Journal of Computer Science

Research Article

Enhancing SQL Injection Detection: A Machine Learning Approach Using Network Flow Data

P. Vinoth, K. Muthamil Sudar and S. Muthukumar

Department of Computer Science and Engineering, Mepco Schlenk Engineering College, Sivakasi, Tamil Nadu 626005, India

Article history
Received: 13-03-2026
Revised: 27-06-2026
Accepted: 15-07-2026

Corresponding Author:
K. Muthamil Sudar
Department of Computer Science and Engineering, Mepco Schlenk Engineering College, Sivakasi, Tamil Nadu 626005, India
Email: k.muthamilsudar@gmail.com

Abstract: Background: SQL injection is one of the cyberattacks which occurs in web application vulnerabilities. It targets the poor input validation and enables the attackers to inject malicious queries into the database fields in any application. It will grant access to unauthorized users and data modification, or they will take control over the complete database. To prevent this, there are several traditional methods, like Intrusion Detection Systems (IDS) and packet inspection methods, which will detect these attacks by each and every network packet in the network traffic. Methods: The existing methods also have some limitations due to computational overhead due to monitoring each and every packet. It makes it challenging for high-end routers and large-scale networks as it monitors all the individual traffic. To resolve this problem, propose a flow-based detection method, making use of lightweight protocols like NetFlow and sFlow to identify SQLI attacks. Unlike traditional methods, which require more inspection on individual packets, flow-based analysis uses mandatory communication metadata, like source and destination IP addresses, port numbers, etc. It minimizes the need for computationally expensive packet inspection, which is going to render the process of detection more trustworthy and economical, particularly within high-traffic conditions. Results: In order to validate this approach, we gathered two collections of data, which included net flow data of SQLI attacks of database systems such as SQL, postgres SQL, etc. It has both normal and bad traffic, and we applied machine learning models to the traffic. Conclusion: The results indicate that the proposed flow-based method is more efficient and scalable for detecting SQL injection attacks than packet-based inspection.

Keywords: SQL Injection Detection, Web Application Security, Flow-Based Analysis, Machine Learning Classification, Network Traffic Monitoring

Introduction

Background

The use of cyber-attack has become a significant menace to people, businesses, and organisations across the globe (Casmiry et al., 2026). With the number of activities involving the use of web applications such as financial transactions, online shopping and access of information, attackers often find themselves targeting these applications because of their high exposure. SQL Injection Attack (SQLI) is one of the most threatening risks that should not be overlooked among other cyber threats. SQLI enables the hackers to use input vulnerabilities in the back end of the web applications to inject malicious SQL queries in it. This facilitates access to sensitive data by unauthorized users like user information and passwords and causes data leaks, system control, or data manipulation (Wang et al., 2026). Open Web Application Security Project (OWASP), an established expert in software security, continuously lists SQL injection on its OWASP Top 10 list as a leading risk, and place it at number three in 2024 (Satar et al., 2026). Likewise, this vulnerability is also threatening to the world as the MITRE CWE Top 25 Most Dangerous Software Weaknesses report has listed SQL injection attacks in the third position. Such ranks point to the fact that SQLIs remain common and may have an influence on a broad spectrum of web-based systems. The other methods of SQL injection attacks have been reported widely both in learning and security (Elshewey, 2026). As an illustration, to get access to the hidden data, attackers can alter the query to come up with unauthorized answers, interfere with the application logic by injecting

conditional statements, or attack it by using UNIONs to retrieve data across many tables. Blind SQL injection is where the attacker is deducing database information based on behavioral data as opposed to direct output, and is therefore more difficult to identify by the traditional means.

SQLI Injection Mechanism

In the database of the web application, SQLI vulnerabilities can be found. There are many types in the SQL Injection. Some of them are:

SQL Injection Through Cookies: Attackers can exploit SQLI by manipulating cookies stored in the user's browser (Sahbi et al., 2026). Web applications that trust and use cookie data directly in SQL queries without proper sanitization are vulnerable to this form of attack. The attacker modifies the cookie value to inject SQL code, which is then interpreted by the server when the cookie is read. This method is particularly dangerous because it often goes unnoticed by the user and is processed automatically by the server. Example: A web app stores user ID in a cookie: User_id = 10. It is modified by an attacker to user_id = 10 OR 1 = 1, and access is gained unauthorized.

Injection Via the User Input: It enables the attacker to manipulate the queries which an application sends to its database (Balogun et al., 2025). In cases where the input fields such as the login form, search form or even feedback form fail to validate or sanitize user input, the input field becomes a direct entry point to SQL commands. The attacker will insert malicious SQL statements that will change the logic of the SQL query, and it will be convenient in gaining unauthorized access or sensitive information of the system. Example: Inputting ' OR '1'='1 into a login field can trick the database into bypassing authentication.

SQL Injection Using Server Variables: By usage of server variables which contains network headers and environment variables (Mariettou et al., 2025). In web applications we use these server variables like recording usage statistics and identifying browsing trends. Without proper validation if these variables are recorded in database, it will create an SQLI vulnerability.

Example: An attacker sets the User-Agent header to ' OR 1 = 1--, and the server logs this directly into a vulnerable SQL query.

Second-Order or Stored SQL Injections: This is a more complex and challenging attack. It involves injecting data that does not immediately trigger an attack but is stored and later executed in a different context (Smrity and Muntaqim, 2025). In the first phase, the attacker stores a malicious SQL fragment in the database. In the second phase, another functionality in the application processes that data without sanitization, triggering the attack. Example: A user signs up with a bio containing ' OR '1'='1. The injection occurs later when an admin is viewing the profile of the user, the app makes a SQL query with the bio which is stored, resulting in the injection.

SQLI Types

Error attack: This technique uses lengthy error messages that are sent by the database. The attacker develops queries in a manner that they result in error generation by the database. Such messages may contain information on the database structure such as names of table, name of fields and the types of data. It is normally applied at the initial stages of an attack to conduct reconnaissance and customize subsequent injection according to the database schema (Anwar, 2025).

Example:

Query : SELECT * FROM users WHERE id = 1'
Error 1: Syntax error near 1 - shows the structure of the query

Union attack: This is a method that involves the use of SQL UNION operator to combine the result of a second query with the first query. It enables the attacker to access information on other tables and columns that would otherwise not be accessible to it via the application interface. It is common to perform a combination of the vulnerable query results and such information as the usernames, passwords, or credit cards (Casmiry et al., 2025).

Example:

Original query : SELECT name FROM products1 WHERE id = '1267'
Injected query : 1 UNION SELECT user_name, passwd FROM attendee

Blind attack: Blind attack is the most sophisticated and elaborate way of attack among the said types of attacks.This method functions when an application produces no visible or error-based indications during execution. The attacker initiates true or false condition-generating payloads that allow the laser type evaluation of application behavior changes or duration variations (Alnemari and Alajmani, 2025).

Challenges

Conventional methods of detection like manual code review and deep packet inspection can be quite ineffective and not viable in large scale or real time implementation as they can be resource intensive (Abubakar et al., 2025). A more scalable solution is to employ flow-based network telemetry, which records important metadata of network connections but does not examine individual packets. Efficient monitoring of network behavior is achieved with lightweight protocols such as NetFlow, sFlow and IPFIX since they can process high traffic volumes (Chen et al., 2025).

Objectives

This study introduces a framework that uses the flow-based network analysis and machine learning technology to identify SQL injection attacks. The system is based on two publicly available labelled datasets which are under the Creative Commons Attribution 4.0 International license. Such datasets consist of NetFlow Version 5 logs that record benign traffic as well as the SQLI generated traffic.

Contributions

The major contribution of proposed framework are as follows:

(i) A lightweight detection framework for SQL injections through NetFlow V5 flows is suggested that bypasses expensive packet inspection process
(ii) A comprehensive analysis of various classifiers based on machine learning and deep learning is performed to detect SQL injection in flows such as Logistic Regression, Random Forest, SGD, Voting Classifier, and Regularized ANN (R-ANN)
(iii) A new hybrid approach for feature engineering by selecting important features through statistical, ensemble, and neural network techniques is proposed to detect SQL injections in flows
(iv) An R-ANN classifier with regularization using L2, dropout, and batch normalization is developed to classify the SQL injection traffic
(v) The results indicate that it is possible to efficiently detect SQL injection using flow-level telemetry data instead of packet-based deep packet inspection methods

Literature Survey

There are so many findings about the detection of SQLI using Machine Learning approaches through this literature survey.

Machine Learning-Based SQL Injection Detection

An SQL injection attack detection and stage identification method, termed SDSIOT (Siale et al., 2025). Dynamic detection methods use penetration testing for vulnerability detection. The limitation for this methodology has high false negative rates for unknown SQLI and struggle with runtime behavior vulnerabilities. The use of decision tree and random forest algorithms for SQL injection attack detection was proposed (Mannuru et al., 2026). The research done by Preprocessing to handle missing values and duplicates. By using Decision Tree and Random Forest models and evaluating the model performance using performance metrics. The limitation for this methodology that Decision Tree algorithm is less accurate than Random Forest and Decision Tree may lack transparency in complex decision-making.

Machine learning algorithms such as Logistic Regression, Stochastic Gradient Descent, Sequential Minimal Optimization, Naïve Bayes, Multilayer Perceptron, Instance-Based Learner, and Bayesian Network were implemented for SQL injection attack detection (Bakır, 2025). Limitation for this methodology that Machine learning approaches require large data for model preparation and limited number of attack types handled by machine learning methods. A hybrid approach for detecting SQL injection attacks using machine learning techniques was proposed (Mathew and Vidhate, 2025). Supervised learning is used to train the model. Limitations for this methodology for most existing approaches fail to cover the entire problem scope and are limited to a small dataset for training and testing.

Deep Learning and NLP-Based Approaches

SQL injection attack detection and prevention techniques using a natural language processing model were suggested (Okello et al., 2026). Convolution Neural Network (CNN) is employed for classification. Multilayer Perceptron (MLP) is used for identifying malicious requests. The limitation for this methodology by traditional filtering systems face many problems with evolving attack methods. SQL injection detection requires high machine performance for effective results. Attack detection for web applications using deep learning was implemented, and a systematic literature review and quality assessment of selected primary studies (Naga Charan et al., 2022). But it was limited by the lack of comparison between selected studies performance and different datasets and performance measures used in this paper.

A hybrid CNN–BiLSTM model for SQL injection detection integrated with an ensemble learning algorithm was proposed (Floris et al., 2025). It was not implemented to SQL injection detection generated using AI techniques like Generative Adversarial Networks (GANs). A bidirectional LSTM–CNN architecture with a consensus function for SQL injection attack detection was proposed (Afrin et al., 2025). To interpret and analyze, they used LIME. It was limited to not used to detect different injection attacks like Cross-Site Scripting (XSS) and it used only the static monitoring approaches.

Ensemble, Hybrid and Rule-Based Approaches

An ensemble-based SQL injection attack detection approach for IoT ecosystems using nine base classifiers (Lo et al., 2025). For feature extraction, they used CBOW and SKG Word2Vec methods. Naive Bayes classifiers for probabilistic classification and logistic regression for SQLI probability assessment. This method fails at camouflaged attacks. Methods didn't address overfitting effectively. A proxy-based static analysis approach for identifying SQL statements and injection points was proposed (Muthamil and Deepalakshmi, 2021). Data

between web applications and databases captured by dynamic execution. The research was done by developing the matching rules detect first and second-order SQL injection vulnerabilities. Escaping and truncating malicious payloads mitigate negative database impacts. SQLPsdem was limited by can't classify four attack types effectively. Performance was limited detecting zero-day attacks due to rule-based approach.

Advanced Deep Learning Models

A deep learning based method for identifying SQL injection attacks was proposed (Zimmer et al., 2024). The model uses neural networks to learn patterns of SQL queries to differentiate between bad entries and valid ones. Pre processing of the data and extraction of features were undertaken in order to improve the accuracy of the models and training was done on a sample of SQL injection. The study has shown a high detection rate as opposed to conventional means, which were able to identify common attack vectors. The use of the model was however limited by the fact the model depends on the quality and variety of the training dataset and may not be able to deal with new or zero-day SQL injection techniques. A proposal was made with regards to an unsupervised deep learning model that involves RNN autoencoders to detect SQL injection (Arasteh et al., 2025). The architecture gets to know the normal forms of legitimate SQL queries and detects anomalies based on reconstruction error, which is a measure that an injection attack is possible. It was trained on a labeled set and tested on it showing great accuracy and resilience in identifying known and never-before-seen SQL injection patterns. The use of reconstruction thresholds, however, also creates the issue of fine-tuning sensitivity, and the method can generate false positives in the case of legitimate queries that are not normal by the learned distribution. It proposed a deep learning based on recurrent neural networks to identify SQL injection attacks based on the sequential format of SQL queries (Sudar et al., 2024). The model takes queries in the form of character or token sequences, it is trained to learn their characteristics over time to differentiate between benign and malicious inputs. The study employed a labeled dataset that consisted of normal and injection-laden queries with high classification performance with high accuracy and precision. Although it has been shown to be effective in capturing sequential patterns, the efficiency of the model relies on the quality of the preprocessing of data and can be limited by the ability to deal with obfuscated or highly diverse techniques of injection not found in the training data.

Flow-Based and Network-Level Approaches

To assist in malicious traffic detection, a system based on the NetFlow sensors was proposed to analyze flow-level network traffic (Muthamil and Deepalakshmi, 2021). This paper demonstrates that the significance of real-time flow information capture and feature recognition as a basis of training viable detection systems. The suggested solution can be used to generate high-quality datasets that are optimized to machine learning-based intrusion detection and increase the capacity to detect abnormal behavior in network traffic. Although the method is scalable and efficient, the method depends on correct labelling and feature selection and its operation might be less efficient when dealing with encrypted or obfuscated traffic patterns.

Adversarial and Generative Learning Approaches

An innovative method to increase the resilience of web application firewalls to the adversarial SQL injection attacks was suggested (Alghawazi et al., 2023). The paper presents the ModSec-AdvLearn, a learning model that combines adversarial training and ensemble to automatically fine-tune and provide selections of ModSecurity rules. The model can use adversarial examples during training to obtain increased detection accuracy and better generalization to advanced, obfuscated attack examples. The experimental findings showed that the framework was capable of working better compared to classic signature-based systems. Nevertheless, the quality and the variety of adversarial examples are conditioned on the effectiveness of the model, and the model also comes with extra computational costs at the training stage.

It proposed an upgraded framework of detection, which used generative models to enhance the detection and prevention of SQL injection attacks (Kakisim, 2024). The paper discusses the application of Variational Autoencoders (VAEs) and Conditional Wasserstein GANs (CWGANs) to produce realistic queries generated via SQL synthesis to augment the training data of machine learning classifiers. The method of data augmentation improves the generalization of models and decreases the chance of over fitting, thus improving the accuracy of detection of known and unknown attack patterns. Although the method is particularly helpful in improving the flexibility of the detection systems, it also adds complexity to training the generative models and requires a keen tuning to ensure that it produces neither too simple nor unrealistic samples of the attacks.

NLP and High-Throughput Detection Systems

A new cascaded NLP-based system to find SQL injection attacks in high-throughput systems was introduced (Aulianoor and Koprawi, 2024). They use a combination of tokenization, semantic analysis, and embedding in context as a better way to comprehend the structure of the SQL query and detect malicious behavior. With the multiple-stage architecture of applying transformers and attention algorithms, the system is capable of delivering a high detection rate and a low latency, thus being applicable in real-time deployment in

data center systems. Although it is efficient, the model is complex to operate in terms of resource usage and must be constantly tuned to meet the changing query patterns and evasion strategies. The authors of Fu et al. (2022) proposed a machine learning-based method of detection of adversarial phishing attacks with the help of several classification algorithms. This system was trained on the basis of features of URL structures, domain features and lexical patterns in order to detect phishing attempts very well. The model was also compared with adversarial examples designed to evade conventional filters, which demonstrates its strength in adversarial evasion examples. Nevertheless, the research predominantly looked at phishing threats and URL-based features, meaning that it was not as flexible as payload-based attacks, such as SQL injection. It also did not perform well in real-time detection situations because input features were static, and behavioral analysis was not dynamic.

Existing SQL injection detection studies primarily focus on payload-based inspection, query parsing, or deep packet inspection techniques. Although these methods achieve high detection rates, they often incur high computational overhead and are difficult to deploy in high-throughput environments. Recent deep learning approaches improve detection capability but require extensive training data and computational resources. Furthermore, limited attention has been given to lightweight flow-based telemetry for SQL injection detection. Motivated by these gaps, the present study investigates whether NetFlow-level metadata combined with machine learning can provide scalable and accurate SQL injection detection without relying on full packet inspection. Table 1 presents a comparative analysis of existing SQL injection detection approaches. Most existing studies rely on payload inspection, query parsing, or computationally intensive deep learning methods. In contrast, the proposed work focuses on lightweight NetFlow-level telemetry combined with machine learning and Regularized ANN-based classification to achieve scalable and efficient SQL injection detection with reduced computational overhead.

In the present work, an improved deep learning algorithm that involves SQLI detection is offered to overcome the weakness of contemporary algorithms. Conventional methods like outbound traffic strategy and dynamic detection strategies tend to have high false-negative when facing unknown attacks and runtime vulnerabilities. Although such classification models as Random Forests are more efficient than the Decision Trees in detecting the SQLI, they have drawbacks when it comes to detecting the intricate attack patterns. In the same way, algorithms like the Logistic Regression and the Naive Bayes involve the use of large datasets limiting their application in different scenarios of attacks. Hybrid machine learning methods strive to increase detection accuracy but work usually on small datasets which restricts their extrapolation. SQL query classification has been applied using deep learning algorithms such as CNNs; although they need large computing resources, they might not be able to handle evolving attack patterns. In a bid to address these issues, our suggested solution will use a deep learning model that has several dense layers, batch normalization, drop out regularization, and optimized Adam optimizer.

Table 1: Comparative Analysis of Existing SQL Injection Detection Approaches

Methodology	Detection Approach	Feature Type	Strength	Limitation
SDSIOT (Fu et al., 2023)	Dynamic outbound traffic analysis	Traffic behavior	Detects staged SQL injection attacks	High false-negative rate for unknown attacks
Decision Tree and Random Forest (Aulianoor and Koprawi, 2024)	Traditional Machine Learning	Statistical query features	Good classification performance	Decision Tree lacks robustness for complex patterns
CNN-based SQL Injection Detection (Okello et al., 2026)	Deep Learning (CNN + NLP)	SQL query payloads	High detection accuracy	High computational overhead
BiLSTM-CNN Hybrid Model (Floris et al., 2025)	Deep Learning Hybrid	Sequential query patterns	Improved pattern learning	Limited evaluation against AI-generated attacks
RNN Autoencoder (Alghawazi et al., 2023)	Unsupervised Deep Learning	Query sequence behavior	Detects unknown attack patterns	Sensitive to reconstruction threshold tuning
Ensemble IoT SQLI Detection (Gowtham and Pramod, 2022)	Ensemble Learning	Word embedding features	Robust against multiple attack styles	Weakness against camouflaged attacks
ModSec-AdvLearn (Floris et al., 2025)	Adversarial Learning	Web request payloads	Improved resilience to adversarial attacks	Increased training complexity
NetFlow-based IDS (Muthamil and Deepalakshmi, 2021)	Flow-based Network Analysis	Flow-level metadata	Scalable and lightweight monitoring	Limited payload visibility

Through combining these methods, we will have improved generalization, less overfitting, and improved detection of accuracy by preserving efficiency of computation. It makes our approach better than the other in recognizing SQLI threats, as it is powerful and efficient in the current cyber security issues.

Methods

System Design

The proposed workflow consists of five main stages: Data generation using benign and malicious datasets, preprocessing and normalization, feature selection using multiple statistical and neural methods, classification using diverse ML/DL models, and evaluation to determine the accuracy of benign vs. malicious request detection. Figure 1 shows the system diagram of proposed work organized internet traffic into two categories which include innocent Benign traffic and hostile Malicious traffic. Our goal is to obtain network traffic data through SoftFlowd together with nfcapd and Wireshark capture programs over a period of 72 hours. This process yielded approximately 57299 labeled flows, evenly balanced between benign and malicious instances.

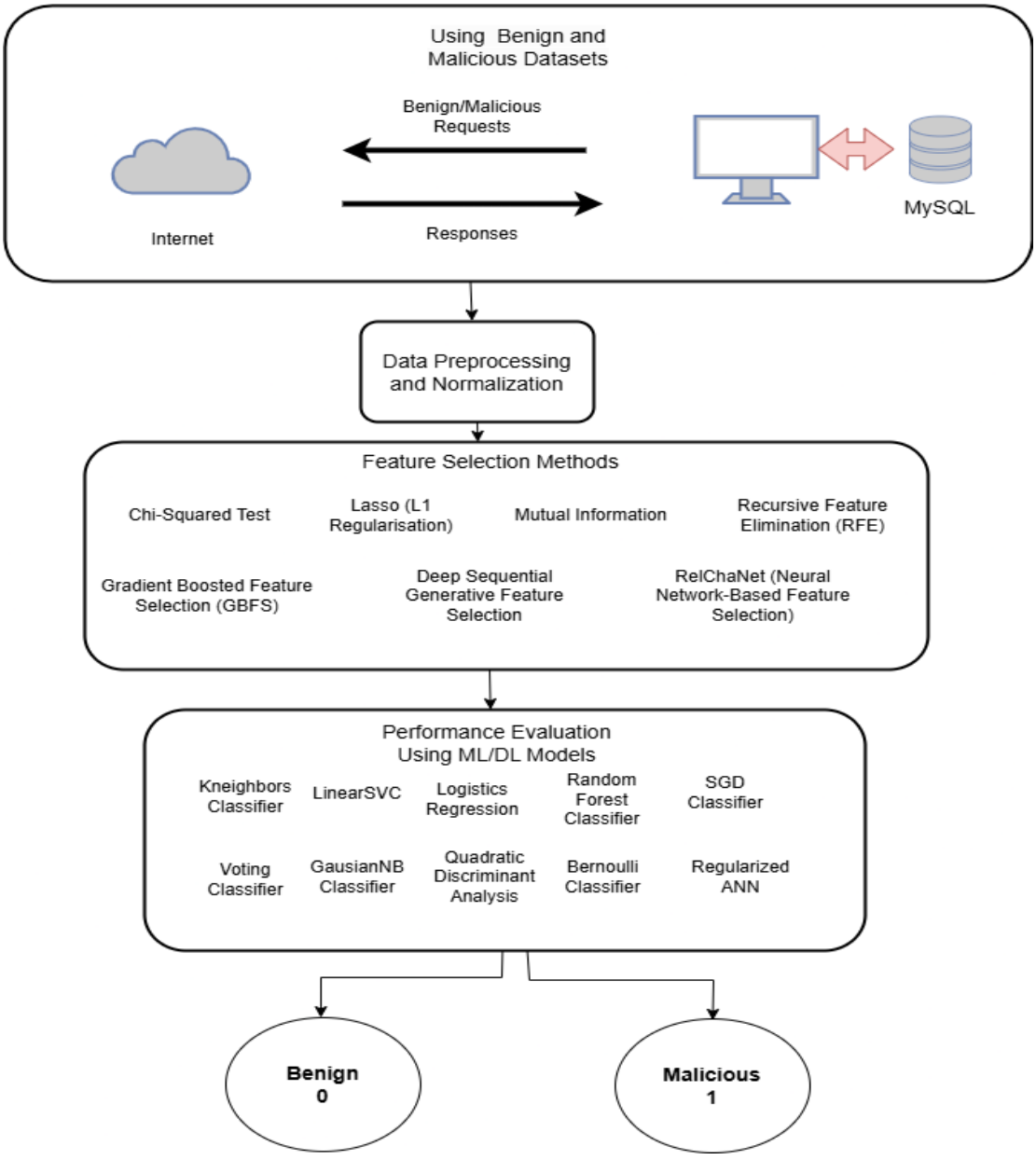

Fig. 1: System Design of proposed Work

The method involved defining the creation of both benign and malicious traffic patterns through which packets undergo the simulated systems' communication process. The process uses simulated traffic packets for benign emulation. The same steps are followed during malicious traffic processing as well. Python scripts execute the operations required for generating both benign and malicious traffic. The acquired data undergoes labeling process where the information gets categorized into benign [0] and malicious [1] types. Data preprocessing along with normalization forms the second step because it stops overfitting and prepares the dataset for usage with different machine learning techniques. Feature selection follows data preprocessing by implementing seven techniques which include Chi-Squared Test and Lasso (L1 Regularisation) and Mutual Information and Recursive Feature Elimination (RFE) and Gradient Boosted Feature Selection (GBFS) and Deep Sequential Generative Feature Selection and RelChaNet as a neural network-based feature selection approach. The selected methods serve two purposes: They simplify the data while maintaining the most important components. Performance evaluation using selected Machine Learning and Deep Learning models consists of K-Nearest Neighbors Classifier, Linear SVC, Logistic Regression, Random Forest Classifier, SGD Classifier, Voting Classifier, Gaussian Naive Bayes, Quadratic Discriminant Analysis, Bernoulli Classifier and Regularized Artificial Neural Networks (ANN). At the end of the process the systems predicts if the traffic arises from benign or malicious activities. The performance metrics including Accuracy and F1 Score among others help evaluate the effectiveness of each model. Our system detects SQL Injection attacks through the implementation of injection payloads into vulnerable cookies to effectively analyze and classify injection-based anomalies.

Data Preprocessing

The model refinement process happened during data processing while removing data bias that naturally occurred from the data generation method. There are three sequential steps for data processing techniques in this study.

Feature Cleaning: The IP addresses undergo number conversion and blank rows and columns receive a check during the feature cleaning process to maintain model accuracy.

Dimensionality reduction: It exists to simplify the complexity found in models. Each feature in the model system requires an exponential increase in complexity levels. For each feature, the complexity of the models increases exponentially. In practice, the addition of features increases the dimensionality of the data, which can lead to the “curse of dimensionality” a phenomenon where higher dimensions require more data and computational resources to achieve reliable performance. It goes with the decreasing detection capacity. For the Netflow V5 version it has total 24 features. To reduce redundancy and computational overhead, Principal Component Analysis (PCA) was employed, ensuring that the most informative components were retained while minimizing exponential increases in complexity.

Data normalization: To increase the accuracy normalizing the values in the features which is mandatory and also it will manage the error which is caused due to size of data. The typical number of digits in an IP address reaches ten but packet counts in individual flows stay near two digits. Model training produces biases when it gives preference to the IP address when the data remains unnormalized. This work adopts min-max-based linear data normalization as its technique:

$$X_{normalized} = \frac{X - Min(A)}{Max(A) - Min(A)} \quad (1)$$

Where A is the dataset and X is the individual data point being normalized and can take any value from the dataset *A. Min(A)* and *Max(A) Max(A) Max(A)* are constants because they are determined by the dataset A and remain unchanged throughout the normalization process for all values of X.

Features Selection Methods

Chi-Squared Test

The chi-squared Test (χ^2 Test) is done by test that helps deciding whether it has relationship between two categorical variables. Zimmer, (2024) It is used to compare observed data with expected data assuming independence. The test assists in analyzing whether differences in expected and observed frequencies are a result of chance or actually an association:

$$\chi^2 = \Sigma \frac{(O_i - E_i)^2}{E_i} \quad (2)$$

O_i - Represents the observed count in the $i^{\text{-th}}$ category or cell of the contingency table

E_i - Represents the expected count in the $i^{\text{-th}}$ category or cell, calculated based on the null hypothesis

χ^2 - Is the output of the formula, the test statistic itself, which varies depending on the values of $O_i$and E_i

Lasso (L1 Regularisation)

The lasso (Least Absolute Shrinkage and Selection Operator) is a regression that does feature selection as well as regularization to enhance model performance and avoid overfitting Arasteh et al. (2026). It does this by including an L1 penalty in the loss function, which compels some of the coefficients to reduce to exactly zero, essentially dropping less significant features:

$$\text{Minimize}\left(\frac{1}{2n}\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j)x_{ij})^2 + \alpha \sum_{j=1}^{p} |\beta_j|\right) \quad (3)$$

y_i - For the i^{-th} observation it observes the responses
x_{ij} -Value of the j^{-th} predictor for the i^{-th} observation
β_0 -Intercept parameter to be estimated
β_j -Coefficient for the j^{-th} predictor, to be estimated
n - Number of observations
p - Number of predictors
α - Regularization parameter
$\frac{1}{2}$ - Fixed numerical scaling factor

Mutual Information

The mutual information measures the mutual dependency of two random variables. It calculates the degree to which the knowledge of the value of one variable decrease the uncertainty about the other (Sudar et al., 2024). It differs from correlation in that it captures both linear and non-linear relationships. *MI* is commonly employed in feature selection, information theory, and machine learning to quantify the relationship between variables.$x\epsilon X x\epsilon X x\epsilon X$:

$$I(X;Y) = x\epsilon X \sum x\epsilon Y \sum (x,y) log(\frac{p(x,y)}{p(x)p(y)}) \quad (2)$$

x - Outcomes of X
y - Outcomes of Y
$p(x,y)$ - x and y which comes in joint probability
$p(x)$ - x (marginal probability)
$p(y)$ - y (marginal probability)
$I(X;Y)$ - Mutual Information, the result
X - Random variable X from sample space
Y - Random variable Y from sample space

Recursive Feature Elimination (RFE)

The recursive feature elimination is a machine learning feature selection method that eliminates the least relevant features recursively in order to find the most critical features. Muthamil et al. (2021) RFE trains a model, sorts feature according to their relevance, and repeatedly discards the least relevant features until the specified number of features are obtained. RFE is normally employed with those algorithms that support feature importance scoring, including Support Vector Machines (SVMs), linear regression, and logistic regression.

Deep Sequential Generative Feature Selection

The deep sequential generative feature selection is a feature selection method that performs feature selection which employs deep learning models like neural networks to dynamically and sequentially pick the most significant features for a prediction problem (Alghawazi et al., 2023). As opposed to conventional methods such as Recursive Feature Elimination (RFE), DSGFS utilizes generative models and reinforcement learning to choose the optimal set of features. The method is particularly well-suited forhigh-dimensional data as well as in scenarios where feature interactions are complex and non-linear.

Gradient Boosted Feature Selection (GBFS)

The gradient boosted feature selection refers to feature selection using Gradient Boosting Machines (GBM), i.e., XGBoost, LightGBM, or CatBoost, for ranking and selecting the most impactful features (Kakisim et al., 2024). GBFS is a high-performance computational algorithm for high-dimensional datasets that takes advantage of the strengths of ensemble learning in selecting the most informative features to utilize when building predictive models.

RelChaNet (Neural Network Based Feature Selection)

The RelChaNet (Relevant Characteristic Network) is a neural network-based feature selection technique that chooses the most significant features through learning feature importance with deep learning methods (Aulianoor et al., 2024). Compared to conventional feature selection techniques based on statistical testing or tree models, RelChaNet employs deep learning architectures to compute feature importance dynamically.

Table 2 shows the feature selection ranking table presents a comparison of different methods applied to rank the most significant features of a dataset based on the most number of counts.

Table 2: Feature Selection ranking table

Features	Chi Square	Lasso	Mutual Information	RFE	RelChaNet	DeepSeq	GBFS	Count
Dpkts	✓	✓	✓	✓	✓		✓	6
Dockets	✓	✓	✓		✓		✓	5
Srcaddr	✓	✓	✓	✓	✓	✓	✓	7
Dstaddr	✓	✓	✓	✓	✓	✓	✓	7
Output	✓	✓	✓	✓	✓	✓		6
Prot	✓		✓	✓	✓	✓		5
Tos	✓							1
Tcp flags	✓	✓	✓	✓	✓	✓	✓	7
inpu t		✓	✓	✓	✓	✓	✓	6
Srcport		✓		✓		✓	✓	4
dstport						✓	✓	2

The methods vary from Chi-Square, Lasso, Mutual Information, Recursive Feature Elimination (RFE), RelChaNet, DeepSeq, and Gradient Boosted Feature Selection (GBFS). Each of the methods applies a different selection criterion, and the checkmarks indicate which features are chosen by each method. The last column, "Count," indicates the number of methods that chose a given feature, representing a measure of its general significance. Notably, features like "srcaddr" and "dstaddr" have the highest selection count, indicating high relevance, and features like "dpkts," "output," and "tcp_flags" also have high (count of 7), indicating importance in the dataset. On the other hand, the selection counts of other variables such as tos and dstport are the least implying that they might not be significant predictive variables when modeling. The table also indicates that some methods, like Mutual Information choose fewer features, and GBFS and DeepSeq choose a larger set of significant features. Overall, this ranking is helpful in the identification of the most predictive features for the construction of an effective predictive model, as highly ranked features will contribute more to performance, and lower-ranked features will be less significant or redundant.

Classification Methods

Ridge Classifier

The Ridge Classifier is particularly effective in detecting SQL injection attacks due to its ability to handle multicollinearity among features using L2 regularization. In flow-based datasets where attributes can be highly correlated, Ridge prevents overfitting and provides stable decision boundaries, making it suitable for classification in high-dimensional network environments.

Figure 2 shows how each network-based input feature receives its own weight assignment in the SQL Injection (SQLI) detection system using Logistic Regression modeling. The Net Input Function (Σ) processes these features by adding weighted sums in order to characterize the linear relationship between features and the target. The resulting output proceeds to the Logistic Activation Function that converts it into probability scores within the range of 0 to 1. Finally, the model outputs a decision (SQLI or benign), making it suitable for binary classification. All system components empower precise identification of malicious network activities.

Linear SVC

The Linear Support Vector Classifier (Linear SVC) is suitable for SQL injection detection as it creates an optimal linear hyperplane that separates benign and malicious traffic (Fu et al., 2023). It performs well when features are scaled and relatively linearly separable, as is often the case in normalized NetFlow-based traffic data used for detecting injection anomalies.

Figure 3 shows a Linear Support Vector Classifier (SVC) which separates malicious network traffic (red) and benign traffic (green) through two selected features at X_1 and X_2. Each network flow specimen appears as a point whose placement according to feature values determines its position. The support vector classifier establishes the black line as the best separating boundary by maximizing the distance between the data sets. Support vectors define the margin boundaries since they represent the points which lie closest to the hyperplane. The visual representation shows Linear SVC separating SQL Injection traffic from normal network behavior with accuracy.

Random Forest Classifier

Random forest classifier is ideal for SQL injection detection because of its ability to model complex, non-linear patterns and reduce overfitting through ensemble learning (Boyd et al., 2004). By aggregating predictions from multiple decision trees, it captures intricate variations in flow-level features that differentiate malicious injections from benign traffic.

Figure 4 depicts the workflow of Random Forest Classifier which utilizes multiple decision trees to analyze one input record (network flow) until reaching a decision (Boyd et al., 2004). Tree-1 among other trees makes autonomous predictions by using flow duration and packet size as features.

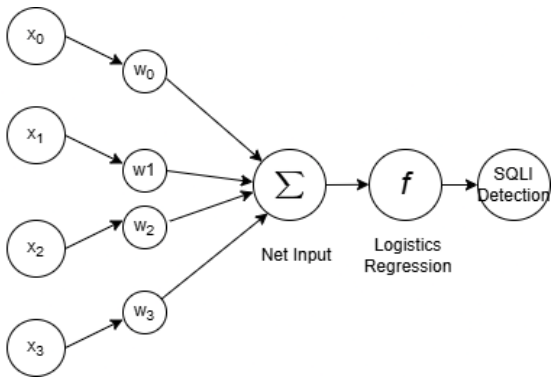

Fig. 2: Ridge Classifier

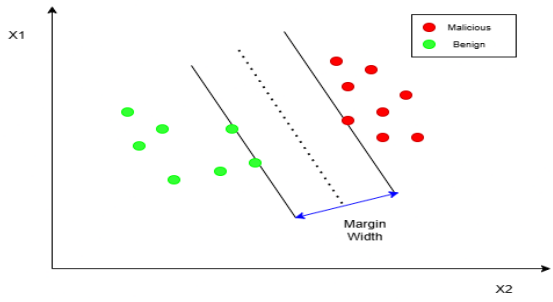

Fig. 3: Linear SVC

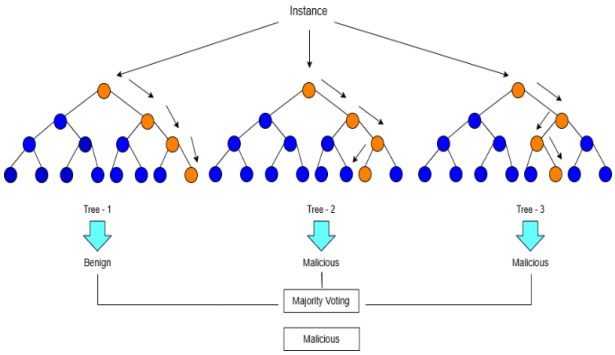

Fig. 4: Random Forest Classifier

The majority voting process compiles individual predictions to decide whether the traffic is SQL Injection or benign. So, when we used to implement ensemble learning it will results in better accuracy and robustness for identifying intricate attack patterns.

SGD Classifier

The appropriate classifier that should be used to detect large-scale and real-time SQL injection is the Stochastic Gradient Descent classifier. To its rapid convergence and the capacity to update models at a time (Al-Shammare et al., 2024). This renders it useful in high-throughput. applications in which data is received in real-time, e.g. in live network traffic monitoring systems.

Figure 5 shows the optimization of the SGD classifier which is designed to minimize the error of SQL injection detection by optimizing weights. The figure has two axes on the weight update layer that relies on network flow data as well as the measure of the loss value (misclassification rate dimension). The red steps are the weight adjustments as a result of the analysis processes of the network flow conducted by a person. The model follows a course towards the global minimum accuracy in detecting malicious and benign traffic considering the possibility of temporary local minima.

Voting Classifier

The Voting Classifier is an improvement of SQL injection detection that uses prediction by a collection of base classifiers, and hence the collective strength of the base classifiers (Zhang et al., 2024). This grouping technique enhances strength, minimizes fluctuation and offer a balanced trade-off amid accuracy and recall, which is essential in classifying complex attack patterns.

Figure 6 demonstrates a voting classifier system which detects SQL Injection attacks. The Training Set labeled network traffic dataset allows researchers to train three classification models including Logistic Regression and SVM and Random Forest. The models make benevolent or malicious predictions about incoming network data before a majority voting process aggregates them into one final prediction. A collective decision-making method links personal decisions to create an absolute prediction between Benign and Malicious hence improving both accuracy and robustness in identifying dangerous SQLI traffic.

Kneighbors Classifier

The K Nearest Neighbors (KNN) Classifier is useful for identifying anomalies in SQL injection traffic based on proximity to known patterns. It classifies traffic based on similarity to labeled instances, making it suitable for scenarios where SQLI behaviors deviate significantly from legitimate flows in feature space (Chen et al., 2021).

Figure 7 demonstrates to identify the classification of new data through nearest instance distance. The X_1 and X_2 coordinate axes consist of extracted features that include request size and flow duration.

Fig. 5: SGD Classifier

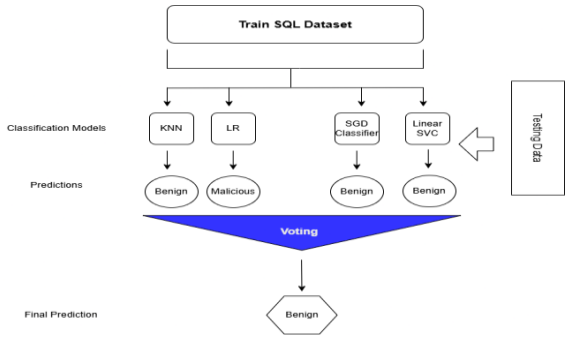

Fig. 6: Voting Classifier

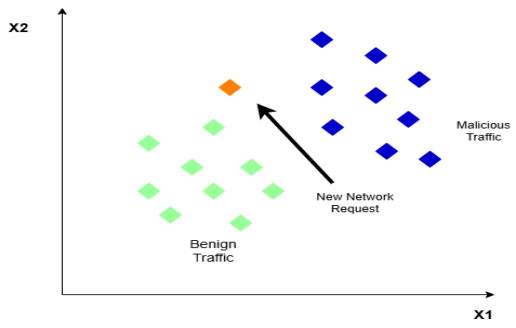

Fig. 7: Kneighbors Classifier

The points represented by green and blue colors correspond to pre-labeled benign network traffic and SQLI attacks. A new network request with the orange point gets its predicted class by determining the majority class of its closest neighbors. The approach makes it possible to detect SQLI successfully by analyzing patterns between fresh and documented network traffic data.

Gaussian NB Classifier

Gaussian Naive Bayes is effective in SQL injection detection when features follow a normal distribution, as is often achieved after normalization. Its probabilistic nature and simplicity allow for rapid classification of flow records, making it ideal for lightweight and interpretable intrusion detection systems.

Figure 8 shows how Gaussian Naive Bayes determines the classification of new data points x by evaluating their likelihood distribution between benign traffic (Class A red curve) and SQLI attack traffic (Class B blue curve). The distances $(x - \mu_A)/\sigma_A$ and $(x - \mu_B)/\sigma_B$ show the deviation distance of the new data point from each class mean. The newer data point will get categorized into the density areas of $p(x|A)$ or p(x|B) that demonstrates highest probability density. The probabilistic system identifies SQL injection attempts through statistical traffic analysis of features including request length and keyword frequencies as well as symbol count patterns.

Quadratic Discriminant

QDA is suitable for SQL injection detection because it allows for modeling different covariance structures for each class. This flexibility helps in capturing more complex relationships in the flow data, especially when benign and malicious traffic have distinct statistical characteristics.

Figure 9 explains the implementation of Quadratic Discriminant Analysis (QDA) for detecting SQL injection (SQLI). The bytes_transferred appear along the X-axis and packet_size appears along the Y-axis in this diagram since both aspects are network-traffic-derived features. Each network session appears as a colored dot in the diagram which identifies its category either as malicious-red or benign-green or remaining blue for ambiguous or other traffic. In QDA decision boundary marking traffic through statistical patterns the light blue curved line functions as the separation method. Through this model the data points can be properly categorized into SQLI or benign so new data points achieve accurate classification.

Bernoulli Classifier

The Bernoulli Classifier is especially appropriate for datasets with binary features, such as protocol flags or presence/absence of certain patterns in SQL queries. It effectively models binary-valued inputs and is well-suited for lightweight classification of attack traffic where categorical indicators are used.

Figure 10 demonstrates how Bernoulli and Gaussian Naive Bayes classifiers evaluate a new data point (yellow dot) for SQL injection (SQLI) prediction. The traffic data is shown through blue and red circular dots which represent Benign and Malicious categories respectively. The Bernoulli model determines the data point to be Benign with 0.99 probability.

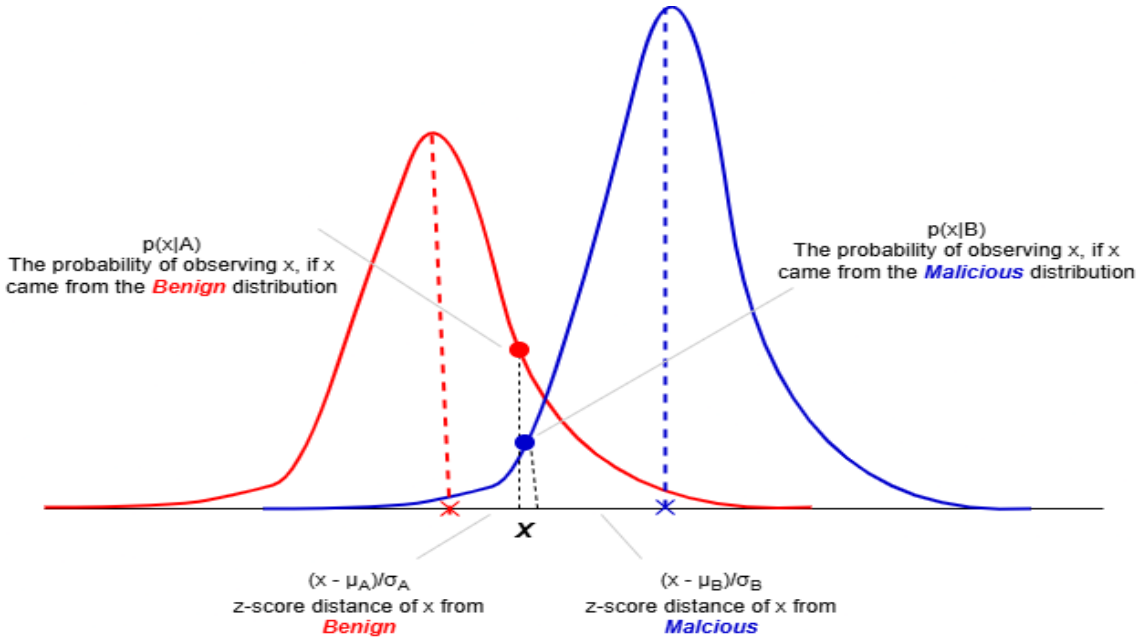

Fig. 8: Gaussian NB Classifier

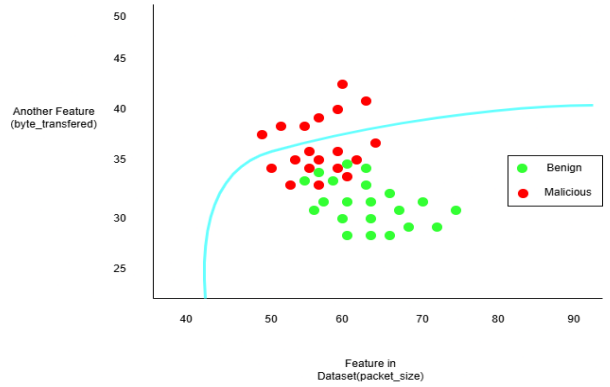

Fig. 9: Quadratic Determinant Analysis

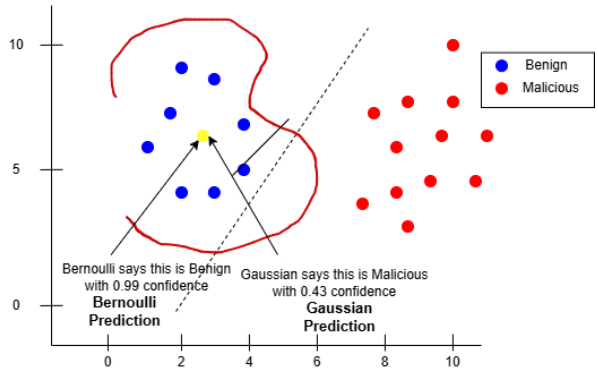

Fig. 10: Bernoulli Classifier

The Gaussian model assigns 0.43 confidence to malicious status for the data point while using continuous features (package size and byte transferred). The decision boundary and red contour display which classification criteria will identify threats as SQLI while keeping normal traffic paths clear.

Regularized Artificial Neural Network

The Regularized Artificial Neural Network (R-ANN) is effective in identifying SQL injection attacks because it has a deep learning attribute, building and standardization methods. Incorporating L2 weight decay, dropout, and batch normalization, R-ANN generalizes well even in complex feature spaces. Its ability to learn non-linear and hierarchical patterns from flow data ensures high classification accuracy, making it a powerful tool in modern network-based threat detection systems.

Figure 11 shows the Regularized Artificial Neural Network (R-ANN) model serves for detecting SQL Injection (SQLI) incidents through the diagram. The network begins by processing input layer signals that consist of packet_size, byte_transferred, and duration features sent from the network traffic. Multiple dense layers process the data before Adam optimizer applies its learning optimization alongside Binary Cross-Entropy loss for processing.

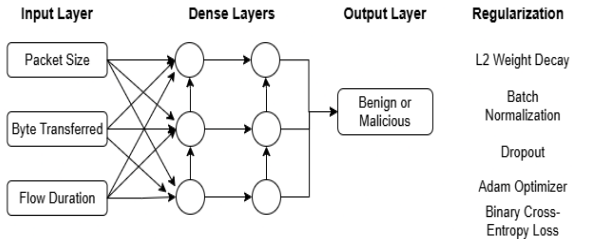

Fig. 11: Regularized ANN

Three techniques - L2 weight decay and Batch Normalization and Dropout - are used to stop overfitting and maintain the detection of SQLI patterns in unseen situations. Traffic samples reach a final output layer which decides between benign and malicious SQLI (SQL injection) content through identified patterns.

Experimental Setup

This experiment was conducted in a Linux-based OS (Ubuntu 20.04 LTS), which ran on a workstation with an Intel Core i7 processor, 16 GB of RAM, and 1 TB SSD storage. The used topology included client–server architecture, where the traffic was created and captured using SoftFlowd, nfcapd, and Wireshark software for benign and malicious traffic monitoring. For the traffic generation and analysis purpose, 57,229 NetFlow V5 data flows were recorded over 72 hours continuously. These data records contained both malicious and benign traffic of SQL injections. While the benign traffic involved normal browsing, authentication requests, and other web interactions, the malicious traffic involved SQL injection attacks including automated SQL injection payloads, malformed queries, authentication by-passing, and interaction with the vulnerable MySQL-based web application via the client-server connection. As it is known, there are currently few publicly available NetFlow datasets for analyzing SQL injection attacks. Therefore, the traffic generation procedure was employed to create a synthetic dataset reflecting real SQL injections. It should be emphasized that such synthetic traffic can hardly reflect the real-world conditions, and further research will be devoted to validation of the proposed approach on benchmark intrusion datasets. The collected packets were captured in PCAP format using Wireshark and then exported in NetFlow format using SoftFlowd and nfcapd. The obtained flow data were transformed into CSV format to be preprocessed and used for machine learning. The data preprocessing comprised the steps of dealing with missing values, cleaning features, normalizing them using Min-Max scaling, and reducing their dimensionality by applying feature selection algorithms. An 80:20 ratio was used to partition the dataset into training and testing datasets using stratified sampling. Cross-validation with five folds was applied when training and evaluating the models to increase statistical significance and avoid evaluation bias. The

reported performance results are the mean ± standard deviation across all folds. For the proposed R-ANN, we used TensorFlow/Keras libraries, while for other ML algorithms, we utilized Scikit-learn libraries. We used the Adam optimizer with a learning rate of 0.0005, a batch size of 64, binary cross-entropy loss function, and early stopping strategy with patience value 3 when training the ANN model. Random seed 42 was chosen to reproduce the experimental results.

Models Fitting Through Classification

The detection models received their parameterization through MoEv. MoEv operates as a general interface that interacts with Scikit-learn API functions. MoEv provides an automatic model building capability for classification tasks utilizing Scikit-learn library through YAML-based configuration file modification. Through its Grid Search CV class MoEv performs data preprocessing and normalization as well as dimensionality reduction and enables hyperparameter search. The system creates an output document with results that show accuracy, false alarm rates and detection statistics including Matthew's correlation coefficient, Cohen's kappa coefficient, detection rate, re_call and F1_score values. Researchers have successfully applied MoEv for detecting jamming attacks in real-time location systems as well as other research domains.

Performance Evaluation Metrics

The confusion matrix enables the calculation of familiar Key Performance Indicators (KPIs) which determine the most precise classification method. First, the models' performance was gauged their accuracy through this formula (5) while the calculation method defines accuracy as: The mathematical representation shows T_P as the successfully identified malicious traffic while T_N defines correct benign traffic identification. Finally, F_P notes incorrect benign classification as malicious and substance corresponds to wrong malicious classification as benign. T_N points to the number of benign flows correctly the number of correctly detected benign samples is designated as F_P incorrectly classified as malicious. Finally, F_N points out the models mistook malicious traffic as benign traffic:

$$\text{Accuracy} = \frac{T_P+T_N}{T_P+F_N+T_N+F_P} \quad (5)$$

T_P – Number of instances predicted as positive
T_N – Number of instances predicted as negative
F_N – Number of instances predicted as negative which is actually positive
F_P – Number of instances predicted as positive which is actually negative

The evaluation uses False Alarm Rate (FAR) together with Mathew's correlation coefficient as multiple KPIs. Binary systems find their preferred classification through selecting what prediction occurs most frequently. The assessment includes simultaneous DR and R calculations for flow data classes 0 and 1 with F1_score evaluation. The model provides F1_score evaluation for two output categories of benign and malicious flow data results. FAR stands for the method to calculate incorrect positive classifications relative to the full number of negative instances. Under assessment we observe that negative occurrences get identified as positive instances (false positives) exist during validation. positives and the total number of actual negative events (regardless of classification). The equation is given below:

$$\text{FAR} = \frac{F_P}{T_N+F_P} \quad (6)$$

Mathematical models use the correlation coefficient for assessing performance quality among binary classification systems. It can be calculated as shown in below:

$$\phi = \frac{T_P \times T_N - F_P \times F_N}{\sqrt{(T_P+F_P)(T_P+F_N)(T_N+F_P)(T_N+F_N)}} \quad (7)$$

The *DR* evaluation method determines the accuracy of positive results. It can be calculated as shown in below:

$$\text{DR} = \frac{T_P}{T_P+F_N} \quad (8)$$

The ratio between accurate detections of positive instances constitutes *R* also known as sensitivity and true positive rate. It can be calculated as shown in below:

$$\text{R} = \frac{T_P}{T_P+F_N} \quad (9)$$

DR and *R* can be combined to provide a single statistic called F1_score (F1), which is useful for anyone who needs a straightforward way to compare two classifiers. The F1 value is the harmonic mean of *DR* and *R*. The harmonic mean gives low numbers a lot more weight than the standard mean, which treats all values equally:

$$\text{F1} = 2\frac{DR \times R}{DR+R} \quad (10)$$

Results and Discussion

Table 2 gives a comparative summary of various machine learning classification models based on performance metrics like Accuracy, Recall, etc., the models to be compared here are Neighbors Classifier, Linear SVC, Logistic Regression, Random Forest Classifier, SGD Classifier, Voting Classifier, Gaussian NB Classifier, Quadratic Discriminant, and Bernoulli Classifier. The comparison of performance of each of the

models on three classes namely MWFF (Models without Feature Filtering), MFF (Models with Feature Filtering), and MFS (Models Feature Selection) is done. The Logistic Regression model performs consistently high on all the performance metrics with the values of Precision, Recall, F1-Score, and Accuracy being closer to or even at 0.97 for most. Neighbors Classifier is the next in performance and is superior even for MWFF with F1-Score and Accuracy of 0.93. However, models such as Quadratic Discriminant and Gaussian NB Classifier have lesser values in comparison with the rest especially based on Recall and F1-Score. The table provides a description of how the various classifiers will perform on various classification tasks and assists in the determination of the suitable model to use based on an application.

Figure 12 shows the confusion matrices compare the performance between nine different ML methods and one deep learning method in distinguishing between benign and malicious instances. Each matrix shows the true positives, false positives, false negatives, and true negatives, providing insight into the strengths and weakness of each model. False positive rate (15,292) of the K-Nearest Neighbors (KNN) Classifier is very high and this means that it tends to be false positives when benign cases are taken to be malicious cases. Simultaneously, it reaches the zero false negatives meaning that all the malicious cases were accurately detected. Although this is an extraordinarily sensitive indicator in identifying malicious traffic, the usability cost is considerable; since the false positive rate is high, this inhibits the model to be practically applied to detect SQL injections in real-world conditions. Linear SVC is better in this regard because it has fewer false positives (10,155) and fewer false negatives (151), which is a more reasonable trade-off between false positives and false negatives. The most accurate model in benign classification is the Logistic Regression, which makes only 1, 017 false positives and slightly less accurate in recall, 571 false negatives. Random Forest has a more balanced result, as it has 9,114 false positives, and the number of false negatives is only 156, which is why it is a good classifier in general.

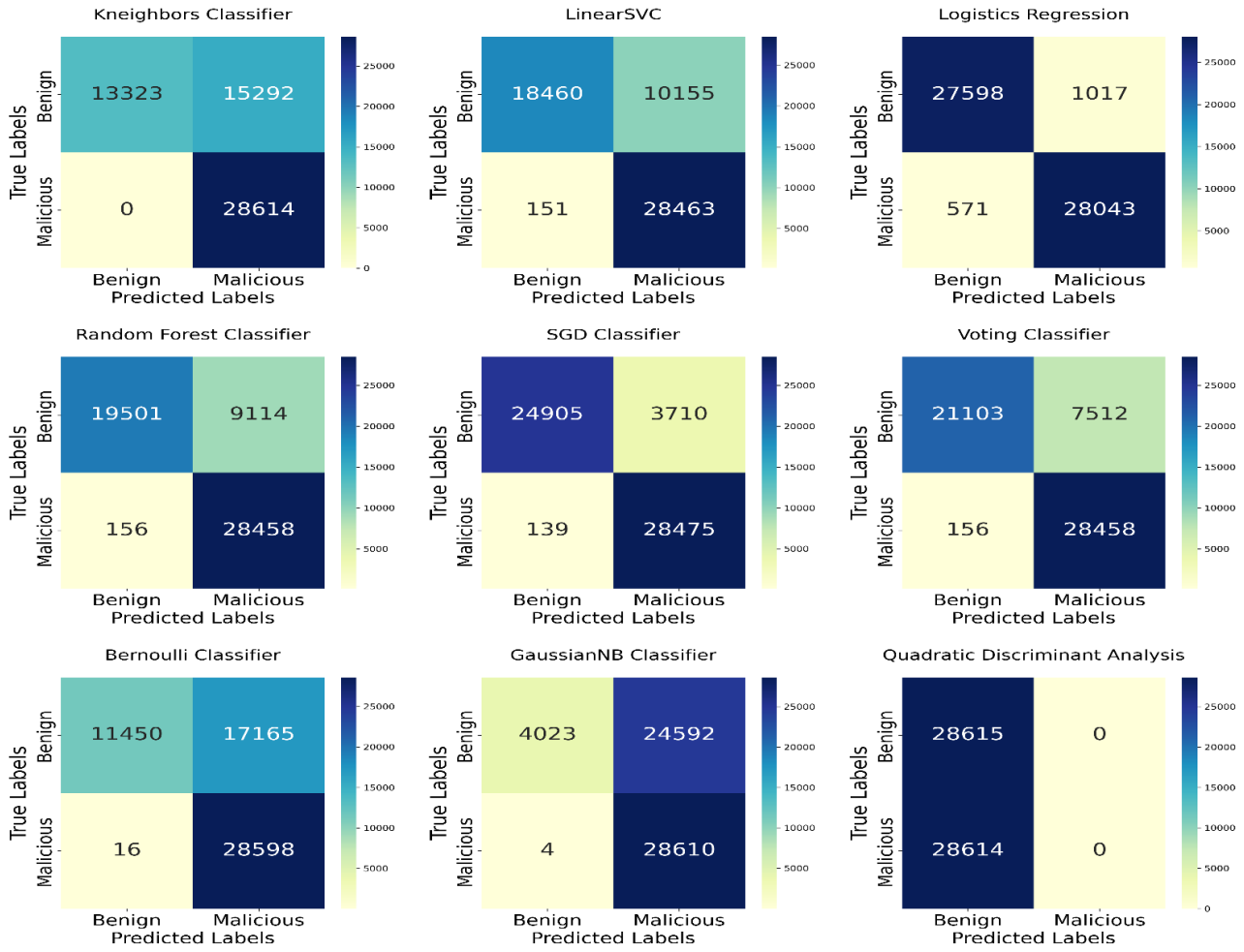

Fig. 12: Confusion Matrix Representation

The SGD Classifier is doing very well in recognizing the cases of maliciousness and its lowest false negative (140) and a comparatively low false positive (2,906) make it recall large. Lastly, the Voting Classifier that is the combination of several models, has a balanced score, and the false positives (9,796) and false negatives (142) decreased, making it a solid option. To conclude, Logistic Regression is the most precise, SGD is most effective in recall and Voting Classifier is a stable trade-off of the two thus good performer on average.

Inference Time Analysis

One of the primary objectives of the proposed framework is to reduce computational overhead compared with payload-based deep packet inspection approaches. To evaluate computational efficiency, inference latency was measured during classification experiments. The proposed flow-based detection framework achieved an average inference time of approximately 2.8 ms per flow for traditional machine learning classifiers and 4.6 ms per flow for the Regularized ANN model under the experimental hardware configuration. The lightweight nature of NetFlow-level feature extraction significantly reduced processing overhead compared with payload inspection techniques, making the proposed approach suitable for scalable and high-throughput network environments.

The deep learning model proposed will combine dropout and L2 regularization to facilitate a strong performance by detecting SQL injection attacks. Fully connected layers are added to allow the model to identify complicated patterns amongst the input data. To stabilize training process and hasten convergence, batch normalization is used and improves training efficiency. In order to overcome the threat of overfitting, a dropout rate of 40% is used, randomly removing neurons in the course of training. In addition, the L2 regularization is employed to discourage big weight values and again reduce the overfitting tendency. The adam optimizer with the learning rate of 0.0005 is chosen due to its adaptability and use in deep learning activities. The binary loss (cross-entropy) is used as the loss of the binary benchmark of benign and malicious traffic. The training set-up is a batch size of 64, early stopping with a patience of 3 and validation loss (val_loss) monitored to optimize the computational time and model generalization.To improve the reliability and statistical validity of the experimental results, 5-fold cross-validation was employed during model evaluation. The dataset was partitioned into five equal subsets, where four folds were used for training and one fold for testing in each iteration. The process was repeated five times such that each subset was used exactly once for testing. The final performance metrics were reported as the mean ± Standard Deviation (SD) across all folds. This approach reduces performance bias caused by a single train–test split and improves the robustness and reproducibility of the evaluation.

The performance values reported in Table 3 represent the mean ± standard deviation obtained using 5-fold cross-validation. The low standard deviation values observed for Logistic Regression, Voting Classifier, and Regularized ANN indicate stable and consistent performance across different folds. In particular, the proposed Regularized ANN achieved the highest overall classification accuracy and F1-score under the feature selection setting, demonstrating strong generalization capability and robustness for SQL injection detection using flow-level network traffic data.

Experimenting with various methods of detecting SQL injections, this paper reveals that linear and ensemble learning were more intricate in the area, agreeing with the concept that structured, highly-organized attack data are more effective with models that reduce nonlinearity and generalise properly. Flying at a tangent to the research, it is found that the statistical nature of SQL injection patterns does not change with the removal of unnecessary features, and that is in accordance to the concept that such types of attacks possess a specific signature which does not change. Nonetheless, the inaccuracy of probabilistic classifiers under this study is not new to other intrusion detection research studies and it is because such models are not suited to the unpredictability and messiness of malicious traffic. The method of feature filtering and selection made our results more robust, and complies with the thought of more current security analysts who opine the relevance of features within a model is what is most significant in intrusion detection, as opposed to the complexity of the model.

Table 3: Performance Evaluation Analysis

Models	Precision	Recall	F1-Score	Accuracy
	MWFF	MFF	MFS	MWFF
KNeighbors Classifier	0.82±0.03	0.81±0.04	0.94±0.02	0.73±0.05
Linear SVC	0.86±0.02	0.87±0.02	0.25±0.06	0.81±0.03
Logistic Regression	0.97±0.01	0.97±0.01	0.90±0.02	0.97±0.01
Random Forest Classifier	0.87±0.02	0.87±0.02	0.85±0.03	0.83±0.03
SGD Classifier	0.46±0.06	0.96±0.01	0.88±0.03	0.49±0.05
Voting Classifier	0.87±0.02	0.88±0.02	0.90±0.02	0.84±0.03
Gaussian NB Classifier	0.76±0.04	0.76±0.04	0.25±0.05	0.57±0.05
Quadratic Discriminant	0.25±0.05	0.25±0.05	0.25±0.05	0.50±0.04
Bernoulli Classifier	0.81±0.03	0.81±0.03	0.81±0.03	0.69±0.04
Regularized ANN	0.76±0.03	0.85±0.02	0.97±0.01	0.77±0.03

Table 4: Comparative Evaluation of Methodological Trends in SQL Injection Detection

Methodology / Study	Learning Paradigm	Feature Handling	Key Insight	Limitation
Distance-based classifier approach (Boyd et al., 2004)	Instance-based ML	Manual feature selection	High sensitivity to malicious patterns	Elevated false alarm rate
Probabilistic modeling framework (Al-Shammare et al., 2024)	Statistical ML	Distribution-based features	Effective under ideal data assumptions	Poor robustness on real-world traffic
Ensemble tree-based IDS (Zhang et al., 2024)	Ensemble ML	Feature importance ranking	Balanced generalization and stability	Computational overhead
Linear classifier with feature filtering (Chen et al., 2021)	Linear ML	Filter-based feature reduction	Strong interpretability and robustness	Limited nonlinear modeling
Regularized deep learning IDS (Chen et al., 2021)	Deep learning	Automated + selected features	Improved generalization and stability	Requires careful tuning
Proposed Work	Hybrid ML and deep learning	Feature filtering and feature selection	Robust and deployment-oriented SQL injection detection	Designed to reduce overfitting and false alarms

Familiar problems of optimisation-based and deep learning architectures were also noted in our study, and demonstrate that such architecture will be most applicable to high-stakes settings. However, past research has cautioned us to follow them keenly to ensure that they do not give false alarms. The results here contribute to the accumulating evidence that the well-balanced designs, intelligent feature engineering and consideration to the context under which the system is being operated increase intrusion detection performance. Instead of having all your eggs in one basket and having one type of classification. All of these trends are put into perspective in the Table 4 which compares our methods and the ones in the other studies.

The experiments were conducted on a workstation equipped with an Intel Core i7 processor, 16 GB RAM, running Windows 11 and Python 3.11 with Scikit-learn and TensorFlow libraries. The reported inference latency values represent the average prediction time per flow instance during the testing phase. Compared with conventional payload-based inspection techniques that require packet content parsing and deep feature extraction, the proposed flow-based approach operates on compact NetFlow features, thereby significantly reducing computational complexity and processing overhead. Although the inference-time results indicate promising computational efficiency, the current evaluation was conducted under controlled experimental conditions using synthetically generated traffic data. Future work will include large-scale benchmarking using real-world network traces and operational SDN environments to further validate scalability, throughput performance, and deployment readiness under production workloads.

Limitation of Proposed Work

Although the proposed framework achieved strong classification performance under controlled experimental conditions, several limitations remain. The dataset was generated within a controlled environment using synthetic traffic generation techniques, which may not fully capture the complexity and variability of real-world enterprise traffic. In addition, highly imbalanced traffic distributions commonly observed in operational networks were not extensively evaluated. The current work also does not comprehensively address adversarially generated SQL injection payloads, encrypted traffic analysis, or sophisticated obfuscation strategies. Synthetic traffic may not fully capture real-world network dynamics. Future validation using benchmark datasets and live SDN deployments is planned. Future research will therefore focus on benchmark dataset validation, real-time deployment analysis, adversarial robustness evaluation, and large-scale enterprise traffic experimentation.

Conclusion

As part of the evaluation process, it was revealed that the proposed Regularized Artificial Neural Network (R-ANN) proved to have a higher accuracy and strength compared to any other model, which includes Logistic Regression (LR), Random Forest, and conventional Neural Networks. The high performance of the R-ANN was owed to its deep structure and weight decay in L2 and L2 weight decay, dropout and batch normalization strategies which ensured that the R-ANN performed well in high-dimensional feature space. The R-ANN outperformed both linear regression and random forest in the assumptions of the feature-target linear relationships due to its capability to identify hierarchical and non-linear trends of network flow data. This approach made the accurate estimation of the probability, the reduced sensitivity to noise and outliers, and the reduction of the number of classification errors possible. The process of feature selection in the model allowed eliminating any unnecessary or redundant inputs into the system, which contributed to an increase in the

accuracy potential of the system. According to their outstanding features, the regularized ANN was found to be an ideal solution in the detection of SQL injections, which was superior to all the tested models in this study. Concerning the SQL injection detection, the R-ANN framework proposed is an excellent tool because of its better detection features. The proposed Regularized ANN demonstrated strong generalization capability within the evaluated dataset and experimental conditions. However, broader validation against previously unseen attack variants, adversarial payloads, and real-world enterprise traffic remains an important direction for future work

Abbreviations

Abbreviation	Description
ANN	Artificial Neural Network
R-ANN	Regularized Artificial Neural Network
ML	Machine Learning
DL	Deep Learning
IDS	Intrusion Detection System
SQL	Structured Query Language
SQLI	SQL Injection
LR	Logistic Regression
RF	Random Forest
KNN	K-Nearest Neighbors
SGD	Stochastic Gradient Descent
NB	Naïve Bayes
QDA	Quadratic Discriminant Analysis
MWFF	Models Without Feature Filtering
MFF	Models With Feature Filtering
MFS	Models With Feature Selection

Acknowledgment

The authors sincerely thank Mepco Schlenk Engineering College, Sivakasi, for providing the necessary facilities, support, and encouragement to carry out this research work.

Funding Information

The authors declare that no specific funding was received for conducting this study.

Authors Contributions

P. Vinoth: Software implementation, experimentation, data curation, write review and edited.

K. Muthamil Sudar: Conceptualization, methodology, investigation, formal analysis, write original draft, supervision.

S. Muthukumar: Experimentation, visualization, resource support, write review and edited.

Ethics

This study does not involve human participants, personal data, or animal subjects, and therefore does not require ethical approval.

Conflict of Interest

The authors affirm that there are no conflicts of interest regarding the publication of this manuscript.

Declaration of Artificial Intelligence (AI) Assistance

The authors declare that no AI tool was used for idea generation, data analysis, experimental design, or interpretation of results. The authors take full responsibility for all content, findings, and conclusions presented in the manuscript.

Data Availability

The datasets used during the current study are available from the corresponding author on reasonable request.

References

Balogun, S. A., Ijiga, O. M., Okika, N., Enyejo, L. A., & Agbo, O. J. (2025). Machine Learning-Based Detection of SQL Injection and Data Exfiltration Through Behavioral Profiling of Relational Query Patterns. *International Journal of Innovative Science and Research Technology*, *10*(8), 49–63. https://doi.org/10.38124/ijisrt/25aug324

Abubakar, A. I., Bisallah, H. I., & Kabir, KhadijatM. (2025). Enhancing Web Application Security Through Automated SQL Injection Detection Using Neural Networks. *UNIABUJA Journal of Engineering and Technology*, *2*(2), 368–376. https://doi.org/10.70118/ujet.2025.0202.34

Afrin, S., Elsayed, M. A., & Zincir-Heywood, N. (2025). SQL-GENIE: SQL Protection using GENerative Modeling for Anomaly Detection against Injection and Evolved Adversarial Attacks. *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, 459–464. https://doi.org/10.1109/compsac65507.2025.00066

Alghawazi, M., Alghazzawi, D., & Alarifi, S. (2023). Deep Learning Architecture for Detecting SQL Injection Attacks Based on RNN Autoencoder Model. *Mathematics*, *11*(15), 3286. https://doi.org/10.3390/math11153286

Alnemari, A. S., & Alajmani, S. H. (2025). Collaborative SQL and JSON injection detection system using machine learning. *Journal of Theoretical and Applied Information Technology*, *103*, 11.

Al-Shammare, H., Al-Otaiby, N., Al-Otabi, M., & Alshayeb, M. (2024). An Empirical Investigation of the Security Weaknesses in Open-Source Projects. *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, 634–642. https://doi.org/10.1145/3661167.3661279

Anwar, I. (2025). Machine Learning Approaches for Detection of SQL Injection Attacks. *Jurnal Sistem Informasi Dan Teknik Informatika (JAFOTIK)*, *3*(1), 1–6. https://doi.org/10.70356/jafotik.v3i1.50

Arasteh, B., Karimi, M., Kusetogullari, H., Arasteh, K., & Kiani, F. (2025). A cybersecurity method to detect SQL injection attacks using heuristic-driven feature selection and machine learning algorithms. *The Journal of Supercomputing*, *82*(1), 31. https://doi.org/10.1007/s11227-025-08165-y

Aulianoor, A. B., & Koprawi, M. (2024). Comparative Analysis of the Performance of Decision Tree and Random Forest Algorithms in SQL Injection Attack Detection. *Journal of Applied Informatics and Computing*, *8*(1), 194–202. https://doi.org/10.30871/jaic.v8i1.8136

Bakır, R. (2025). UniEmbed: A Novel Approach to Detect XSS and SQL Injection Attacks Leveraging Multiple Feature Fusion with Machine Learning Techniques. *Arabian Journal for Science and Engineering*, *50*(19), 15591–15604. https://doi.org/10.1007/s13369-024-09916-4

Boyd, S. W., & Keromytis, A. D. (2004). SQLrand: Preventing SQL Injection Attacks. *Applied Cryptography and Network Security*, *3089*, 292–302. https://doi.org/10.1007/978-3-540-24852-1_21

Casmiry, E. N., Sinde, R. S., & Mduma, N. M. (2026). A Hybrid Deep Learning Model for SQL Injection Attack Detection. *IEEE Access*, *14*, 6450–6463. https://doi.org/10.1109/access.2026.3651991

Casmiry, E., Mduma, N., & Sinde, R. (2025). Enhanced SQL injection detection using chi-square feature selection and machine learning classifiers. *Frontiers in Big Data*, *8*, 1686479. https://doi.org/10.3389/fdata.2025.1686479

Chen, D., Yan, Q., Wu, C., & Zhao, J. (2021). SQL Injection Attack Detection and Prevention Techniques Using Deep Learning. *Journal of Physics: Conference Series*, *1757*(1), 012055. https://doi.org/10.1088/1742-6596/1757/1/012055

Chen, Y., Liang, G., & Wang, Q. (2025). Research on SQL Injection Detection Technology Based on Content Matching and Deep Learning. *Computers, Materials & Continua*, *84*(1), 1145–1167. https://doi.org/10.32604/cmc.2025.063319

Elshewey, A. M. (2026). An Enhanced Approach for Edge-Based Intrusion Detection Based on a Hybrid Deep Learning Model CNN-DNN. *SN Computer Science*, *7*(3), 271. https://doi.org/10.1007/s42979-026-04805-z

Floris, G., Scano, C., Montaruli, B., Demetrio, L., Valenza, A., Compagna, L., Ariu, D., Piras, L., Balzarotti, D., & Biggio, B. (2025). ModSec-AdvLearn: Countering Adversarial SQL Injections With Robust Machine Learning. *IEEE Transactions on Information Forensics and Security*, *20*, 6693–6705. https://doi.org/10.1109/tifs.2025.3583234

Fu, H., Guo, C., Jiang, C., Ping, Y., & Lv, X. (2023). SDSIOT: An SQL Injection Attack Detection and Stage Identification Method Based on Outbound Traffic. *Electronics*, *12*(11), 2472. https://doi.org/10.3390/electronics12112472

Gowtham, M., & Pramod, H. B. (2022). Semantic Query-Featured Ensemble Learning Model for SQL-Injection Attack Detection in IoT-Ecosystems. *IEEE Transactions on Reliability*, *71*(2), 1057–1074. https://doi.org/10.1109/tr.2021.3124331

Kakisim, A. G. (2024). A deep learning approach based on multi-view consensus for SQL injection detection. *International Journal of Information Security*, *23*(2), 1541–1556. https://doi.org/10.1007/s10207-023-00791-y

Lo, R.-T., Hwang, W.-J., & Tai, T.-M. (2025). SQL Injection Detection Based on Lightweight Multi-Head Self-Attention. *Applied Sciences*, *15*(2), 571. https://doi.org/10.3390/app15020571

Mannuru, V., Al-Sinayyid, A., Kadiyala, S., & Battula, R. R. (2026). Real-Time Detection of Time-Based Blind SQL Injection Using Machine Learning for Critical Infrastructure. *Leadership, Business and Management in STEM*, 38–57. https://doi.org/10.1007/978-3-032-10539-4_4

Mariettou, S., Koutsojannis, C., & Triantafyllou, V. (2025). A Secure Prescription System with Machine Learning for SQL Injection Detection. *Computer Networks and Communications*, *3*(2), 59–72. https://doi.org/10.37256/cnc.3220257145

Mathew, R. R., & Vidhate, A. (2025). Deep learning prediction model for DoS and SQL injection attack in SDN. *International Journal of Information and Computer Security*, *28*(3), 327–347. https://doi.org/10.1504/ijics.2025.149446

Muthamil S., K., & Deepalakshmi, P. (2021). An intelligent flow-based and signature-based IDS for SDNs using ensemble feature selection and a multi-layer machine learning-based classifier. *Journal of Intelligent & Fuzzy Systems*, *40*(3), 4237–4256. https://doi.org/10.3233/jifs-200850

Naga Charan, N. (2022). Predictive Sql Injection Detection and Prevention Using Machine Learning Across Aws, Azure, and Google Cloud Platforms. *International Journal of Engineering Science and Advanced Technology*, *22*(8), 68–74. https://doi.org/10.64771/ijesat.2022.v22.i08.pp68-74

Okello, F. O., Kipkebut, A., & Oginga, R. (2026). A Hybrid Deep Learning Framework with Honeypot-Assisted Intelligence for SQL Injection Detection. *Journal of Computer Science and Technology (JCST)*, *4*(1), 1–12. https://doi.org/10.51317/jcst.v4i1.919

Sahbi, A., Jaidi, F., & Bouhoula, A. (2026). SDN-Net: Data fusion and artificial intelligence for enhancing intrusion detection within SDN/NFV network. *Journal of Communications and Networks*, *28*(2), 233–249. https://doi.org/10.23919/jcn.2025.000098

Satar, Y. A., Mohamed, A., & Hassanain, A. A. (2026). NSGA-II Optimization of Probabilistic Neural Networks for Robust SQL Injection Attack Detection. *Engineering Systems and Intelligent Technologies (ESIT*, *1*(1), 29–41.

Siale, A. Y. D., Hassan, Q. M. Z., Kadekle, M. F. A. S., & Veena, B. S. (2025). Enhancing Large-Scale Network Security with a VGG-Net-Based DCNN: A Deep Learning Approach to Anomaly Detection. *Journal of Robotics and Control (JRC)*, *6*(3), 1316–1331. https://doi.org/10.18196/jrc.v6i3.25169

Smrity, T. A., & Muntaqim, M. Z. (2025). SQLGuardNet: Deep learning adaptive processing-based pattern recognition for enhanced SQL injection detection. *Signal, Image and Video Processing*, *19*(13), 1152. https://doi.org/10.1007/s11760-025-04642-2

Sudar, K. M., Rohan, M., & Vignesh, K. (2024). Detection of adversarial phishing attack using machine learning techniques. *Sādhanā*, *49*(3), 232. https://doi.org/10.1007/s12046-024-02582-0

Wang, S., Liu, X., Liu, H., Zhao, C., & Ding, J. (2026). SQL Injection Detection Method Based on Multi-Dimensional Feature Modeling and Deep Learning. *KSII Transactions on Internet and Information Systems*, *20*(1), 482–502. https://doi.org/10.3837/tiis.2026.01.021

Zhang, B., Ren, R., Liu, J., Jiang, M., Ren, J., & Li, J. (2024). SQLPsdem: A Proxy-Based Mechanism Towards Detecting, Locating and Preventing Second-Order SQL Injections. *IEEE Transactions on Software Engineering*, *50*(7), 1807–1826. https://doi.org/10.1109/tse.2024.3400404

Zimmer, F. (2024). RelChaNet: Neural Network Feature Selection using Relative Change Scores. *ArXiv*. https://doi.org/10.48550/arXiv.2410.02344