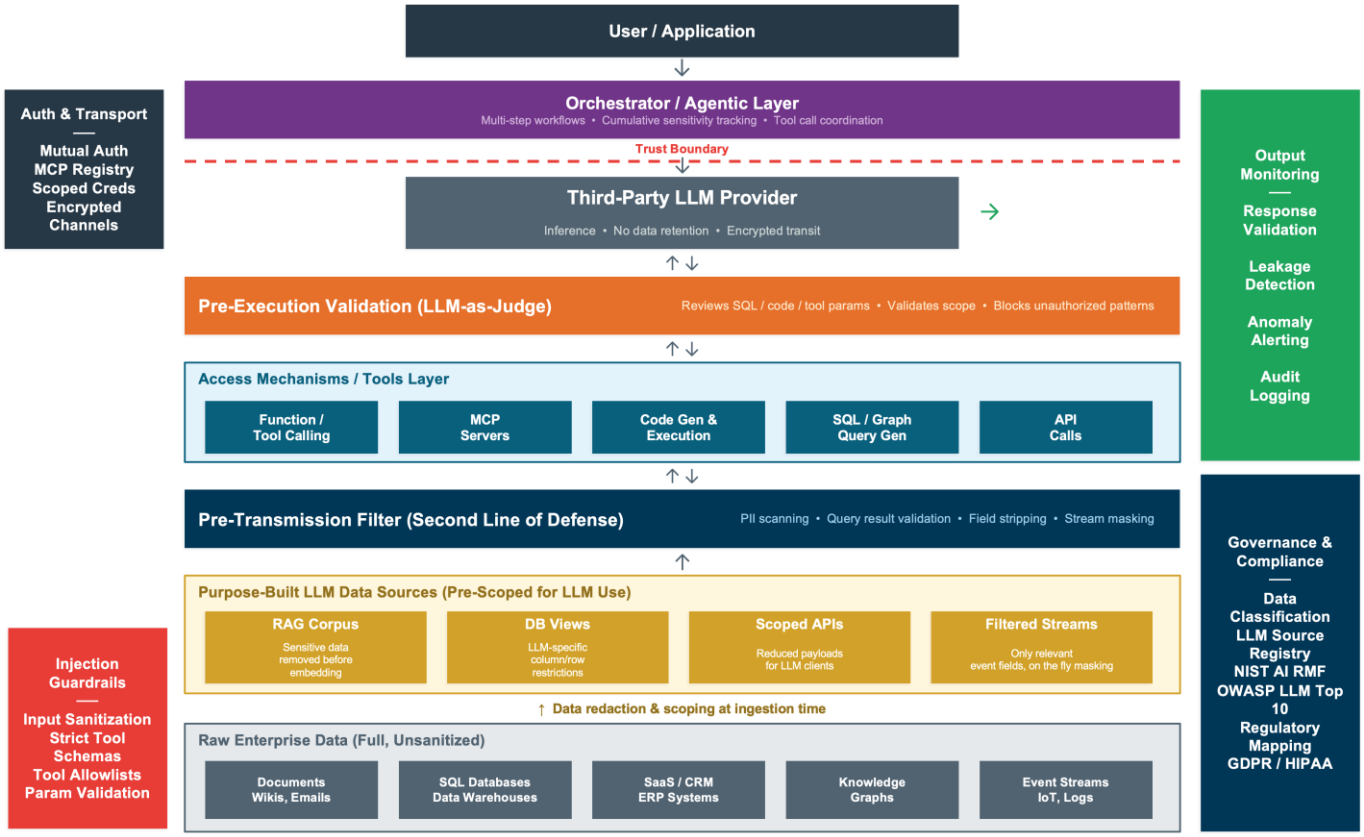

Fig 1:Reference Architecture for Inference-Time Data Protection

7. Discussion and Practical Recommendations

Most organizations already have data classification, access control, and monitoring infrastructure. The shift required is extending these capabilities to cover LLM inference pipelines.

The most impactful starting point is at the data source level. Before investing in downstream filters, organizations should audit what data is currently accessible to their LLM applications and whether all of it needs to be. Reducing scope at the source eliminates entire categories of risk before they arise, and it is a governance decision rather than a technology purchase.

Next, if MCP servers are in use, establishing a registry, enforcing authentication, and reviewing tool descriptions should be an immediate priority. For applications that generate SQL or execute code, pre-execution validation catches unauthorized access before it happens. This can start with rule-based checks and evolve toward model-based review over time.

Monitoring and audit should be non-negotiable from the start. Logging every tool call, query, and response creates the foundation for incident investigation and continuous improvement. Organizations that defer logging until after a security incident find themselves unable to determine what was accessed or how.

Trade-offs exist. Purpose-built knowledge domains require upfront effort. Pre-execution validation adds latency. MCP governance may slow development teams. A risk-based approach works best: apply the deepest controls to the highest-sensitivity data first, and extend coverage as the security program matures.

8. Conclusion

Enterprise data reaches large language models through multiple paths and mechanisms. Each one is a potential point of exposure, and securing just one leaves the others unprotected.

This paper presents a reference architecture organized around three layers: the data sources being accessed, the mechanisms used to access them, and the governance controls that wrap around both. The architecture draws on the NIST AI Risk Management Framework and the OWASP Top 10 for LLM Applications, and is designed to be practical, technology-agnostic, and adaptable. Organizations can start with source-level scoping, the highest-impact and lowest-cost control, and build outward toward mechanism the layer validation and monitoring as their security program matures.

Funding: This research received no external funding.
Conflicts of Interest: The authors declare no conflict of interest.
ORCID iD: 0009-0000-8207-9493

References

[1]. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., & Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. (arXiv:2312.10997v5). https://arxiv.org/abs/2312.10997v5

[2]. Zeng, S., Zhang, J., He, P., Liu, Y., Xing, Y., Xu, H., Ren, J., Chang, Y., Wang, S., Yin, D., & Tang, J. (2024). The Good and The Bad: Exploring privacy issues in retrieval-augmented generation (RAG). *Findings of the Association for Computational Linguistics: ACL 2022*, 4505–4524. https://doi.org/10.18653/v1/2024.findings-acl.267

[3]. Radosevich, B., & Halloran, J. (2025). *MCP safety audit: LLMs with the model context protocol allow major security exploits*. arXiv. https://arxiv.org/abs/2504.03767

[4]. MarketsandMarkets. (2025). Retrieval-augmented Generation (RAG) - Global Forecast to 2030. *Retrieval-augmented generation (RAG) market*. https://www.marketsandmarkets.com/Market-Reports/retrieval-augmented-generation-rag-market-135976317.html

[5]. Lasso Security. (2025), LLM data privacy: Protecting enterprise data in the world of AI. https://www.lasso.security/blog/llm-data-privacy

[6]. Siddharth Nandagopal. (2025). Securing retrieval-augmented generation pipelines: A comprehensive framework. *Journal of Computer Science and Technology Studies*, *7*(1), 17–29. https://doi.org/10.32996/jcsts.2025.7.1.2

[7]. OWASP. (2025). OWASP top 10 risk & mitigations for LLM & Gen AI apps. *LLMRisks.* https://genai.owasp.org/llm-top-10/

[8]. NIST. (2023). Artificial intelligence risk management framework (AI RMF 1.0). *NIST AI 100-1*. https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf

[9]. Hou, X., Zhao, Y., Wang, S., & Wang, H. (2025). *Model Context Protocol (MCP): Landscape, security threats, and future research directions*. arXiv. https://arxiv.org/abs/2503.23278

[10]. *CSA*. (2025). *MCP Can Be RCE for you and me.* Cloudsecurityalliance.org. https://cloudsecurityalliance.org/blog/2025/11/25/mcp-can-be-rce-for-you-and-me

[11]. Gulyamov, S., Gulyamov, S., Rodionov, A., Khursanov, R., Mekhmonov, K., Babaev, D., & Rakhimjonov, A. (2026). Prompt injection attacks in large language models and AI agent systems: A comprehensive review of vulnerabilities, attack vectors, and defense mechanisms. *Information*, *17*(1), 54. https://doi.org/10.3390/info17010054

[12]. Arzanipour, A., Behnia, R., Ebrahimi, R., & Dutta, K. (2025). RAG security and privacy: Formalizing the threat model and attack surface. arXiv. https://arxiv.org/abs/2509.20324

[13]. Red Hat. (2025). Model Context Protocol (MCP): Understanding security risks and controls. https://www.redhat.com/en/blog/model-context-protocol-mcp-understanding-security-risks-and-controls

[14]. EDPS. (2025). Retrieval-augmented generation (RAG). *European Data Protection Supervisor*. https://www.edps.europa.eu/data-protection/technology-monitoring/techsonar/retrieval-augmented-generation-rag_en