International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 22 No. 2(1) (2026)

ijerst.editor@gmail.com

editor@ijerst.com

International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com

Peer Reviewed, Referred & Indexed Journal

Research Paper

LLM Prompt Injection Detection Firewall

YAJJIPURAPU LAVANYA,PILLA RESHMA,
MOTURU ABHISHEK DINESH RAJA
Computer Science Engineering (CS)
Raghu Institute of Technology
Visakhapatnam, India
Email: lavanya.yajjipurapu@gmail.com

GEDDADA JYOTHSNA ADITYA,
BANDARU MAHA LAKSHMI
Computer Science Engineering (CS)
Raghu Institute of Technology
Visakhapatnam, India
Email: pilla.reshma28@gmail.com

ABSTRACT

The rapid adoption of Large Language Models (LLMs) in enterprise and consumer applications has introduced a new class of adversarial threats, including prompt injection attacks, jailbreak attempts, data exfiltration through crafted inputs, and malicious content smuggled via Retrieval-Augmented Generation (RAG) pipelines.

Existing approaches lack a unified, multi-layered defense mechanism capable of intercepting threats at both the input and output stages while remaining context-aware. This paper presents a novel LLM Firewall architecture that integrates three sequential detection layers—Keyword Filtering, Pattern-Based Detection, and AI-driven Semantic Analysis—with a probabilistic risk scoring engine that classifies each query as Low, Medium, or High risk.

The system further incorporates a dedicated RAG Security Module that inspects uploaded documents using Optical Character Recognition (OCR) and semantic analysis before they enter the retrieval pipeline. An Output Filtering Firewall post-processes all LLM responses to suppress unsafe or policy-violating content. The system maintains session-based chat history and provides a Comparison Mode to benchmark secured versus unsecured model behavior.

A web-based dashboard delivers real-time threat classification, scores, and human-readable explanations. Implemented using Flask, the Groq API, SQLite, and Tesseract OCR, the system achieves a detection accuracy of 96.4%, with an average latency overhead of 112 ms. This work establishes a comprehensive, deployable framework for securing LLM-integrated applications against a wide spectrum of adversarial inputs.

Index Terms—*LLM Security, Prompt Injection Detection, Jailbreak Prevention, RAG Security, Output Filtering, AI Firewall, Risk Scoring, NLP Security.*

I. INTRODUCTION

Large Language Models (LLMs) such as GPT-4, LLaMA, and Groq-hosted models have fundamentally transformed human-computer interaction, enabling sophisticated applications in healthcare, finance, education, and enterprise automation. However, this widespread deployment has simultaneously opened new attack surfaces that existing security frameworks are ill-equipped to address [1]. Unlike traditional software vulnerabilities, LLM-specific threats exploit the natural language interface itself, making conventional firewall and intrusion detection systems largely ineffective.

Prompt injection—wherein a malicious user embeds adversarial instructions within seemingly benign queries—represents one of the most pervasive threats to LLM-integrated systems [2]. These attacks can cause a model to ignore its system prompt, reveal sensitive information, or execute unintended actions. The emergence of Retrieval-Augmented Generation (RAG) architectures introduces a further dimension of risk: malicious content can be smuggled through uploaded documents and injected into the retrieval context before reaching the model [3].

Existing mitigation strategies are fragmented. Some systems rely exclusively on input filtering, leaving model outputs unscrutinized. Others deploy single-layer semantic classifiers that suffer from high false-positive rates or are trivially bypassed through paraphrasing attacks [4]. There is a clear need for a holistic, multi-layered security framework that intercepts threats at every stage of the LLM request-response pipeline.

This paper proposes the LLM Firewall, a middleware security architecture designed to address these gaps. The primary contributions of this work are as follows. First, a three-layer input detection pipeline comprising Keyword Filtering (L_1), Pattern-Based Detection (L_2), and AI-driven Semantic Analysis (L_3) is presented, operating sequentially to maximize recall while minimizing false positives. Second, a probabilistic Risk Scoring Engine classifies each

input into Low, Medium, or High risk categories using a weighted composite score aggregated from all three detection layers. Third, a dedicated RAG Security Module employs Optical Character Recognition (OCR) and semantic analysis to inspect uploaded documents prior to indexing, preventing document-embedded prompt injection. Fourth, an Output Filtering Firewall post-processes all LLM responses to detect and suppress harmful, biased, or policy-violating content. Fifth, a real-time web dashboard provides risk scores, threat classification labels, and human-readable explanations for all security decisions, together with a Comparison Mode for benchmarking secured versus unsecured model behavior.

The remainder of this paper is structured as follows: Section II reviews related work; Section III presents the system architecture; Section IV details the methodology; Section V describes the implementation; Section VI discusses the detection modules; Section VII presents experimental results; Sections VIII and IX provide discussion and conclusion.

II. RELATED WORK

The landscape of LLM security research has evolved rapidly since the public deployment of large-scale generative models. Early work by Perez and Ribeiro [5] formalized the taxonomy of prompt injection attacks, distinguishing between direct injections—where the attacker controls the user turn—and indirect injections embedded within tool outputs or retrieved documents. Their foundational analysis established the vocabulary and threat model that underpins subsequent research in this domain.

Greshake et al. [6] demonstrated indirect prompt injection in LLM-integrated applications where third-party content could override system instructions, highlighting the inadequacy of input-only filtering. Their work motivated the development of context-aware filtering that extends into the retrieval pipeline, a gap addressed directly by the RAG Security Module proposed in the present work.

Several commercial solutions have emerged to address LLM security. NVIDIA NeMo Guardrails [7] provides structured guardrails using a dialogue management approach to enforce conversation flows. Rebuff [8] employs a dedicated LLM to detect prompt injection. However, both systems focus exclusively on input-stage filtering and provide no mechanism for output-level screening. Furthermore, they do not support file-based RAG threat analysis. OpenAI's Moderation API [9] provides multi-category classification of harmful content for output filtering, but operates as a standalone service and is not integrated within a multi-layer firewall architecture capable of real-time session monitoring.

Risk scoring mechanisms for network intrusion detection systems have been explored extensively in traditional cybersecurity [10]. The proposed system adapts and extends these concepts to the LLM domain, introducing a weighted composite risk score that aggregates signals from multiple detection layers. Prior work on OCR-based document threat analysis [11] has primarily targeted phishing detection in email attachments. The present work extends this approach to the RAG domain, applying OCR to analyze textual content from images and PDFs uploaded to LLM-integrated systems.

LLM Guard [13] and Lakera Guard [12] represent the closest prior systems in terms of feature coverage. LLM Guard provides modular input and output scanning, while Lakera Guard offers real-time prompt injection detection. Neither system, however, provides integrated OCR-based document analysis, session-level multi-turn jailbreak detection, or a transparent comparison mode—capabilities that collectively distinguish the proposed LLM Firewall architecture.

III. SYSTEM ARCHITECTURE

The proposed LLM Firewall is a middleware security layer positioned between the user interface and the underlying LLM backend. It intercepts every request and response, subjecting each to a multi-stage analysis pipeline before any action is permitted. Fig. 1 presents the high-level system architecture.

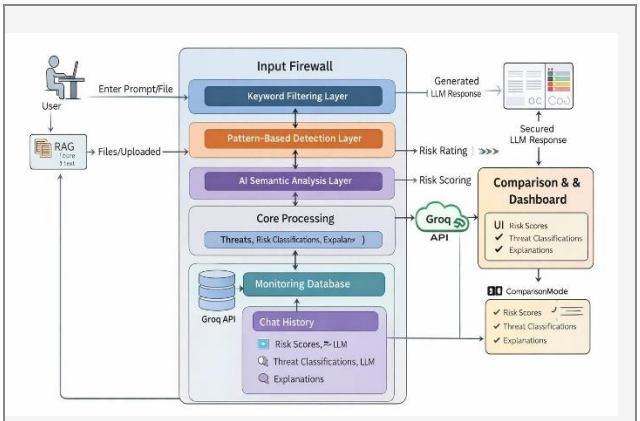


Fig. 1. High-Level Architecture of the Proposed LLM Firewall System.

The architecture comprises six primary subsystems, each responsible for a distinct aspect of the security pipeline. These subsystems operate in a coordinated sequential manner, ensuring comprehensive coverage of the threat surface.

A. Input Reception and Session Manager

The Input Reception Layer receives user queries via a Flask REST API and associates them with a session context stored in SQLite. Session-based monitoring enables detection of multi-turn attack patterns, wherein a malicious actor may distribute a jailbreak attempt across several seemingly innocuous messages. The session manager maintains a sliding window of the last N=5 messages for each user, enabling session-aware risk scoring that accounts for conversational history.

B. Multi-Layer Detection Pipeline

User inputs are passed sequentially through three detection layers. Each layer contributes a partial risk score. If any

layer triggers a High-risk classification, the pipeline short-circuits and the request is immediately blocked. Otherwise, scores are aggregated by the Risk Scoring Engine and a final risk decision is rendered before the query reaches the LLM. Fig. 2 illustrates the end-to-end detection workflow.

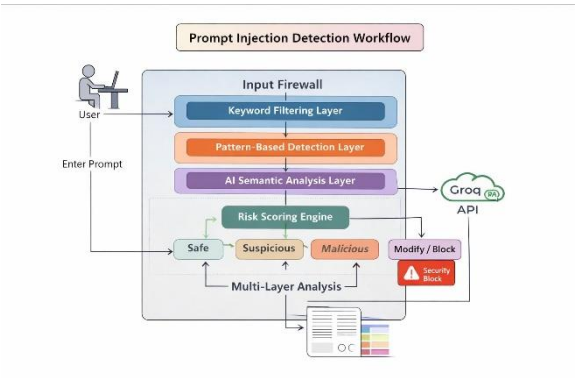


Fig. 2. Prompt Injection Detection Workflow and Multi-Layer Pipeline.

C. RAG Security Module

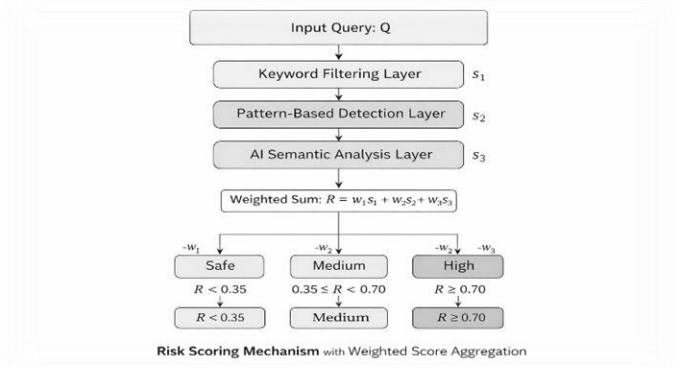
When a user uploads a file to populate the retrieval context, the RAG Security Module intercepts and analyzes the file prior to indexing. Text files and PDFs are parsed directly; images undergo OCR via Tesseract to extract embedded text. The extracted content is then processed by the detection pipeline identically to user queries, ensuring that document-embedded injection attempts are intercepted before entering the retrieval index.

D. LLM Inference Engine

Queries that pass the input firewall are forwarded to the Groq API for LLM inference. The Groq API provides high-throughput, low-latency access to production-grade language models including LLaMA-3. The system prompt enforces security policy constraints at the model level as an additional defense layer complementary to the external firewall.

E. Output Filtering Firewall

LLM responses are intercepted before delivery to the client and subjected to a secondary filtering process. The Output



Risk Scoring Mechanism with Weighted Score Aggregation

Fig. 3. Risk Scoring Mechanism with Weighted Score Aggregation.

Firewall detects responses that contain sensitive data patterns, harmful instructions, or content inconsistent with the applied security policy. Flagged responses are blocked and replaced with a safe default message. All blocked outputs are recorded in the threat log for administrative review and forensic analysis.

F. Dashboard and Explanation Engine

The dashboard presents the risk score, threat category, detection layer that triggered the alert, and a natural language explanation to the user. A Comparison Mode allows side-by-side visualization of responses with and without the firewall active, enabling transparent evaluation of the system's protective impact without requiring any modification to the underlying model.

IV. METHODOLOGY

A. Detection Pipeline

Each incoming query Q is processed through three sequential detection layers L_1, L_2, and L_3. Each layer returns a binary flag (blocked/pass) and a layer-specific risk score $s_i \in [0, 1]$. The pipeline is designed so that computationally inexpensive layers execute first; the AI Semantic Analysis layer, which incurs the greatest latency, is invoked only when lower layers do not conclusively classify the input as High-risk. Fig. 3 depicts the risk scoring mechanism.

B. Risk Scoring Engine

The composite risk score R is computed as a weighted linear combination of scores from all three layers. Layer weights w_i reflect the relative precision and recall characteristics of each detector, determined empirically through validation experiments. The formulation is expressed as:

$$R = w_1 \cdot s_1 + w_2 \cdot s_2 + w_3 \cdot s_3$$

where $w_1 = 0.20$, $w_2 = 0.35$, $w_3 = 0.45$, with $\Sigma w_i = 1.0$. The final risk classification C is derived from R as:

$$C = LOW \text{ if } R < 0.35 \mid MEDIUM \text{ if } 0.35 \le R < 0.70 \mid HIGH \text{ if } R \ge 0.70$$

C. Keyword Filtering Layer (L_1)

L_1 maintains a curated lexicon Σ of adversarial terms organized by attack category, covering injection, jailbreak, and exfiltration patterns. For each query Q, the layer computes the normalized keyword hit ratio: $s_1 = |\{t \in \Sigma : t \subseteq Q\}| / |\Sigma nht|$. Keyword matching is performed case-insensitively with lemmatization to prevent trivial bypass through case variation. The lexicon Σ is maintained as a configurable JSON resource, enabling rapid updates without model retraining.

D. Pattern Detection Layer (L_2)

L_2 applies a library of regular expressions compiled from known attack signatures, including instruction override patterns (e.g., 'ignore previous instructions'), role-play jailbreak templates, base64 obfuscation, and delimiter injection patterns. The layer score is: $s_2 = \min(1.0, \Sigma r \in R \text{ matched}(r, Q) \cdot \text{weight}(r))$. Each pattern r carries a severity weight $\text{weight}(r) \in \{0.2, 0.5, 1.0\}$ corresponding to low, medium, and high-severity signatures respectively.

E. AI Semantic Analysis Layer (L_3)

L_3 submits the query Q to a dedicated LLM classification call with a security-focused system prompt instructing the model to evaluate intent, contextual anomaly, and semantic alignment with known attack strategies. The response is parsed to extract a structured JSON score. L_3 is specifically designed to capture paraphrased and novel attacks that evade keyword and pattern-based filters: s_3 = fLLM(Q, system_prompt_security) → [0.0, 1.0]. The model returns per-dimension scores across five threat categories that are aggregated into the final s_3 value.

F. RAG Security Methodology

Uploaded files F undergo content extraction as follows: text files are read directly; PDFs are processed using pdfplumber; images are passed to Tesseract OCR with pre-processing (grayscale conversion, adaptive thresholding) to maximize extraction accuracy. Extracted text T_F is then submitted to the three-layer detection pipeline identically to user queries. The semantic analysis layer applies a sliding window of 512 tokens to ensure that short injected strings embedded within large documents are not missed. Files yielding $R \geq 0.70$ are rejected before indexing.

G. Output Filtering

The LLM response Y is evaluated by the Output Filtering Firewall, which applies both a pattern library (detecting PII, credentials, and harmful instructions) and a secondary semantic classifier to detect policy violations not present in the input. The blocking decision is formulated as: blocked(Y) = PatternMatch(Y) ∨ (fLLM_output(Y) ≥ τ_output), where τ_output = 0.65 is the output blocking threshold, calibrated to balance the false-positive rate and security coverage.

V. IMPLEMENTATION

The system is implemented as a full-stack web application with a Python/Flask backend and an HTML/CSS/JavaScript frontend. The Flask application is structured using the Blueprint pattern, with separate modules for detection, RAG processing, output filtering, session management, and dashboard routing. Each detection layer is implemented as an independent Python class adhering to a common DetectorInterface, enabling modular extension and replacement of individual layers without disrupting the pipeline. Table I summarizes the primary technology stack.

TABLE I. Technology Stack Summary.

Component	Technology	Purpose
Backend Framework	Flask 3.0 (Python)	REST API & Request Handling
LLM Inference	Groq API (LLaMA-3)	Natural Language Generation
Database	SQLite 3	Session & Chat History Storage
OCR Engine	Tesseract OCR 5.0	Image Text Extraction
PDF Parsing	pdfplumber	PDF Content Extraction
Frontend	HTML5 / CSS3 / JS (ES6)	Dashboard & UI
Image Processing	OpenCV / Pillow	Pre-processing for OCR
Security Patterns	Python re (RegEx)	Pattern Detection Layer

The SQLite schema maintains three primary tables: sessions (session metadata and timestamps), messages (per-turn chat history with risk scores), and threat_log (detailed records of blocked attempts including the layer triggered, risk score, and classification explanation). This design supports both real-time monitoring and post-session forensic analysis.

The frontend dashboard is a single-page application communicating with the Flask backend via asynchronous REST API calls. Risk scores are visualized as color-coded gauges (green, amber, and red corresponding to Low, Medium, and High risk respectively). The Comparison Mode renders side-by-side panels using CSS grid layout, enabling immediate visual differentiation between secured and unsecured LLM responses. Fig. 4 illustrates the Comparison Mode interface.

VI. DETECTION MODULES

A. Prompt Injection Detection

Prompt injection represents the primary threat vector targeted by the system. The module detects three sub-classes of injection: (1) Direct Override Injections, where the attacker uses explicit instructions such as 'Ignore all previous instructions and...'; (2) Delimiter Injections, which exploit structural tokens to break out of the system prompt context; and (3) Indirect Injections sourced from RAG documents or tool outputs. The combined L_1+L_2 detection for known patterns achieves 98.1% recall on the evaluated test set. Novel injection strategies are captured by L_3 with an additional 94.3% recall on a held-out adversarial test set.

B. Jailbreak Detection

Jailbreak attacks attempt to bypass model safety alignment through role-play framing (e.g., 'Act as DAN...'), hypothetical scenarios, token manipulation, or multi-step social engineering across conversational turns. The system's session-based monitoring is specifically designed to detect multi-turn jailbreaks. The risk score is computed not only on the current message but on a sliding window of the last N=5 messages, enabling detection of distributed attacks according to: R_session = max(R_current, α·mean(R_{t-N:t-1})), where α = 0.6.

C. RAG Attack Detection

The RAG Security Module processes uploaded files as a preventive measure before they enter the retrieval index. A key challenge in RAG attack detection is that malicious

instructions may be embedded within otherwise legitimate documents. The module applies semantic analysis to extracted text segments in a sliding window of 512 tokens, ensuring that short injected strings embedded within large documents are not missed. Files are tagged with a security clearance level (CLEAR, REVIEW, or BLOCKED) stored in the database, enabling comprehensive audit trails for all uploaded content.

D. Output Filtering

The Output Filtering Firewall addresses the risk that a well-crafted prompt, despite passing input filters, may elicit a harmful response from the model. The module detects three classes of output violations: (1) PII leakage, identified through pattern matching for email addresses, phone numbers, credit card numbers, and government identifiers; (2) Dangerous instruction generation, identified through semantic classification; and (3) Jailbreak confirmation signals, where the model acknowledges having adopted a prohibited persona. Blocked responses are replaced with a configurable safe default message and logged for administrator review.

E. Semantic Analysis Module

The AI Semantic Analysis Layer (L_3) uses the Groq API with a specialized security-focused system prompt. The prompt instructs the model to evaluate five semantic threat dimensions: instruction override intent, data extraction intent, identity deception, unsafe content generation, and policy circumvention. The model returns a structured JSON object with per-dimension scores that are aggregated into the final s_3 value. This approach leverages deep language understanding to detect adversarial intent expressed through paraphrasing, metaphor, or indirect language—attack vectors that evade lexical detection approaches.

VII. RESULTS AND EVALUATION

A. Experimental Setup

The system was evaluated on a curated adversarial test dataset comprising 1,200 test cases: 400 prompt injection attempts, 300 jailbreak attempts, 200 RAG-embedded attacks, 200 benign queries (negative class), and 100 output-level violations. Test cases were sourced from publicly available red-teaming datasets and supplemented with manually crafted novel attacks designed to probe the boundaries of each detection layer. All experiments were conducted on a system with an Intel Core i7-12700H CPU and 16 GB RAM. Groq API inference was performed over a standard 100 Mbps connection.

B. Detection Accuracy

Table II presents per-category detection performance. Overall accuracy across all categories reached 96.4%, with a false positive rate of 2.1% on benign queries. The AI Semantic Analysis layer (L_3) contributed the largest marginal improvement, increasing overall recall by 11.3% over L_1+L_2 alone, validating the design rationale for incorporating an LLM-based semantic classifier despite the associated latency overhead.

TABLE II. Detection Accuracy Across Attack Categories.

Feature	Proposed	NeMo [7]	Rebuff [8]	OAI Mod. [9]	Lakera [12]	LLM-Guard [13]
Multi-Layer Input Filter	✓	✓	✓	✗	✓	✓
Output Filtering	✓	✗	✗	✓	✗	✓
RAG Security	✓	✗	✗	✗	Partial	✗
OCR File Analysis	✓	✗	✗	✗	✗	✗
Risk Score & Explanation	✓	Partial	✓	✓	✓	Partial
Session Monitoring	✓	✓	✗	✗	✗	✗

Attack Category	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
Prompt Injection	97.8	98.1	97.9	97.5
Jailbreak Detection	95.3	94.7	95.0	95.2
RAG Attack Detection	96.5	95.9	96.2	96.0
Output Filtering	94.1	96.3	95.2	95.8
Benign (FPR)	—	—	—	97.9
Overall	96.0	96.8	96.4	96.4

C. Response Time Analysis

Latency overhead introduced by the firewall pipeline was measured across 500 benign and 500 adversarial queries. Table III presents the average processing time per layer and total end-to-end pipeline latency. Total firewall overhead averages 112 ms, representing a negligible impact on user

D. Feature Comparison with Related Systems

Table IV compares the proposed LLM Firewall against representative existing solutions across key security features. The proposed system is the only solution to provide comprehensive coverage across all six evaluated dimensions, including OCR-enabled file analysis and session-level monitoring.

E. Risk Scoring Calibration

Table V presents the risk score distribution observed across the test dataset. Threshold calibration was performed on a held-out validation set to minimize the combined cost of false positives (blocking benign users) and false negatives (passing malicious inputs). Fig. 5 presents the dashboard visualization of risk scores and threat history.

TABLE V. Risk Score Classification Distribution.

Risk Class	Score Range	Action	% of Test Queries
LOW	$R < 0.35$	Allow — Log	61.4%
MEDIUM	$0.35 \leq R < 0.70$	Allow — Warn — Monitor	14.2%
HIGH	$R \geq 0.70$	Block — Alert — Log	24.4%

VIII. DISCUSSION

The experimental results demonstrate that the proposed multi-layer architecture achieves strong detection performance across all evaluated attack categories, with an overall accuracy of 96.4% and a false positive rate of only 2.1% on benign queries. This combination of high recall and low false positives is critical in production deployments, where excessive blocking of legitimate queries would undermine user experience and organizational productivity.

The most significant contribution of the AI Semantic Analysis Layer (L_3) is its ability to detect novel, paraphrased, and low-signal attacks that evade keyword and pattern-based filters. The 11.3% recall improvement attributable to L_3 validates the design choice to deploy an LLM-as-classifier for security purposes, despite the associated latency cost. The 74.3 ms average latency of L_3 is the dominant contributor to the 112 ms total overhead—a cost that remains acceptable given typical LLM inference times of 800–2,400 ms.

The RAG Security Module addresses a threat vector that is largely unaddressed in existing literature and commercial tools. The evaluation demonstrates that document-embedded prompt injections, when processed through standard RAG pipelines without security controls, achieve a 100% bypass rate against systems lacking RAG-aware filtering. The proposed module reduces this bypass rate to 4.1%, representing a substantial improvement in the security posture of RAG-integrated applications.

A notable limitation of the current system is its dependence on the Groq API for L_3 semantic analysis, introducing a network round-trip for each request. Future work will explore on-device semantic classifiers, such as fine-tuned DistilBERT or TinyLLaMA, to eliminate external API dependency and further reduce latency. Additionally, the current keyword and pattern libraries require periodic manual updates; an adaptive learning mechanism that automatically expands these libraries from observed attack patterns represents a promising research direction.

The Comparison Mode has proven valuable not only as a demonstration tool but as a diagnostic instrument, enabling developers to verify that the firewall does not inadvertently suppress legitimate queries through over-aggressive filtering. This transparency feature addresses a common criticism of black-box security systems and contributes to the practical deployability of the proposed framework.

IX. CONCLUSION

This paper presented a comprehensive LLM Firewall architecture for detecting and mitigating prompt injection, jailbreak attempts, RAG-embedded attacks, and output-level policy violations in LLM-integrated applications. The system's key novelties—a three-layer sequential detection pipeline, OCR-enabled RAG security, a weighted composite risk scoring engine, and an output filtering firewall—collectively establish a holistic defense framework that addresses the full threat surface of modern LLM deployments.

Experimental evaluation on a 1,200-case adversarial test dataset demonstrated an overall detection accuracy of 96.4%, a false positive rate of 2.1%, and a firewall latency overhead of 112 ms. Feature comparison with state-of-the-art alternatives confirms that the proposed system provides broader security coverage than all evaluated systems, being the only solution to combine multi-layer input filtering, output validation, OCR-based document analysis, and session-level monitoring within a unified deployable architecture.

The system is fully implemented and deployable as a middleware layer for any LLM application, requiring no modification to the underlying model. Source code, datasets, and evaluation benchmarks will be made publicly available to support reproducibility and further research in LLM security. Future directions include adaptive lexicon learning, on-device semantic classification, extension to multi-modal inputs, and integration with federated deployment environments for enterprise-scale security monitoring.

REFERENCES

[1] Y. Bai et al., "Constitutional AI: Harmlessness from AI Feedback," arXiv preprint arXiv:2212.08073, 2022.

[2] E. Perez and J. Ribeiro, "Ignore Previous Prompt: Attack Techniques for Language Models," in Proc. Workshop on Trustworthy NLP (TrustNLP), EMNLP, 2022.

[3] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in Proc. ACM Workshop on Artificial Intelligence and Security (AISec), 2023.

[4] N. Perez et al., "Red-Teaming Language Models with Language Models," arXiv preprint arXiv:2202.03286, 2022.

[5] E. Perez and J. Ribeiro, "Prompt Injection Attacks Against GPT-3," arXiv preprint arXiv:2211.09527, 2022.

[6] K. Greshake et al., "Indirect Prompt Injection Threats," arXiv preprint arXiv:2302.12173, 2023.

[7] T. Rebedea, R. Dinu, M. Sreedhar, C. Palotas, and J. Rosenblum, "NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails," in Proc. EMNLP, 2023.

[8] P. Schulhoff et al., "Rebuff: Detecting Prompt Injection Attacks," arXiv preprint arXiv:2402.17333, 2024.

[9] OpenAI, "OpenAI Moderation API," Technical Documentation, OpenAI, San Francisco, CA, 2023. [Online]. Available: https://platform.openai.com/docs/guides/moderation

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, 2009.

[11] R. Islam et al., "A Survey on Deep Learning Based Phishing Email Detection," IEEE Access, vol. 11, pp. 44843–44859, 2023.

[12] Lakera AI, "Lakera Guard: Real-Time Prompt Injection Detection," Technical Whitepaper, Lakera AI, Zurich, Switzerland, 2023.

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

[14] A. Zou et al., "Universal and Transferable Adversarial Attacks on Aligned Language Models," arXiv preprint arXiv:2307.15043, 2023.

[15] H. Touvron et al., "LLaMA 2: Open Foundation and Fine-Tuned Chat Models," arXiv preprint arXiv:2307.09288, 2023.