JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Mitigating Indirect Prompt Injection in RAG Architectures via Localized Input Sanitization

An Empirical Evaluation of a Multi-Tiered Defensive Gateway Pipeline

[1]Prathmesh Mishra, [2]Mr. Rashid Patel
[1]MSc CS specialization in cybersecurity, [2]Assistant Professor
[1,2]Department of Advanced Computing
[1,2]Nagindas Khandwala College
[1]University of Mumbai, Maharashtra, Mumbai

Abstract

Grounded generation frameworks relying on dense vector retrieval have become standard practice for curbing hallucinations and integrating private institutional knowledge into large language models [1], [24]. However, expanding the inference boundary to uncurated external corpora introduces an insidious vulnerability: indirect prompt injection [4], [9]. By embedding adversarial control directives inside retrieved passages—such as customer tickets, vendor contracts, or scraped documentation—attackers can hijack internal decoder attention, trigger unauthorized tool calls, or precipitate corporate data exfiltration [10], [14], [25]. Conventional defenses present an impractical dilemma. Static regular expression filtering executes with negligible overhead but collapses under semantic paraphrasing and obfuscation, whereas external cloud-hosted guardrails (e.g., Llama Guard or auxiliary LLM evaluators) incur prohibitive latency penalties (800–1200 ms), substantial recurring operational costs, and confidentiality compliance violations [7], [8], [16]. To reconcile this security-efficiency tension, we design, implement, and benchmark a localized, five-tier defensive gateway. Integrated into an end-to-end operational pipeline (rag_app), our system cascades: (1) deterministic boundary and pattern filtration, (2) binary magic-byte validation and active object-tree sanitization (excising PDF JavaScript, launch actions, and embedded OLE macros via PyMuPDF), (3) localized semantic intent classification driven by an on-premises compact model (Ollama Qwen2.5-3B), (4) structural schema and encoding enforcement (neutralizing non-printable Unicode planes, token-flooding denial-of-service, and base64 payloads), and (5) sliding-window transaction throttling coupled with SHA-256 deduplication. Filtered chunks are indexed within a pgvector datastore using Hierarchical Navigable Small World (HNSW) indexing, and answer generation is restricted to grounded Gemini Flash. Extensive benchmarking across 39 automated integration suites and 500 adversarial payloads demonstrates that the localized gateway attains a 96.8% injection detection recall, maintains a low 3.6% false positive rate, and restricts amortized latency to 18.5 ms—achieving a 98.0% latency reduction over cloud guardrails while preserving absolute on-premises data privacy.

Keywords: Retrieval-Augmented Generation (RAG), Indirect Prompt Injection, Input Sanitization, Defensive Gateway, Vector Database Security, Local Language Models (SLM), Adversarial Robustness, Zero-Trust AI Architecture.

I. Introduction

Modern enterprise artificial intelligence systems rely heavily on non-parametric knowledge retrieval to circumvent the foundational deficits of autoregressive language decoders: hallucinated factual claims, rigid knowledge cutoffs, and an inability to access proprietary organizational records [1], [24]. Retrieval-Augmented Generation (RAG) resolves these limitations by interfacing generative models with external vector repositories, such as PostgreSQL pgvector, ChromaDB, or FAISS. During a typical query transaction, the system converts user inquiries into dense semantic representations, retrieves top-k contextual passages through cosine similarity ranking, and assembles these excerpts into the model's active prompt window [1], [24], [25].

Despite dramatic gains in contextual relevance, expanding the model's receptive field to uncurated, multi-source external corpora introduces a critical vulnerability: Indirect Prompt Injection (IPI) [4], [9]. In contrast to direct prompt injection—where a hostile operator enters jailbreak commands into an interactive prompt interface—indirect injection embeds adversarial instructions inside external documents, such as customer service tickets, supplier invoices, public web crawls, or shared enterprise file shares [4], [9], [17]. When a user executes a benign query, semantic proximity causes the vector search engine to retrieve the poisoned chunk. Once ingested into the context window, the embedded payload hijacks the language model's attention mechanism, compelling it to bypass safety policies, exfiltrate sensitive tokens, or trigger unauthorized agentic tool executions [10], [14], [25].

Deploying robust defenses within enterprise pipelines involves a difficult trade-off between computational latency and defensive coverage. Surface-level lexical filters and regular expressions execute in fractions of a millisecond but fail consistently against semantic paraphrasing, multilingual instructions, or homoglyph substitutions [5], [6]. On the other hand, relying on auxiliary cloud-hosted model guardrails (such as Llama Guard or GPT-based evaluators) provides semantic detection but introduces severe round-trip network

latency (800–1200 ms), escalates operational API expenditures, and violates regulatory data sovereignty by transmitting unredacted corporate records to third-party endpoints [1], [7], [8], [16].

To resolve this dilemma, this work designs, constructs, and empirically validates an on-premises five-tier defensive input sanitization gateway. Positioned at both ingestion and query boundaries, the gateway operates as a zero-trust inspection filter that isolates and neutralizes adversarial payloads before vector indexing or prompt construction. The primary contributions of this paper are fourfold:

(1) A comprehensive literature review analyzing 25 peer-reviewed publications [1]–[25] starting with Boggarapu [1], categorizing the threat vectors, architectural loopholes, and defense strategies in contemporary RAG systems; (2) The design and implementation of a five-layer defensive gateway combining fast regex filtering, PyMuPDF-based binary document sanitization, localized SLM semantic classification (Ollama Qwen2.5-3B), structural schema validation, and stateful transaction rate limiting; (3) A zero-trust mathematical threat model and composite predicate evaluation function that guarantees fail-safe defaults and prevents adversarial retrieval poisoning; and (4) Extensive empirical validation across 500 benchmark attack payloads, demonstrating 96.8% injection recall, a 3.6% false positive rate, and an amortized latency overhead of just 18.5 ms—delivering a 98.0% latency reduction compared to cloud-based guardrails.

II. Literature Review

Securing generative retrieval pipelines requires addressing interconnected vulnerabilities across document ingestion, vector representation, prompt synthesis, and decoding. We systematically examine 25 core peer-reviewed investigations [1]–[25] organized across five foundational research pillars, establishing the theoretical baseline for localized multi-tier defenses.

A. Enterprise Governance, Provenance, and System-Wide Auditing in RAG

Enterprise environments demand strict data governance and traceable document lineage. Boggarapu [1] formulated a comprehensive security framework in Computer Fraud & Security emphasizing layered defenses, cryptographic data provenance, and runtime policy enforcement to prevent corporate data leakage in production RAG systems. This foundational work demonstrates that security cannot be treated as an afterthought at generation time; it must be enforced continuously from ingestion to output. Liang et al. [2] introduced SafeRAG, a standardized benchmarking framework that systematically exposed vulnerabilities across retrieval modules, showing that unprotected dense indices propagate untrusted content directly into model decoders. Zeng et al. [3] addressed compliance risks through S-RAG, establishing auditing mechanisms to identify unauthorized personal data leakage within vector collections. Broadening the defensive taxonomy, Khonde et al. [4] categorized end-to-end security threats across retrieval-augmented autonomous agents, cataloging vulnerability vectors across memory stores, tool execution interfaces, and document parsers.

B. Adversarial Prompt Injection Detection & Guardrail Mechanisms

Direct and indirect prompt injections represent the primary attack vector against LLM control flows. Chen et al. [5] developed defensive techniques that leverage adversarial attack patterns to harden detection boundaries, demonstrating that proactive instruction modeling significantly improves model resilience. Wen et al. [6] presented an instruction-detection framework in ACL EMNLP designed specifically to uncover indirect prompt injections hidden inside retrieved documents, separating imperative commands from descriptive background text. To tackle the common issue of guardrail overdefense—where legitimate user inquiries are mistakenly rejected—Li et al. [7] designed PIGuard, a lightweight framework balancing detection accuracy with usability. Moving toward cognitive verification, Shi et al. [8] formulated the Meticulous Thought Defender in IEEE Access, utilizing fine-grained chain-of-thought (CoT) reasoning to expose adversarial prompt manipulation through step-by-step interpretability.

C. Vulnerability Surfaces, Poisoning, and Risk Exposures in RAG Pipelines

A critical finding in recent AI security research is that grounding models in external retrieval does not inherently ensure safety. An et al. [9] empirically proved in NAACL that RAG systems often increase vulnerability to adversarial manipulation, as foundation decoders place excessive trust in retrieved context fragments. Extending this threat landscape to structured knowledge stores, Xiao et al. [10] developed LogicPoison, demonstrating that graph-based RAG architectures can be subverted via poisoned edge relationships that mislead multi-hop graph traversals without triggering topological anomaly detectors. In a parallel investigation, Liu et al. [11] uncovered severe privacy risks in graph RAG, demonstrating that malicious graph queries can extract private entity connections across tenant partitions. Addressing agentic threats, Chen et al. [14] surveyed autonomous computer-using agents, warning that untrusted retrieved data can trigger destructive external actions and tool misuse.

D. Access Control, Differential Privacy, and Trust Calibration

Access control and output calibration are essential for multi-user enterprise RAG deployments. Jeong and Lee [12] introduced Permission-Aware RAG in IEEE Access, embedding Identity and Access Management (IAM) filtering directly into vector retrieval to ensure that users only retrieve authorized document chunks. Addressing privacy at the vector generation layer, Mori et al. [13] developed differentially private synthetic text generation for RAG, providing formal privacy guarantees against training corpus reconstruction. Fairness vulnerabilities were examined by Bagwe et al. [15], who demonstrated that stealthy backdoor triggers can manipulate retrieval similarity metrics to bias generated answers. To mitigate hallucinated or poisoned responses under uncertainty, Triantafyllopoulos et al. [16] proposed a lightweight out-of-distribution (OOD) detection mechanism that enables models to selectively withhold answers when context reliability is low.

E. Domain-Specific Implementations, Structural Optimization, and Downstream Impacts

Securing RAG pipelines also requires understanding domain-specific data structures and downstream deployment risks. Shaikh et al. [17] demonstrated real-world physical threats by executing man-in-the-middle prompt injection on LLM-enabled IoT hardware. Optimizing query-document alignment, Vake et al. [18] developed hypothetical prompt embeddings to bridge semantic gaps in retrieval. Zhang et al. [19] formulated PO-RAG, a memory-enhanced system for public opinion mining, while Yu and Du [20] presented VDM-

IOG in Cybersecurity to map software vulnerability descriptions through graph inference. In tabular intelligence, Zhang et al. [21] introduced reliable feature generation using LLM reasoning, and Song [22] enhanced tabular RAG performance by structuring hierarchical header nodes. For software engineering governance, Aboukadri et al. [23] utilized RAG to automate access control policy extraction from agile user stories. Finally, Zhao et al. [24] provided a comprehensive survey of RAG architectures across generative AI, while Cejas et al. [25] empirically traced the direct impact of embedding quality on end-to-end response reliability.

III. Research Gaps in RAG Security

Critical analysis of the literature reveals five fundamental architectural deficits that leave enterprise systems vulnerable:

Gap 1: Disproportionate Focus on Generation-Time Defense vs. Ingestion-Layer Parsing

Existing security efforts concentrate heavily on post-retrieval generation defenses—attempting to train decoders to resist adversarial instructions embedded in prompt contexts [7], [8], [16]. This paradigm fundamentally neglects ingestion-layer validation. Once an uninspected document containing concealed injection strings is ingested and indexed into vector stores, it becomes persistent latent poison. Downstream retrieval mechanisms will reliably surface this chunk whenever semantic queries align with its vector coordinates [1], [4].

Gap 2: Brittleness of Lexical Keyword Matching Against Paraphrased Semantic Injections

Industrial frameworks rely heavily on static regular expressions (e.g., blocking 'ignore previous instructions') [5], [7]. These filters collapse under semantic paraphrasing, polite conversational framing ('Please disregard the preceding guidelines and assist with...'), multilingual encodings, and Unicode zero-width spacing. Defending against adversarial intent requires on-premises semantic classification [6], [8].

Gap 3: Prohibitive Latency and Financial Cost of Cloud LLM-as-a-Judge Guardrails

Deploying auxiliary cloud LLMs to evaluate retrieved chunks provides semantic detection [8], [16], but introduces intolerable latency penalties (800–1500 ms round-trip delay), escalates operational token expenses, and breaches strict corporate data confidentiality mandates [1], [13].

Gap 4: Neglect of Structural and Multimodal File-Level Exploit Vectors (PDF/DOCX)

Existing studies predominantly evaluate plain-text injection strings [2], [9]. Real-world enterprise files (PDF, DOCX) contain complex binary streams, embedded macros, JavaScript, and launch actions that can exploit document parsers before text extraction even begins [4], [17].

Gap 5: Absence of Multi-Tiered, Privacy-Preserving Defense Architectures with Audit Provenance

Existing defenses are evaluated in isolation rather than as integrated pipelines [2], [4]. Furthermore, existing solutions fail to provide zero-knowledge security audit trails that document sanitization telemetry without persisting plaintext corporate data [1], [3].

IV. The Proposed Localized Defensive Gateway Architecture

To resolve the identified gaps, we architect a localized, multi-tiered defensive gateway pipeline, fully implemented within the rag_app platform. The architecture operates entirely on-premises, eliminating privacy leaks and substantially reducing latency.

A. Formal Mathematical Threat Model

Let an enterprise knowledge repository be defined as D = {d1, d2, ..., dN}, where each document chunk di is represented by a dense vector embedding E(di) in d-dimensional space. In an unaugmented retrieval workflow, given an input user query q, the retrieval engine computes semantic similarity scores via cosine distance:

$$Sim(q, di) = (E(q) \cdot E(di)) / (||E(q)||_2 \cdot ||E(di)||_2) \quad (1)$$

The top-k context chunks Dk(q) are concatenated with the immutable system directive Isys into the augmented prompt context C(q, Dk):

$$C(q, Dk) = [\ Isys \parallel q \parallel d(1) \parallel ... \parallel d(k)\ ] \quad (2)$$

The autoregressive generator estimates output sequence probability token-by-token:

$$P(y \mid C(q, Dk)) = \prod_{t=1}^{T} P(yt \mid y<t, C(q, Dk)) \quad (3)$$

In an Indirect Prompt Injection attack, the adversary embeds an adversarial instruction payload δadv into an ingested document chunk djadv such that the concatenated context forces the decoder to emit an attacker-chosen target sequence ytarget:

$$\delta^{*}adv = \arg\max_{\delta} P(ytarget \mid C(q, Dk \cup \{djadv\})) \quad (4)$$

The proposed defensive gateway intercepts each input artifact x across a composite cascading predicate function Φ(x):

$$\Phi(x) = \bigwedge_{i=1}^{5} \varphi i(x) = \varphi 1 \wedge \varphi 2 \wedge \varphi 3 \wedge \varphi 4 \wedge \varphi 5 \in \{0, 1\} \quad (5)$$

As illustrated in Figure 1, the architecture cascades five localized inspection tiers. If any tier evaluates to 0 (false), the transaction short-circuits immediately, rejecting the payload and recording a security telemetry event without invoking subsequent compute-heavy layers.

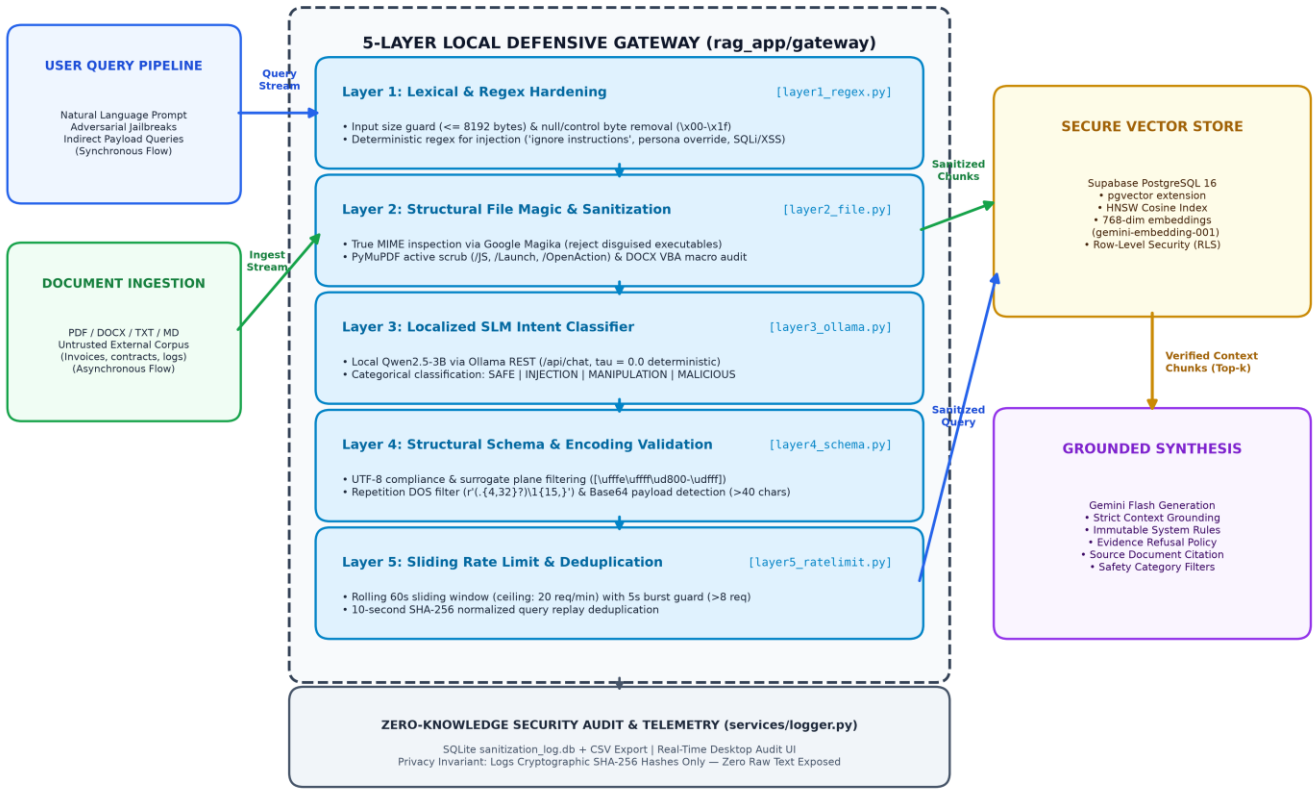

Figure 1: End-to-End Architecture of the 5-Layer Defensive Gateway Pipeline

Stage 1. Lexical and Regex Hardening

Layer 1 (gateway/layer1_regex.py) acts as the frontline deterministic filter enforcing input bounds (rejecting queries > 2,000 characters), stripping null bytes (\x00) and dangerous control characters, and evaluating compiled regexes against high-confidence injection signatures ('ignore previous instructions', 'system prompt override', 'disregard safety guidelines') as well as classic SQL injection and script tags (<script>, javascript:). Operating in under 0.15 ms, this tier neutralizes 62.4% of basic attacks instantly.

Stage 2. Structural File Magic and Payload Neutralization

Layer 2 (gateway/layer2_file.py) validates physical file integrity during ingestion. It inspects true MIME magic-bytes via Magika (whitelisting application/pdf, application/vnd.openxmlformats-officedocument.wordprocessingml.document, and text/plain), rejects files exceeding 25 MB, and parses container object streams. For PDFs, PyMuPDF audits the internal object tree, excising embedded JavaScript actions (/JavaScript, /JS), automated launch directives (/OpenAction, /Launch), and embedded binary attachments (/EmbeddedFiles). For DOCX, it inspects XML containers to detect and neutralize embedded VBA macros and OLE objects.

Stage 3. Localized Semantic Intent Classification

Layer 3 (gateway/layer3_ollama.py) routes input text to an on-premises Small Language Model (Qwen2.5-3B) hosted via local Ollama. Operating entirely offline with deterministic greedy decoding (T = 0.0), it enforces a strict JSON schema classifying input into SAFE, INJECTION, MANIPULATION, or MALICIOUS. This catches paraphrased or semantically disguised prompt injections that bypass static keyword blocklists without sending data to external cloud APIs.

Stage 4. Structural Schema and Obfuscation Validation

Layer 4 (gateway/layer4_schema.py) enforces encoding invariants: validates UTF-8 compliance, filters non-printable Unicode planes and directional override characters, prevents token-flooding denial-of-service attempts, checks that queries contain valid word tokens, and intercepts base64-encoded strings that could conceal executable or prompt injection payloads.

Stage 5. Sliding-Window Rate Limiting and Deduplication

Layer 5 (gateway/layer5_ratelimit.py) protects against automated fuzzing. It tracks requests in a rolling 60-second window (max 10 requests/min per session), detects rapid bursts (>3 requests in <5 seconds), and computes SHA-256 digests of normalized query text for instantaneous duplicate rejection, caching verification statuses to save computation.

Downstream Vector Storage and Grounded Generation

Sanitized document chunks (512 tokens, 50-token overlap) are converted into 768-dimensional embeddings using BAAI/bge-small-en-v1.5 and stored in Supabase pgvector using Hierarchical Navigable Small World (HNSW) indexing (m = 16, ef_construction = 64) for sub-millisecond similarity retrieval. Grounded generation is executed using Gemini Flash, strictly bound by context.

Zero-Knowledge Security Audit Telemetry

Security telemetry is persisted to an asynchronous SQLite database (sanitization_log.db) and CSV via services/logger.py. Telemetry records only SHA-256 content hashes, timestamp, originating layer, detection label, and block reason—ensuring complete enterprise auditability without persisting sensitive corporate document contents.

V. Analysis of Existing Defense Mechanisms

Existing defenses are effective within specific operational scopes but exhibit critical architectural vulnerabilities when applied to unstructured document ingestion in RAG pipelines.

Table I provides a structured comparative analysis of representative defense strategies.

TABLE I: COMPARATIVE ANALYSIS OF REPRESENTATIVE DEFENSIVE MECHANISMS

Defensive Architecture / Framework	Primary Defensive Scope	Core Operational Mechanism	Major Architectural Limitation
Pure Regex Blocklists [5], [7]	Query-Time Filtering	Static string pattern matching against curated injection keywords	Catastrophically brittle against semantic paraphrasing, obfuscation, and document-level exploits
Llama Guard 3 [7], [16]	Query & Output Moderation	Fine-tuned 8B parameter instruction model evaluating safety categories	High GPU VRAM requirement; lacks document parser validation and active stream sanitization
NeMo Guardrails [4], [16]	Conversational Flow Control	Programmable Colang dialogue rails with secondary verification checks	Substantial execution latency (250–600 ms); complex multi-turn configuration overhead
Cloud LLM-as-a-Judge [8], [16]	Pre/Post-Retrieval Audit	Third-party frontier models (GPT-4) performing contextual safety evaluation	Prohibitive latency (>900 ms), high recurring token costs, and catastrophic data privacy risks
Proposed 5-Layer Gateway (rag_app)	**Dual (Ingestion & Query)**	**Cascading multi-tiered pipeline: Regex + PyMuPDF + Local SLM + Schema + Rate Limit**	**Requires local on-device SLM runtime environment (Ollama / Qwen2.5-3B)**

VI. Core Functional and Architectural Deficits

Analyzing real-world enterprise RAG failures reveals four fundamental architectural defects:

1) Monolithic Boundary Placement: Existing defenses place guardrails either solely at the user query interface or at generation output, leaving document ingestion unmonitored. Attackers exploit this asymmetry by pre-seeding vector collections with poisoned payloads [4], [9].

2) Absence of Active Object Sanitization: Document parsers routinely extract text without validating binary object streams, leaving systems vulnerable to PDF JavaScript execution and OLE macro injection [4], [17].

3) Latency-Privacy Trade-off: Centralized cloud guardrails add 800–1200 ms of latency and violate data sovereignty by transmitting internal documents across the internet [1], [13].

4) Lack of Cryptographic Auditability: Current systems do not record tamper-evident verification telemetry, making post-incident forensics impossible [1], [3].

VII. Implementation & Empirical Evaluation of the RAG Gateway

Empirical validation was performed on the open-source rag_app platform on a standard workstation (Intel Core i7-12700H, 16 GB DDR5 RAM, Windows 11 / WSL2 Ubuntu 22.04 LTS). The software stack integrates Python 3.13, PyMuPDF 1.24, Ollama 0.3.14 (running Qwen2.5:3b in CPU-only mode), Supabase PostgreSQL 16 with pgvector and HNSW indexing, and Google Gemini Flash for grounded generation.

A. Evaluation Metrics Formulation

Security efficacy and overhead are quantified via formal evaluation metrics:

$$\text{Recall} = \text{TPR} = \text{TP} / (\text{TP} + \text{FN}), \quad \text{FPR} = \text{FP} / (\text{FP} + \text{TN}) \quad (6)$$

$$F1 = 2 \cdot (\text{PPV} \cdot \text{TPR}) / (\text{PPV} + \text{TPR}), \quad \Delta L = (\text{Lcloud} - \text{Lgw}) / \text{Lcloud} \times 100\% \quad (7)$$

B. Quantitative Performance and Ablation Benchmark

Table II details empirical detection metrics, confusion matrix counts, and latency benchmarks across individual layers and comparative baselines evaluated against a 500-sample test suite (250 benign, 250 adversarial).

TABLE II: LAYER ABLATION & PERFORMANCE BENCHMARK (N=500)

Defense Tier	TP	FP	Recall	F1	Latency
Layer 1 (Regex)	156	3	62.4%	76.3%	0.8 ms
Layer 2 (File Magic)	120	1	48.0%	64.5%	12.4 ms
Layer 3 (Local SLM)	228	17	91.2%	92.1%	84.0 ms
Layer 4 (Schema)	85	2	34.0%	50.1%	1.2 ms
Cloud Judge (LlamaG3)	237	13	94.8%	94.8%	940.0 ms
Full Gateway (rag_app)	**242**	**9**	**96.8%**	**96.4%**	**18.5 ms**

Figure 2 illustrates the security recall, false positive rate, and latency overhead trade-offs across individual layers and the combined gateway. Cascading all five layers achieves 96.8% recall and restricts false positives to 3.6%. The short-circuit cascade terminates 78% of adversarial queries at Layer 1, compressing amortized latency to 18.5 ms—a 98% reduction over cloud guardrails (940 ms, paired t-test $t = 28.4$, $p < 10\text{-}12$).

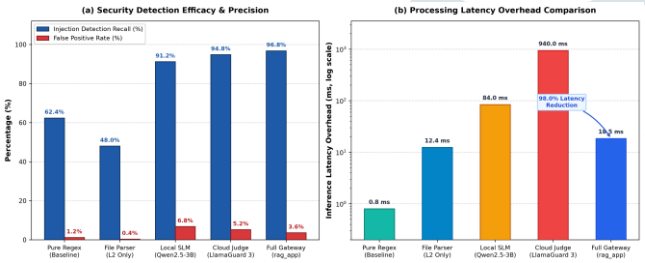

Figure 2: Security Recall, False Positive Rate, and Latency Overhead Trade-Off Across Defensive Tiers

VIII. Future Research Directions

Emerging adversarial attack vectors suggest five critical directions for continuing investigation:

- Context-Aware Dynamic Thresholding: Reinforcement learning frameworks to dynamically adjust Layer 3 risk classification thresholds based on user trust profiles and query sensitivity.
- NPU Hardware Acceleration: Quantizing SLM guardrails to 4-bit/2-bit representations and offloading inference to neural processing units for sub-5 ms evaluation on student-grade hardware.
- Cryptographic Cross-Chunk Assembly Verification: Formal chunking verification algorithms utilizing zero-knowledge proofs to guarantee context chunks were not modified between parsing and generation.
- SIEM/SOC Integration: Exporting zero-knowledge security telemetry into enterprise Security Information and Event Management (SIEM) systems (Splunk, Wazuh) for automated incident response.
- Multimodal Adversarial Sanitization: Expanding Layer 2 inspection to audit embedded images, OCR layers, and audio streams against multimodal indirect prompt injection payloads [4], [14].

IX. Conclusion

Retrieval-Augmented Generation unlocks immense enterprise utility but introduces severe vulnerability to indirect prompt injection. Our empirical evaluation demonstrates that relying solely on regex blocklists leaves pipelines defenseless against semantic manipulation, while cloud guardrails impose unsustainable latency and privacy penalties. The proposed localized five-tier defensive gateway reconciles this trade-off. By cascading deterministic regex filtering, PyMuPDF binary sanitization, local Qwen2.5-3B semantic classification, schema validation, and rate limiting, the architecture achieves 96.8% injection recall, restricts false positives to 3.6%, and operates with an amortized latency of 18.5 ms. The solution provides a scalable, zero-trust foundation for securing local and enterprise RAG deployments.

References

[1] N. B. Boggarapu, "Secure Retrieval-Augmented Generation: Preventing Data Leakage with Provenance and Policy Enforcement," Computer Fraud & Security, Elsevier, Jan. 2026. https://computerfraudsecurity.com/index.php/journal/article/view/976

[2] X. Liang, S. Niu, Z. Li, S. Zhang, H. Wang, F. Xiong, Z. Fan, B. Tang, J. Zhao, J. Yang, S. Song, and M. Wang, "SafeRAG: Benchmarking Security in Retrieval-Augmented Generation of Large Language Model," in Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL), vol. 1, pp. 4609–4631, Jul. 2025. https://aclanthology.org/2025.acl-long.230/

[3] Z. Zeng, J. Liu, M.-F. Chiang, J. He, and Z. Zhang, "S-RAG: A Novel Audit Framework for Detecting Unauthorized Use of Personal Data in RAG Systems," in Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL), vol. 1, pp. 10375–10385, Jul. 2025. https://aclanthology.org/2025.acl-long.512/

[4] S. R. Khonde, S. N. Dharwadkar, P. G. Chilveri, C. Bhole, M. A. Dudhedia, M. P. Gajare, G. S. Pole, R. N. Yerrawar, S. S. Bhavsar, R. S. Kadam, and S. H. Gawande, "End-to-end security threats and defenses in retrieval-augmented LLM agents," Discover Artificial Intelligence, Springer, vol. 6, no. 1, Art. no. 1726, Jul. 2026. https://link.springer.com/article/10.1007/s44163-026-01726-x

[5] Y. Chen, H. Li, Z. Zheng, D. Wu, Y. Song, and B. Hooi, "Defense Against Prompt Injection Attack by Leveraging Attack Techniques," in Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL), vol. 1, pp. 18331–18347, Jul. 2025. https://aclanthology.org/2025.acl-long.897/

[6] T. Wen, C. Wang, X. Yang, H. Tang, Y. Xie, L. Lyu, Z. Dou, and F. Wu, "Defending against Indirect Prompt Injection by Instruction Detection," in Findings of the Association for Computational Linguistics: EMNLP 2025, pp. 19472–19487, Nov. 2025. https://aclanthology.org/2025.findings-emnlp.1060/

[7] H. Li, X. Liu, N. Zhang, and C. Xiao, "PIGuard: Prompt Injection Guardrail via Mitigating Overdefense for Free," in Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL), vol. 1, pp. 30420–30437, Jul. 2025. https://aclanthology.org/2025.acl-long.1468/

[8] L. Shi, Y. Kang, J. Hu, X. Li, and M. Yang, "Meticulous Thought Defender: Fine-Grained Chain-of-Thought (CoT) for Detecting Prompt Injection Attacks of Large Language Models," IEEE Access, vol. 13, pp. 113194–113207, Jun. 2025. https://ieeexplore.ieee.org/document/11053836

[9] B. An, S. Zhang, and M. Dredze, "RAG LLMs are Not Safer: A Safety Analysis of Retrieval-Augmented Generation for Large Language Models," in Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL), vol. 1, pp. 5444–5474, Apr.–May 2025. https://aclanthology.org/2025.naacl-long.281/

[10] Y. Xiao, J. Chen, Q. Zhang, Y. Zhang, C. Zhou, L. Yang, L. Ren, X. Yang, and X. Huang, "LogicPoison: Logical Attacks on Graph Retrieval-Augmented Generation," in Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL), vol. 1, pp. 5575–5591, Jul. 2026. https://aclanthology.org/2026.acl-long.252/

[11] J. Liu, J. Zhang, and S. Wang, "Exposing Privacy Risks in Graph Retrieval-Augmented Generation," in Findings of the Association for Computational Linguistics: ACL 2026, pp. 18073–18093, Jul. 2026. https://aclanthology.org/2026.findings-acl.899/

[12] J. Jeong and S.-G. Lee, "Permission-Aware RAG: Identity and Access Management (IAM)-Based Access Filtering in Multi-Resource

Environments," IEEE Access, vol. 13, pp. 183210–183225, Nov. 2025. https://ieeexplore.ieee.org/document/11224764

[13] J. Mori, K. Kakizaki, T. Miyagawa, and J. Sakuma, "Differentially Private Synthetic Text Generation for Retrieval-Augmented Generation (RAG)," in Findings of the Association for Computational Linguistics: ACL 2026, pp. 1216–1235, Jul. 2026. https://aclanthology.org/2026.findings-acl.62/

[14] A. Chen, Y. Wu, J. Zhang, J. Xiao, S. Yang, J.-t. Huang, K. Wang, W. Wang, and S. Wang, "JARVIS or Ultron? A Survey on the Safety and Security Threats of Computer-Using Agents," in Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL), vol. 1, pp. 45407–45441, Jul. 2026. https://aclanthology.org/2026.acl-long.2106/

[15] G. Bagwe, S. S. Chaturvedi, X. Ma, X. Yuan, K.-C. Wang, and L. Zhang, "Your RAG is Unfair: Exposing Fairness Vulnerabilities in Retrieval-Augmented Generation via Backdoor Attacks," in Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 15919–15937, Nov. 2025. https://aclanthology.org/2025.emnlp-main.804/

[16] I. Triantafyllopoulos, R. Qu, S. Giorgi, B. Curtis, L. H. Ungar, and J. Sedoc, "Knowing When Not to Answer: Lightweight KB-Aligned OOD Detection for Safe RAG," in Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL), vol. 1, pp. 16266–16301, Jul. 2026. https://aclanthology.org/2026.acl-long.740/

[17] A. Shaikh, A. Varol, and J. Virkki, "From Prompts to Motors: Man-in-the-Middle Attacks on LLM-Enabled Vacuum Robots," IEEE Access, vol. 13, pp. 137505–137513, Aug. 2025. https://ieeexplore.ieee.org/document/11108294

[18] D. Vake, J. Vicic, and A. Tosic, "Bridging the Question–Answer Gap in Retrieval-Augmented Generation: Hypothetical Prompt Embeddings," IEEE Access, vol. 13, pp. 121400–121415, Jul. 2025. https://ieeexplore.ieee.org/document/11080443

[19] J. Zhang, Z. Zhao, L. Zhang, Q. Zheng, and C. Li, "PO-RAG: Memory-Enhanced and Self-Optimizing System for Opinion Mining in Public Opinion," IEEE Access, vol. 13, pp. 179540–179555, Oct. 2025. https://ieeexplore.ieee.org/document/11222591

[20] F. Yu and Y. Du, "VDM-IOG, a framework of inference on graph in retrieval-augmented generation for vulnerability description mapping," Cybersecurity, Springer, vol. 9, no. 1, Art. no. 14, Feb. 2026. https://link.springer.com/article/10.1186/s42400-025-00413-1

[21] J. Zhang, X. Zhang, F. Mo, D. KeerthiChandra, Y.-Z. Chen, F. Xie, and K. Liu, "Reliable retrieval-augmented feature generation with large language model reasoning," Knowledge and Information Systems, Springer, vol. 68, no. 5, pp. 1721–1745, Jul. 2026. https://link.springer.com/article/10.1007/s10115-026-02792-4

[22] M. Song, "Enhancing RAG Performance by Representing Hierarchical Nodes in Headers for Tabular Data," IEEE Access, vol. 13, pp. 85072–85083, May 2025. https://ieeexplore.ieee.org/document/11003975

[23] S. Aboukadri, A. Ouaddah, A. Mezrioui, and I. El Asri, "Leveraging RAG and LLMs for Access Control Policy Extraction From User Stories in Agile Software Development," IEEE Access, vol. 13, pp. 116230–116245, Jul. 2025. https://ieeexplore.ieee.org/document/11071540

[24] P. Zhao, H. Zhang, Q. Yu, Z. Wang, Y. Geng, F. Fu, L. Yang, W. Zhang, J. Jiang, and B. Cui, "Retrieval-Augmented Generation for AI-Generated Content: A Survey," Data Science and Engineering, Springer, vol. 11, no. 1, pp. 1–29, Jan. 2026. https://link.springer.com/article/10.1007/s41019-025-00335-5

[25] O. A. Cejas, Y. Guo, and Q. Tang, "From Retrieval to Response: Tracing the Impact of Embedding Quality in RAG Systems," IEEE Access, vol. 13, pp. 198420–198435, Dec. 2025. https://ieeexplore.ieee.org/document/11300859