Detection and Mitigation of SQL Injection-based Attacks in Web Security

Nisha P. Shetty, Vinayak Kothari, Eva Hemantkumar Shah, Prashanth J. Kumar and Jayashree Shetty*

Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal-576104 Karnataka, India
E-mail: jayashree.sshetty@manipal.edu
**Corresponding Author*

Received 31 March 2026; Accepted 17 June 2026

Abstract

Crafty tactics like nested request bodies, encoding schemes, and JavaScript Object Notation (JSON) operators are now used by attackers to trick and bypass conventional Web application firewalls. The proposed machine learning-based system for detecting and mitigating SQL injection attacks is designed not just to protect against conventional SQLi attacks but also against JSON-based SQLi attacks, NoSQL injection attacks, hybrid attacks, and conventional WAF evasion techniques. The proposed system utilizes a stacking ensemble of Random Forest, Gradient Boosting, and Logistic Regression classifiers with manually constructed features that represent various properties of queries instead of using conventional static rule-based techniques or deep learning models. The detector is integrated into an application process that facilitates query inspection, batch analysis, decision explanation, and mitigation actions. The application is made available via a Flask-based REST API. To add structural variety, the dataset is constructed from public payload sources and augmented methodically. To support detections and to provide potential WAF rules, feature-level explanations are employed. The proposed

Journal of Mobile Multimedia, Vol. 22_3, 373–412.
doi: 10.13052/jmm1550-4646.2234

work is also extended to incorporate a privacy-preserving federative learning framework to show its efficacy in collaborative environments. The system's overall goal is to provide a modern, API-driven application as a complementary injection attack detection and monitoring layer suitable for quick, easy deployment.

Keywords: Cybersecurity, machine learning, intrusion detection, SQL injection, anomaly detection, deep learning, web security, network security, threat detection.

1 Introduction

Poor input handling procedures in data-dependent applications lead to SQL injection, a persistent security flaw. SQLi was first identified in the late 1990s [1], and it is still included in the OWASP Top 10 vulnerabilities [2]. SQLi is a technique used by attackers to manipulate backend queries in order to obtain unauthorized access, disclose private information, change records, or even take down a database infrastructure. After more than 30 years of research and mitigations, the utilization of complex data format variants and growing application architectures continues to create fresh opportunities for SQLi to survive.

For data flexibility and scalability, structured data formats, particularly JavaScript Object Notation (JSON) [3], are frequently used in contemporary Web applications. Relational databases simplify API development by providing native support for nested structures and JSON operators [4]. In order to get beyond the filtering features of legacy systems [5], which now seem unnecessary, JSON-based SQLi attacks incorporate malicious payloads into JSON fields or even use JSON-specific operators. This change pushes SQLi beyond traditional string-concatenation vulnerabilities to more sophisticated context-sensitive attack strategies.

The failure of enterprise-grade WAFs, such as those offered by Palo Alto Networks, AWS, Cloudflare, F5, and Imperva, to recognize SQLi payloads constructed using JSON syntax was demonstrated by Claroty's Team82 in 2023 [5]. The aforementioned findings have demonstrated that most commercial-grade WAFs allow nested injection strings because they incorrectly parse or interpret JSON syntax. These defects prove that rule-based detection is fundamentally flawed, as most solutions rely on blacklist-based techniques, regex, and signatures [6].

Because JSON-based searches are widespread and schemas are often changing, this issue is made worse in cloud-native and API-driven systems. Monitoring every potential mutation with static rules is impractical and often leads to significant false-negative rates. The illustrated injection, SELECT * FROM users WHERE data @>{"id": 1 OR 1=1}; may appear syntactically correct but, if it were incorporated into an acceptable JSON payload, it would be semantically harmful. Because there are so many parameters involved, traditional methods do not provide the additional context of the query structure that is necessary for the detection of such hidden discrepancies.

Due to these flaws, machine learning (ML) and deep learning (DL) are becoming more often used for SQLi detection. Regardless of hard-coded signatures, ML models are capable of identifying abnormalities and learning distributional patterns from huge datasets [7]. They are able to identify disguised or undetected payloads by capturing the semantic links between tokens [8]. The possibility of contextual modeling has been studied in the past. For instance, Lu et al. [9] discovered that semantic embeddings performed noticeably better than pattern-based methods, and Lo et al. [10] employed multi-head self-attention mechanisms for the same.

However, detection is not enough. This is because, given that the real world will definitely pose queries that were not used to train any database used for building an application, an adequate security solution must be able to mitigate threats in real-time. This includes blocking queries, logging, and alerting, as well as supporting updates for WAF rules. One of the easiest ways of integrating ML models with existing application stacks is through their deployment as light-weight REST APIs, for example, using Flask [11]. It is quite simple to implement so that it intercepts and investigates application SQL queries before execution. Passive monitoring, therefore, becomes active defense with the integration of ML-based detection mechanisms and adequate response mechanisms [12].

In light of these insights, the contributions of this work are as follows:

- While frameworks like parametrized queries and Object-Relational Mapping (ORM) frameworks are most suitable for such attacks at the code level, this work proposes a lightweight model that complements these frameworks, suitable for runtime detection of such attacks in API driven environments.
- We propose a ML-based framework to support the detection of both classical and JSON-based SQLi attacks, focusing on evasion-resistant (both semantically and structurally) behavioral modeling.

- We create a realistic dataset with various SQLi payloads, including JSON-specific exploits that have been shown to bypass traditional WAFs.

With the surge in shift to cloud-native designs and increased use of JSON formats [13], there are active research to upgrade existing query blocking and alerting strategies. This proposed approach aims to address this gap by providing the ML solution as a complementary add on to the exiting protection infrastructure.

The rest of this document is structured as follows. Section 2 reviews prior literature and identifies key research gaps, and outlines the motivation for this work. Section 3 outlines the materials and methods employed in this study, covering the combined data-generation pipeline, system architecture, feature extraction process, machine-learning detection engine, and mitigation strategy. Section 4 details the outcomes of the developed application, including real-time analysis, batch processing, WAF-evasion management, explainability results, and system-wide performance indicators. Section 5 provides a detailed discussion of the findings and concludes the research.

2 Literature Survey

Fathi et al. [14] developed a working DL-based SQLi detector model based on Long Short-Term Memory (LSTM) networks to work on imbalanced datasets. With polymorphous resampling, their model was able to attain near-perfect accuracy with three public Kaggle datasets of different benign-malicious ratios (99.7–100%) with no resampling. The experiment showed that LSTM had the capability of sequence query dependencies, but was not interpretable and had not been analyzed to run in real time. The article by Mustapha et al. [15] is a review of more than 25 SQLi detection ML models in e-commerce applications, which comprised of the Random Forest, SVM, and Naïve Bayes classifiers. Their review showed potential detection accuracy (89-99%) but reported that real-time detection and JSON-based injection management remained a difficult issue, providing theoretical knowledge but not concrete details.

Pasini et al. [16] investigated the strength of SQLi detectors produced by large language models (LLMs). Their experiments, based on building a dataset of adversarial payloads and adversarial fine-tuning, increased model accuracy by 71%. Even though they were innovative, their research did not have a runtime analysis or explainability. Equally, Dasari et al. [17] proposed

the use of generative models like GANs and VAEs to generate different SQLi payloads to augment data. The method enhanced the resistance to zero-day attacks but failed to measure the performance in real-time.

Zulu et al. [18] utilized contextual embeddings with conventional classifiers in order to improve the semantic detection of SQLi. Their hybrid system was able to identify subtle differences in the linguistic characteristics of queries with an accuracy of about 99%, but embedding computation was very expensive. Sun et al. [19] suggested a two-fold DL pipeline that combines TextCNN and Bi-LSTM layers along with attention frameworks, with high precision and low false positives. The interpretability and computational overhead were not, however, evaluated. Alghawazi et al. [20] used an RNN-autoencoder to detect anomalies, training normal query patterns and detecting maximum variations as an attack with an F1 score of close to 92%. In a different publication, Alghawazi et al. [21] provided a systematized review of various ML-based SQLi detectors, summarized the performance measures, but did not include experimental validation or deployment details of the experiments.

The ensemble model presented by Alarfaj and Khan [22] considered probabilistic neural networks and bio-inspired feature optimization and attained 99.19% accuracy on a large custom dataset. Their strategy did well with precision, but never assessed against hidden, obfuscated payloads. Xu et al. [23] used Deep Isolation Forests to detect anomalies, which is effective at detecting rare attacks with higher accuracy when compared to ensemble models. Paul et al. [24] suggested SQLR34P3R, a CNN-LSTM hybrid architecture, which analyzed more than four lakh traffic samples, and yielded a high F1 score (approximately 97). The system is also connected to DVWA and Wireshark to track exfiltration in real-time, but was not tested with preventive strategies.

ModSecurity WAF was upgraded to a hybrid ModSec-ML engine by Floris et al. [25], which increased its accuracy of detection by 30% of adversarial SQLi. Although good, the model was very time-consuming and had to be manually tuned as well as labeled. A cascaded NLP-based SQLi detector optimized to work with a data center was suggested by Tasdemir et al. [26] and achieved 99.86% accuracy and high throughput without focusing on interpretability and scaling of costs. Muduli et al. [27] incorporated two novel CNN architectures for effective classification of SQLi. Their models achieved an accuracy of over 97% when tested on various datasets.

Gandhi et al. [28] constructed a CNN-BiLSTM model to identify multi-layered and noisy SQLi payloads with an overall accuracy of about 98. It

was not tested on modern JSON-based injections despite such good generalization. SqliGPT [29] is an LLM-based black-box SQLi scanner by Gui et al. that has a Strategy Selection Module and a Defense Bypass Module. Assessed on 45 targets of SQLi-Lab and real CVEs, SqliGPT outperformed six commercial scanners with high coverage, but was prone to hallucinations. The TextCNN model, with Word2Vec embeddings, put forward by Xia et al. [30] had above 95% accuracy and was sufficient to resist obfuscated payloads, but did not investigate JSON-based and Unicode-encoded evasions.

Thalji et al. [31] introduced an autoencoder-based architecture that extracted distinguishing features from SQL payloads. These features were further given as input into various ML and DL models for effective categorization. Le et al. [32] conducted a literature review on ensemble models used to detect SQLi with a focus on explainable AI (XAI) models like SHAP and LIME. Although useful to understand transparency and the model underlying it, they did not test the feasibility of deploying the model empirically.

2.1 Research Gaps and Motivation

1. Imbalanced Dataset Challenges: Real-world SQLi datasets are heavily imbalanced, leading to biased detection.
2. Inadequate Detection of JSON-Based SQL Injections: Emerging JSON operators in SQL create unseen injection vectors.
3. Low Robustness Against Evasive Payloads: Models often fail against obfuscated or WAF-bypassing queries.
4. Lack of Real-Time Deployment Evaluation: Accuracy metrics ignore latency and scalability performance.
5. Lack of Semantic Understanding in Query Analysis: Existing systems overlook meaning-preserving variations.
6. Limited Use of XAI: Lack of transparency limits trust.
7. Underutilization of Hybrid Learning Models: Few works combine supervised, unsupervised, and semi-supervised learning.
8. Scarcity of Realistic Benchmark Datasets: Public datasets lack diversity and modern attack samples.
9. Absence of Runtime monitoring and integration with modern application security stacks: Most of the enterprises rely on preventive methods like ORMs and WAFs. Runtime analysis is not employed on a large scale. ML algorithms, even though used in a variety of works, are mostly used in isolation and not shown how they can be integrated into application security workflows.

3 Materials and Methods

3.1 Data Generation and Augmentation

3.1.1 Existing dataset sources

The primary dataset is sourced from Kaggle, with over 30,000 labeled queries [33]. The dataset contains both safe and malicious SQL payloads. In addition, several data sets from security community-developed GitHub repositories are utilized, including PayloadAllTheThings [34] and NoSQL_Gen [35].

In order to ensure major prominent techniques of attacks are not overlooked, query patterns are also collected from OWASP resources, such as the SQL Injection Prevention Cheat Sheet [36] and query parameterization guidelines [37]. Finally, to demonstrate how attacks happen in real time, payloads that are generated or captured by various security testing tools, such as Burp Suite [38] and SQLMap [39], are included.

All queries collected from various sources are then standardized to create a single internal schema with four columns: raw_query, label (0 for "safe," 1 for "malicious"), attack_type, and source. Multiple datasets are combined in this way to avoid bias towards any particular source, so that the final dataset represents a wide range of actual SQL and API attacks.

Instead of learning known attack methods, the model thus learns distributions across query behavior (such as abnormal entropy, unexpected operator usage, stacked statements, or anomalous JSON structures – as defined in Table 4). Because of its design, the system can generalize to payloads that have never been encountered before, such as proprietary or internal query forms, provided that they include unusual structural features. In actuality, this method is more appropriate for managing new injection attempts since it is more akin to anomaly-aware detection than signature matching.

3.1.2 Python libraries for augmentation

Many prominent Python libraries are used for augmentation, such as the pandas [40] library to eliminate duplicates, NumPy [41] for vectorized operations and randomized sampling, and Python's re [42] for regular-expression-based transformations. Additionally, the base64 [43] library along with the urllib.parse module [44] is used to create Base64-encoded and URL-encoded versions of payloads in order to replicate encoding-based evasion strategies that are frequently seen in actual attacks. The random [45] module accounts for the stochastic selection of augmentation rules, the collections. Counter

Table 1 Augmentation template used

Transformation Category	Description	Example Variants
Comment Injection	Comments are added to terminate or alter query execution.	Appending comments like OR 1=1 -- OR a=a /*
Case and Spacing Variations	Modify keywords with case mismatch and irregular spacing [36]	SeLeCt * FrOm users UNION/**/SELECT username, password FROM usersSELECT+*+FROM+users
Encoding-Based Transformations	Obfuscation using character functions, hex-encoded using escape sequences, or URL-encoded	%27 OR 1=1 -- -- URL encoding (→ %27)CHAR(83)+CHAR(81)+ CHAR(76) -- "SQL"0x53454C454354 -- Hex for "SELECT"
Logical Operator Variants	Tautological and contradictory conditions are inserted into queries	SELECT * FROM products WHERE product_id=(SELECT product_id FROM products WHERE 1=1); Or OR 1=1 AND 1=0 OR 1=1 OR TRUE
Nested Sub-query Wrapping	Malicious queries are hidden inside nested subqueries.	SELECT * FROM users WHERE id=(SELECT id FROM users WHERE 1=1);

utility [46] is used to assess diversity and make sure that no single template dominates the dataset.

3.1.3 SQL augmentation rules

The rules generated by the augmentation engine are tabulated in Table 1.

When combined, these guidelines guarantee that the model learns a variety of obfuscated and polymorphic SQLi variations in addition to evident attacks.

3.1.4 JSON and NoSQL augmentation

Separate augmentation algorithms are created especially for JSON and NoSQL-style payloads because many contemporary apps expose JSON-based APIs [47]. The engine adds operators like $where, $gt, $lt, $gte, $regex, $or, and $and – which are frequently used in document databases – to otherwise benign JSON documents.

For instance, adding a $where condition that always evaluates to true to a basic authentication payload with a username and password field can successfully circumvent authentication logic. To mimic WAF-bypass attempts, encoded variations are also created, such as partially encoded payloads or permissive regular expressions.

Additionally, structural differences are included. JSON payloads can be recast as arrays with operator expressions, deeply nested objects, or realistic fields like filter, conditions, or query. This aids in guaranteeing that the detection system maintains its resilience in the face of densely nested JSON structures, which are frequently encountered in actual API traffic [48].

3.1.5 Block diagram overview of the data generation workflow

A block diagram of the data generation procedure is shown in Figure 1.

3.1.6 Dataset balancing and finalization

To ensure syntactic correctness, SQL queries are parsed using the sqlparse library [49]. JSON payloads are validated against JSON schemas [50]. The length of queries varies from 10 to 5000 characters, reflecting a realistic query length distribution. The final exported dataset contains the raw query, label, type of attack (conventional SQLi, JSON/NoSQL, WAF bypass, hybrid, or safe), and 20 precomputed feature columns, as described in the next section. The entire dataset composition is tabulated in Table 2. As shown in Table 2, the dataset is balanced with a 50–50 ratio of benign and malicious queries. Roughly, the four attack classes are in the ratio of 2 : 1.5 : 1 : 0.5 (classical : JSON/NoSQL : WAF-bypass : hybrid).

3.2 Detection Engine Architecture and Model Selection

3.2.1 Classifier configurations

The scikit-learn framework is utilized to implement all of the application's classifiers [51]. Table 3 describes the hyperparameters utilized in the deployed system.

3.2.2 Reasoning for choosing ensemble classification

On the held-out test set, both the Random Forest and Logistic Regression models achieve nearly 98% accuracy, and the ensemble performs equally too. The choice of ensemble as the final classifier is justified for the following reasons. Owing to differences in the working principles of each model, i.e., Gradient Boosting learns from previous residual errors [52], Random Forest

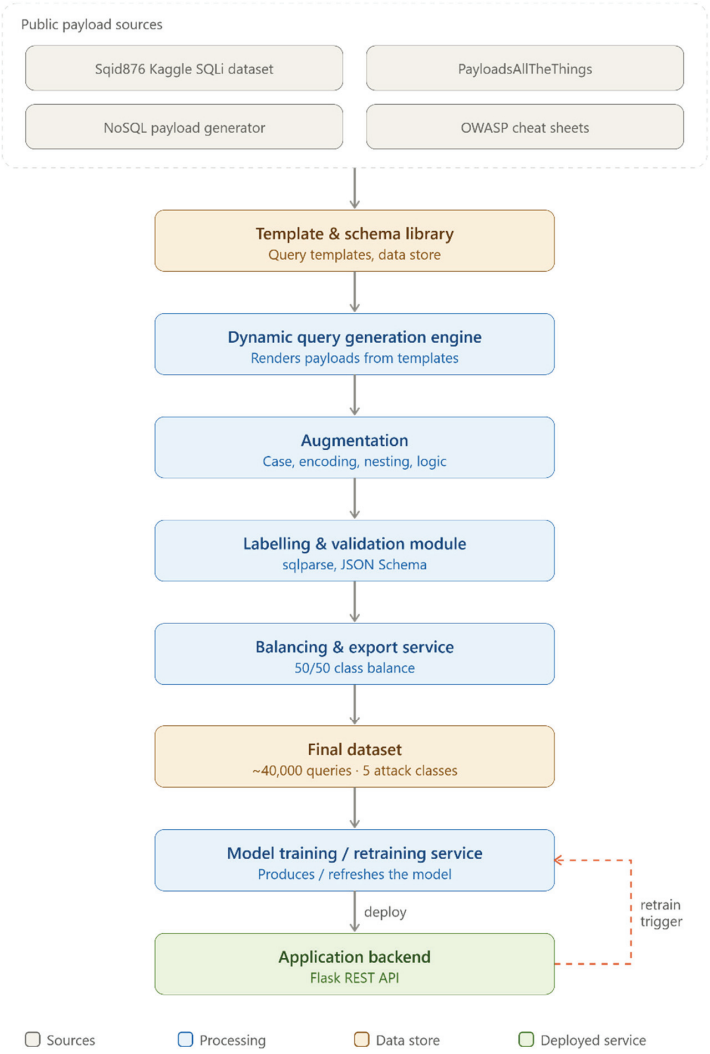

Figure 1 Proposed data generation pipeline.

stresses non-linear feature interactions [53], and Logistic Regression concentrates on global linear structure [54], they usually disagreed on edge cases. Soft voting, as opposed to blindly adhering to one model, produces a stable choice by aggregating the prediction probabilities of individual learners [55]. Alternatively, ensembles are proven to be stable against future drift and noisy data [56]. The proposed classifier is evaluated further against two prominent

Table 2 Dataset composition

Class	Total	Train (80%)	Test (20%)	Primary Source
Safe/benign queries	15,000	12,000	3000	Sajid576 [33], parameterized templates [37]
Classical SQLi	6000	4800	1200	Sajid576 [33], PayloadsAllTheThings [34], sqlmap [39]
JSON / NoSQL injection	4500	3600	900	No-SQL_Gen [35], Claroty patterns [5]
WAF-bypass / obfuscated SQLi	3000	2400	600	PayloadsAllTheThings [34], Burp Suite [38]
Hybrid (SQL + JSON)	1500	1200	300	Generated via augmentation pipeline
Total	30,000	24,000	6000	

DL architectures - BiLSTM and BERT. As the stacking ensemble offered comparable accuracy with better interpretability and quicker speed, it is ultimately chosen as the final detector.

3.2.3 Explainability text and suggestion generation

From an implementation standpoint, explainability primarily depends on the built-in introspection methods of scikit-learn. The Random Forest and Gradient Boosting classifiers' feature importance scores are combined with the Logistic Regression coefficient vector. Basic statistics like feature means, standard deviations, and rankings are calculated using NumPy [41] and pandas [40].

There are two primary phases involved in creating the explanation that is displayed to the user. The system first ascertains which features are most important for the current query. The explainer examines the scaled feature vector for that query after the ensemble has generated its probability score. It considers the global importance of each feature as well as the degree to which the current value deviates from the training distribution. Each feature is given a straightforward contribution score based on this. For explanation, only the top five characteristics that most strongly influenced the prediction to fall into the malicious category are kept.

Secondly, a series of handwritten explanation templates is matched with these features. These are a few manually authored text templates for each engineered feature. For instance, a message regarding many SQL statements in a single request is triggered by stacked queries, whereas an explanation indicating the presence of significant obfuscation or encoding is triggered by

Table 3 Configuration of classifiers

Classifier	Key Configuration	Design Rationale
Random Forest (RF)	n_estimators=100 max_depth=20 max_features=sqrt bootstrap=True class_weight=balanced random_state=42 n_jobs=-1	Chosen to capture complex, non-linear interactions between engineered features such as entropy, stacked queries, and JSON nesting depth. Balanced class weights penalize false negatives even if future data distribution drifts slightly.
Gradient Boosting (GB)	n_estimators=100 learning_rate=0.1 max_depth=7 subsample=1.0 random_state=42	Included because it focuses iteratively on hard-to-classify samples. In early experiments, it often flagged borderline or unusual payloads missed by other models.
Logistic Regression (LR)	penalty=l2 C=1.0 solver=lbfgs max_iter=1000 class_weight=balanced random_state=42	Serves as a linear baseline with well-calibrated probabilities. Captures global trends and stabilizes the ensemble when tree-based models over-emphasize non-linear patterns. Very fast and reliable in production.
Soft Voting Ensemble	Base models: RF, GB, LR voting=soft Equal weights	Final deployed detector. Averages class probabilities from all base models and selects the class with the highest mean probability. Preserves probability information, which is important for confidence-based decisions, explainability, alert thresholds, and mitigation logic.
BiLSTM	Embedding dimension=128 Hidden units per direction=128 Layers=1 (bidirectional) Dropout=0.3 Batch size=64 Optimizer=Adam lr=1e-3 Epochs=20 (early stopping, patience=3) Loss=binary cross-entropy	Chosen to capture bidirectional positional and contextual relationships between tokens
BERT (bert-base-uncased)	Hidden size=768 Attention heads=12 Layers=12 Max sequence length=128 tokens Batch size=16 Optimizer=AdamW lr=2e-5 weight decay=0.01 Warm-up=10% of total steps Epochs=4 (fine-tuning) Loss=binary cross-entropy on the [CLS] token	Chosen to learn from deep bidirectional contextual representations, unlike BiLSTM which learns through strong subword co-occurrence patterns

abnormally high entropy. Explanations highlighting possible NoSQL injection attempts are produced by JSON or NoSQL operators like $where or $gt, and explanations noting anomalous query structure are produced by dense usage of SQL keywords.

A sample template reads as follows: "The query has unusually high character entropy, which often indicates heavy obfuscation or encoding rather than normal application SQL". The system also offers remediation recommendations in addition to an explanation of why a query was detected. A second set of handwritten templates that rely on the identified feature patterns is used to create these. For instance, to prevent user input from changing the query structure, the program recommends using parameterized queries or ORM techniques in place of string-concatenated SQL when OR-based injection patterns are identified. The system endorses prohibiting arbitrary operators in user-controlled fields and validating request bodies against strict schemas when it detects problems unique to JSON or NoSQL. The explainer module thus offers explanations based on SQL based attack relevant feature parameters, and returns human-understandable mitigation schemes with the prediction outcomes. The outcomes are linked to the engineered features, and the language used is simple to understand.

3.3 Training and Evaluation Methodology

As noted in Table 2, the train-test split is 80:20 with stratified resampling. The core features aiding the classification are tabulated in Table 4.

Table 4 Selected features

Category	Feature	Description
Structure	Query length	Captures abnormally long payloads typically produced by obfuscation or automated attack tools
	Character entropy	High entropy indicates packed payloads; low entropy often reflects simple keyword probing
	Non-alphanumeric ratio	Injection payloads rely heavily on symbols, unlike normal inputs
	Semicolon count	Indicator of stacked queries and multi statement injection attacks
	Comment token count	Comment tokens (e.g. --, /**/) are commonly used to truncate or neutralize legitimate query logic
SQL-Focused	SQL keyword count	Aggregated count of reserved SQL tokens (e.g. SELECT, DROP, WHERE) present in the payload

(*Continued*)

Table 4 Continued

Category	Feature	Description
	UNION count	Signal for UNION-based SQLi in which a secondary query is used to extract data
	Boolean tautology flag	Binary indicator for always-true or always-false patterns (e.g. OR 1=1, AND a=a) used in authentication bypass
	Function call density	Counts calls linked to time-based (e.g. SLEEP, WAITFOR) and file-based (e.g. LOAD_FILE) attacks.
	Identifier anomaly score	Measures deviation of column/table names from expected schema
JSON/NoSQL	JSON depth	Deep nesting is typical in injections
	NoSQL operator count	Counts operators associated with NoSQL injection
	Key value ratio	Abnormal ratios suggest operator-heavy payloads rather than normal data objects
	Array length	Oversized arrays often indicate mass exploitation
Encoding and Obfuscation	URL-encoding density	Measures frequency of percent-encoded characters
	Base64-like token flag	Detects long, high entropy Base64 character distribution substrings
	Mixed encoding score	Captures simultaneous use of multiple encodings, a strong bypass signal
Behavioral/ Contextual	Repeated token ratio	High repetition of keywords reflects automated or scripted attacks
	Digit ratio anomaly	Unusual numeric density suggests probing behavior
	Endpoint risk prior	Lightweight prior based on the historical attack frequency of the targeted endpoint

3.4 Operational Workflows

3.4.1 Implementation pipeline

The implementation pipeline is documented in Algorithm 1.

Algorithm 1 Ensemble-Based Injection Detection with Explainability

Require: Raw query/API payload q
Ensure: Predicted label y, confidence score conf, attack category, and feature explanations
Perform basic sanity checks
Extract feature vector:

$$x = \Phi(q) = [x_1, x_2, \ldots, x_d] \in R^d; \text{d} \approx 20$$

for each feature f do

Scaled value=

$$\underline{x}_f = (x_f - \mu_f) \div \sigma_f$$

end for

Form scaled feature vector $\underline{x}$

Obtain probability predictions:

$$p_{rf} = C_{rf}(\underline{x}), \quad p_{gb} = C_{gb}(\underline{x}), \quad p_{lr} = C_{lr}(\underline{x})$$

Each probability vector has the form:

$$p = [p(safe), p(malicious)]$$

Compute ensemble probabilities:

$$p_{ens}(safe) = [p_{rf}(safe) + p_{gb}(safe) + p_{lr}(safe)] \div 3$$
$$p_{ens}(malicious) = [p_{rf}(malicious) + p_{gb}(malicious) + p_{lr}(malicious)] \div 3$$

Form ensemble output:

$$p_{ens} = [p_{ens}(safe),\ p_{ens}(malicious)]$$

Predict final class:

$\hat{y} = arg_{c \in \{safe, malicious\}} max\ p_{ens}$(c)

Compute confidence score:

$conf = max(p_{ens}(safe),\ p_{ens}(malicious))$

for each feature x_f normalize

$$z_f = (x_f - \mu_f) \div (\sigma_f + \epsilon)$$

Compute feature contribution

$$effect[f] = imp[f].z_f$$

end for

Rank features using:

$|effect[f]||$

Select top 5 contributing features

for each selected feature do
  if $|effect[f]| > 0$ then
    Mark feature as malicious contributor
  else
    Mark feature as safe contributor
  end if
end for

Generate explanations:

• SQL keyword/UNION count → SQL Injection
• JSON depth/NoSQL operators → NoSQL/JSON Injection
• Encoding flags/high entropy → WAF bypass or obfuscation

return y, conf, top contributing features, attack category, and explanations

3.4.2 Logging and WAF-rule candidate creation

The application creates a structured log entry for each analyzed query that includes the query (raw or sanitized), the ensemble probability, the predicted label $\hat{y}$, the top contributing features with their contribution scores and, if relevant, the inferred attack type. If the malicious probability exceeds a high threshold $(p_{ens}(malicious) = 0.95)$, the same information is also written to a dedicated WAF rule candidate store. Each candidate entry contains a representative pattern, usually a regular expression, the associated attack type, the endpoint where it was observed, confidence score, a counter tracking how often it has appeared, and timestamps for first and last occurrence.

When a new high-confidence detection arrives, the system checks whether a matching candidate already exists for the same endpoint and pattern. If so, counters and timestamps are updated. Otherwise, a new candidate entry is created. Over time, this store accumulates patterns that security engineers can review and promote into actual WAF rules. The reason for not automating this promotion yet is that these arbitrary WAF rules may increase processing costs and latency. An increase in the number of rules may even increase managerial complexity, so it is best left to the interest of the security managers to process them post-manual scrutiny.

The relatively small set of 20 features is what makes this entire workflow practical. Feature extraction remains fast enough for real-time use, every feature has a clear security meaning, and the same vector supports detection, explainability, and WAF-candidate generation. As a result, the full pipeline, from query analysis to explanation and rule suggestion, remains consistent and easy to maintain.

3.4.3 Workflow

The general process for managing and classifying incoming requests is shown in Figure 2(a). The pipeline starts with the user submitting their SQL query or API request. The request then goes through an intensive feature extraction and input validation process. The features obtained from this process are then used by the ML detection engine to identify the intent of the request. If the system determines that the request is harmless, it will forward and process it normally. However, if it detects a malicious request, not only will it detect the issue, but it will also provide explanations and suggestions for mitigation. This rigorous approach ensures real-time protection while providing actionable intelligence for security forensics.

Figure 2(b) shows the integration pipeline. Each incoming request is filtered through five layers of existing controls in the modern Web stack,

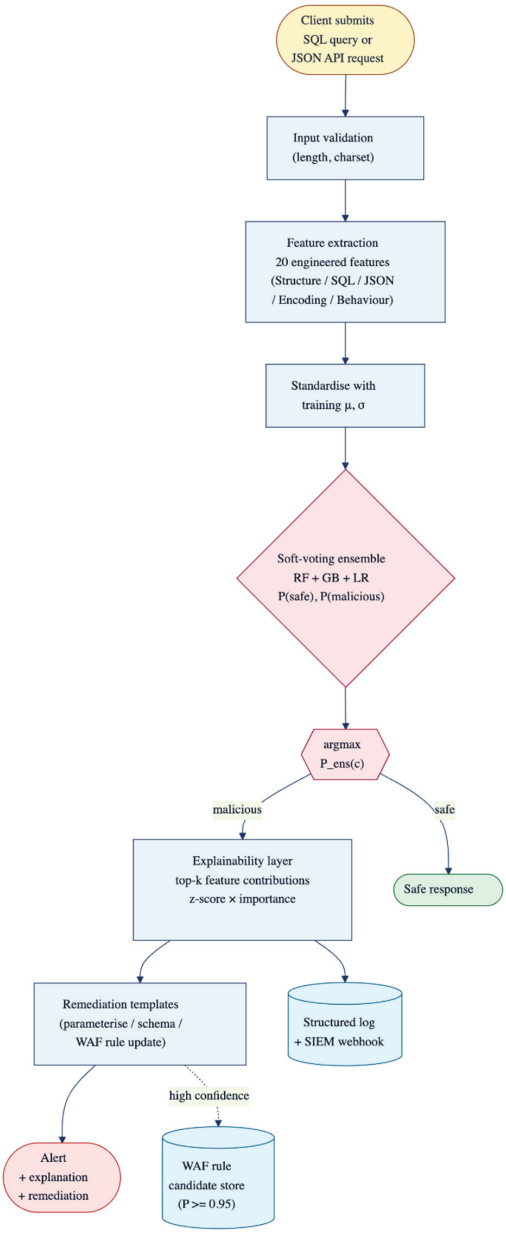

Figure 2(a) End-to-end workflow of the proposed SQLi detection application.

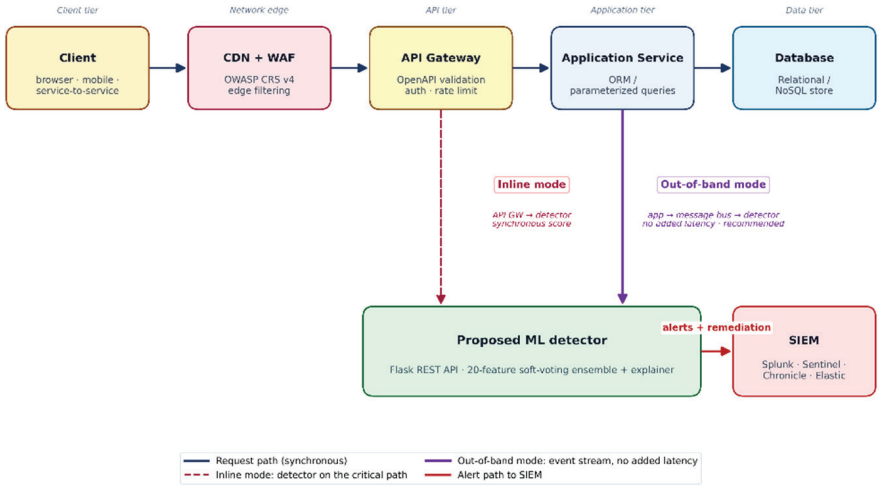

Figure 2(b) Integration pipeline.

i.e., a content delivery network and edge Web application firewall running the OWASP Core Rule Set v4 [57], which filters known malicious traffic; an API gateway that handles authentication, schema validation and rate limiting; the application service layer which processes the business rules and resources handling; and ORM with parametrized binding that aids with database interaction to the Relational/No SQL store. The proposed ML detection pipeline is integrated into the framework in two modes. In the inline mode, it is positioned between the API gateway and the application service, while in the out-of-band mode, the traffic is forwarded by the application layer to this layer via a message bus (such as Apache Kafka). In the inline mode, the detector extracts the 20 engineered features from the request and computes the risk score as shown in Figure 2(a). The latency trade-off incurred in the detection module is countered in the out-of-band mode, wherein a copy of the traffic is sent by the application layer, with the processing happening in parallel. While the inline mode offers better defense, ensuring no malicious request reaches the application layer at all, the out-of-band mode is better due to its responsiveness in production environments (it doesn't offer real-time defense but alerts post the attack). To integrate the module in real-time operations, regardless of the mode, each flagged request is alerted to the connected SIEM platform like Splunk so that these outputs can be reviewed by security engineers, and new rules formulated from these detections, if warranted, can be pushed into edge WAF.

4 Results

4.1 System Performance Overview

Figure 3 shows the system dashboard, which displays performance metrics, the system overview, and identified attacks. The real-time monitoring section processed 90 queries, with 27 threats successfully blocked, with throughput and latency values.

The query analysis page in Figure 4 has two components: a window in which we input the query and an analysis module.

The examples of different types of queries handled are depicted in Figures 5–10. This highlights the comprehensive features offered by the application.

Figure 11 is an example of the federated learning framework, which allows privacy-preserving distributed training on a number of different clients. From the Dashboard View, it is clear to see that there are five active clients currently involved in training, achieving a cumulative accuracy of 99.2% after 10 rounds of training. The model employed here is the FedAvg (Federated Averaging) [58] model, in which the five clients operate on disjoint sets of the corpus. In every round, a local copy of the ensemble is trained by each client on its private local dataset. Post-training, each client then transmits its model weights to a central aggregator. The central global model computes the weighted mean of these weights proportional to its

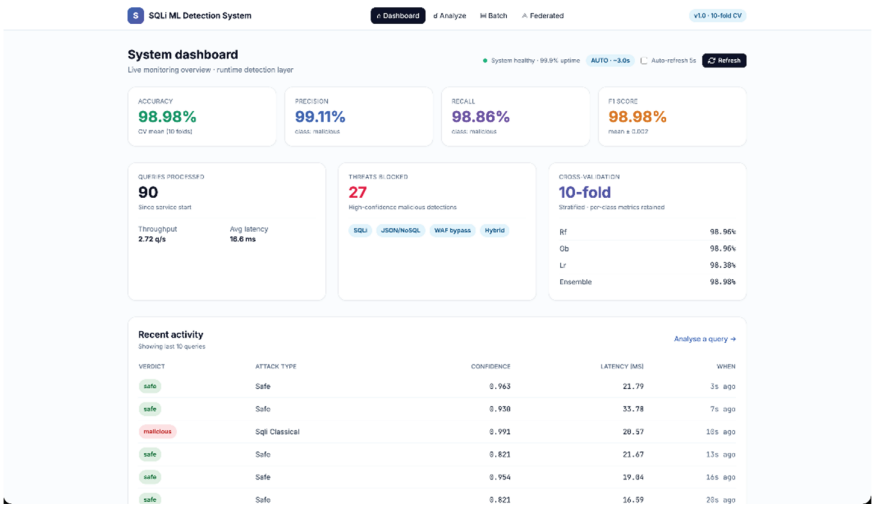

Figure 3 System dashboard providing an overview of real-time performance metrics.

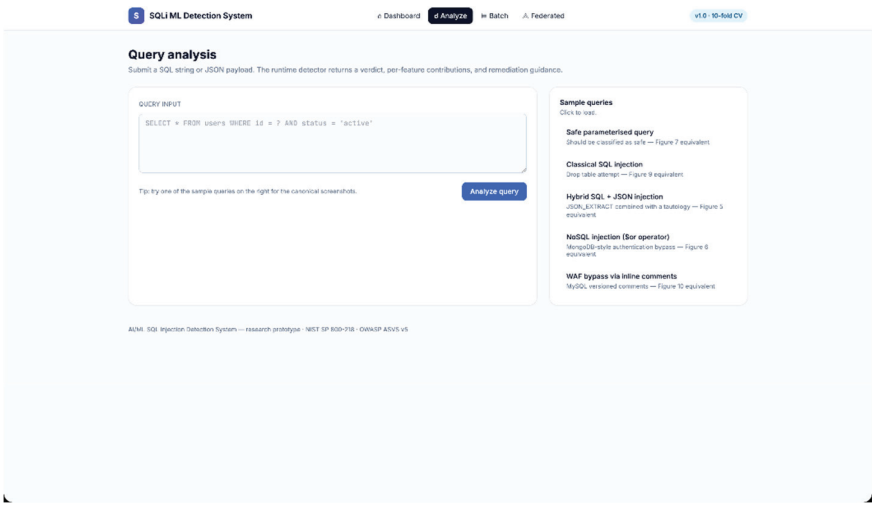

Figure 4 Application landing page and query input interface.

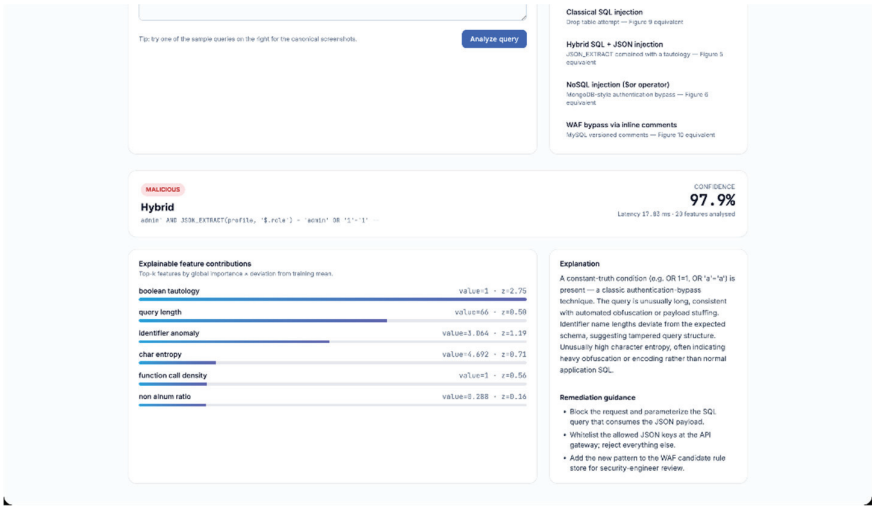

Figure 5 Detection module for hybrid SQL and JSON injection techniques.

dataset size and redistributes the merged model to all clients for the next round. For a total of 10 rounds, the proposed system computes both local and global accuracy. The achieved accuracy here is comparable to the centralized baseline model, making this method extensible to the real-world collaborative WAF deployment environments where sensitive query logs cannot be shared.

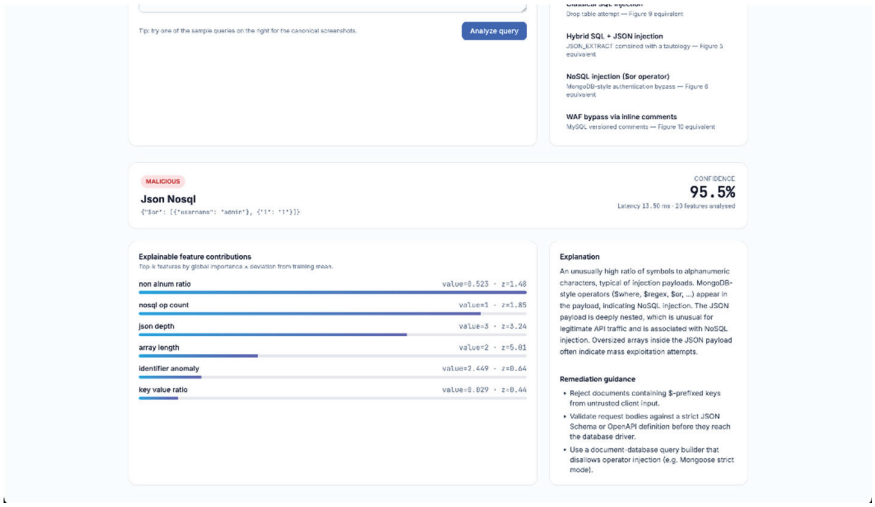

Figure 6 Identification results for high-risk JSON NoSQL injection patterns.

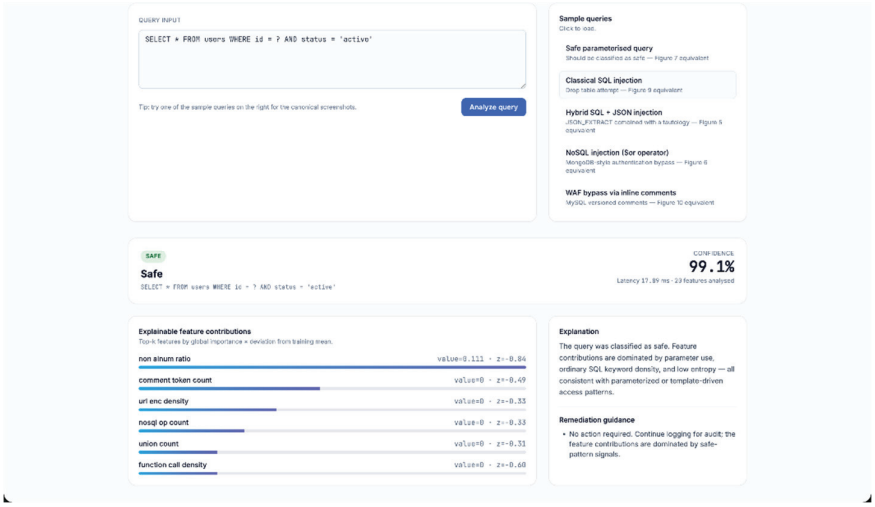

Figure 7 Real-time SQL query analysis interface demonstrating safe query detection.

4.2 Machine Learning Related Results

The classification results are depicted in Figures 12 and 14. Figure 13 provides the most informative features ranked by the Random Forest Model. The contribution of various features in classification is shown in Figure 15.

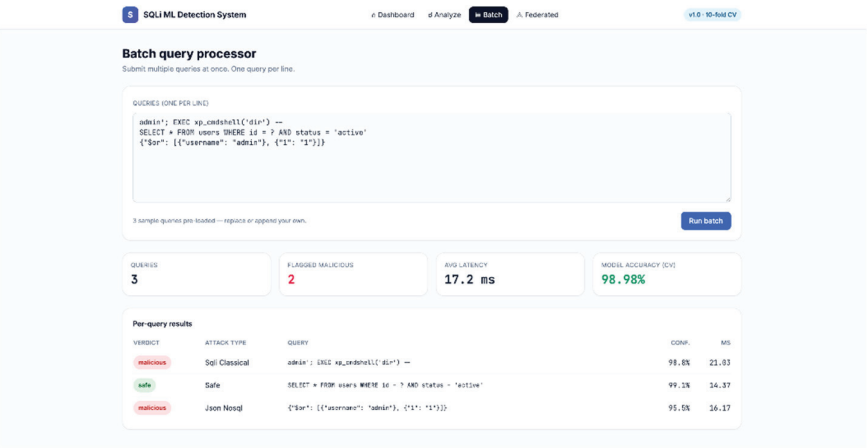

Figure 8 Batch query processing interface for SQL and NoSQL classification.

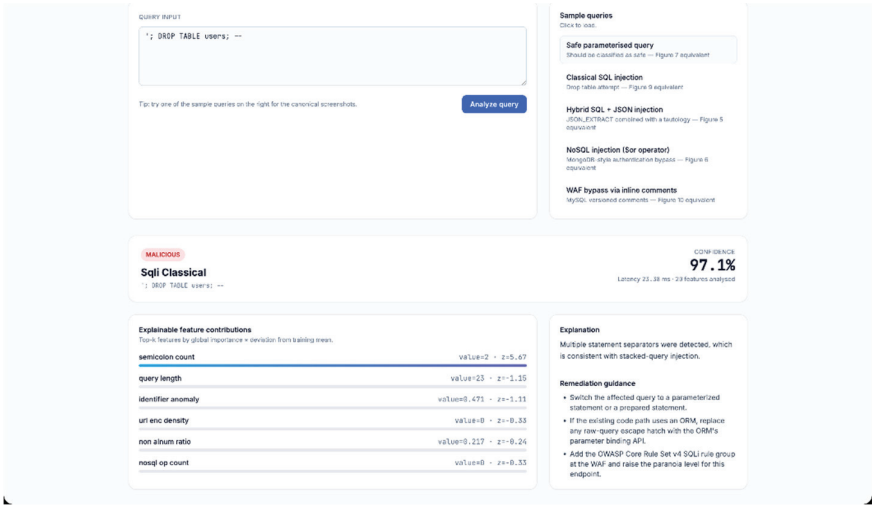

Figure 9 Classic SQLi detection output highlighting high-risk payload patterns.

The heatmap in Figure 16 indicates the detection efficiency of different architectural approaches against six main classes of attacks.

As such, Figure 17 is an illustration of the accuracy trends of five different clients and the global model over 10 rounds of training. From the trends, it is evident that there is a consistent and stable improvement in performance for each and every node in the network as training progresses. In the last round, the global model is able to effectively reach a peak accuracy of 99.2%, thus

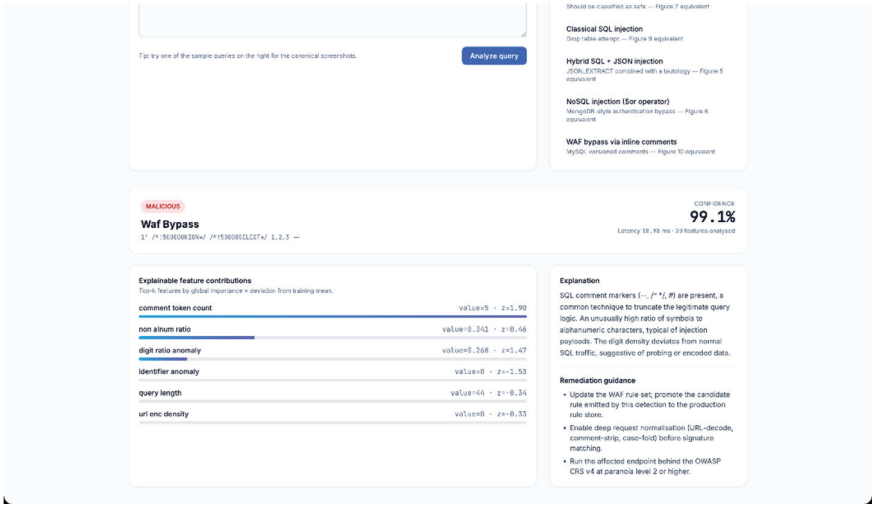

Figure 10 WAF bypass detection panel identifying obfuscated SQLi attempts.

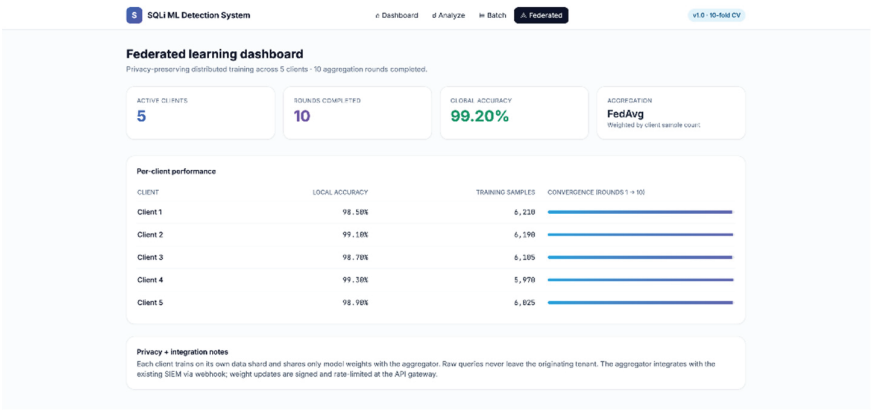

Figure 11 Federated learning dashboard summarizing distributed training and client performance.

demonstrating scalability and efficiency in privacy preservation. This consistent convergence shows that the model can keep local data decentralized while maintaining high-fidelity detection.

Figure 18 illustrates how feature engineering affects model performance by contrasting the outcomes of three different feature sets: baseline (10 features), intermediate (15 features), and proposed (20 features). As feature dimensionality rises, the data shows a notable rising trend in predictive power.

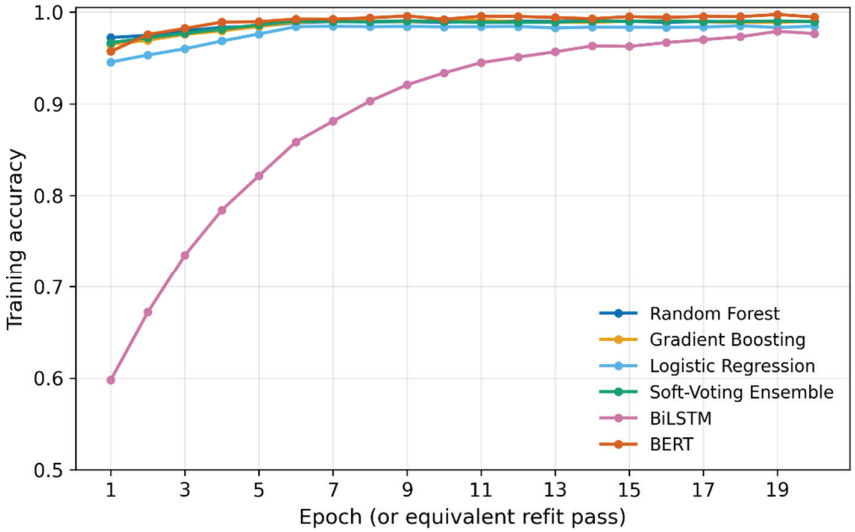

Figure 12 Comparison of training accuracy progression across 20 epochs for various model architectures.

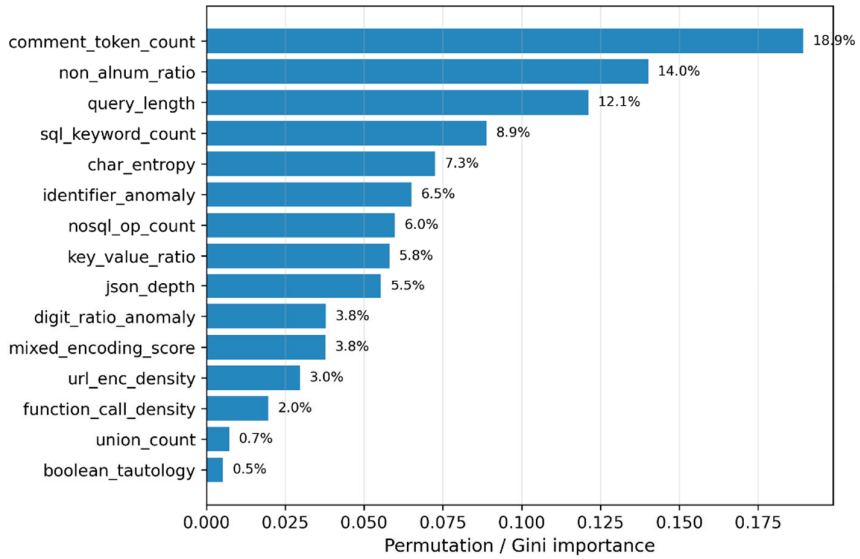

Figure 13 Top 15 most informative features ranked by importance as determined by the Random Forest model.

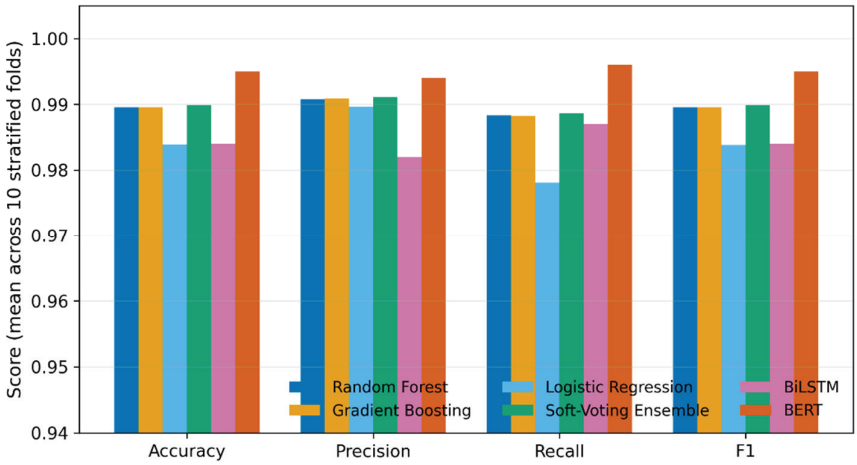

Figure 14 Evaluation of classification metrics across multiple machine learning and deep learning models.

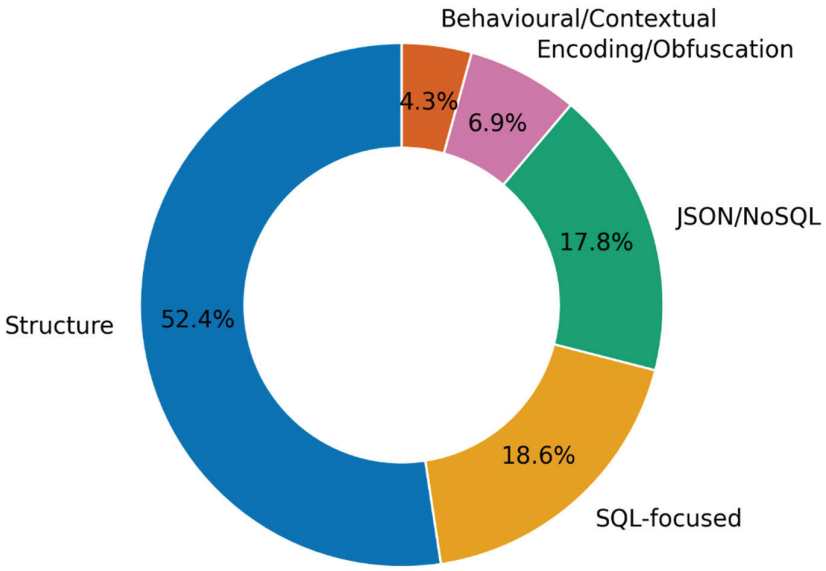

Figure 15 Proportional contribution of various feature engineering categories to detection performance.

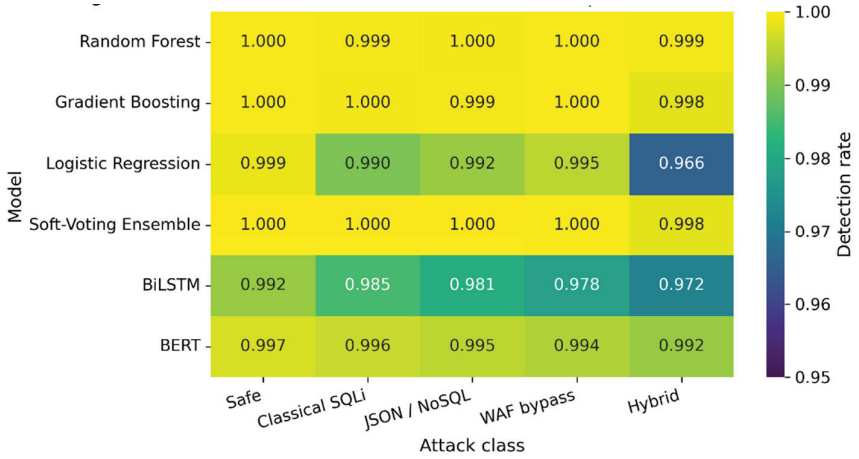

Figure 16 Heatmap of detection rates across various attack classes and machine learning models.

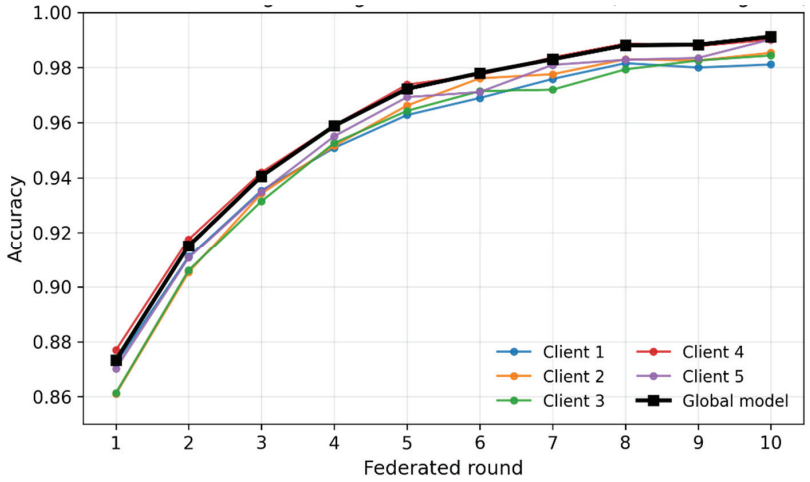

Figure 17 Accuracy convergence across 10 rounds of federated learning for local clients and the global model.

Notably, when compared to the baseline model, the suggested complete feature set achieves a significant 10% gain in both accuracy and F1 score.

Figure 19 shows the trade-offs between model complexity and deployment efficiency. Although BERT performs better overall in terms of accuracy and resilience to adversarial attacks, it is slower in real-time processing. On the other hand, the Random Forest model shows better explainability and

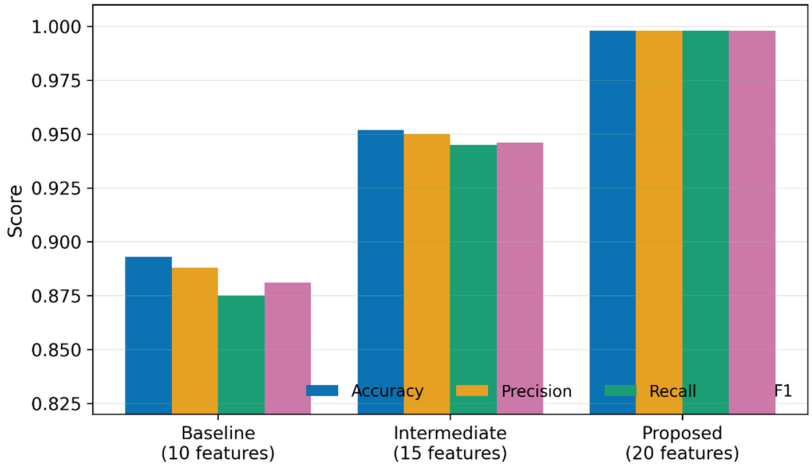

Figure 18 Performance metrics comparison across different feature set complexities.

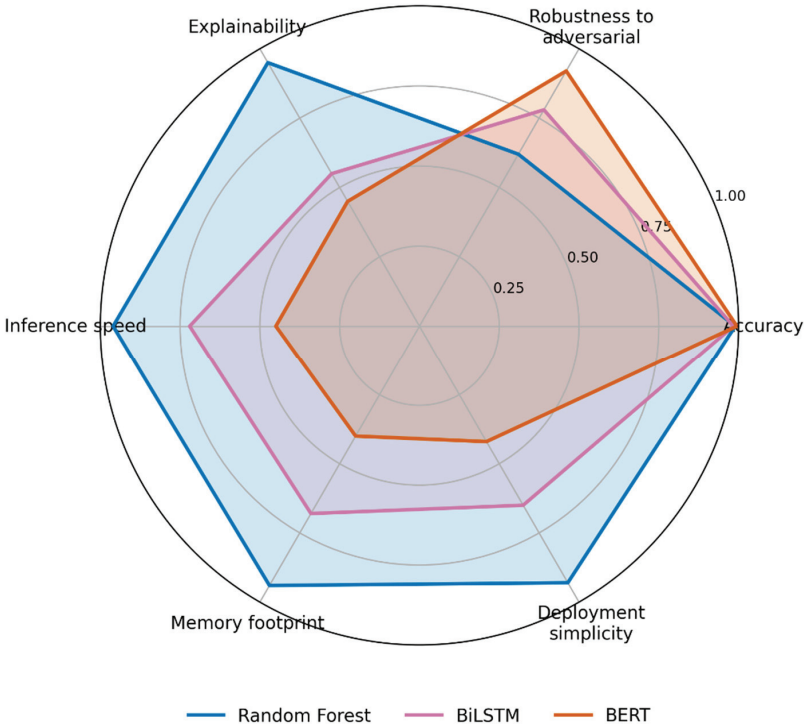

Figure 19 Radar chart comparison of Model Capability across performance and security metrics.

quicker execution durations, which makes it a better option for settings with limited resources. These findings highlight the necessity to strike a compromise between the practical demands of interpretability and privacy and the requirement for high-fidelity detection when choosing an architecture.

5 Discussion and Conclusion

The proposed methodology is a holistic combination of the detection of various injection attacks with lightweight ML classifiers. The system aims to be resilient to the latest attacks with the incorporation of the latest detected threats into its WAF rules. The explainability and the feature engineering layer offer interpretability without requiring high computational power, unlike CNNs and GNNs. Another main contribution here is the introduction of structural data augmentation rules addressing the common data scarcity issues faced in this research. With the explainability module translating feature importance outputs into human-readable rationale and candidate WAF rule suggestions, it proves itself as a practical alternative to both static rule-based WAFs and opaque neural network detectors, suited for modern API-driven application architectures. The proposed work should not be treated as an alternative to secure coding practices like ORMs. However, it presents itself as a lightweight complementary model that can be integrated easily into the runtime environment as a behavioral monitoring layer. A detailed comparative analysis of popular security mechanisms is tabulated in Table 5.

Looking at this in light of existing WAF solutions such as ModSecurity and NAXSI, which are heavily reliant upon manually written rules, this solution can adjust to new attack patterns within the realm of variants of attacks it is not aware of, without the need for additional rules. Furthermore, with a low inference time, this is ideal for a production environment where delays can have negative consequences with respect to uptime. Furthermore, a reduced number of false negatives and a high performance level on multi-type attacks with better generalization capability can be achieved using hybrid architectures [68] combining semantic knowledge with relevant contextually chosen features, which prevents overfitting in a model strongly biased towards tokenization or optimized for a particular dataset. Future research can concentrate on incremental and online learning methods to counter zero-day SQLi variants. More robust adversarial pipelines with automated WAF rule generation can be explored in the future.

One of the immediate future scopes would be addressing the major study limitation, i.e., evaluating the model on live anonymized traffic from

Table 5 Summary comparison of security mechanisms and ML-based monitoring

Technique	Layer	Injection Prevention	Unknown Attack Detection	Major Strength	Major Limitation
Parameterized Queries [59]	App	High	No	Eliminates SQLi through parameter binding	No behavioral anomalies
Prepared Statements [60]	App/DB	High	No	Secure and efficient query execution	Limited to pre-defined queries
ORM Frame-works [61]	App	Medium	Low	Reduces coding errors	Unsafe when raw queries are used
Stored Procedures [62]	DB	High	Low	Controlled database execution	Dynamic SQL may be vulnerable
DB Packages/Function	DB	High	Low	Strong access control	Limited attack visibility
API Gateway [63]	API	Medium	Low	Authentication, rate limiting	No query-level inspection
Schema Validation [64]	API/App	Medium	Low	Blocks malformed inputs	Cannot detect semantic attacks
Cloud-native WAF [65][66]	Network	High (Known)	Medium	Signature-based attacks blocking	Limited against zero-day attacks
ML Monitoring Layer [67]	Cross-layer	Adaptive	High	Detects anomalies, insider threats, and data exfiltration	Requires training and may produce false positives

real-world applications. Furthermore, the use of LLM models to synthesize traffic, and measures to prune the model with reduced latencies can be looked at. In summary, a WAF assisted with ML can go beyond fixed rules and think in an adaptive and context-dependent manner when it comes to protecting against intrusions. The model is meant to be dynamic with regard to evolving query trends.

Author Statement

Nisha P. Shetty: Conceptualization, Data Curation, Methodology, Software, Formal Analysis, Writing – Original Draft.

Vinayak Kothari: Methodology, Validation, Investigation, Writing – Review & Editing.

Eva Hemantkumar Shah: Data Curation, Formal Analysis, Visualization, Writing – Review & Editing.

Prashanth J. Kumar: Resources, Supervision, Validation, Writing – Review & Editing.

Jayashree Shetty: Conceptualization, Supervision, Project Administration, Funding Acquisition, Writing – Review & Editing.

All authors have read and approved the final manuscript

References

[1] ExtraHop Networks, Inc., "SQL Injection (SQLi) Attacks: Definition, Examples, and Prevention," ExtraHop. [Online]. Available: https://www.extrahop.com/resources/attacks/sqli. [Accessed: 09-Jan-2026].

[2] OWASP, "OWASP Top 10:2025," The Open Web Application Security Project, 2025. [Online]. Available: https://owasp.org/Top10/2025/. [Accessed: 09-Jan-2026].

[3] J. Erickson, "What Is JSON?," Oracle India, Apr. 4, 2024. [Online]. Available: https://www.oracle.com/in/database/what-is-json/. [Accessed: 09-Jan-2026].

[4] S. L. Bjeladinovic, M. S. Asanovic, and N. M. Gospic, "An analysis of the JSON functionalities evolution across different versions of Oracle relational DBMS," 2021. https://www.eventiotic.com/eventiotic/files/Papers/URL/ecfe7c82-5f54-416e-929d-d52ad9e993e4.pdf.

[5] N. Moshe, "{JS-ON: Security-OFF}: Abusing JSON-Based SQL to Bypass WAF," Claroty Team82 Research, Dec. 8, 2022. [Online]. Available: https://claroty.com/team82/research/js-on-security-off-abusing-json-based-sql-to-bypass-waf. [Accessed: 09-Jan-2026].

[6] M. A. D. Varma, G. S. Vaasist, B. C. Reddy, M. P. Reddy, and R. Nair, "Intrusion detection system using signature and anomaly based algorithm," 2025 International Conference on Inventive Computation Technologies (ICICT), Kirtipur, Nepal, 2025, pp. 838–842, doi: 10.1109/ICICT64420.2025.11004762.

[7] K. Mithran and C. Gopi, "Anomaly detection in IoT sensor networks using machine learning," 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), Kochi, India, 2022, pp. 1–7, doi: 10.1109/IC3SIS54991.2022.9885575.

[8] K. Mani and A. K. B. Shenoy, "Machine learning models in Web applications: A comprehensive review," ICT Express, vol. 11, no. 6, 2025, pp. 1110–1119, ISSN 2405-9595, https://doi.org/10.1016/j.icte.2025.09.001.

[9] D. Lu, J. Fei, and L. Liu, "A semantic learning-based SQL injection attack detection technology," Electronics, vol. 12, p. 1344, 2023. https://doi.org/10.3390/electronics12061344.

[10] R.-T. Lo, W.-J. Hwang, and T.-M. Tai, "SQL injection detection based on lightweight multi-head self-attention," Applied Sciences, vol. 15, no. 2, p. 571, 2025. https://doi.org/10.3390/app15020571.

[11] C. Caudill and D. Sury, "Visualize AI/ML model results using Flask and AWS Elastic Beanstalk," Amazon Web Services (AWS) Prescriptive Guidance, [Online]. Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/visualize-ai-ml-model-results-using-flask-and-aws-elastic-beanstalk.html. [Accessed: Mar. 21, 2026]..

[12] G. Lazrek, K. Chetioui, Y. Balboul, S. Mazer, and M. El Bekkali, "An RFE/Ridge-ML/DL based anomaly intrusion detection approach for securing IoMT system", Results in Engineering, vol. 23, p. 102659, 2024. https://doi.org/10.1016/j.rineng.2024.102659.

[13] A. Rao, D. Khankhoje, U. Namdev, C. Bhadane, and D. Dongre, "Insights into NoSQL databases using financial data: A comparative analysis," Procedia Comput. Sci., vol. 215, pp. 8–23, 2022.

[14] K. S. Fathi, S. Barakat, and A. Rezk, "An effective SQL injection detection model using LSTM for imbalanced datasets," Computers & Security, vol. 153, p. 104391, 2025. https://doi.org/10.1016/j.cose.2025.104391.

[15] A. A. Mustapha, A. S. Udeh, T. A. Ashi, O. S. Sobowale, M. J. Akinwande, and A. O. Oteniara, "Comprehensive review of machine learning models for SQL injection detection in e-commerce," World Journal of Advanced Research and Reviews, vol. 23, no. 1, pp. 451–465, July 2024, doi: 10.30574/wjarr.2024.23.1.2004.

[16] S. Pasini et al., "Evaluating and improving the robustness of security attack detectors generated by LLMs," Empirical Software Engineering, vol. 31, no. 2, p. 35, 2026.

[17] N. S. Dasari et al., "Enhancing SQL injection detection and prevention using generative models," 2025. arXiv:2502.04786.

[18] J. Zulu, B. Han, I. Alsmadi, and G. Liang, "Enhancing machine learning based SQL injection detection using contextualized word embedding." Proceedings of the 2024 ACM Southeast Conference (ACMSE '24). Association for Computing Machinery, New York, NY, USA, pp. 211–216, 2024. https://doi.org/10.1145/3603287.3651187.

[19] H. Sun, Y. Du, and Q. Li, "Deep learning-based detection technology for SQL injection research and implementation," Appl. Sci., vol. 13, p. 9466, 2023. https://doi.org/10.3390/app13169466.

[20] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Deep learning architecture for detecting SQL injection attacks based on RNN autoencoder model," Mathematics, vol. 11, no. 15, p. 3286, 2023. https://doi.org/10.3390/math11153286.

[21] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Detection of SQL injection attack using machine learning techniques: A systematic literature review," Journal of Cybersecurity and Privacy, vol. 2, no. 4, pp. 764–777, 2022. https://doi.org/10.3390/jcp2040039.

[22] F. K. Alarfaj and N. A. Khan, "Enhancing the performance of SQL injection attack detection through probabilistic neural networks," Applied Sciences, vol. 13, no. 7, p. 4365, 2023. https://doi.org/10.3390/app13074365.

[23] H. Xu, G. Pang, Y. Wang, and Y. Wang, "Deep Isolation Forest for anomaly detection," IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 12, pp. 12591–12604, 2023.

[24] A. Paul, V. Sharma, and O. Olukoya, "SQL injection attack: Detection, prioritization & prevention," Journal of Information Security and Applications, vol. 85, p. 103871, 2024. https://doi.org/10.1016/j.jisa.2024.103871.

[25] G. Floris et al., "ModSec-AdvLearn: Countering adversarial SQL injections with robust machine learning," IEEE Transactions on Information

Forensics and Security, vol. 20, pp. 6693–6705, 2025, doi: 10.1109/TIFS.2025.3583234.

[26] K. Tasdemir, R. Khan, F. Siddiqui, S. Sezer, F. Kurugollu, S. B. Yengec-Tasdemir, and A. Bolat, "Advancing SQL injection detection for high-speed data centers: A novel approach using cascaded NLP," 2023, arXiv:2312.13041.

[27] D. Muduli et al., "SIDNet: A SQL injection detection network for enhancing cybersecurity," IEEE Access, vol. 12, pp. 176511–176526, 2024, doi: 10.1109/ACCESS.2024.3502293.

[28] N. Gandhi, J. Patel, R. Sisodiya, N. Doshi, and S. Mishra, "A CNN-BiLSTM based approach for detection of SQL injection attacks," 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, United Arab Emirates, pp. 378–383, 2021, doi: 10.1109/ICCIKE51210.2021.9410675.

[29] Z. Gui, E. Wang, B. Deng, M. Zhang, Y. Chen, S. Wei, W. Xie, and B. Wang, "SqliGPT: Evaluating and Utilizing Large Language Models for Automated SQL Injection Black-Box Detection" Applied Sciences, vol. 14, no. 16, p. 6929, 2024. https://doi.org/10.3390/app14166929.

[30] Z. Xia, J. Shao, L. Yu, J. Sun, X. Yang, H. Xu, C. Liu, and J. Ren, "SQL injection attack detection method based on textCNN," Proc. SPIE 13222, International Conference on Signal Processing and Communication Security (ICSPCS 2024), 1322219 (22 July 2024). https://doi.org/10.1117/12.3038647.

[31] N. Thalji, A. Raza, M. S. Islam, N. A. Samee, and M. M. Jamjoom, "AE-Net: Novel autoencoder-based deep features for SQL injection attack detection," IEEE Access, vol. 11, pp. 135507–135516, 2023, doi: 10.1109/ACCESS.2023.3337645.

[32] T.-T.-H. Le, Y. Hwang, C. Choi, R. W. Wardhani, D. S. C. Putranto, and H. Kim, "Enhancing structured query language injection detection with trustworthy ensemble learning and boosting models using local explanation techniques," Electronics vol. 13, no. 22, p. 4350, 2024. https://doi.org/10.3390/electronics13224350.

[33] Sajid576, "SQL Injection Dataset," Kaggle, 2021. [Online]. Available: https://www.kaggle.com/datasets/sajid576/sql-injection-dataset. [Accessed: 09-Jan-2026].

[34] swisskyrepo, "PayloadsAllTheThings," GitHub repository, 18 Oct. 2016-present. [Online]. Available: https://github.com/swisskyrepo/PayloadsAllTheThings. [Accessed: 09-Jan-2026].

[35] capnmav77, "No-SQL_Gen: No-SQL Injection Dataset," GitHub repository, 17 Aug. 2023. [Online]. Available: https://github.com/capnmav77/No-SQL_Gen. [Accessed: 09-Jan-2026].

[36] OWASP, "SQL Injection Prevention Cheat Sheet," OWASP Cheat Sheet Series, 2025. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. [Accessed: 09-Jan-2026].

[37] OWASP, "Query Parameterization Cheat Sheet," OWASP Cheat Sheet Series, 2025. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html. [Accessed: 09-Jan-2026].

[38] PortSwigger Ltd., "Burp Suite: Web application security testing software," 2003-present. [Online]. Available: https://portswigger.net/burp. [Accessed: 09-Jan-2026].

[39] sqlmap "Automatic SQL injection and database takeover tool sqlmap.org," 2006-present. [Online]. Available: https://sqlmap.org/. [Accessed: 09-Jan-2026].

[40] Pandas Development Team, "Pandas: Python Data Analysis Library," 2008-present. [Online]. Available: https://pandas.pydata.org/. [Accessed: 09-Jan-2026].

[41] NumPy Developers, "NumPy," 2006-present. [Online]. Available: https://numpy.org/. [Accessed: 09-Jan-2026].

[42] Python Software Foundation, "re — Regular expression operations," Python 3. [Online]. Available: https://docs.python.org/3/library/re.html. [Accessed: 09-Jan-2026].

[43] Python Software Foundation, "base64 – Encode and decode with base64," Python 3. [Online]. Available: https://docs.python.org/3/library/base64.html. [Accessed: 09-Jan-2026].

[44] Python Software Foundation, "urllib.parse – Parse URLs into components," Python 3. [Online]. Available: https://docs.python.org/3/library/urllib.parse.html. [Accessed: 09-Jan-2026].

[45] Python Software Foundation, "random – Generate pseudo-random numbers," Python 3. [Online]. Available: https://docs.python.org/3/library/random.html. [Accessed: 09-Jan-2026].

[46] Python Software Foundation, "collections – Container datatypes: Counter class," Python 3. [Online]. Available: https://docs.python.org/3/library/collections.html#collections.Counter. [Accessed: 09-Jan-2026].

[47] JSON:API, "Implementations," 2025. [Online]. Available: https://jsonapi.org/implementations/. [Accessed: 09-Jan-2026].

[48] "Nested objects in real apps," freeCodeCamp Forum, Jul. 8, 2022. [Online]. Available: https://forum.freecodecamp.org/t/nested-objects-in-real-apps/526638. [Accessed: 09-Jan-2026].

[49] "sqlparse," PyPI, 2025. [Online]. Available: https://pypi.org/project/sqlparse/. [Accessed: 09-Jan-2026].

[50] JSON Schema, "Documentation," 2025. [Online]. Available: https://json-schema.org/docs. [Accessed: 09-Jan-2026].

[51] Scikit-Learn Developers, "scikit-learn: Machine Learning in Python," 2007-present. [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 09-Jan-2026].

[52] S. Xia, F. Zhang, and C. Zhang, "A Gradient Boosting based classification technique for assisted prediction algorithm research," 2023 IEEE International Conference on Image Processing and Computer Applications (ICIPCA), Changchun, China, pp. 371–375, 2023, doi: 10.1109/ICIPCA59209.2023.10257866.

[53] J. K. Jaiswal and R. Samikannu, "Application of Random Forest algorithm on feature subset selection and classification and regression," 2017 World Congress on Computing and Communication Technologies (WCCCT), Tiruchirappalli, India, pp. 65–68, 2017, doi: 10.1109/WCCCT.2016.25.

[54] V. Madaan, N. Sharma, R. S. Bangari, and S. Aluvala, "Fraudulent job posting detection using Logistic Regression," 2024 International Conference on Information Science and Communications Technologies (ICISCT), Seoul, Korea, Republic of, pp. 1–6, 2024, doi: 10.1109/ICISCT64202.2024.10956218.

[55] Amriana, A. A. Ilham, A. Achmad, and Y. Yusran, "Ensemble soft-voting model for classification optimization of medicinal plants leaves," 2023 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT), Malang, Indonesia, pp. 147–152, 2023, doi: 10.1109/COMNETSAT59769.2023.10420635.

[56] V. Agate, A. De Paola, S. Drago, P. Ferraro, and G. L. Re, "Enhancing IoT network security with concept drift-aware unsupervised threat detection," 2024 IEEE Symposium on Computers and Communications (ISCC), Paris, France, pp. 1–6, 2024, doi:10.1109/ISCC61673.2024.10733733.

[57] OWASP Core Rule Set (CRS), "OWASP CRS Project – The 1st Line of Defense," OWASP Foundation. [Online]. Available: https://coreruleset.org/. [Accessed: Jun. 3, 2026].

[58] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," Proc. 20th International Conf. on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, FL, vol. 54, pp. 1273–1282, 2017.

[59] R. F. Sidik, S. N. Yutia, and R. Z. Fathiyana, "The effectiveness of parameterized queries in preventing," Proceedings of the International Conference on Enterprise and Industrial Systems (ICOEINS 2023), Cham, Switzerland: Springer Nature, p. 204, 2023.

[60] J. Clarke, SQL Injection Attacks and Defense, 2nd ed. Burlington, MA, USA: Syngress/Elsevier, 2012.

[61] C. Bauer and G. King, Java Persistence with Hibernate, 2nd ed. Shelter Island, NY, USA: Manning Publications, 2015.

[62] M. Howard and D. LeBlanc, Writing Secure Code, 2nd ed. Redmond, WA, USA: Microsoft Press, 2003.

[63] Oracle Corporation, Oracle Database Security Guide. Austin, TX, USA: Oracle Corporation, 2024.

[64] C. Richardson, Microservices Patterns: With Examples in Java. Shelter Island, NY, USA: Manning Publications, 2018.

[65] F. Pezoa, J. L. Reutter, F. Suárez, M. Ugarte, and D. Vrgoč, "Foundations of JSON Schema," Proc. 25th Int. Conf. World Wide Web (WWW), Montréal, QC, Canada, pp. 263–273, 2016.

[66] I. Ristić, ModSecurity Handbook: The Complete Guide to the Popular Open Source Web Application Firewall. London: Feisty Duck, 2023.

[67] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," Proc. IEEE Symp. Security and Privacy, Berkeley, CA, USA, pp. 305–316, 2010.

[68] A. K. Mousa and M. N. Abdullah, "An improved deep learning model for DDoS detection based on hybrid stacked autoencoder and checkpoint network," Future Internet, vol. 15, no. 8, art. no. 278, 2023, doi: 10.3390/fi15080278. [Online]. Available: https://www.mdpi.com/1999-5903/15/8/278. [Accessed: 09-Jan-2026].

Biographies

Nisha P. Shetty received the B.E. and M.Tech. degrees in Computer Science and Engineering from Visvesvaraya Technological University (VTU), India, in 2013 and 2015, respectively, and the Ph.D. degree in Data Privacy and Security from Manipal Institute of Technology (MIT), Manipal Academy of Higher Education (MAHE), in 2023. She is currently an Associate Professor with the School of Computer Engineering (SCE), MIT, MAHE. She is a dedicated Academician. She is a Faculty Advisor for Project Cryptonite, one of the top-performing student projects at MIT, which has won accolades nationally and internationally. Her research interests include data privacy and security, with a focus on privacy-preserving frameworks in social networks, deep learning for medical diagnostics, and enhanced security measures for online data.

Vinayak Kothari is a final-year B.Tech. student in Information Technology at Manipal Institute of Technology, Manipal, India. His research interests include cybersecurity, Web security, and software engineering. He is currently an intern at Tessell, where he contributes to technology-driven solutions in industry. He is passionate about applying research and innovation to address real-world computing and security challenges and intends to pursue a career in the technology industry.

Eva Hemantkumar Shah is currently pursuing the B.Tech. degree in Information Technology (Honors) with a minor in cybersecurity at the Manipal Institute of Technology, Manipal, Karnataka, India, with an expected graduation in 2026. She is currently a Software Engineer at Microsoft, India. She previously completed internships at Microsoft, CNLABS, and Give. Her work includes various research projects applying machine learning and deep learning to security domains. Her current research interests include machine learning and cybersecurity. Shah is a member of the Institute of Engineering and Technology (IET). She received the IET Prize 2024 for showing outstanding performance in her engineering course.

Prashanth J. Kumar is a Bachelor of Technology student in Information Technology at Manipal Institute of Technology, India. He has served as team leader of Cryptonite, one of India's top Capture-the-Flag (CTF) teams and ethical hacking projects. His interests lie in Web-based attacks, cloud security, and embedded systems, and he is passionate about exploring and advancing cybersecurity.

Jayashree Shetty earned the B.E. degree in Computer Science and Engineering from CMRIT, Bangalore, India, and the M.Tech. degree in Computer Science and Engineering from SDIT, Mangalore. From 2014 to 2016, she worked as an Assistant Professor in the Department of Computer Science and Engineering, SDIT, Mangalore. In 2016, she joined the Manipal Institute of Technology as an Assistant Professor and is currently employed in the same institution. Her research interests include the field of medical image processing, artificial intelligence, and machine learning.