

Article

Deployment-Oriented Multi-Embedding Machine Learning Framework for SQL Injection Detection and Prevention in a Web Application Firewall

Sahar Saadallah Ahmed * and Mohand Lokman Al dabag *

Technical Engineering College, Northern Technical University, Mosul 41001, Iraq

* Correspondence: sahar_saadallah@ntu.edu.iq (S.S.A.); mohandaldabag@ntu.edu.iq (M.L.A.d.)

Abstract

Structured Query Language injection (SQLi) remains a persistent threat to web applications due to the obfuscation, diversity, and evolving structure of malicious payloads, which limit the effectiveness of conventional rule and signature-based Web Application Firewalls (WAFs). Although prior studies have reported high detection performance using individual feature extraction methods or offline classification models, limited work has addressed deployment-oriented SQLi prevention through an integrated real-time inspection framework. This paper proposes a Machine Learning (ML)-based SQLi detection and prevention framework that combines hybrid feature representation, supervised dimensionality reduction, Genetic Algorithm (GA)-based hyperparameter optimization, and real-time WAF validation. Multiple public SQLi datasets were merged, cleaned, and deduplicated to improve exposure to diverse query patterns. SQL queries were encoded using Term Frequency–Inverse Document Frequency (TF-IDF), Word2Vec, and FastText features, which were fused and transformed through a Supervised Autoencoder into a compact discriminative representation. GA was then employed to optimize multiple classifiers, including Random Forest (RF), Support Vector Machine (SVM), Decision Tree (DT), and Multi-Layer Perceptron (MLP). The MLP achieved the best overall performance, with an accuracy of 0.998681. The optimized model was deployed within a lightweight Flask-based WAF for real-time Hypertext Transfer Protocol (HTTP) request inspection and malicious input blocking. SQLMap v1.8.4-based robustness testing and runtime analysis demonstrate that the proposed framework provides effective SQLi prevention with practical deployment efficiency beyond conventional offline benchmark evaluation.

Keywords: SQL injection detection; web application firewall (WAF); hybrid feature fusion; supervised autoencoder; genetic algorithm optimization; word embedding; real-time web security; SQLMap-based robustness evaluation



Academic Editor: Leandros Maglaras

Received: 17 May 2026

Revised: 30 May 2026

Accepted: 2 June 2026

Published: 5 June 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

In recent years, the growing dependence on web-based systems for processing sensitive information has substantially increased exposure to cyberattacks. In interconnected industrial and cyber-physical environments, web-based attacks such as SQL injection, DDoS, and cross-site scripting can lead to sensitive data loss, operational disruption, and financial damage [1]. The Open Web Application Security Project (OWASP) Top 10 reports (2017 and 2021) consistently highlight injection-related risks as critical web application security threats. Notably, the Injection category moved from A1 (2017) to A03 (2021), while

SQL injection remains a prominent injection vector discussed within this category [2]. SQL injection exploits improper handling and validation of user input in SQL statements, enabling attackers to alter the intended query logic and manipulate database queries, which may result in authentication bypass, unauthorized data access, and unauthorized insertion, modification, or deletion of database records [3,4]. Despite long-standing awareness, effective defense remains challenging because SQLi payloads are highly variable and can be deliberately obfuscated to evade static detection mechanisms [5,6].

In operational settings, SQLi protection is commonly enforced through Web Application Firewalls (WAFs), which inspect incoming HTTP requests before they reach application logic [7]. Conventional WAFs rely largely on rule- and signature-based inspection (e.g., keyword matching and regular expressions). While efficient and interpretable, such mechanisms can be brittle against evasive SQLi variants, where alternate encodings, inline comments, case variation, and query rewriting may bypass signatures and impose a practical trade-off between tightening rules (increasing false positives) and relaxing them (increasing missed attacks) [8].

To mitigate these limitations, recent research has increasingly shifted toward data-driven web attack detection, where incoming HTTP traffic is represented through selected features and parameters, and then classified by pre-trained deep learning models to support real-time allow/block decisions [9]. Common text representations include sparse lexical features such as TF-IDF [10], dense semantic embeddings such as Word2Vec [11], and subword-aware embeddings such as FastText [12]. In addition, autoencoder-based representation learning has been explored for SQLi detection to extract compact and meaningful deep features from SQL query data, which can improve downstream classification performance [13]. However, a considerable portion of the literature primarily reports results under offline or benchmark-based evaluation, without fully bridging the gap between experimental performance and integration within realistic request-inspection workflows [14].

Despite these advances, practical integration into real-time web protection environments remains limited. Existing studies still tend to address representation learning, classifier optimization, and deployment validation separately, which means that strong benchmark accuracy does not necessarily translate into effective protection capability in realistic request-inspection workflows. To address this limitation, the present study proposes a deployment-oriented framework that integrates hybrid SQL query representation, supervised autoencoder-based dimensionality reduction, GA-based optimization, and real-time WAF validation within a unified practical pipeline.

Accordingly, the main contributions of this study are as follows: (1) constructing a unified multi-source SQLi dataset to reduce dataset-specific bias and improve exposure to diverse attack patterns; (2) developing a hybrid representation that combines TF-IDF, Word2Vec, and FastText to capture complementary lexical, contextual, and subword-level characteristics of SQL queries; (3) employing supervised autoencoder-based dimensionality reduction to obtain a compact latent space aligned with the classification objective; (4) applying GA-based hyperparameter optimization under a unified validation protocol; and (5) extending the evaluation beyond benchmark classification through real-time WAF deployment, SQLMap-based robustness testing, and runtime response-time/throughput analysis.

2. Related Work

2.1. Deployment-Oriented SQL Injection Detection

Research on SQL injection (SQLi) detection has increasingly expanded beyond offline benchmark classification toward deployment-oriented protection settings that better reflect operational web security requirements. Within this direction, Peralta-Garcia et al. [15] examined SQL injection detection in web microservices using conventional machine learning

classifiers, including Random Forest, Decision Tree, and Support Vector Machine, on a public dataset of 22,764 records. Their study is particularly relevant because it combined classifier evaluation with a microservices-based deployment architecture for request validation and attack detection. Among the tested models, Random Forest achieved the best results, with 99% accuracy and precision, 97% recall, and 98% F1-score.

Similarly, Ogini et al. (2022) [16] developed a deep learning-based feedforward neural network (FFNN) for SQL injection detection with Flask deployment. Their approach used a Kaggle dataset of approximately 30,635 queries, together with preprocessing, CountVectorizer-based vectorization, TF-IDF feature selection, and FFNN training, and reported an accuracy of 97.65%. This study is relevant because it supports the practical feasibility of deploying learning-based SQLi detectors within lightweight web protection workflows.

Taken together, these studies highlight the growing importance of deployment realism in SQLi detection research. At the same time, they show that practical validation has often been implemented using conventional classifiers or single-model pipelines, leaving room for more integrated representation-driven approaches.

2.2. Feature Representation for SQLi Detection

Pramono et al. (2024) [17] compared Naïve Bayes and Support Vector Machine using a Kaggle dataset, standard preprocessing, and TF-IDF feature extraction under three different data-split scenarios. Their results reached an accuracy of 96.67%, showing that conventional text preprocessing with classical classifiers can still provide competitive SQLi detection performance. However, the study remained limited to traditional feature engineering and did not incorporate richer representation learning, hybrid embedding fusion, or deployment-oriented validation.

A second important direction in the literature concerns the representation of SQL queries for machine learning-based attack detection. In this context, Bakır (2025) [18] introduced UniEmbed, a unified machine learning framework for detecting XSS and SQL injection attacks using a hybrid representation derived from Word2Vec, the Universal Sentence Encoder (USE), and FastText. Experimental results showed that the MLP classifier achieved particularly strong SQLi detection performance, with accuracy exceeding 0.9980 across two SQLi datasets. This study is especially relevant because it demonstrates that multi-embedding fusion can outperform individual feature extraction methods. However, the framework remained primarily benchmark-focused and did not extend to WAF-oriented deployment or runtime validation.

Taken together, these studies show that SQL query representation is a major factor in SQLi detection effectiveness. Because malicious queries often contain lexical, contextual, and subword-level patterns simultaneously, especially under obfuscation or structural variation, richer feature modeling has become increasingly important. Accordingly, hybrid representation has emerged as a promising direction for improving generalization and robustness in SQLi detection.

2.3. Autoencoder and Semantic Representation Learning for SQLi Detection

In parallel with feature-fusion research, representation learning has been explored more explicitly in deep learning-based SQLi detection. Alghawazi et al. (2023) [19] proposed an RNN autoencoder-based architecture in which latent representations were learned and subsequently used for SQLi classification. This study is closely related to the present work because it demonstrates the usefulness of autoencoder-based dimensionality reduction in reducing feature redundancy and enhancing discriminative learning for SQLi detection. However, the approach mainly relies on a single deep representation-learning

pathway, rather than a framework that first integrates handcrafted and embedding-based textual representations and then transforms them into a task-aligned latent space.

Thalji et al. (2023) [13] proposed AE-Net, an autoencoder-based feature engineering framework for SQL injection detection. Using a dataset of 46,392 SQL queries, the study extracted high-level deep features and reported an accuracy of 0.99 with the extreme gradient boosting classifier. The study confirms the value of autoencoder-derived representations in SQLi detection, but remains mainly benchmark-focused without hybrid feature fusion or deployment-oriented validation.

Viewed together, these studies indicate that learned representations, whether through autoencoders or contextual semantic models, have become increasingly important in SQLi detection. At the same time, the literature remains divided between model-centric semantic solutions and more traditional feature-based pipelines, suggesting a continuing need for integrative approaches that combine representational diversity with compact latent learning.

2.4. Transformer-Based SQLi Detection

Recent SQLi detection studies have increasingly explored transformer-based and attention-based architectures to improve the semantic modeling of malicious query patterns. Ammar and Alharbi [20] fine-tuned CodeBERT on 30,919 SQL queries and reported high benchmark performance, including 99.90% accuracy and 99.86% F1-score. However, their evaluation was limited to offline testing and did not include WAF-oriented deployment, runtime overhead analysis, or adversarial request-level validation. Liu and Dai [21] proposed a hybrid BERT–LSTM model and evaluated it using a simulated vulnerable platform and SQL attack experiments, reporting 0.973 accuracy and 0.958 F1-score. Lo et al. [22] further incorporated SQL-specific tokenization, semantic labels, CNN, and multi-head self-attention, while also reporting inference time and model weight size. These studies demonstrate the growing relevance of semantic and attention-based learning in SQLi detection. Nevertheless, they also show that high benchmark accuracy alone is insufficient for evaluating practical SQLi protection, where model efficiency, deployment feasibility, and real-time WAF suitability are also critical.

To provide a clearer comparison of the most closely related studies, Table 1 summarizes the datasets, methods, deployment validation, and achieved performance of prior SQLi detection approaches.

Table 1. Comparative summary of related SQLi detection studies.

Study	Dataset	Method	Deployment Validation	Best Reported Performance
Peralta-Garcia et al. [15]	Public SQLi dataset; 22,764 balanced records	TF-IDF/CountVectorizer with RF, SVM, and DT	Microservices-based request validation architecture	RF achieved 99% accuracy and precision, 97% recall, and 98% F1-score
Ogini et al. [16]	Kaggle SQLi dataset; approximately 30,635 queries	CountVectorizer, TF-IDF feature selection, and FFNN	Python Flask web deployment	97.65% accuracy
Pramono et al. [17]	Kaggle SQLi dataset; 4187 samples	TF-IDF with Naïve Bayes and SVM	Evaluation was limited to offline benchmark testing	Best accuracy: 96.67% with Naïve Bayes

Table 1. *Cont.*

Study	Dataset	Method	Deployment Validation	Best Reported Performance
Bakır [18]	Two SQLi datasets; 33,726 and 30,919 SQL queries	UniEmbed: Word2Vec + USE + FastText with multiple ML classifiers	Evaluation was limited to offline benchmark testing	MLP achieved accuracy exceeding 0.9980 across two SQLi datasets.
Alghawazi et al. [19]	Kaggle SQLi dataset; 30,907 records after cleaning	RNN autoencoder with LSTM-based classification	Evaluation was limited to offline benchmark testing	94% accuracy and 92% F1-score
Thalji et al. [13]	Combined SQLi dataset; 46,392 SQL queries	AE-Net deep features with ML/DL classifiers	Evaluation was limited to offline benchmark testing	AE-Net + XGB achieved 0.99 accuracy
Ammar & Alharbi [20]	30,919 SQL queries	Fine-tuned CodeBERT	Evaluation was limited to offline benchmark testing	Accuracy: 99.90%, Precision: 99.96%, Recall: 97.75%, F1-score: 99.86%, AUC-ROC: 0.9987
Liu and Dai [21]	30,156 samples: 10,852 malicious SQLi + 19,304 normal requests	Hybrid BERT-LSTM	Simulated vulnerable platform, graphical interface, netflow-based detection, and 1307 SQL attack experiments conducted over one week	Accuracy: 0.973, Precision: 0.963, Recall: 0.962, F1-score: 0.958
Lo et al. [22]	Training set T: 98,275 queries; Test set S: 24,707 queries; Test set U: 500 queries longer than 1000 characters	SQL-specific tokenizer, semantic labels, CNN stage, multi-head self-attention stage, and output stage	Average inference time measured on UP Board and PC	Test set S: Accuracy 98.98%, Precision 98.01%, Recall 99.84%, F1-score 98.92%; Weight size: 69,269; PC inference time: 51.55 ms; UP Board inference time: 175.92 ms

As summarized in Table 1, previous studies have achieved strong SQLi detection performance using conventional machine learning, autoencoder-based representation learning, and transformer-based models. However, most of these studies either remained limited to offline benchmark evaluation or provided only partial deployment validation. This limitation motivates the research gap and positioning discussed in the following subsection.

2.5. Research Gap and Positioning of the Present Study

The closely related literature shows meaningful progress in deployment-oriented SQLi detection, hybrid feature modeling, autoencoder-based representation learning, and transformer-based detection. However, these directions have generally been investigated separately. Deployment-oriented studies often rely on conventional textual representations or single-model pipelines, whereas feature-fusion, autoencoder-based, and transformer-based studies mainly emphasize benchmark performance, with limited attention to practical WAF validation, runtime efficiency, and adversarial request-level robustness.

Against this background, the present study is positioned at the intersection of these directions. It combines multi-source SQLi data construction, hybrid feature extraction using TF-IDF, Word2Vec, and FastText, supervised autoencoder-based latent representation learning, GA-based hyperparameter optimization, and lightweight WAF-oriented deployment validation.

3. Materials and Methods

The proposed framework follows a sequential workflow that includes data preprocessing, hybrid feature extraction, supervised autoencoder-based dimensionality reduction to obtain a compact latent space, and GA-based hyperparameter optimization under a unified evaluation protocol, as shown in Figure 1.

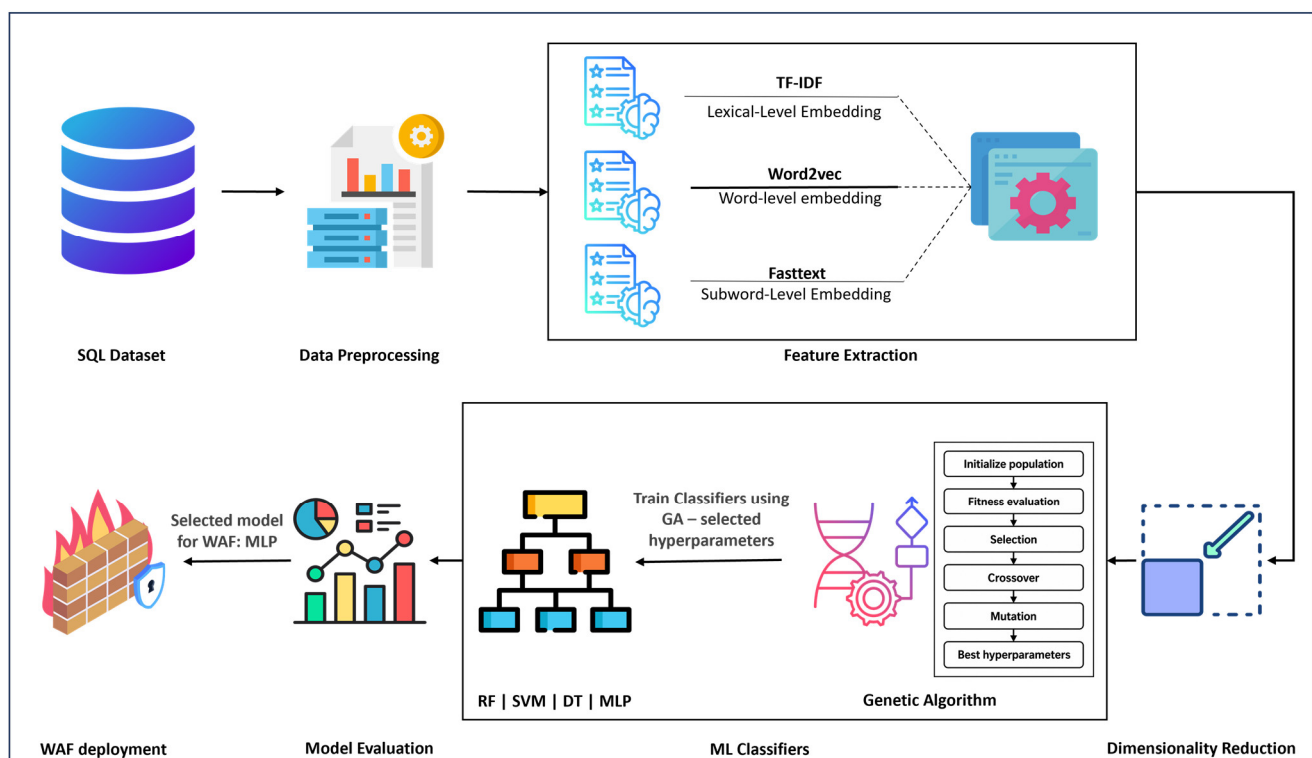


Figure 1. Block diagram of the proposed method.

3.1. SQL Injection Attacks

SQL injection occurs when malicious or untrusted input is directly concatenated with dynamically generated SQL queries, enabling attackers to manipulate query logic and perform unauthorized operations [23].

3.2. Datasets

This study was constructed by merging three publicly available Kaggle SQL injection datasets, namely SQLi, SQLiV2, and SQLiV3, which provided 68,880 query samples [24]. In addition, 11,011 samples from the BCCC-SFU-SQLi dataset were incorporated to increase the diversity of SQL injection patterns [25]. Therefore, the collected raw dataset contained 79,891 SQL query samples.

A controlled preprocessing procedure was applied before feature extraction. All query samples were converted to lowercase, and redundant whitespace characters, including repeated spaces, tabs, and line breaks, were normalized into a single space. Empty or null query entries were removed, and exact duplicate queries were removed based on the cleaned query text before data splitting. Importantly, SQL-relevant special characters

were preserved because quotation marks, parentheses, comparison operators, semicolons, comment markers, and wildcard characters may represent discriminative evidence for SQL injection detection. After preprocessing and duplicate removal, the final dataset contained 56,838 unique queries, including 34,349 benign queries and 22,489 SQL injection queries, as shown in Table 2.

Table 2. Final dataset distribution after preprocessing.

Class	Number of Samples	Percentage
Benign queries	34,349	60.43%
SQL injection queries	22,489	39.57%
Total	56,838	100%

To further describe the diversity of malicious samples, the SQL injection queries in the final preprocessed dataset were grouped according to common SQLi attack patterns. The resulting distribution is presented in Table 3. This analysis was used only for dataset characterization and was not used as an additional training label.

Table 3. SQL injection attack-type distribution in the final dataset.

Attack Type	Sample Size	Percentage of Malicious Samples
Boolean-based blind SQLi	10,268	45.66%
Error-based/metadata-extraction SQLi	6706	29.82%
UNION query-based SQLi	2212	9.84%
Time-based blind SQLi	1819	8.09%
Miscellaneous SQLi/comment-tampering payloads	1471	6.54%
Stacked-query SQLi	13	0.05%
Total malicious samples	22,489	100%

These subtype categories were used only to characterize the diversity of malicious samples and were not used as training labels, since the proposed framework focuses on binary SQLi detection rather than multi-class attack-type classification.

The final deduplicated dataset was divided into training, validation, and testing subsets using a fixed stratified hold-out split with a 70:10:20 ratio. The split was neither time-based nor source-based; instead, all datasets were first merged, cleaned, and deduplicated, and then a unified stratified random partition was applied. Stratification preserved the benign/malicious class distribution across all subsets, and the split indices were saved and reused across all experiments to ensure reproducibility.

To avoid data leakage, duplicate removal was performed before data partitioning, and all feature-learning steps were restricted to the training subset. Specifically, the TF-IDF vectorizer was fitted only on the training subset and then applied to the validation and test subsets. Similarly, Word2Vec and FastText embedding models were trained only using the training queries. The validation set was used for hyperparameter tuning and model selection, whereas the test set was held out for final performance evaluation only.

3.3. Feature Extraction

Raw input queries must be converted into machine-readable feature representations so that the learning model can identify discriminative patterns for SQLi detection [26]. Therefore, this study employs a hybrid feature extraction strategy combining TF-IDF with embedding-based models, namely Word2Vec and FastText, to improve robustness against diverse and obfuscated SQLi patterns.

TF-IDF assigns weights to terms based on their frequency within a text sample and their rarity across the corpus, thereby highlighting lexically relevant terms that can support text representation and classification [27]. It is commonly computed as:

$$\text{TFIDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t) \quad (1)$$

where $\text{TF}(t, d)$ represents the frequency of term t in document d , while $\text{IDF}(t)$ reflects the rarity of that term across the corpus.

However, because TF-IDF is mainly frequency-based, it may not fully capture contextual relationships among terms. To address this limitation, Word2Vec was employed to learn dense vector representations from contextual co-occurrence. Word2Vec provides two common architectures, CBOW and Skip-gram [11]. In this study, the Skip-gram model was adopted to learn representations by predicting surrounding context words from a given target word. The Skip-gram objective is expressed as:

$$\sum_{t=1}^T \sum_{c \in C_t} \log p(w_c | w_t) \quad (2)$$

where T denotes the total number of words in the training corpus, w_t is the target word at position t , C_t represents the set of context words surrounding w_t , and $p(w_c | w_t)$ is the probability of predicting a context word w_c given the target word w_t . By maximizing this objective, words appearing in similar contexts are encouraged to learn closer vector representations [11].

FastText represents words using character n-grams, which helps generate vector representations for out-of-vocabulary words and captures subword-level information [12]. For the Skip-gram setting adopted in this study, FastText incorporates subword information through the following scoring function:

$$s(w, c) = \sum_{g \in G_w} z_g^T v_c \quad (3)$$

where $s(w, c)$ denotes the association score between the target word w and the context word c , G_w is the set of character n-grams appearing in w , z_g is the vector representation of the character n-gram g , and v_c is the context-word vector [11].

By summing subword representations, FastText can generate informative representations for rare or unseen tokens. In the context of SQLi detection, this property is relevant because obfuscated payloads may contain rare, modified, or previously unseen token forms.

3.4. Dimensionality Reduction

An autoencoder was employed as an automatic feature engineering method to extract high-level deep features from SQL textual query data. The extracted features were then used as inputs to machine learning models for SQL injection detection [13]. In this study, dimensionality reduction was performed using a supervised autoencoder to transform the high-dimensional fused feature vector into a compact and discriminative latent representation. The input to the autoencoder consisted of the concatenated TF-IDF, Word2Vec, and FastText features.

The encoder mapped the fused input into a lower-dimensional latent space through a fully connected hidden layer of 1024 neurons followed by a latent layer. The decoder was designed to reconstruct the original fused input from the latent representation, while an additional classification head was attached to the latent layer to guide the learning process using class-label information. Therefore, the supervised autoencoder was trained using a

combined objective function consisting of reconstruction loss and weighted classification loss. The total loss was defined as [28]:

$$L_{total} = L_{rec} + \lambda L_{Cls} \quad (4)$$

where L_{rec} denotes the reconstruction loss computed using mean squared error, and L_{cls} denotes the classification loss computed using cross-entropy. The hyperparameter λ controls the contribution of the classification objective relative to the reconstruction objective.

The supervised autoencoder was trained using the Adam optimizer with a learning rate of 1×10^{-3} , a batch size of 256, and ReLU activation functions in the encoder and decoder hidden layers. Adam is an adaptive gradient-based optimization algorithm that updates the model weights during training [29]. In this study, Adam was used to train the autoencoder by minimizing the reconstruction loss between the original fused feature representation and its reconstructed output. It was selected because it adaptively adjusts the learning rate for different model parameters, which supports stable and efficient convergence when learning compact representations from high-dimensional fused features. The model was trained for a maximum of 30 epochs with early stopping based on validation F1-score. After training, the learned latent vectors were extracted and used as compact feature representations for RF, SVM, DT, and MLP.

3.5. Genetic Algorithm

A Genetic Algorithm (GA) is a metaheuristic optimization method inspired by natural selection. It searches for improved candidate solutions through iterative selection, crossover, and mutation operations [29].

In this study, GA was employed to optimize the hyperparameters of the machine learning classifiers used for SQLi classification. Each individual represented a candidate hyperparameter configuration, and validation F1-score was used as the fitness criterion. The GA was executed with a population size of 20, selected parents of 10, a mutation rate of 0.15, elitism with the best two individuals retained, and a maximum of 30 generations. During the optimization process, candidate solutions were iteratively evaluated, selected, crossed over, and mutated, while GA-level early stopping was applied after seven generations without improvement. This optimization strategy enabled efficient convergence and supported improved generalization and detection performance.

3.6. Classification Models

After feature extraction and representation learning, the final stage of the proposed framework performs supervised classification to discriminate between benign and SQL injection (SQLi) queries. Multiple machine learning classifiers were evaluated to ensure that the performance gains are not tied to a single learner and to provide a fair comparison between linear, margin-based, and ensemble decision mechanisms.

Random Forest (RF) is an ensemble learning method that constructs multiple decision trees using bootstrap samples and aggregates their predictions to improve accuracy and reduce overfitting [30]. Support Vector Machine (SVM) is a supervised learning algorithm that builds an optimal decision boundary (hyperplane) to separate classes while maximizing the margin between them, improving classification performance and generalization [31]. Decision Tree (DT) represents the classification process as a hierarchical structure of decision rules that split the data until a final class label is obtained [32]. Finally, a Multilayer Perceptron (MLP) is a feed-forward neural network composed of input, hidden, and output layers, where model parameters are updated through backpropagation to learn complex non-linear relationships in the data [33].

4. Results

4.1. Hybrid Feature Fusion and Supervised Autoencoder-Based Dimensionality Reduction

The proposed hybrid fusion was implemented through feature concatenation, where TF-IDF vectors (30,432 dimensions) were combined with Word2Vec and FastText embeddings (200 dimensions each), yielding a fused feature space of 30,832 dimensions.

To reduce the high-dimensional fused representation, a supervised autoencoder was employed to learn a compact latent space while preserving class-discriminative information. The autoencoder was trained using a joint objective that combined reconstruction loss and classification loss. To prevent the high-dimensional TF-IDF block from dominating the reconstruction process, a block-wise reconstruction objective was adopted, in which the reconstruction errors were computed separately for the TF-IDF, Word2Vec, and FastText feature blocks and then averaged to form the overall reconstruction loss. In addition, the TF-IDF representation was L2-normalized using the default setting of the TfidfVectorizer, further reducing scale imbalance among feature groups.

Several latent dimensions were experimentally evaluated, including 25, 50, 75, 100, 125, 150, 175, and 200, to determine the most suitable compact representation for the proposed framework. The classification-loss weight λ was selected separately for each latent dimension using the validation set. Five candidate values were evaluated: 0.01, 0.1, 0.3, 0.5, and 0.7. For each latent dimension, the supervised autoencoder was first trained using each candidate value, and the value that achieved the highest validation F1-score was selected as the best λ . The selected λ was then used to train the final autoencoder model for that latent dimension.

As shown in Table 4, the validation F1-scores were highly comparable across several latent dimensions, indicating stable representation learning. However, the 150-dimensional latent representation achieved the best overall performance, with the highest test accuracy of 0.998593 and the highest test F1-score of 0.998219. Although larger latent dimensions such as 175 and 200 maintained strong performance, they did not provide additional improvement over the 150-dimensional configuration. Therefore, the 150-dimensional latent space was selected as the final compact representation for the downstream classifiers.

Table 4. Effect of latent dimensionality on supervised autoencoder performance and training time.

Latent Size	Best λ	Validation F1	Test Accuracy	Test F1-Score	Training Time (s)
25	0.01	0.997999	0.997713	0.997103	185.82
50	0.01	0.998221	0.997801	0.997215	176.37
75	0.01	0.998221	0.998241	0.997773	291.62
100	0.30	0.998000	0.998505	0.998108	329.12
125	0.01	0.998221	0.997889	0.997327	208.78
150	0.10	0.998221	0.998593	0.998219	324.77
175	0.01	0.998221	0.998153	0.997662	202.50
200	0.01	0.998221	0.998153	0.997662	203.08

As illustrated in Figure 2, the learning curves show rapid and stable convergence, with both reconstruction and classification losses decreasing smoothly. The close alignment between training and validation trends indicates stable optimization without clear signs of overfitting.

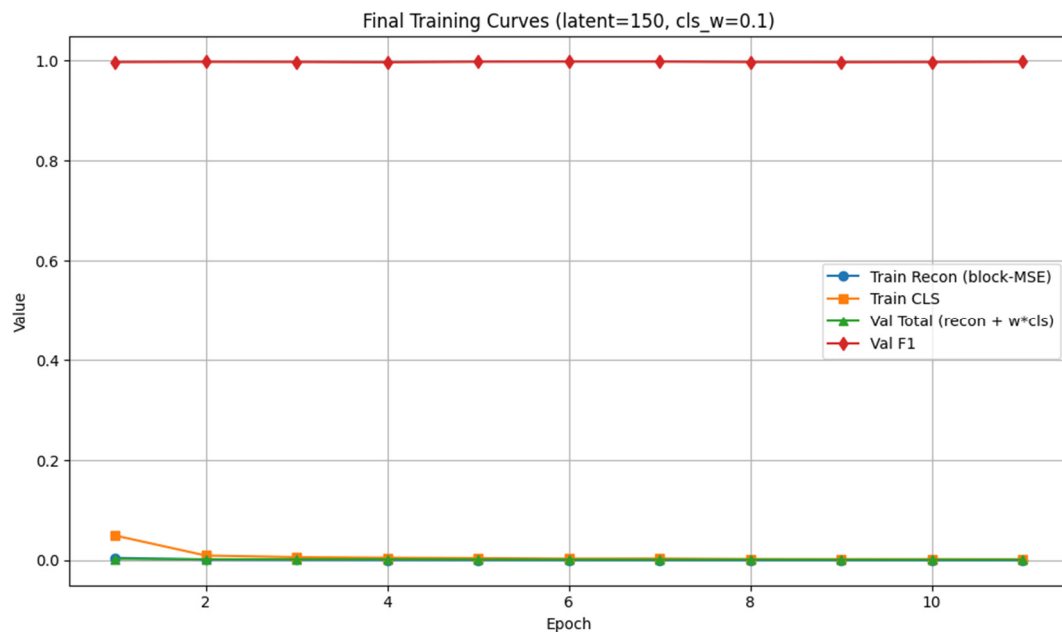


Figure 2. Training and validation loss curves of the supervised autoencoder. Here, $w \times \text{cls}$ denotes the weighted classification loss, where w is the classification-loss weight and cls is the classification loss.

4.2. Genetic Algorithm-Based Hyperparameter Optimization

In addition to improving classification performance, the Genetic Algorithm (GA) contributed to reducing the overall hyperparameter search time by efficiently exploring the parameter space. Instead of exhaustively evaluating all possible combinations, GA iteratively searched the solution space and identified a near-optimal set of hyperparameters that yielded strong validation performance, as shown in Table 5.

Table 5. Best hyperparameters of the ML classifiers optimized using GA.

Classifier	Best Hyperparameters
RF	n_estimators: 50; max_depth: 20; min_samples_split: 10; min_samples_leaf: 4; criterion: entropy
SVM	Kernel: linear; Degree: 3; Gamma: 0.01; C: 10
DT	Criterion: entropy; Max depth: 6; Min samples split: 2; Min samples leaf: 1
MLP	Hidden layers: Single hidden layer, 128, neurons; Activation: relu; Alpha: 0.001; Learning rate: 0.01; Solver: adam; Max iterations: 150

4.3. Ablation Analysis of the Proposed Framework

Table 6 compares the classification accuracy achieved using individual feature extraction techniques versus the proposed hybrid representation. Across all classifiers, the hybrid TF-IDF, Word2Vec, and FastText representations consistently outperformed the individual feature extraction methods. These findings indicate that combining lexical, contextual, and subword-level information provides a stronger representation for SQL injection detection than relying on any single feature space alone.

Table 6. Comparison of classification accuracy across different feature extraction methods.

ML Classifier	TF-IDF	Word2Vec	FastText	Proposed Framework
RF	0.9964	0.9962	0.9970	0.9985
SVM	0.9943	0.9967	0.9956	0.9985
DT	0.9947	0.9952	0.9922	0.9985
MLP	0.9963	0.9981	0.9970	0.9986

To justify the use of three-feature fusion, an ablation-style analysis was conducted using the Decision Tree (DT) classifier as a lightweight baseline under identical data split and training conditions. The DT classifier was selected because of its low computational cost, fast training process, and suitability for controlled comparative evaluation. This allowed the effects of feature fusion and SAE-based dimensionality reduction to be examined without the additional optimization burden associated with more complex classifiers. As shown in Table 7, all two-feature combinations achieved strong performance; however, the full fusion of TF-IDF, Word2Vec, and FastText produced the best result. Moreover, SAE-based dimensionality reduction consistently improved performance across all fusion settings, supporting the effectiveness of the proposed fusion and dimensionality reduction strategy.

Table 7. Ablation analysis of feature fusion with and without SAE-based dimensionality reduction using DT.

Feature Combination	Accuracy Without SAE	Accuracy with SAE
Word2Vec + FastText	0.9958	0.9979
TF-IDF + Word2Vec	0.9956	0.9978
TF-IDF + FastText	0.9949	0.9974
TF-IDF + Word2Vec + FastText	0.9961	0.9985

4.4. Classification Performance of the Proposed Framework

Table 8 summarizes the performance of the classifiers trained using the proposed framework. Among the evaluated models, MLP achieved the best overall performance, reaching an accuracy of 0.998681 and an F1-score of 0.998330, while also maintaining strong recall and precision. In terms of error behavior, it produced only one false positive and fourteen false negatives, supporting its selection as the final classifier for deployment within the proposed WAF framework.

Table 8. Performance comparison of ML classifiers using the proposed framework.

ML Classifier	Accuracy	F1-Score	Recall	Precision	FP	FN	ROC-AUC
RF	0.998593	0.998219	0.996888	0.999554	2	14	0.999407
SVM	0.998593	0.998219	0.996888	0.999554	2	14	0.999071
DT	0.998505	0.998107	0.996443	0.999777	1	16	0.998255
MLP	0.998681	0.998330	0.996888	0.999777	1	14	0.999024

The confusion matrices and ROC-AUC curves for the evaluated classifiers are presented in Figures 3 and 4, respectively, to provide additional visual support for the classification results.

4.5. Comparison with Fine-Tuned CodeBERT Baseline

An additional experiment was conducted to compare the proposed MLP-based model with a transformer-based fine-tuned CodeBERT baseline using the same dataset and identical train/validation/test split. The CodeBERT baseline was fully fine-tuned for binary SQL injection classification using microsoft/codebert-base, a maximum sequence length of 128, a learning rate of 2×10^{-5} , a training batch size of 8, an evaluation batch size of 16, weight decay of 0.01, and a warm-up ratio of 0.1. Training was allowed for a maximum of five epochs with early stopping based on validation F1-score. The model stopped after four effective epochs, with the best checkpoint selected according to validation F1-score. The comparison included classification performance, parameter count, saved model size, raw tensor weight size, training/optimization time, inference latency, and throughput. The results are presented in Table 9.

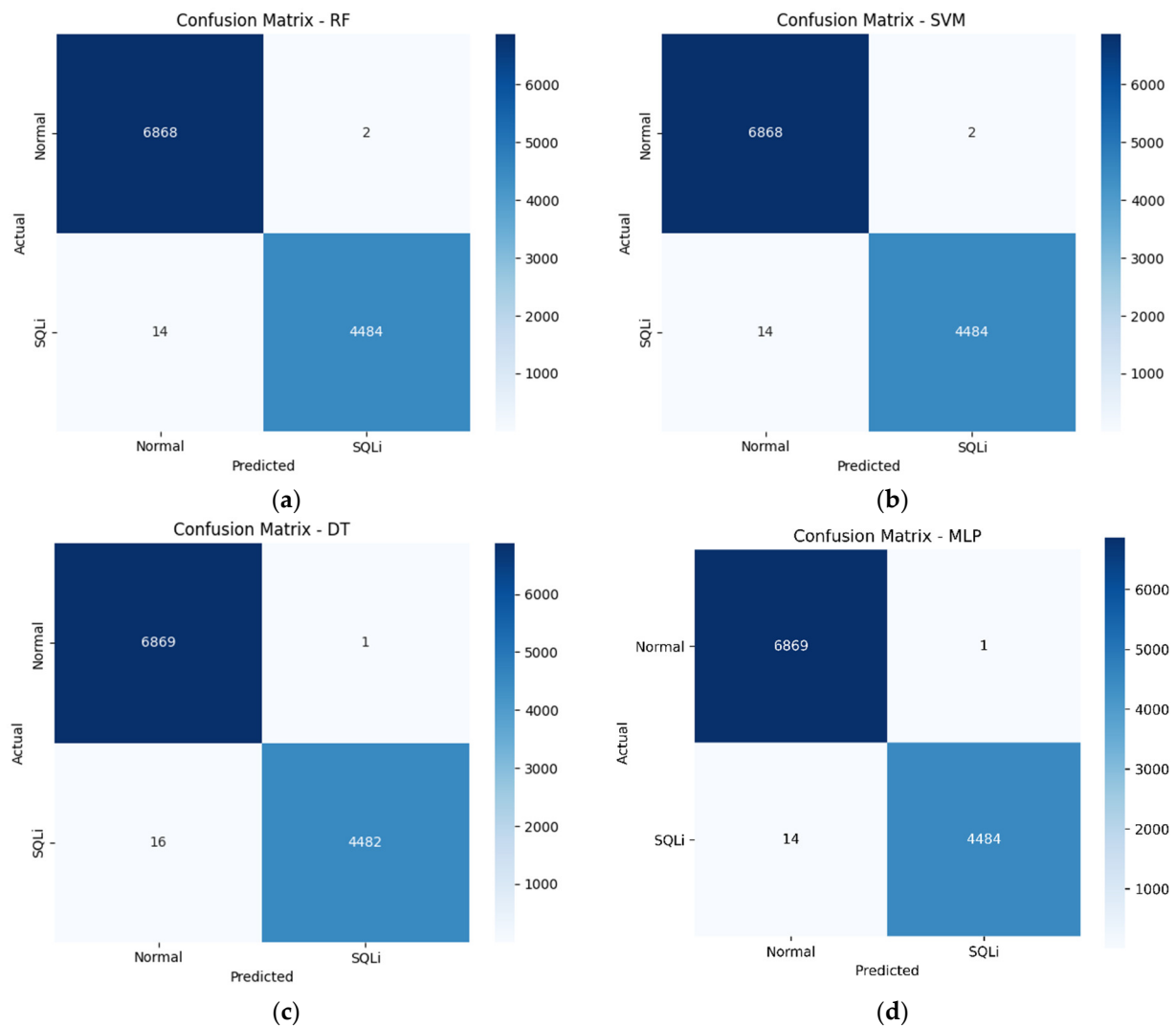


Figure 3. Confusion Matrices of the evaluated classifiers: (a) Random Forest (RF); (b) Support Vector Machine (SVM); (c) Decision Tree (DT); (d) Multi-Layer Perceptron (MLP).

Table 9. Comparison of the Proposed Model with CodeBERT Baseline.

Metric	Proposed Framework	Fine-Tuned CodeBERT
F1-score	99.8330%	99.9333%
Total parameters	19,586	124,647,170
model size	0.0814 MB	480.113 MB
weight size	0.0747 MB	475.491 MB
Training/optimization time	3436.31 s/57.27 min	4737.74 s/78.96 min
Inference time per request	3.4014 ms	41.2737 ms
Throughput	287 samples/s	257.69 samples/s

4.6. Deployment of the Proposed ML-WAF Framework

The best-performing classifier, namely the MLP, was integrated into an inline reverse-proxy inspection setup, in which a Flask-based ML-WAF was positioned between the Apache web server and the backend PHP application. Incoming HTTP GET and POST requests were intercepted, user-supplied parameters were extracted, and each value was evaluated by the trained SQL injection detector. If a request was classified as malicious, it was blocked at the WAF layer, an automated Gmail notification was sent to the administra-

tor, and an HTTP 403 (Forbidden) response was returned to the user. In contrast, benign requests were forwarded to the backend, as illustrated in Figure 5.

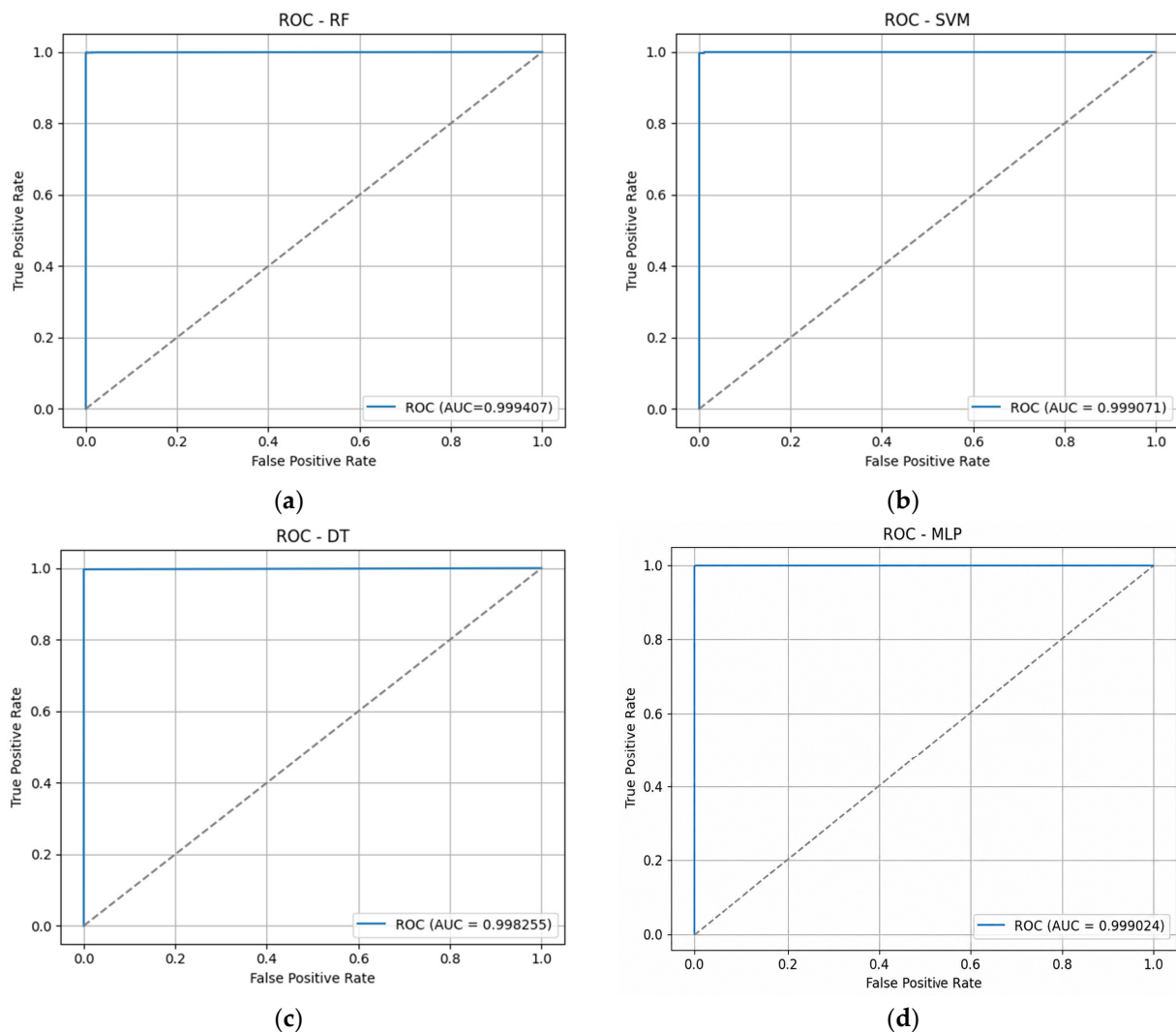


Figure 4. ROC curves of the evaluated classifiers: (a) Random Forest (RF); (b) Support Vector Machine (SVM); (c) Decision Tree (DT); (d) Multi-Layer Perceptron (MLP). The gray dotted diagonal line represents the random classifier baseline (AUC = 0.5).

The deployed WAF was implemented on a Linux-based server running an Apache–MySQL–PHP stack at Northern Technical University to validate integration under realistic hosting conditions. Experiments were conducted on a system with a 13th Gen Intel Core i7-13620H CPU and 16 GB RAM, while model training and evaluation were performed on a GPU-enabled system to reduce computational time. These deployment results support the feasibility of integrating the proposed SQLi detector into a lightweight real-time WAF workflow.

4.7. SQLMap-Based Robustness Evaluation

To further assess the robustness and generalization ability of the proposed system under realistic adversarial conditions, an additional SQLMap-based evaluation was conducted. SQLMap version 1.8.4 was used to generate a large and diverse set of malicious HTTP requests, including both obfuscated and non-obfuscated SQL injection payloads. These payloads covered several common SQL injection techniques, such as Boolean-based

blind SQLi, UNION query SQLi, time-based blind SQLi, error-based SQLi, stacked-query SQLi, and other miscellaneous SQLi syntax patterns.

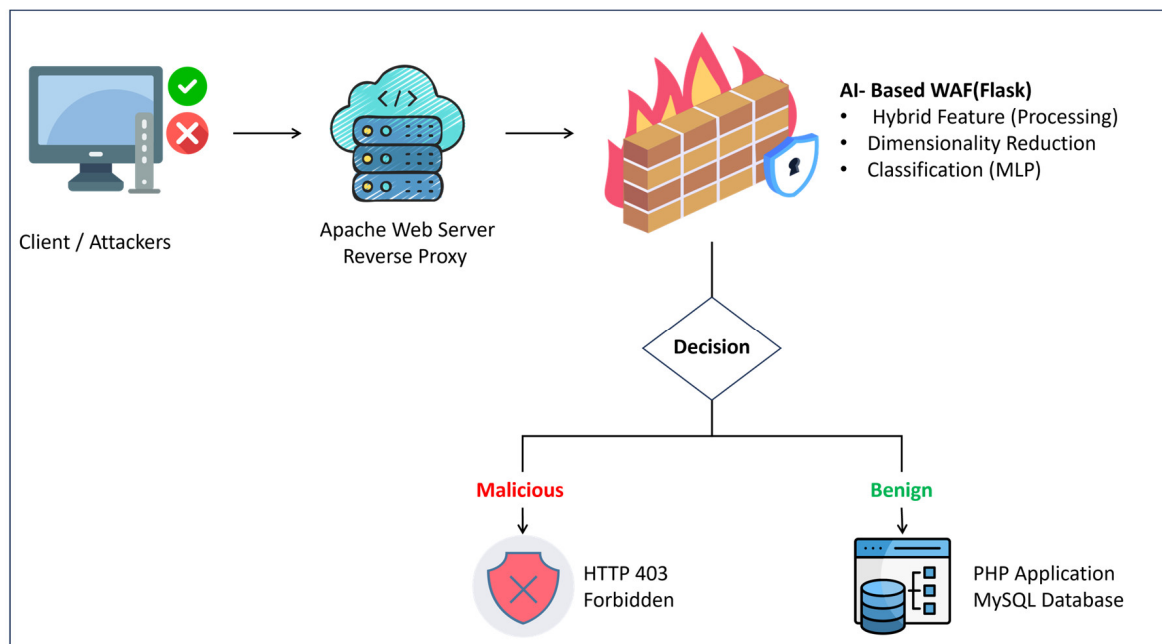


Figure 5. System architecture of the proposed AI-based WAF framework.

A total of 19,990 unique malicious requests were submitted to the deployed inspection workflow during this evaluation. These SQLMap-generated requests were not included in the original training or testing datasets, ensuring that the evaluation was performed on unseen adversarial samples. Among them, the proposed detector correctly identified 19,496 attacks as malicious, corresponding to true positives (TP), whereas 494 attack instances were incorrectly classified as benign, corresponding to false negatives (FN). Based on these results, the system achieved a detection rate/recall of 97.53%, with an overall false negative rate (FNR) of 2.47%.

The SQLMap test set was further analyzed according to attack type, as shown in Table 10. This analysis provides a more detailed view of the detector's behavior across different SQL injection patterns and highlights the types of attacks that were more challenging for the proposed framework.

Table 10. SQLMap attack-type distribution and false negative analysis.

Attack Type	Sample Size	TP	FN	FNR
Boolean-based blind SQLi	5551	5151	400	7.21%
UNION query SQLi	5465	5418	47	0.86%
Time-based blind SQLi	3238	3225	13	0.40%
Miscellaneous SQLi/malformed payloads	2827	2798	29	1.03%
Error-based SQLi	2428	2423	5	0.21%
Stacked-query SQLi	481	481	0	0.00%
Total	19,990	19,496	494	2.47%

In addition to the attack-type analysis, the SQLMap payloads were also categorized according to their obfuscation status. This analysis was performed separately because obfuscation is not an independent attack type; rather, it is a cross-cutting property that may

appear within different SQL injection categories, including Boolean-based, UNION-based, time-based, and error-based SQLi payloads, as shown in Table 11.

Table 11. Distribution of obfuscated and non-obfuscated SQLMap queries.

Query Category	Count	Percentage
Obfuscated SQLi queries	12,881	64.44%
Non-obfuscated SQLi queries	7109	35.56%
Total	19,990	100%

These results indicate that the proposed framework retained strong detection capability when evaluated on a large and diverse set of automatically generated SQL injection requests. The high detection rate demonstrates that the model was able to identify the majority of SQL injection attempts despite variations in attack structure, payload syntax, and obfuscation. Moreover, the separate analysis of obfuscated and non-obfuscated payloads provides additional evidence that the system was not only evaluated on direct SQLi patterns, but also on evasive payloads that are commonly used to bypass traditional signature-based protection mechanisms.

4.8. Runtime Performance and Throughput Evaluation

The runtime performance of the deployed WAF prototype was evaluated under three scenarios: Without WAF, WAF + benign, and WAF + malicious. As shown in Table 12, the Without WAF scenario achieved the lowest mean latency of 28.1 ms and the highest QPS of 36.56 req/s in the single-request evaluation, as requests were served directly without ML-based inspection. In comparison, the WAF + benign scenario increased the mean latency to 69.5 ms and reduced QPS to 18.80 req/s because legitimate requests were intercepted, transformed into feature representations, classified, and then forwarded to the backend application. The WAF + malicious scenario achieved a lower mean latency of 42.7 ms and a higher QPS of 23.63 req/s than WAF + benign because detected SQL injection requests were rejected directly at the WAF layer with an HTTP 403 response without backend forwarding. These results show the expected inspection overhead introduced by inline ML-based filtering while maintaining acceptable latency in the single-request laboratory evaluation.

Table 12. Single-request runtime performance of the deployed WAF.

Scenario	Mean Latency (ms)	Min Latency (ms)	Max Latency (ms)	Std Latency (ms)	QPS (req/s)	HTTP Status
Without WAF	28.1	18.9	73.4	13.6	36.56	200
WAF + benign	69.5	56.9	97.4	10.6	18.80	200
WAF + malicious	42.7	40.4	52.1	3.2	23.63	403

Apache Bench from Apache HTTP Server 2.4.58 (Ubuntu) with keep-alive enabled was further used to evaluate runtime performance under concurrent traffic at 10, 50, and 100 concurrent connections. As summarized in Table 13, the Without WAF scenario achieved the highest QPS across all concurrency levels because requests were processed without WAF inspection. At 100 concurrent connections, without WAF, benign achieved 183.51 req/s with a mean latency of 544.930 ms, whereas WAF + benign achieved 26.70 req/s with a mean latency of 3745.420 ms. WAF + malicious achieved 35.08 req/s with a lower mean latency of 2850.855 ms, reflecting the lower processing cost of blocking malicious requests directly at the WAF layer without backend forwarding.

Table 13. Apache Bench keep-alive benchmark under selected concurrency levels.

Concurrency	Metric	Without WAF	WAF + Benign	WAF + Malicious
10	QPS (req/s)	346.60	32.01	37.07
	Mean latency (ms)	28.851	312.375	269.738
50	QPS (req/s)	190.64	26.28	35.27
	Mean latency (ms)	262.275	1902.539	1417.588
100	QPS (req/s)	183.51	26.70	35.08
	Mean latency (ms)	544.930	3745.420	2850.855

5. Discussion

The experimental results suggest that the main strength of the proposed framework lies in the representation-learning stage rather than in the complexity of the final classifier. The hybrid fusion of TF-IDF, Word2Vec, and FastText provided complementary views of SQL query patterns, where TF-IDF captured discriminative lexical indicators, Word2Vec represented contextual token co-occurrence, and FastText improved sensitivity to rare, modified, and partially obfuscated token forms through subword modeling. This complementary representation is particularly important for SQLi detection because malicious payloads may appear in short, irregular, encoded, or syntactically altered forms that cannot be fully captured by a single feature extraction method.

The ablation analysis provides direct evidence for the value of both feature fusion and supervised autoencoder-based dimensionality reduction. The full three-feature fusion achieved better performance than all two-feature combinations, indicating that the improvement resulted from integrating complementary feature perspectives rather than simply increasing feature dimensionality. Moreover, applying the supervised autoencoder consistently improved the results across all fusion settings. This suggests that the SAE was not only useful for reducing dimensionality, but also for refining the fused representation by reducing redundant and noisy features while preserving information relevant to SQLi classification.

The close performance observed among MLP, RF, SVM, and DT can also be explained by the quality of the SAE-generated latent representation. After supervised dimensionality reduction, the fused features were projected into a compact and more discriminative latent space, making the separation between benign and malicious queries clearer for different classifiers. As a result, even classifiers with different learning mechanisms achieved highly comparable performance. This indicates that the SAE contributed to producing a robust representation that reduced dependence on a specific classifier architecture.

Nevertheless, the MLP was selected as the final detector because it provided the best overall balance across the evaluation metrics while maintaining a lightweight structure suitable for real-time WAF deployment.

To further evaluate practical suitability, the proposed framework was compared with a fine-tuned CodeBERT baseline under the same dataset and identical train/validation/test split. CodeBERT achieved a higher F1-score, confirming the strong capability of transformer-based models in capturing complex SQLi patterns. In contrast, the proposed framework achieved competitive detection performance while requiring fewer computational and storage resources. This difference highlights the distinct strengths of the two approaches: CodeBERT provides strong semantic modeling capacity, whereas the proposed framework offers a lightweight and efficient alternative for real-time WAF deployment. Therefore, the proposed model provides a favorable balance between detection performance, model compactness, inference efficiency, and practical inline request inspection.

From a practical perspective, the deployment-oriented evaluation shows that the proposed framework extends beyond offline benchmark performance and can operate under realistic request-inspection conditions. The SQLMap-based assessment demonstrated strong robustness against unseen adversarial payloads, including both obfuscated and non-obfuscated SQLi patterns. Although Boolean-based blind SQLi produced the largest share of false negatives, the framework maintained low false-negative rates across most other attack categories, indicating stable generalization to structurally diverse payloads.

The runtime analysis provides additional evidence of the operational behavior of the proposed WAF prototype under controlled laboratory conditions. In the single-request evaluation, WAF-filtered benign traffic maintained a mean latency below 0.1 s, while malicious requests were processed faster because they were blocked directly at the WAF layer without backend forwarding. Under concurrent traffic, Apache Bench showed the expected reduction in QPS and increase in mean latency compared with the backend-only configuration. However, these results should be interpreted as prototype-level operational behavior rather than evidence of production-scale scalability. The selected ML model remained lightweight, as indicated by its compact size and very low standalone inference time. Therefore, the higher end-to-end latency mainly reflects the complete Flask-based request-processing pipeline, including request interception, feature transformation, forwarding/blocking decisions, backend communication, and HTTP response handling, rather than the isolated cost of ML inference alone. Further optimization using production-grade deployment components would be required for large-scale operational use.

Compared with the related studies summarized in Table 14, the proposed framework achieves competitive detection performance while providing a more comprehensive deployment-oriented evaluation.

Table 14. Performance comparison with related SQLi detection studies.

Study	Method	Best Reported Performance	Runtime and Deployment Validation
Peralta-Garcia et al. [15]	TF-IDF/CountVectorizer with RF, SVM, and DT	RF achieved 99% accuracy, 97% recall, and 98% F1-score	Microservices-based validation was reported; however, model size, inference latency, and throughput were not reported.
Ogini et al. [16]	CountVectorizer, TF-IDF feature selection, and FFNN	97.65% accuracy	Flask web deployment was reported; however, model size, inference latency, and throughput were not reported.
Pramono et al. [17]	TF-IDF with Naïve Bayes and SVM	Best accuracy: 96.67% with Naïve Bayes	Offline benchmark only; deployment validation was not reported
Bakır [18]	UniEmbed: Word2Vec + USE + FastText with multiple ML classifiers	MLP achieved accuracy above 0.9980 on both SQLi datasets	Offline benchmark only; deployment validation was not reported
Alghawazi et al. [19]	RNN autoencoder with LSTM-based classification	94% accuracy and 92% F1-score	Offline benchmark only; deployment validation was not reported
Thalji et al. [13]	AE-Net deep features with ML/DL classifiers	AE-Net + XGB achieved 0.99 accuracy	Offline benchmark only; deployment validation was not reported

Table 14. *Cont.*

Study	Method	Best Reported Performance	Runtime and Deployment Validation
Ammar & Alharbi [20]	Fine-tuned CodeBERT	Accuracy: 99.90%, Precision: 99.96%, Recall: 97.75%, F1-score: 99.86%, AUC-ROC: 0.9987	Transformer-based model; deployment validation, model size, inference latency, and throughput were not reported
Liu and Dai [21]	Hybrid BERT-LSTM	Accuracy: 0.973, Precision: 0.963, Recall: 0.962, F1-score: 0.958	Simulated deployment, but model size and per-request WAF latency were not reported
Lo et al. [22]	SQL-specific tokenizer, semantic labels, CNN stage, multi-head self-attention stage, and output stage	Test set S: Accuracy 98.98%, Precision 98.01%, Recall 99.84%, F1-score 98.92%;	Weight size: 69,269; PC inference time: 51.55 ms; UP Board inference time: 175.92 ms
Proposed framework	TF-IDF + Word2Vec + FastText with supervised autoencoder dimensionality reduction and GA-optimized MLP deployed in ML-WAF	Accuracy: 0.998681, Precision: 0.999777, Recall: 0.996888, F1-score: 0.998330	Weight size: 0.0747 MB, inference time: 3.4014 ms

As shown in Table 14, previous studies have achieved strong SQLi detection performance using classical machine learning, autoencoder-based feature extraction, embedding-based fusion, and transformer-based architectures. However, most of these studies mainly focused on offline classification metrics, with limited consideration of runtime behavior, deployment feasibility, and real-time request inspection. In contrast, the proposed framework combines competitive detection performance with WAF-oriented runtime validation, demonstrating that the lightweight representation can operate efficiently within an operational request-inspection workflow. Therefore, the contribution of this study extends beyond high classification accuracy by showing that the proposed framework remains practical, compact, and efficient when transferred from benchmark evaluation to a real-time protection environment.

Limitations

Despite the promising results, several limitations should be acknowledged. First, the proposed framework was designed and evaluated specifically for SQL injection detection and prevention; therefore, extending the framework to other web attack categories, such as XSS, command injection, and DoS/DDoS attacks, remains a direction for future work.

Second, although the proposed model produced only one false positive on the held-out test set, this result should not be interpreted as evidence of false-positive-free behavior in real WAF deployment. Production traffic may contain application-specific or uncommon benign inputs that are not fully represented in public datasets. Therefore, strict blocking should be preceded by an initial monitoring phase, request logging, administrator review, and threshold calibration using real benign traffic to reduce the risk of unnecessarily blocking legitimate users.

Third, the SQLMap-based robustness evaluation using 19,990 unseen malicious HTTP requests provided additional deployment-oriented evidence under adversarial-style conditions. Future work may extend this evaluation using more diverse attack-generation tools and real-world traffic traces to further assess robustness under broader operational settings.

Fourth, the runtime evaluation was conducted on a single laboratory server using a Flask-based WAF prototype and selected concurrency levels. Although this setup was

suitable for validating the proposed approach, Flask-based prototype applications may show limited scalability under high concurrent traffic when deployed in a single-server setting, where requests can accumulate in queues and increase latency. Future work should evaluate the system on larger and more diverse deployment environments to better assess its scalability.

Finally, the proposed framework adopts lightweight textual representations, including TF-IDF, Word2Vec, and FastText, to maintain compact model size, low inference latency, and practical real-time WAF deployment. Future studies may explore optimized lightweight transformer models or hybrid semantic representations to further improve contextual understanding while preserving deployment efficiency.

6. Conclusions

This study presented a practical and deployment-oriented framework for SQL injection detection and prevention in Web Application Firewall environments. Unlike evaluations limited to offline classification, the proposed work connected model development with real-time request inspection, thereby addressing an important gap between benchmark accuracy and operational applicability.

The findings show that combining lightweight textual representations with supervised dimensionality reduction can produce a compact and highly discriminative detection model suitable for inline security filtering. The selected MLP model achieved strong detection performance while maintaining a small model footprint and fast inference behavior, making it appropriate for latency-sensitive WAF deployment.

The comparison with fine-tuned CodeBERT further highlighted the different strengths of transformer-based and lightweight machine learning approaches. While CodeBERT achieved a higher F1-score, the proposed framework offered a more favorable balance between detection performance, model size, inference speed, and operational efficiency. This balance is particularly important for real-time WAF environments, where accurate detection must be combined with low-latency request inspection.

The deployed WAF evaluation, including SQLMap-based unseen attack testing and runtime measurements, showed that the framework can support practical SQLi prevention under realistic request-inspection conditions. Overall, the study demonstrates that lightweight machine learning, when carefully integrated and validated, remains a viable and effective option for real-time SQLi protection.

Future work may extend the evaluation through larger-scale deployment testing, broader web attack coverage, and optimized lightweight contextual representations to further improve generalization while preserving real-time efficiency.

Author Contributions: Conceptualization, S.S.A. and M.L.A.d.; methodology, S.S.A. and M.L.A.d.; software, S.S.A. and M.L.A.d.; validation, S.S.A. and M.L.A.d.; formal analysis, S.S.A. and M.L.A.d.; investigation, S.S.A.; resources, S.S.A. and M.L.A.d.; data curation, S.S.A. and M.L.A.d.; writing—original draft preparation, S.S.A.; writing—review and editing, S.S.A. and M.L.A.d.; visualization, S.S.A. and M.L.A.d.; supervision, M.L.A.d.; project administration, S.S.A. and M.L.A.d. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: The authors acknowledge the Computer Center at Northern Technical University (Iraq) for providing server resources used to deploy and evaluate the WAF prototype.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Salam, A.; Ullah, F.; Amin, F.; Abrar, M. Deep learning techniques for web-based attack detection in industry 5.0: A novel approach. *Technologies* **2023**, *11*, 107. [\[CrossRef\]](#)
2. OWASP Foundation. OWASP Top 10—2021: The Ten Most Critical Web Application Security Risks. 2021. Available online: <https://owasp.org/Top10/> (accessed on 21 February 2025).
3. Alghawazi, M.; Alghazzawi, D.; Alarifi, S. Detection of sql injection attack using machine learning techniques: A systematic literature review. *J. Cybersecur. Priv.* **2022**, *2*, 764–777. [\[CrossRef\]](#)
4. Katole, R.A.; Sherekar, S.S.; Thakare, V.M. Detection of SQL injection attacks by removing the parameter values of SQL query. In *Proceedings of the 2018 2nd International Conference on Inventive Systems and Control (ICISC)*; IEEE: New York, NY, USA, 2018; pp. 736–741. [\[CrossRef\]](#)
5. Chen, D.; Yan, Q.; Wu, C.; Zhao, J. Sql injection attack detection and prevention techniques using deep learning. *J. Phys. Conf. Ser.* **2021**, *1757*, 012055. [\[CrossRef\]](#)
6. Demetrio, L.; Valenza, A.; Costa, G.; Lagorio, G. WAF-A-MoLE: Evading web application firewalls through adversarial machine learning. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, Brno, Czech Republic, 30 March–3 April 2020; pp. 1745–1752. [\[CrossRef\]](#)
7. Alotaibi, F.M.; Vassilakis, V.G. Toward an SDN-Based Web Application Firewall: Defending against SQL Injection Attacks. *Future Internet* **2023**, *15*, 170. [\[CrossRef\]](#)
8. Appelt, D.; Nguyen, C.D.; Panichella, A.; Briand, L.C. A machine-learning-driven evolutionary approach for testing web application firewalls. *IEEE Trans. Reliab.* **2018**, *67*, 733–757. [\[CrossRef\]](#)
9. Dawadi, B.R.; Adhikari, B.; Srivastava, D.K. Deep learning technique-enabled web application firewall for the detection of web attacks. *Sensors* **2023**, *23*, 2073. [\[CrossRef\]](#)
10. Liu, C.-Z.; Sheng, Y.-X.; Wei, Z.-Q.; Yang, Y.-Q. Research of text classification based on improved TF-IDF algorithm. In *Proceedings of the 2018 IEEE International Conference of Intelligent Robotic and Control Engineering (IRCE)*; IEEE: New York, NY, USA, 2018; pp. 218–222. [\[CrossRef\]](#)
11. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguist.* **2017**, *5*, 135–146. [\[CrossRef\]](#)
12. Badri, N.; Kboubi, F.; Chaibi, A.H. Combining fasttext and glove word embedding for offensive and hate speech text detection. *Procedia Comput. Sci.* **2022**, *207*, 769–778. [\[CrossRef\]](#)
13. Thalji, N.; Raza, A.; Islam, M.S.; Samee, N.A.; Jamjoom, M.M. Ae-net: Novel autoencoder-based deep features for sql injection attack detection. *IEEE Access* **2023**, *11*, 135507–135516. [\[CrossRef\]](#)
14. Durmuşkaya, M.E.; Bayraklı, S. Web application firewall based on machine learning models. *PeerJ Comput. Sci.* **2025**, *11*, e2975. [\[CrossRef\]](#)
15. Peralta-Garcia, E.; Quevedo-Monsalbe, J.; Tuesta-Monteza, V.; Arcila-Diaz, J. Detecting structured query language injections in web microservices using machine learning. *Informatics* **2024**, *11*, 15. [\[CrossRef\]](#)
16. Ogini, P.; Taylor, E.; Nwiabu, N. A deep learning approach for the detection of structured query language injection vulnerability. *Int. J. Inf. Secur.* **2022**, *11*, 211–217.
17. Pramono, P.; Irawan, R.D.; Sari, A.A. Comparative Analysis of SQL Injection Attack Clasification Using Naïve Bayes Method And Support Vector Machine (SVM). In *Proceedings of the International Conference on Science, Health, And Technology*; UDB Surakarta: Surakarta, Indonesian, 2024; pp. 208–217. [\[CrossRef\]](#)
18. Bakır, R. UniEmbed: A Novel Approach to Detect XSS and SQL Injection Attacks Leveraging Multiple Feature Fusion with Machine Learning Techniques. *Arab. J. Sci. Eng.* **2025**, *50*, 15591–15604. [\[CrossRef\]](#)
19. Alghawazi, M.; Alghazzawi, D.; Alarifi, S. Deep learning architecture for detecting SQL injection attacks based on RNN autoencoder model. *Mathematics* **2023**, *11*, 3286. [\[CrossRef\]](#)
20. Ammar, B.B.; Alharbi, A.M. SQL Injection Detection Using Fine-Tuned CodeBERT. *Eng. Technol. Appl. Sci. Res.* **2025**, *15*, 27852–27857. [\[CrossRef\]](#)
21. Liu, Y.; Dai, Y. Deep learning in cybersecurity: A hybrid BERT–LSTM network for SQL injection attack detection. *IET Inf. Secur.* **2024**, *2024*, 5565950. [\[CrossRef\]](#)
22. Lo, R.-T.; Hwang, W.-J.; Tai, T.-M. Sql injection detection based on lightweight multi-head self-attention. *Appl. Sci.* **2025**, *15*, 571. [\[CrossRef\]](#)
23. Paul, A.; Sharma, V.; Olukoya, O. SQL injection attack: Detection, prioritization & prevention. *J. Inf. Secur. Appl.* **2024**, *85*, 103871. [\[CrossRef\]](#)
24. Hussain, S.S.S. SQL Injection Dataset (Kaggle Dataset). May 2025. Available online: <https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset> (accessed on 3 May 2025).

25. Behaviour-Centric Cybersecurity Center (BCCC). SQL Injection Attack (BCCC-SFU-SQLInj-2023) [Dataset]. May 2023. Available online: <https://www.yorku.ca/research/bccc/ucs-technical/cybersecurity-datasets-cds/sql-injection-attack-bccc-sfu-sqlinj-2023/> (accessed on 10 May 2025).
26. Zhuo, Z.; Cai, T.; Zhang, X.; Lv, F. Long short-term memory on abstract syntax tree for SQL injection detection. *IET Softw.* **2021**, *15*, 188–197. [[CrossRef](#)]
27. Qaiser, S.; Ali, R. Text mining: Use of TF-IDF to examine the relevance of words to documents. *Int. J. Comput. Appl.* **2018**, *181*, 25–29. [[CrossRef](#)]
28. Wang, C.; Liu, H.; Sun, Y.; Wei, Y.; Wang, K.; Wang, B. Dimension reduction technique based on supervised autoencoder for intrusion detection of industrial control systems. *Secur. Commun. Netw.* **2022**, *2022*, 5713074. [[CrossRef](#)]
29. Maazalahi, M.; Hosseini, S. A Novel Hybrid Method Using Grey Wolf Algorithm and Genetic Algorithm for IoT Botnet DDoS Attacks Detection. *Int. J. Comput. Intell. Syst.* **2025**, *18*, 61. [[CrossRef](#)]
30. Aria, M.; Cuccurullo, C.; Gnasso, A. A comparison among interpretative proposals for Random Forests. *Mach. Learn. Appl.* **2021**, *6*, 100094. [[CrossRef](#)]
31. Valkenborg, D.; Rousseau, A.-J.; Geubbelmans, M.; Burzykowski, T. Support vector machines. *Am. J. Orthod. Dentofac. Orthop.* **2023**, *164*, 754–757. [[CrossRef](#)] [[PubMed](#)]
32. Somvanshi, M.; Chavan, P.; Tambade, S.; Shinde, S. A review of machine learning techniques using decision tree and support vector machine. In *Proceedings of the 2016 International Conference on Computing Communication Control and Automation (ICCUBE)*; IEEE: New York, NY, USA, 2016; pp. 1–7. [[CrossRef](#)]
33. Desai, M.; Shah, M. An anatomization on breast cancer detection and diagnosis employing multi-layer perceptron neural network (MLP) and Convolutional neural network (CNN). *Clin. Ehealth* **2021**, *4*, 1–11. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.