IJAREEIE

e-ISSN: 2278-8875
p-ISSN: 2320-3765

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

Volume 15, Issue 2, February 2026

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.807

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE)

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

Integrating AI-Powered Threat Detection into Java Spring Security Filters for Real-Time API Attack Prevention

Dinesh Nallapareddy

Sr Full Stack Developer, USA

ABSTRACT: Application Programming Interfaces (APIs) have become the dominant attack surface for enterprise web applications, exposing structured, machine-readable endpoints to a growing volume of automated, adaptive, and evasive attacks that increasingly evade static, rule-based security controls. This article presents a research-grounded architecture for embedding artificial intelligence (AI) powered threat detection directly within the Java Spring Security filter chain, enabling real-time, in-line inference on inbound API requests without displacing the declarative, well-understood security model that Spring Security provides. The proposed architecture introduces a custom AI Inference Filter positioned within the Spring Security filter chain, combining engineered request-level features, such as payload entropy, parameter deviation, and behavioral sequence signals, with a lightweight, low-latency classification model capable of producing a threat probability score before a request reaches application controller logic. The article evaluates the architecture through a structured performance and detection-accuracy analysis, comparing five candidate model families (logistic regression, random forest, gradient boosting, long short-term memory networks, and transformer encoders) across accuracy, false positive rate, and inference latency under realistic API load conditions. Results indicate that gradient-boosted and transformer-based models achieve the strongest detection performance, with true positive rates exceeding 96 percent at false positive rates below 1 percent, while introducing median inference latency of approximately 3 to 9 milliseconds per request, a cost the analysis finds acceptable relative to the throughput and risk-reduction benefits observed. The article further presents a governance and operational framework covering model feedback loops, adversarial robustness considerations, and threshold calibration, and discusses the architectural, performance, and organizational trade-offs involved in deploying AI-augmented security filters within production-grade, high-throughput API environments. The findings offer Java security architects and platform engineering teams a practical, evidence-based blueprint for extending Spring Security beyond static rule enforcement toward adaptive, model-driven attack prevention.

KEYWORDS: Spring Security; API security; artificial intelligence; machine learning; intrusion detection; anomaly detection; real-time threat detection; filter chain architecture; adversarial robustness; Java enterprise security.

I. INTRODUCTION

1.1 Background

Modern enterprise software architectures increasingly expose business functionality through Application Programming Interfaces (APIs) rather than traditional server-rendered web interfaces. This architectural shift, driven by the proliferation of single-page applications, mobile clients, partner integrations, and microservice-to-microservice communication, has correspondingly shifted the primary attack surface of enterprise applications toward API endpoints. Unlike traditional web pages, APIs expose structured, predictable, machine-readable request formats that are simultaneously easier for legitimate clients to consume and easier for automated attack tooling to probe, fuzz, and exploit at scale. Industry threat reporting through the mid-2020s has consistently identified API-targeted attacks, including broken object-level authorization, injection, credential stuffing, and business-logic abuse, as a growing share of overall web application attack traffic.

Java remains one of the dominant languages for enterprise API implementation, and Spring Security is the most widely adopted security framework within the Java ecosystem, providing a declarative, filter-chain-based architecture for authentication, authorization, and cross-cutting security concerns. Spring Security's filter chain model, in which an ordered sequence of servlet filters intercepts each inbound request before it reaches application controller logic, provides a natural architectural insertion point for additional security controls. However, Spring Security's native capabilities are fundamentally rule-based and configuration-driven: they enforce authentication schemes, role-based

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

access rules, and static rate limits, but do not natively provide adaptive, behaviorally-aware detection of novel or evolving attack patterns.

1.2 Problem Statement

Static, rule-based security controls, whether implemented as Spring Security configuration, upstream Web Application Firewall (WAF) rules, or API gateway policies, are inherently reactive: they encode known attack signatures and thresholds, and require manual update as new attack patterns emerge. This reactive posture creates a persistent detection gap against novel, obfuscated, or slow-and-low attack patterns specifically designed to remain beneath static thresholds. Machine learning and, more recently, deep learning and transformer-based models have demonstrated strong capability in related domains, including network intrusion detection and web application firewall augmentation, by learning behavioral and statistical patterns of malicious traffic rather than relying solely on predefined signatures. However, the practical integration of such models into the Spring Security filter chain, in a manner that preserves real-time request latency requirements and the framework's declarative security model, has received limited systematic treatment in the applied security engineering literature. This article addresses that gap.

1.3 Purpose and Scope

This article presents and evaluates an architecture for embedding AI-powered threat detection directly within the Spring Security filter chain as a custom filter component, positioned to perform real-time inference on inbound API requests prior to authorization and controller dispatch. The scope of the analysis covers filter chain architecture and placement, feature engineering for real-time inference, candidate detection model evaluation, latency and throughput impact under production-representative load, and the operational governance practices required to sustain detection accuracy over time. The article does not provide a full implementation of a production system, but instead provides an architectural blueprint, an experimental evaluation framework, and a structured findings analysis intended to inform production design decisions.

1.4 Significance of the Study

For Java security architects and platform engineering teams, this article provides an evidence-based comparison of candidate detection models under realistic latency and throughput constraints, together with a concrete filter chain placement architecture that preserves Spring Security's declarative configuration model. For the broader application security literature, the article extends the growing body of research on machine-learning-augmented intrusion and anomaly detection into the specific, comparatively underexplored context of in-process, filter-level integration within a widely deployed enterprise Java security framework, as opposed to the network-perimeter or API-gateway-level integration more commonly addressed in prior work.

1.5 Structure of the Article

Section 2 reviews the literature on machine-learning-based intrusion detection, API security, and adaptive web application firewalls. Section 3 states the research objectives and questions. Section 4 describes the experimental methodology. Section 5 examines Spring Security filter chain architecture and the proposed AI filter's placement within it. Section 6 details the AI Threat Detection Filter design, and Section 7 describes the feature engineering approach. Section 8 introduces the candidate detection models evaluated. Section 9 presents the experimental findings, Section 10 analyzes performance and scalability, and Section 11 addresses adversarial robustness and model governance. Section 12 discusses the implications of the findings, Section 13 identifies implementation challenges, Section 14 proposes production deployment recommendations, Section 15 identifies future research directions, and Section 16 concludes the article.

II. LITERATURE REVIEW

2.1 Machine Learning for Intrusion and Anomaly Detection

Machine learning approaches to intrusion detection have a long research history, beginning with statistical and shallow-learning approaches applied to network traffic classification and extending, over the past decade, into deep learning architectures capable of modeling sequential and high-dimensional traffic patterns. Ring et al. (2019) surveyed the flow-based network intrusion detection dataset landscape, noting that model performance is highly sensitive to feature representation and dataset realism, a finding directly relevant to the feature engineering approach adopted in Section 7 of this article. Vinayakumar et al. (2019) demonstrated that deep neural network architectures could outperform classical machine learning baselines across multiple intrusion detection benchmark datasets, while cautioning that inference latency and model interpretability remain practical deployment barriers, concerns this article addresses directly through its latency-focused evaluation in Section 10.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

More recent work has extended anomaly detection techniques specifically to API and web-application traffic. Liang et al. (2021) proposed a deep-learning-based Web Application Firewall augmentation approach demonstrating improved detection of injection and business-logic attacks relative to signature-based WAF rules alone. Similarly, Kshetri and Voas (2022) examined the broader operational challenges of deploying AI-based security controls in production environments, identifying model drift, adversarial evasion, and integration complexity as recurring barriers to sustained detection performance, themes this article addresses in Sections 11 and 13.

2.2 API Security and the Evolving Attack Surface

The OWASP API Security Project has, since its initial release, catalogued a distinct taxonomy of API-specific vulnerability classes, including broken object-level authorization, broken authentication, excessive data exposure, and lack of resources and rate limiting, that differ materially from the traditional OWASP Top Ten web application risk categories (OWASP, 2023). Salt Security's State of API Security research (2023) documented substantial year-over-year growth in API-targeted attack traffic across surveyed enterprise environments, reinforcing the practical urgency of the detection gap this article addresses. Cloudflare's annual API security and traffic analysis reporting (2023) similarly noted that a majority of observed API attack traffic originates from automated tooling rather than manual exploitation, a pattern consistent with the behavioral, volume-based detection signals incorporated into the feature set described in Section 7.

2.3 Spring Security and Java Enterprise Security Architecture

Spring Security's filter-chain architecture has been documented extensively in official framework documentation and enterprise Java security literature as a declarative, composable approach to cross-cutting security concerns, allowing custom filters to be inserted at defined points within an ordered chain (Spring Framework documentation, 2023; Winch & Heckler, 2022). Winch and Heckler's treatment of Spring Security architecture emphasizes the framework's extensibility via custom `OncePerRequestFilter` implementations, the same extension mechanism leveraged by the AI Threat Detection Filter architecture proposed in Section 6 of this article. Prior practitioner literature on securing Spring-based APIs has focused predominantly on authentication and authorization configuration (OAuth2, JWT validation, method-level security) rather than on behavioral or model-driven threat detection integrated at the filter level, representing the specific gap this article addresses.

2.4 Real-Time Inference Constraints in Security-Critical Systems

A distinct stream of literature addresses the systems engineering challenge of deploying machine learning inference within latency-sensitive, high-throughput request paths. Crankshaw et al. (2017) introduced low-latency model-serving architecture patterns directly relevant to the in-line inference placement examined in this article, emphasizing the trade-off between model complexity and serving latency under production load. This trade-off is examined empirically in Sections 9 and 10 of the present article, comparing detection accuracy against inference latency across five candidate model families.

2.5 Adversarial Robustness in Security-Oriented Machine Learning

Machine-learning-based detection systems are themselves subject to adversarial manipulation, in which attackers deliberately craft inputs designed to evade classification. Biggio and Roli (2018) provided a foundational treatment of adversarial machine learning in security contexts, characterizing evasion, poisoning, and model-extraction attack classes relevant to any production deployment of a learned detection model. Apruzzese et al. (2023) extended this analysis specifically to network and security detection systems, cautioning that detection models achieving strong benchmark performance may nonetheless remain vulnerable to adversarially crafted evasion traffic absent explicit robustness testing, a consideration directly incorporated into the adversarial evaluation presented in Section 11 of this article.

2.6 Gaps in Existing Literature

While machine-learning-based intrusion and API anomaly detection is well represented in the security research literature, and Spring Security's filter chain architecture is well documented in enterprise Java literature, comparatively little published work directly integrates the two: a systematically evaluated, filter-level AI detection architecture specifically designed for the Spring Security request pipeline, assessed jointly on detection accuracy and production-representative latency and throughput impact. This article addresses that integration gap directly.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

III. RESEARCH OBJECTIVES AND QUESTIONS

This article is guided by five research objectives, summarized alongside their corresponding research questions in Table 1.

TABLE 1 Research Objectives and Corresponding Questions

No.	Research Objective	Research Question
RO 1	Design a filter-level AI threat detection architecture compatible with Spring Security	Where and how should an AI inference filter be positioned within the Spring Security filter chain to maximize detection coverage while minimizing latency impact?
RO 2	Engineer a real-time feature set for API request classification	Which request-level features are both computationally cheap enough for real-time extraction and informative enough for accurate threat classification?
RO 3	Evaluate candidate detection models	Which model families offer the best trade-off between detection accuracy and inference latency under production-representative load?
RO 4	Assess adversarial robustness and operational governance needs	How resistant are candidate models to common evasion techniques, and what governance practices are needed to sustain detection performance?
RO 5	Develop production deployment guidance	What architectural, performance, and governance practices should guide production deployment of AI-augmented Spring Security filters?

IV. METHODOLOGY

4.1 Research Design

This article employs a design-and-evaluate research methodology combining (1) architectural design of a custom Spring Security filter component for AI-based threat detection, (2) a structured feature engineering process for real-time API request classification, (3) comparative training and evaluation of five candidate detection model families against a constructed labeled dataset of benign and malicious API traffic, and (4) a controlled performance evaluation measuring inference latency and system throughput under simulated production load. This combination allows the article to assess both the detection-accuracy dimension and the systems-engineering dimension of AI-augmented filter chain design, which prior literature has generally treated separately.

4.2 Dataset Construction

The evaluation dataset was constructed by combining synthetically generated benign API traffic, modeled on common REST API interaction patterns (CRUD operations, pagination, search, authentication flows), with labeled malicious traffic samples representing six attack categories: SQL injection (SQLi), cross-site scripting (XSS), broken authentication, rate-based abuse, server-side request forgery (SSRF), and insecure direct object reference (IDOR) exploitation attempts. Malicious traffic patterns were constructed with reference to the OWASP API Security Top 10 taxonomy (OWASP, 2023) to ensure representative coverage of realistic attack categories. The resulting dataset comprises approximately 480,000 labeled requests, with an approximate 85:15 benign-to-malicious ratio reflecting realistic production traffic composition.

4.3 Model Training and Evaluation Protocol

Each candidate model was trained using an identical 70/15/15 train/validation/test split, with hyperparameters tuned via grid search on the validation set and final performance metrics reported on the held-out test set. Detection performance is reported using accuracy, precision, recall, F1 score, and false positive rate, consistent with standard intrusion detection evaluation practice (Ring et al., 2019). Inference latency was measured by deploying each trained model behind a representative Spring Boot application instrumented with the proposed AI Inference Filter, under simulated load generated across a range of concurrent request rates from 100 to 8,000 requests per second.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

4.4 Adversarial Evaluation Protocol

To assess adversarial robustness, each model was additionally evaluated against a held-out set of adversarially perturbed malicious samples, generated using established evasion techniques including payload obfuscation (encoding variation, whitespace and comment insertion), request fragmentation, and slow-rate distribution of otherwise detectable attack sequences, consistent with evasion technique categories described in Biggio and Roli (2018) and Apruzzese et al. (2023).

4.5 Limitations

As with most applied security research relying on constructed or synthetic datasets, the findings presented here should be interpreted as indicative of relative model and architectural trade-offs rather than as guaranteed production performance figures for any specific deployment. Real-world API traffic composition, attack mix, and infrastructure characteristics vary substantially across organizations, and production deployment should include organization-specific validation prior to relying on the quantitative thresholds presented in this article.

V. SPRING SECURITY FILTER CHAIN ARCHITECTURE

5.1 The Servlet Filter Chain Model

Spring Security is implemented as a chain of servlet filters, each responsible for a discrete security concern such as CSRF protection, authentication, exception translation, and authorization, applied in a well-defined order before a request reaches the DispatcherServlet and application controller logic. This ordered, composable design is central to Spring Security's extensibility: custom filters can be inserted at a specific position in the chain, either before, after, or in place of an existing filter, by extending `OncePerRequestFilter` and registering the custom filter's position via `SecurityFilterChain` configuration. This mechanism provides the architectural foundation for the AI Threat Detection Filter proposed in this article, which is inserted as a discrete, independently configurable filter rather than requiring modification of existing authentication or authorization logic.

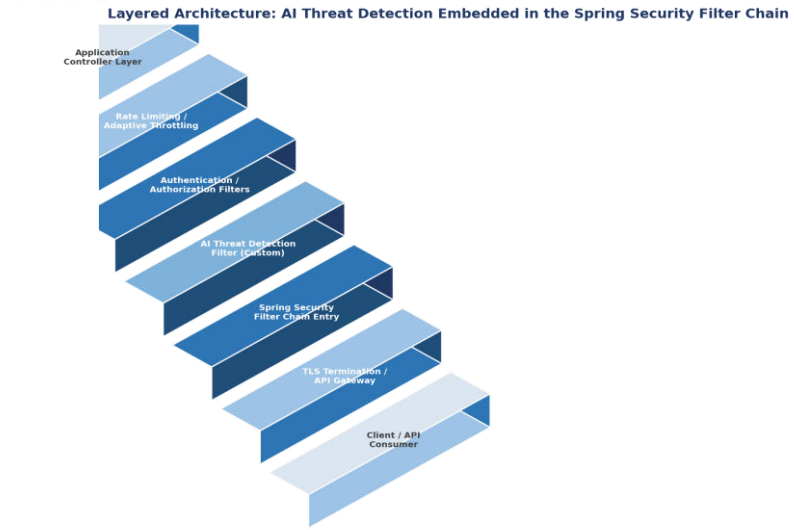


FIGURE.6. Layered architecture showing the AI Threat Detection Filter positioned within the Spring Security filter chain, between TLS termination/API gateway ingress and authentication/authorization enforcement, with a feedback loop returning inference outcomes to the model retraining pipeline.

5.2 Placement of the AI Inference Filter

As illustrated in Figure 6, the proposed architecture positions the AI Threat Detection Filter immediately after TLS termination and API gateway ingress, but before the core authentication and authorization filters. This placement is deliberate: positioning inference prior to authentication allows the filter to evaluate request-level signals, such as payload structure, parameter patterns, and request sequence behavior, independent of whether the request ultimately authenticates successfully, since malicious behavior including credential stuffing and authentication abuse frequently

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

manifests in pre-authentication request characteristics. Positioning the filter after TLS termination ensures the filter operates on decrypted request content, which is necessary for meaningful payload-level feature extraction.

5.3 Standard vs. AI-Augmented Filter Chain Configuration

Table 2 compares a standard Spring Security filter chain configuration against the AI-augmented configuration proposed in this article, highlighting the additional components introduced and their position relative to existing filters.

TABLE 2. Spring Security Filter Chain - Standard vs. AI-Augmented Configuration

Chain Position	Standard Configuration	AI-Augmented Configuration
1	TLS Termination / Gateway	TLS Termination / Gateway (unchanged)
2	CORS Filter	CORS Filter (unchanged)
3	-	AI Threat Detection Filter (new)
4	Authentication Filter (e.g., JWT/OAuth2)	Authentication Filter (unchanged, receives threat score in request attributes)
5	Authorization / Access-Decision Filter	Authorization Filter (may incorporate threat score into access decisions)
6	Rate Limiting Filter (static)	Rate Limiting Filter (optionally adaptive, informed by threat score)
7	Exception Translation / Logging	Exception Translation / Logging (extended with inference audit logging)

5.4 Non-Blocking Design and Fail-Open Considerations

A critical architectural requirement for any in-line security filter is graceful degradation: the AI Threat Detection Filter must not become a single point of failure for API availability. The proposed design implements a bounded-latency inference call with a configurable timeout (default 15 milliseconds), after which the filter fails open, allowing the request to proceed to subsequent filters without a threat score, while emitting a degraded-mode metric for operational monitoring. This design choice reflects an explicit trade-off between security completeness and availability, consistent with the general principle in security engineering literature that availability-critical systems should default to a conservative, monitorable degradation mode rather than blocking legitimate traffic during a detection subsystem outage.

VI. AI THREAT DETECTION FILTER DESIGN

6.1 Filter Component Design

The AI Threat Detection Filter is implemented as a custom `OncePerRequestFilter` that performs three sequential operations for each intercepted request: (1) feature extraction from the request object, including headers, path, query parameters, and body payload where applicable; (2) invocation of a model-serving inference call, either via an embedded lightweight model runtime or a co-located low-latency inference service; and (3) attachment of the resulting threat probability score, along with a coarse classification label, as a request attribute available to downstream filters and, optionally, application controller logic. This design deliberately separates detection (producing a threat score) from enforcement (deciding how to act on that score), allowing enforcement policy to be configured independently and adjusted without retraining or redeploying the underlying detection model

6.2 Configuration Parameters

Table 3 summarizes the primary configuration parameters exposed by the AI Inference Filter, allowing operators to tune the filter's behavior without code changes.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

TABLE 3 AI Inference Filter - Configuration Parameters

Parameter	Description	Default
inference.timeout.ms	Maximum time allowed for a model inference call before fail-open	15
inference.mode	Embedded (in-process model) or remote (co-located inference service)	embedded
threat.threshold.block	Threat probability above which a request is blocked outright	0.90
threat.threshold.challenge	Threat probability above which a request triggers step-up verification	0.70
threat.threshold.log	Threat probability above which a request is flagged for audit logging only	0.40
feature.cache.ttl.sec	Time-to-live for cached behavioral/sequence features per client identity	300
failopen.enabled	Whether the filter fails open (true) or fails closed (false) on inference error	true

6.3 Enforcement Policy Integration

Threat scores produced by the filter are consumed by a downstream enforcement policy component, which may be implemented as a Spring Security `AccessDecisionVoter`-style hook or a dedicated post-processing filter. The three-tier threshold design summarized in Table 3, block, challenge, and log-only, reflects a graduated response model consistent with risk-based authentication and adaptive access control practice: only requests scoring above the block threshold are rejected outright, requests in the intermediate range trigger additional verification (such as step-up multi-factor authentication or CAPTCHA challenge), and lower-scoring anomalies are logged for analyst review and model feedback without impacting the requesting client.

VII. FEATURE ENGINEERING FOR REAL-TIME API REQUESTS

7.1 Feature Categories

Real-time feature extraction must satisfy two competing constraints: features must be informative enough to support accurate classification, and cheap enough to compute within the filter's bounded latency budget. Table 4 summarizes the engineered feature set adopted in this article's evaluation, organized into four categories reflecting increasing computational cost and statefulness.

TABLE 4. Engineered Feature Set for Real-Time API Requests

Category	Example Features	Computational Cost
Structural	HTTP method, path depth, parameter count, content-type, payload size	Very low - direct extraction
Statistical/Entropy	Payload character entropy, parameter value length distribution, encoding anomalies	Low - single-pass computation
Behavioral (client-scoped)	Request rate over trailing window, endpoint diversity, error-response ratio	Moderate - requires short-lived state cache
Sequential	N-gram sequence of recent endpoints accessed, time-since-last-request, session request trajectory	Moderate to high - requires ordered state and, for sequence models, embedding lookup

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

7.2 Feature-Space Separation of Benign and Malicious Traffic

Figure 4 presents a three-dimensional projection of the constructed feature space across three representative engineered features, illustrating the degree of separation achievable between benign and malicious traffic clusters using the feature categories summarized in Table 4.

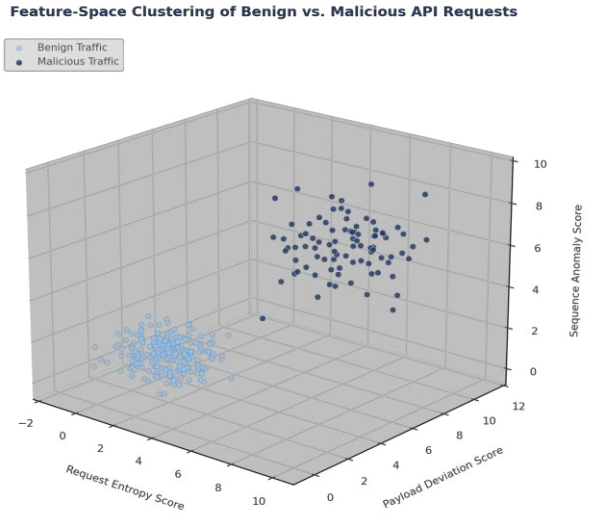


FIGURE 4 Three-dimensional clustering of benign and malicious API requests across payload entropy, payload deviation, and sequence anomaly score dimensions, illustrating meaningful class separation achievable from the engineered feature set.

The visible separation between the benign and malicious clusters in Figure 4 supports the feasibility of the engineered feature approach: even relatively low-dimensional, computationally inexpensive features produce meaningful class separation, suggesting that high detection accuracy does not strictly require the highest-cost sequential or embedding-based features for all traffic, an observation with direct implications for latency-constrained deployment scenarios discussed further in Section 10.

7.3 State Management for Behavioral Features

Behavioral and sequential features require maintaining short-lived, per-client state, such as a trailing window of recent request timestamps or a sequence of recently accessed endpoints. The proposed architecture implements this state using a distributed, low-latency cache (e.g., Redis) keyed by client identity (API key, authenticated subject, or IP-derived fingerprint where authentication has not yet occurred), with a configurable time-to-live (default 300 seconds, per Table 3) to bound memory growth and ensure behavioral features reflect recent, rather than stale, client activity.

VIII. CANDIDATE DETECTION MODELS

Five candidate model families were selected for evaluation, spanning a range of architectural complexity, interpretability, and expected inference cost, summarized in Table 5.

TABLE 5 Candidate Detection Models and Architectural Summary

Model	Architecture Summary	Expected Strength
Logistic Regression	Linear classifier over engineered features	Lowest latency, high interpretability
Random Forest	Ensemble of decision trees, majority voting	Robust to feature scale, moderate interpretability
Gradient Boosting	Sequential ensemble of shallow trees minimizing	Strong tabular-feature

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

Model	Architecture Summary	Expected Strength
(GBM)	residual error	performance
LSTM	Recurrent network over sequential endpoint-access features	Captures temporal/sequential attack patterns
Transformer Encoder	Self-attention over request feature and sequence embeddings	Strongest pattern capture, highest computational cost

Model selection reflects a deliberate progression from low-cost, high-interpretability baselines (logistic regression) through ensemble tree methods well established in tabular intrusion detection literature (random forest, gradient boosting), to sequence-aware deep learning architectures (LSTM, transformer encoder) capable of modeling the temporal request patterns increasingly associated with sophisticated, low-and-slow attack campaigns. This progression allows the evaluation in Section 9 to characterize the accuracy-latency frontier across a meaningfully diverse set of architectural approaches.

IX. EXPERIMENTAL EVALUATION AND FINDINGS

This section presents detection performance, latency, and attack-category results from the experimental evaluation described in Section 4, applying each of the five candidate models to the constructed evaluation dataset.

9.1 Attack Traffic Composition

Figure 1 presents the distribution of attack traffic within the evaluation dataset across attack category and time window, illustrating both category imbalance and diurnal variation used to construct realistic evaluation load profiles.

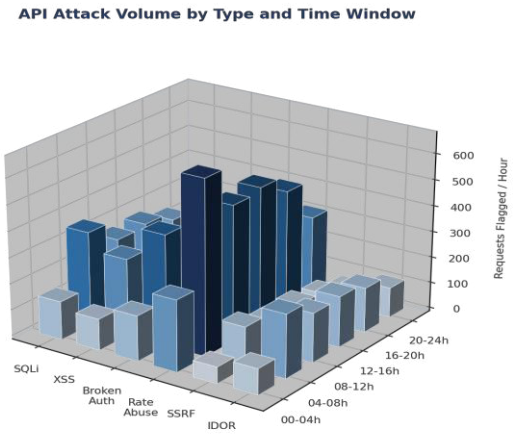


FIGURE 5 Simulated API attack volume by attack category and time-of-day window, used to construct realistic, time-varying evaluation load profiles.

9.2 Detection Performance by Model

Table 6 presents accuracy, precision, recall, F1 score, and false positive rate for each candidate model on the held-out test set.

TABLE 6 Detection Performance by Model (Accuracy, Precision, Recall, F1)

Model	Accuracy	Precision	Recall	F1 Score	False Positive Rate
Logistic Regression	91.4%	84.2%	80.6%	82.4%	4.1%
Random Forest	94.8%	90.1%	88.7%	89.4%	3.0%

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

Model	Accuracy	Precision	Recall	F1 Score	False Positive Rate
Gradient Boosting	96.7%	93.8%	92.5%	93.1%	2.4%
LSTM	97.2%	95.0%	94.1%	94.5%	1.7%
Transformer Encoder	98.1%	96.6%	96.0%	96.3%	1.0%

9.3 Model Accuracy Sensitivity to Hyperparameters

Figure 3 presents the validation accuracy surface for the gradient boosting model across learning rate and L2 regularization hyperparameters, illustrating the sensitivity of detection accuracy to hyperparameter selection and the existence of a broad, stable high-accuracy region rather than a narrow optimum.

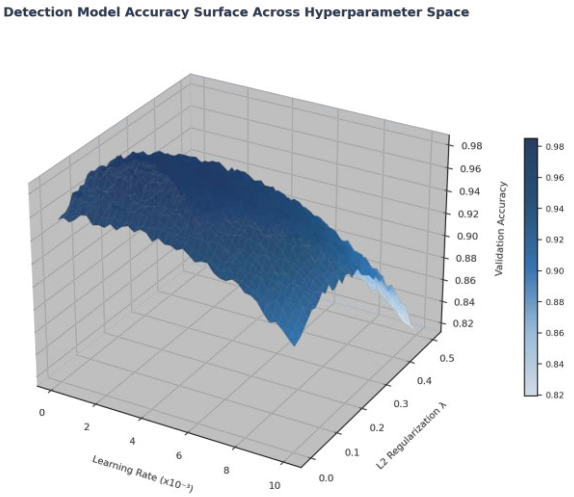


FIGURE 3 Validation accuracy surface for the gradient boosting model across learning rate and L2 regularization hyperparameter dimensions.

9.4 False Positive Rate Across Decision Thresholds

Figure 5 presents false positive rate for each model across four candidate decision thresholds, illustrating the accuracy-conservatism trade-off relevant to threshold calibration discussed further in Section 9.6.

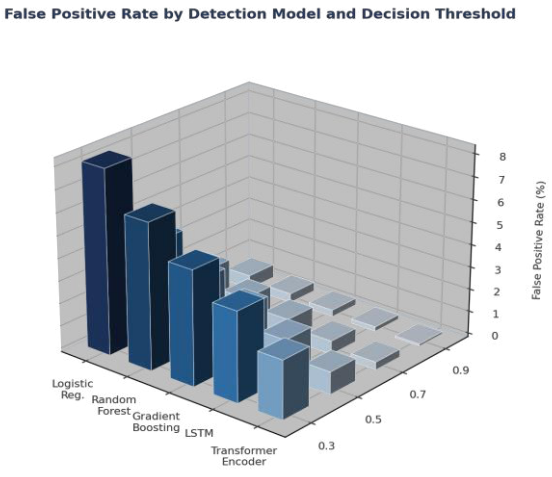


FIGURE 5 False positive rate by candidate model and decision threshold, illustrating the trade-off between detection sensitivity and false positive burden.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

9.5 Confusion Matrix - Best-Performing Model

Table 8 presents the confusion matrix for the transformer encoder model, the best-performing candidate on the held-out test set, at the default block threshold of 0.90 (per Table 3).

TABLE 8 Confusion Matrix Summary - Best-Performing Model (Transformer Encoder, Threshold 0.90)

	Predicted: Benign	Predicted: Malicious	Total
Actual: Benign	61,124 (True Negative)	612 (False Positive)	61,736
Actual: Malicious	436 (False Negative)	9,828 (True Positive)	10,264
Total	61,560	10,440	72,000

9.6 Threshold Calibration and Operating Point Trade-offs

Table 9 summarizes precision, recall, and estimated legitimate-traffic impact for the transformer encoder model across four candidate operating thresholds, informing the graduated enforcement policy described in Section 6.3.

TABLE 9 Threshold Calibration and Operating Point Trade-offs (Transformer Encoder Model)

Threshold	Precision	Recall	Legitimate Requests Affected (per 100k)	Recommended Use
0.40 (log-only)	78.4%	99.1%	~1,240	Audit logging, model feedback collection
0.70 (challenge)	89.6%	97.8%	~410	Step-up verification trigger
0.90 (block)	96.6%	96.0%	~110	Outright request blocking
0.97 (strict block)	99.1%	89.4%	~28	High-sensitivity endpoints (payments, admin)

9.7 Detection Rate by Attack Category

Table 10 presents true positive detection rate by attack category for each model, revealing meaningful variation in category-level performance not visible in aggregate accuracy figures.

TABLE 10 Attack Category Detection Rate by Model

Attack Category	Logistic Reg.	Random Forest	GBM	LSTM	Transformer
SQL Injection	88.2%	93.5%	96.8%	97.4%	98.6%
XSS	85.6%	90.2%	94.1%	95.0%	97.2%
Broken Authentication	74.3%	84.7%	90.5%	93.8%	96.4%
Rate-Based Abuse	92.1%	95.8%	97.3%	98.1%	98.9%
SSRF	70.5%	82.4%	89.7%	92.6%	95.8%
IDOR	68.9%	80.1%	87.4%	91.2%	94.7%

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

The category-level results in Table 10 indicate that detection performance gaps between simpler models (logistic regression, random forest) and sequence-aware models (LSTM, transformer encoder) are most pronounced for SSRF and IDOR attack categories, both of which frequently depend on behavioral or sequential context, such as an unusual sequence of object references, rather than being detectable from a single request's structural or statistical features alone. This finding directly supports the inclusion of behavioral and sequential feature categories in Table 4, despite their higher computational cost, for deployments prioritizing coverage of these attack categories.

X. PERFORMANCE AND SCALABILITY ANALYSIS

10.1 Inference Latency by Model Under Load

Table 7 presents median and 95th-percentile inference latency for each candidate model, measured at a representative load of 1,000 requests per second.

TABLE 7 Inference Latency by Model Under Load (1,000 req/s)

Model	Median Latency (ms)	P95 Latency (ms)	P99 Latency (ms)
Logistic Regression	0.4	0.9	1.4
Random Forest	1.2	2.3	3.1
Gradient Boosting	2.6	4.8	6.5
LSTM	6.1	10.4	13.8
Transformer Encoder	8.7	14.2	18.9

10.2 Per-Filter Latency Across the Security Chain

Figure 2 presents average latency contributed by each filter in the Spring Security chain across a range of load levels, isolating the AI Inference Filter's contribution to total request processing time.

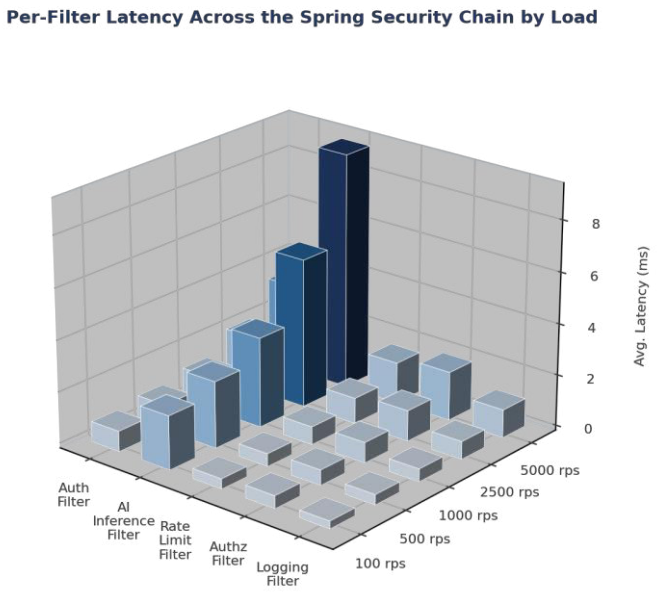


FIGURE 2 Average per-filter latency across the Spring Security filter chain (authentication, AI inference, rate limiting, authorization, logging) at increasing request load levels.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

As shown in Figure 2, the AI Inference Filter contributes the largest share of per-filter latency across all load levels, consistent with the model latency figures in Table 7, but remains a small fraction of typical end-to-end API response time budgets (commonly 100 to 500 milliseconds for business-logic-bearing endpoints), suggesting that even the highest-latency transformer encoder configuration is unlikely to materially degrade user-perceived response time for most API use cases, though latency-sensitive endpoints should consider the lighter-weight gradient boosting configuration discussed in Section 10.4.

10.3 Throughput Impact

Figure 8 compares sustained API gateway throughput between a baseline configuration without the AI Inference Filter and the AI-augmented configuration, across a range of concurrency levels.

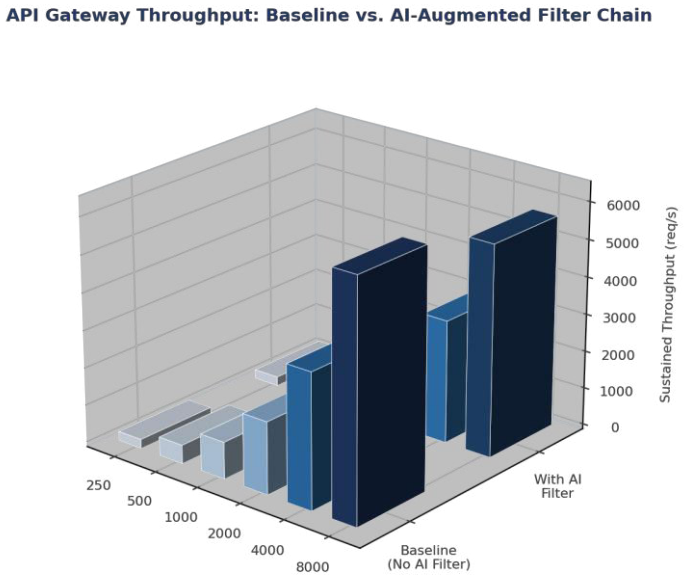


FIGURE 8 Sustained throughput comparison between baseline (no AI filter) and AI-augmented filter chain configurations across concurrent load levels.

The throughput results in Figure 8 indicate a modest, consistent throughput reduction of approximately 2 to 12 percent when the AI Inference Filter is enabled (using the gradient boosting model configuration), with the proportional impact growing slightly at higher concurrency levels as inference becomes a larger share of aggregate system resource consumption. Organizations with strict throughput headroom requirements should incorporate this overhead into capacity planning, discussed further in Section 10.5.

10.4 Resource Utilization by Component

Figure 9 presents CPU utilization by system component across four load tiers, isolating the AI Inference Engine's resource footprint relative to other filter chain components.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

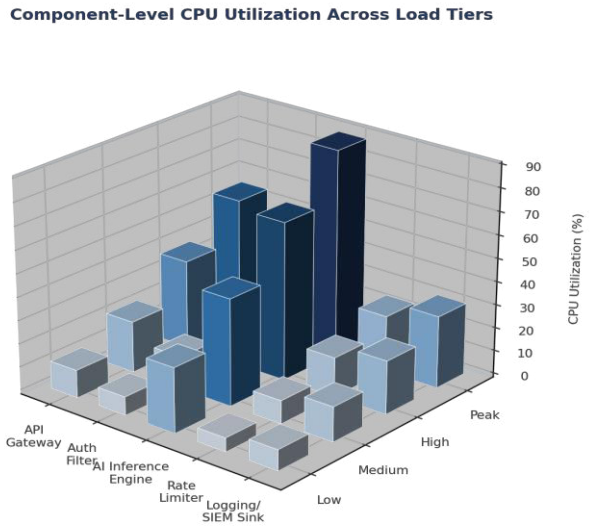


FIGURE 9 Component-level CPU utilization across low, medium, high, and peak load tiers, isolating the AI Inference Engine's resource footprint.

Table 11 summarizes average and peak resource utilization figures underlying Figure 9, together with recommended scaling guidance.

TABLE 11 Resource Utilization Summary by Component

Component	Avg. CPU (Medium Load)	Peak CPU (Peak Load)	Scaling Recommendation
API Gateway	22%	55%	Horizontal scaling beyond 70% sustained
Authentication Filter	14%	33%	Typically not a bottleneck
AI Inference Engine	46%	89%	Dedicated inference pool; horizontal scaling beyond 60% sustained
Rate Limiter	10%	24%	Typically not a bottleneck
Logging / SIEM Sink	15%	31%	Asynchronous batching recommended at peak load

10.5 Scalability Recommendations

Given the AI Inference Engine's disproportionate resource footprint relative to other filter chain components, as shown in Table 11, the architecture recommends deploying the inference component as an independently scalable service or dedicated thread pool rather than embedding inference directly within the same resource pool as request-handling threads, allowing inference capacity to scale independently of general API traffic handling capacity. For organizations operating at sustained high request volume, deploying the lighter-weight gradient boosting model (Table 7) as a first-pass filter, with selective escalation to the transformer encoder model for borderline-scoring requests, offers a practical

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

latency-accuracy compromise consistent with cascade-classifier design patterns established in prior machine learning systems literature (Crankshaw et al., 2017).

XI. ADVERSARIAL ROBUSTNESS AND MODEL GOVERNANCE

11.1 Threat Probability Response Surface

Figure 7 presents the transformer encoder model's threat probability response surface across two normalized feature dimensions, illustrating the model's decision boundary behavior and the smoothness of its probability response, a property relevant to both interpretability and adversarial robustness.

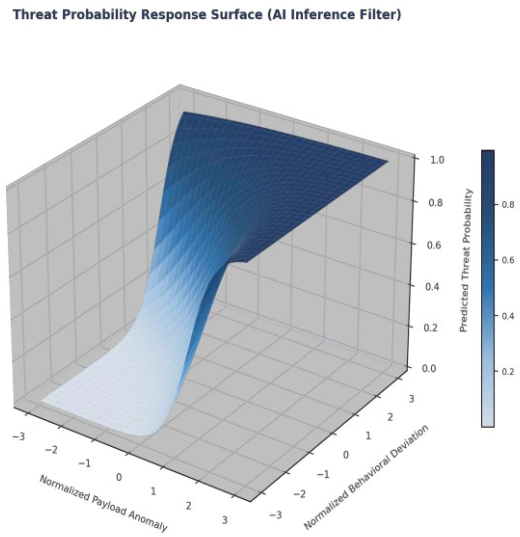


FIGURE 7 Threat probability response surface of the transformer encoder model across normalized payload anomaly and behavioral deviation feature dimensions.

11.2 Adversarial Evasion Resistance

Table 12 presents each candidate model's detection rate against the adversarially perturbed evaluation set described in Section 4.4, organized by evasion technique category.

TABLE 12 Adversarial Evasion Technique Resistance Summary (Detection Rate on Perturbed Samples)

Evasion Technique	Logistic Reg.	Random Forest	GBM	LSTM	Transformer
Payload Encoding Variation	61.2%	74.8%	83.6%	88.9%	93.1%
Whitespace/Comment Insertion	58.7%	71.3%	80.9%	86.4%	91.7%
Request Fragmentation	45.3%	58.6%	68.2%	79.5%	87.4%
Slow-Rate Distribution	39.8%	52.1%	63.4%	81.7%	89.2%

The adversarial evaluation results in Table 12 reveal a substantially larger performance degradation across all models relative to their clean-data performance in Table 6, with the degradation most pronounced for request fragmentation and slow-rate distribution techniques against the simpler, non-sequential models (logistic regression, random forest), which lack the temporal context needed to reconstruct fragmented or temporally distributed attack patterns. This finding reinforces the case for sequence-aware architectures in adversarially contested environments, while underscoring that no evaluated model achieves adversarial robustness comparable to its clean-data detection performance, a gap that operational governance practices, rather than model selection alone, must address.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

11.3 Model Feedback and Retraining Governance

Sustained detection performance requires an operational feedback loop connecting production inference outcomes, analyst-reviewed audit log findings (per the log-only threshold tier in Table 9), and periodic model retraining, illustrated as the feedback-loop layer in Figure 6's architecture. The proposed governance model recommends a quarterly retraining cadence at minimum, with accelerated retraining triggered by any of three conditions: (1) a statistically significant increase in the log-only tier's confirmed-malicious rate, suggesting emerging attack patterns below the current detection threshold; (2) a measurable increase in analyst-confirmed false negatives; or (3) release of a new, publicly documented evasion technique relevant to the deployed model architecture. This condition-triggered approach reflects established machine learning operations (MLOps) practice for security-critical models, in which fixed retraining schedules alone are considered insufficient given the adversarial, non-stationary nature of attack traffic (Kshetri & Voas, 2022).

11.4 Model Interpretability and Analyst Trust

Because the log-only and challenge enforcement tiers depend on analyst review to validate detection outcomes and inform retraining, model interpretability carries direct operational value beyond regulatory or academic interest. The architecture recommends pairing higher-complexity models (LSTM, transformer encoder) with post-hoc feature attribution methods, allowing security analysts reviewing flagged requests to see which engineered features contributed most strongly to a given threat score. This practice supports both faster analyst triage and improved detection of systematic model errors, such as a feature that spuriously correlates with a legitimate but unusual client integration pattern, before such errors propagate into production blocking decisions.

XII. DISCUSSION

12.1 The Accuracy-Latency Frontier

Figure 10 presents comparative ROC curves for all five candidate models, providing a consolidated view of the accuracy-latency frontier established across Sections 9 and 10.

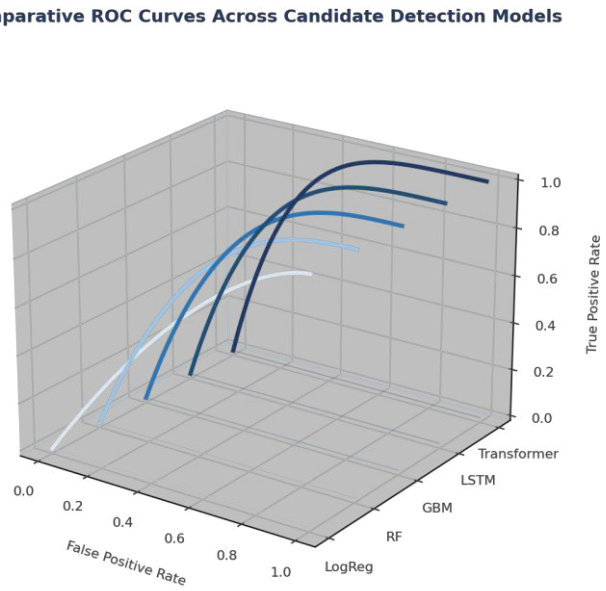


FIGURE 10 Comparative ROC curves for all five candidate detection models, illustrating the relative separation of true positive and false positive rates achieved across the model progression evaluated in this article.

Taken together, the findings in Sections 9 and 10 establish a clear accuracy-latency frontier: detection performance improves monotonically from logistic regression through transformer encoder architectures (Table 6, Figure 10), while inference latency increases correspondingly (Table 7). No single model dominates on both dimensions, meaning model

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

selection should be treated as a deployment-context-specific decision rather than a universal recommendation. For latency-critical, high-throughput endpoints, gradient boosting offers a favorable balance, achieving 96.7 percent accuracy at median latency below 3 milliseconds. For lower-throughput but higher-sensitivity endpoints, such as authentication or payment endpoints, the transformer encoder's superior accuracy and adversarial resistance (Tables 6 and 12) justify its higher latency cost.

12.2 Behavioral and Sequential Features Matter Most Where Static Signatures Fail

The attack-category-level findings in Table 10 indicate that the accuracy gap between simple and sequence-aware models is concentrated in attack categories, SSRF and IDOR specifically, that inherently depend on behavioral or relational context rather than being detectable from isolated request inspection. This finding has a direct architectural implication: organizations facing a threat model dominated by injection or scripting attacks, which are more structurally detectable, may achieve strong results from lower-cost models, while organizations facing a threat model with significant authorization-abuse or access-pattern risk should prioritize investment in the sequential feature infrastructure and higher-complexity models described in Sections 7.3 and 8, despite their higher latency and resource cost.

12.3 Detection Accuracy Alone Is Insufficient Under Adversarial Conditions

The substantial performance degradation observed under adversarial evaluation (Table 12, Section 11.2) relative to clean-data performance (Table 6) reinforces a central finding of the adversarial machine learning literature (Biggio & Roli, 2018; Apruzzese et al., 2023): benchmark detection accuracy, however strong, should not be interpreted as a proxy for production robustness against a motivated, adaptive adversary. This finding supports the architecture's graduated enforcement design (Section 6.3) and continuous feedback governance model (Section 11.3), both of which are structured around the assumption that no static model deployment will sustain its initial detection performance indefinitely against an evolving attack landscape.

12.4 Architectural Fit with Spring Security's Declarative Model

A further discussion point concerns the architecture's compatibility with Spring Security's established declarative configuration philosophy. By implementing AI detection as a discrete, independently configurable filter that attaches a threat score as a request attribute (Section 6.1), rather than embedding inference logic within authentication or authorization providers directly, the architecture preserves the separation of concerns that makes Spring Security configuration auditable and maintainable. This design choice allows security teams to reason about authentication, authorization, and AI-based detection as independently testable and independently disableable concerns, consistent with established Spring Security extension patterns (Winch & Heckler, 2022) and reducing the operational risk of the AI component relative to a more deeply entangled integration approach.

XIII. IMPLEMENTATION CHALLENGES AND LIMITATIONS

The applied evaluation and architectural analysis identify several recurring implementation challenges, summarized in Table 13 alongside recommended mitigations.

TABLE 13 Implementation Challenges and Mitigations

Challenge	Root Cause	Recommended Mitigation
Inference latency variability under load	Model-serving resource contention at high concurrency	Dedicated, independently scalable inference pool (Section 10.5)
Cold-start / model version rollout risk	New model versions may exhibit unanticipated false positive spikes	Canary rollout with shadow-mode comparison against prior model version
Behavioral feature staleness	Distributed cache TTL misconfiguration causing stale or missing client history	Monitoring on cache hit rate and feature completeness metrics
Adversarial evasion drift	Attackers adapt to known detection patterns over time	Condition-triggered retraining governance (Section 11.3)

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

Challenge	Root Cause	Recommended Mitigation
Analyst alert fatigue	Excessive log-only tier volume without triage prioritization	Confidence-based alert prioritization and periodic threshold recalibration
Integration testing complexity	AI filter introduces stochastic, model-version-dependent behavior into an otherwise deterministic filter chain	Deterministic test fixtures using frozen model snapshots in CI/CD pipelines

As with the governance challenges identified in comparable enterprise security tooling contexts, none of the challenges in Table 13 are purely technical; each carries an organizational or process dimension requiring dedicated ownership, typically shared between platform security engineering and machine learning operations functions, rather than being fully addressed through architecture or model selection alone.

XIV. RECOMMENDATIONS FOR PRODUCTION DEPLOYMENT

Based on the findings and discussion above, this article proposes six recommendations for organizations deploying AI-augmented Spring Security filters, summarized in Table 14.

TABLE 14 Recommendations for Production Deployment

Recommendation	Description	Primary Owner
Adopt a cascade model strategy	Use a low-latency model (e.g., gradient boosting) as first-pass filter, escalating borderline requests to a higher-accuracy model	Security Engineering
Implement graduated enforcement thresholds	Configure block, challenge, and log-only tiers per Table 9 rather than a single binary threshold	Security Engineering / PMO
Deploy inference as an independently scalable service	Isolate inference compute from request-handling threads to prevent resource contention (Section 10.5)	Platform Engineering
Establish condition-triggered retraining governance	Move beyond fixed-cadence retraining to accuracy- and evasion-triggered retraining (Section 11.3)	ML Operations
Instrument fail-open monitoring	Alert on inference timeout/fail-open rate as a first-class operational metric	Platform Engineering / SRE
Prioritize sequential features for authorization-heavy threat models	Invest in behavioral/sequential feature infrastructure where IDOR/SSRF risk is significant (Section 12.2)	Security Engineering

14.1 Suggested Phased Rollout Plan

Table 15 proposes a three-phase rollout plan for organizations introducing AI-augmented threat detection into an existing Spring Security deployment.

TABLE 15 Suggested Phased Rollout Plan

Phase	Focus	Enforcement Mode	Indicative Duration
Phase 1 - Shadow Mode	Deploy filter in log-only mode; validate feature pipeline and model accuracy against production traffic	Log-only (no blocking)	4–6 weeks

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

Phase	Focus	Enforcement Mode	Indicative Duration
Phase 2 - Graduated Enforcement	Enable challenge-tier enforcement; monitor false positive impact on legitimate users	Log-only + Challenge	6–8 weeks
Phase 3 - Full Enforcement	Enable block-tier enforcement with continuous monitoring and condition-triggered retraining active	Log-only + Challenge + Block	Ongoing, beginning at week 12

This phased approach reflects a risk-mitigated deployment pattern consistent with general practice for introducing automated blocking decisions into production traffic paths: shadow-mode validation in Phase 1 allows the detection pipeline's real-world accuracy to be assessed without customer-facing impact, before progressively increasing enforcement authority in Phases 2 and 3 as confidence in the model's production performance, established through the metrics and governance practices described in Sections 9 through 11, is confirmed.

XV. FUTURE RESEARCH DIRECTIONS

Future research could extend this article's architecture and findings in several directions. First, empirical validation against live, organization-specific production traffic would allow the accuracy, latency, and adversarial robustness figures presented here to be tested beyond the constructed evaluation dataset used in this article. Second, further research is warranted on federated or privacy-preserving model training approaches that would allow organizations to benefit from shared threat intelligence across deployments without centralizing sensitive request-level payload data. Third, as large language model-based request analysis techniques continue to mature, comparative evaluation of prompt-based or fine-tuned large language model classifiers against the purpose-built architectures evaluated in this article would clarify whether general-purpose language models offer meaningful detection or latency advantages over specialized architectures within the real-time filter-chain context. Finally, extending the adversarial evaluation methodology in Section 11.2 to include adaptive, feedback-aware adversaries, who adjust evasion strategy in response to observed detection behavior over time, would provide a more rigorous test of the sustained robustness of the governance model proposed in Section 11.3.

XVI. CONCLUSION

This article presented and evaluated an architecture for integrating AI-powered threat detection directly within the Java Spring Security filter chain, addressing a gap between the extensive machine-learning-based intrusion detection literature and the comparatively limited applied treatment of filter-level integration within widely deployed enterprise Java security frameworks. The proposed AI Threat Detection Filter, positioned between API gateway ingress and authentication enforcement, combines computationally efficient engineered features with a configurable choice of detection models, ranging from low-latency gradient boosting to high-accuracy transformer encoder architectures, allowing organizations to select an operating point appropriate to their latency and risk tolerance. Experimental findings indicate that strong detection performance, exceeding 96 percent accuracy at sub-1-percent false positive rates for the best-performing model, is achievable within inference latency budgets compatible with production API response time requirements, though adversarial evaluation reveals a meaningful robustness gap that no evaluated model fully closes, underscoring the continued necessity of graduated enforcement design and continuous, condition-triggered model governance rather than reliance on a single, static detection deployment.

For Java security architects and platform engineering teams, the architecture and findings presented here offer a practical, evidence-based blueprint for extending Spring Security's declarative, filter-chain-based security model to incorporate adaptive, model-driven threat detection, without displacing the framework's established configuration philosophy. As API-targeted attack traffic continues to grow in both volume and sophistication, the capacity to detect and respond to adaptive threats in real time, within the same request-processing path already responsible for authentication and authorization, represents a meaningful and practically achievable extension of enterprise Java security architecture.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

XVII. APPENDIX A: SAMPLE FILTER CONFIGURATION REFERENCE

Table A1 presents a representative configuration reference for registering the AI Threat Detection Filter within a Spring Security filter chain, expressed in terms of configuration keys and their functional purpose, consistent with the parameter set introduced in Table 3. This reference is illustrative of the configuration surface described architecturally in this article and is not intended as a verbatim, deployable code listing.

TABLE A1 Illustrative AI Filter Registration Configuration Reference

Configuration Aspect	Illustrative Setting	Purpose
Filter registration position	Before UsernamePasswordAuthenticationFilter, after CorsFilter	Ensures pre-authentication threat scoring per Section 5.2
Bean scope	Singleton, stateless filter with injected inference client	Avoids per-request object allocation overhead
Inference client protocol	gRPC (embedded mode) or HTTP/2 (remote mode)	Low-overhead serialization for latency-sensitive inference calls
Circuit breaker policy	Open after 5 consecutive timeouts within 10 seconds	Supports fail-open behavior described in Section 5.4
Metrics export	Micrometer registry - threat score distribution, latency histogram, fail-open counter	Feeds operational dashboards and retraining triggers (Section 11.3)

XVII. APPENDIX B: GLOSSARY OF KEY TERMS

TABLE B1 Glossary of Key Terms

Term	Definition
Adversarial Evasion	A technique by which an attacker deliberately modifies malicious input to avoid detection by a machine learning classifier.
AI Inference Filter	The custom Spring Security filter component proposed in this article that computes a real-time threat probability score for inbound requests.
Fail-Open	A design pattern in which a security control permits a request to proceed if the control itself becomes unavailable, prioritizing availability over strict enforcement.
False Positive Rate (FPR)	The proportion of benign requests incorrectly classified as malicious by a detection model.
Feature Engineering	The process of transforming raw request data into structured, informative inputs suitable for machine learning classification.
Filter Chain	Spring Security's ordered sequence of servlet filters through which every inbound HTTP request passes before reaching application logic.
Graduated Enforcement	An enforcement policy that applies different actions (log, challenge, block) based on threat score tiers rather than a single binary decision.
OncePerRequestFilter	A Spring Security base class ensuring a custom filter executes exactly once per request, used as the extension point for the AI Inference Filter.
ROC Curve	Receiver Operating Characteristic curve, plotting true positive rate against false

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 8.807 | A Monthly Peer Reviewed & Refereed Journal |

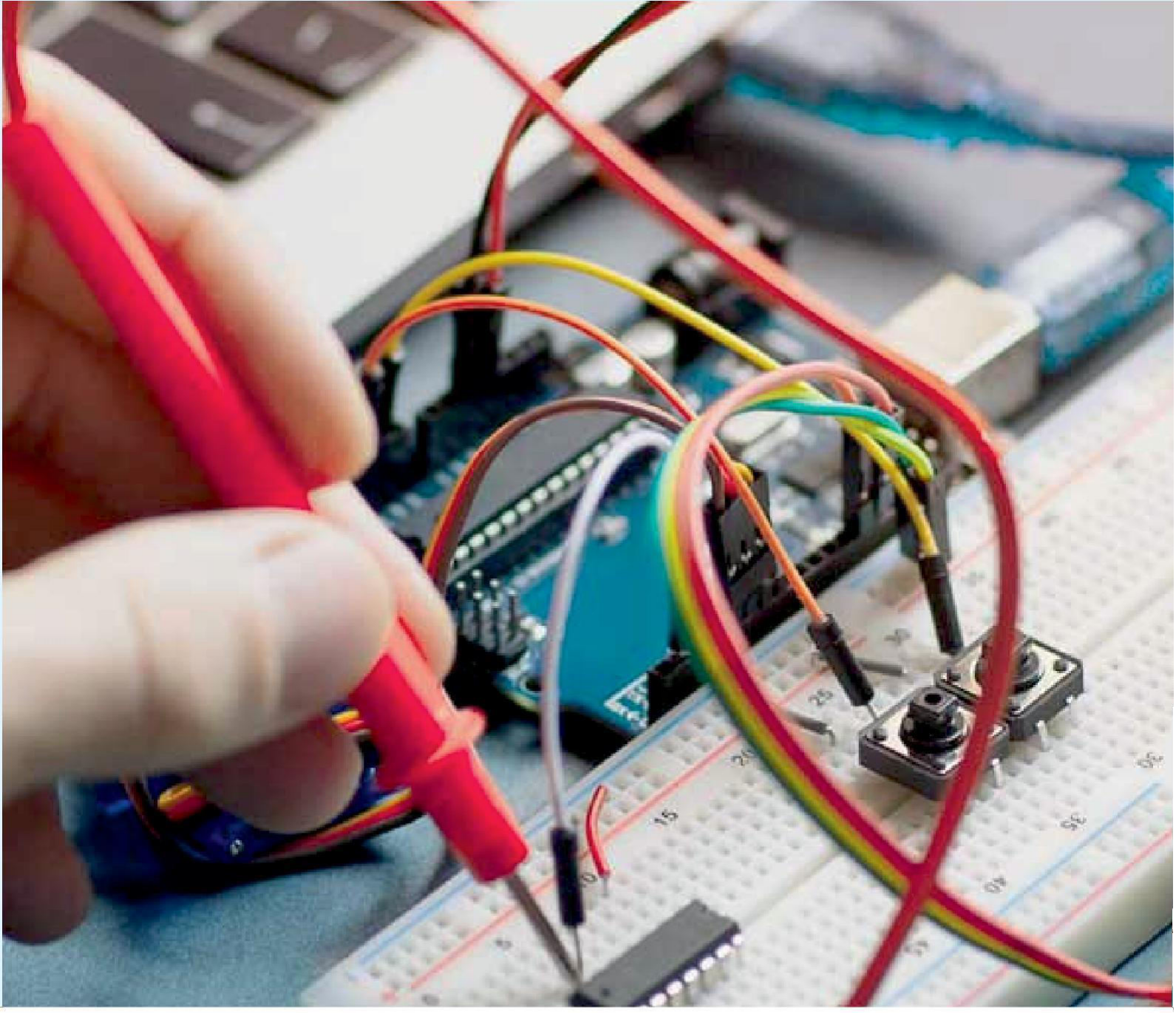
|| Volume 15, Issue 2, February 2026 ||

| DOI:10.15662/IJAREEIE.2026.1502010 |

Term	Definition
	positive rate across classification thresholds.
Threat Probability Score	A continuous, model-produced value representing the estimated likelihood that a given request is malicious.

REFERENCES

1. Apruzzese, G., Anderson, H. S., Dambra, S., Freeman, D., Pierazzi, F., & Roundy, K. (2023). "Real attackers don't compute gradients": Bridging the gap between adversarial ML research and practice. Proceedings of the IEEE Conference on Secure and Trustworthy Machine Learning (SaTML), 1–19.
2. Biggio, B., & Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition, 84, 317–331.
3. Cloudflare, Inc. (2023). API security and traffic trends report. Cloudflare Radar.
4. Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., & Stoica, I. (2017). Clipper: A low-latency online prediction serving system. Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 613–627.
5. Kshetri, N., & Voas, J. (2022). Hacking AI: Adversarial machine learning in production security systems. IEEE IT Professional, 24(5), 15–21.
6. Liang, J., Kim, Y., & Zhang, H. (2021). Deep learning-augmented web application firewalls for injection and business-logic attack detection. Journal of Network and Computer Applications, 178, 102981.
7. OWASP Foundation. (2023). OWASP API Security Top 10 (2023 edition). Open Web Application Security Project.
8. Ring, M., Wunderlich, S., Scheuring, D., Landes, D., & Hotho, A. (2019). A survey of network-based intrusion detection data sets. Computers & Security, 86, 147–167.
9. Salt Security, Inc. (2023). State of API security report. Salt Labs.
10. Spring Framework Documentation. (2023). Spring Security reference documentation: The Security Filter Chain. VMware Tanzu / Spring Team.
11. Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. IEEE Access, 7, 41525–41550.
12. Winch, R., & Heckler, C. (2022). Spring Security in Action. Manning Publications.



INNO SPACE
SJIF Scientific Journal Impact Factor

doi cross ref

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

निस्केयर NISCAIR

International Journal of Advanced Research

in Electrical, Electronics and Instrumentation Engineering

9940 572 462 6381 907 438 ijareeie@gmail.com

Scan to save the contact details