



## Semarak International Journal of Machine Learning

Journal homepage:  
<https://semarakilmu.my/index.php/sijml/index>  
 ISSN: 3030-5241



# SQL Injection Attack Detection using Machine Learning Algorithms

Laila Aburashed<sup>1,\*</sup>, Marah AL Amoush<sup>1</sup>, Wardeh Alrefai<sup>1</sup>

<sup>1</sup> Department of Cyber Security, Faculty of information Technology, Zarqa Technical Intermediate Collage, Zarqa, Jordan

### ARTICLE INFO

#### Article history:

Received 4 April 2024  
 Received in revised form 5 May 2024  
 Accepted 15 June 2024  
 Available online 30 June 2024

#### Keywords:

SQL- injection; machine learning;  
 Random Forest; ANN; datasets; Gradient  
 Boosting; SVM

### ABSTRACT

SQL Injection is one of the most common vulnerabilities exploited for both privacy breaches and financial damage. It remains the top vulnerability on the most recent OWASP Top 10 list, with the number of such attacks on the rise. The SQL Injection Detection Challenge is addressed using machine learning algorithms. By employing a classification method, communications are identified as either SQL Injection or plain text. This research proposes a machine learning framework to assess the feasibility of using a machine learning classifier to detect SQL Injection attacks. Classification algorithms such as Random Forest, Gradient Boosting, SVM, and ANN are utilized. As a result, ANN demonstrated superior performance and required less time to detect SQL Injection attacks.

## 1. Introduction

SQL injection is a longstanding vulnerability that has been featured in the OWASP Top 10 for nearly fifteen years. It allows for the theft and modification of information, whether sensitive or not, stored in millions of databases worldwide. An Imperva Web Application Attack Report (WAAR) issued in February 2020 revealed that 29% of web applications remain vulnerable to SQL injection attacks [1]. Figure 1 displays these statistics.

The majority of the applications we use are web-based. As internet use has increased, so has the size and severity of web attacks. We regularly perform various transactions online as the internet's popularity grows. SQL injection is a type of cyber-attack that uses SQL to exploit and manipulate databases, potentially exposing client information. These attacks are becoming a significant concern for internet administrators. For example, Hetzner, a South African web hosting company, experienced a breach in which 40,000 customer records were compromised [2]. An SQL injection vulnerability could have led to the theft of every client record in their database.

In 2019, a new type of SQL injection called "Voice-Command SQL Injection" emerged, in which the attack is executed using vocal commands [2]. Recently, the application of machine learning algorithms to detect and prevent various cybersecurity threats has gained popularity. However, while the effectiveness of supervised and online learning approaches in detecting security risks is undeniable,

\* Corresponding author.

E-mail address: [lailaburashed90@hotmail.com](mailto:lailaburashed90@hotmail.com)

<https://doi.org/10.37934/sijml.2.1.112a>

the computational demands and time required to run these complex algorithms remain a significant concern for continuously evolving cybersecurity practices.

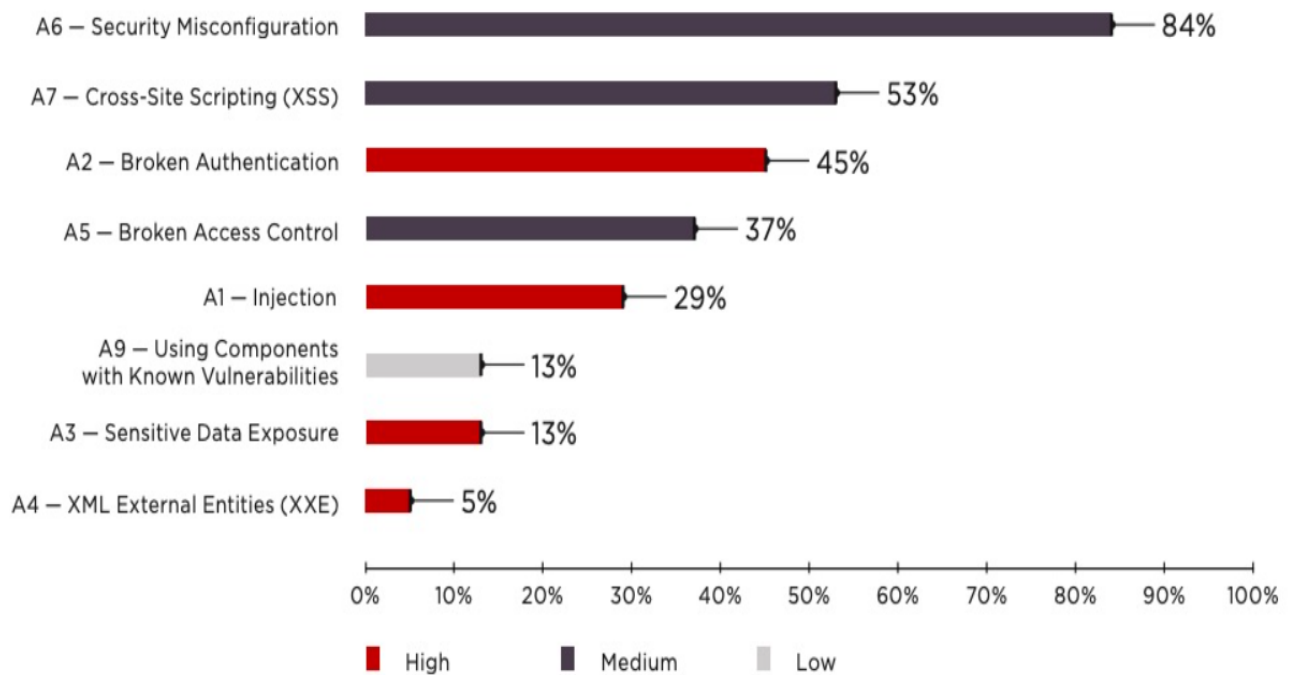


Fig. 1. Web application vulnerability and threats

The following are the primary contributions of the study:

- i. To develop a reliable framework for detecting SQL injection attacks using a robust dataset and a Random Forest ML classifier, alongside Gradient Boosting Classifier, SVM, and ANN.
- ii. To evaluate the frameworks and select the one that best suits the intended purpose of the algorithm.
- iii. To compare the research results with those of other studies.

The end result of this research is an accurate system for detecting SQL injection attacks in web applications. The proposed framework will be implemented in a real-world system in the future. Section II of this paper contains the literature review, while Section III provides background information. The proposed methodology is explained in Section IV, Section V includes the evaluation and results, and Section VI presents the conclusion.

## 2. Literature Review

Several researchers have focused on developing robust frameworks for detecting SQL injection. This section reviews previous studies that have presented various methods for detecting or preventing SQL injection attacks. Sheykhanloo *et al.*, [3] proposed deep learning approaches to detect SQL injection, using a dataset of 25,000 URL addresses, with 12,250 being benign and 12,250 being malicious. The study classified these URLs using neural network models, which were deemed effective for protecting systems against botnet attacks. The accuracy of their results was 95%.

Betarte *et al.*, [4] suggested machine learning framework of web application firewall to detect and prevent web application firewall assaults, the K-NN (K=3), SVM, and Random Forest algorithms were proposed. Using the CSIC-2010, DRUPAL, and PKDD2007 datasets, propose two solutions: first,

a multi-class method for situations in which both valid and attack data are given, and second, a one-class solution for situations in which only valid data is provided, the experiment results achieving a high accuracy of 97%.

George *et al.*, [5] proposed a Token-based Detection framework, a novel solution for detecting and blocking code injections with minimal computational cost. Their experiment achieved a high accuracy of 99.23% using 1,655 queries—451 malicious and 1,204 benign—along with a neural network model.

Kevin *et al.*, [6] discussed a machine learning classifier designed to detect SQL injection flaws in PHP code. To train and evaluate the classifier models, both classical and deep learning techniques were used, along with data validation and feature extraction from source code files. With ten-fold cross-validation, a model trained with a Convolutional Neural Network (CNN) achieved the highest accuracy of 95.4%.

Anandha *et al.*, [7] proposed addressing the SQL Injection Detection Challenge using machine learning algorithms. To classify incoming communication as either SQL Injection or normal text, a classification approach was employed. Five machine learning methods were used: Naive Bayes Classifier, Passive Aggressive Classifier, SVM, and CNN. The Naive Bayes Classifier achieved a 95% accuracy rate, Logistic Regression achieved a 92% accuracy rate, and SVM achieved a 79% accuracy rate. Supervised learning methods are considered to produce more precise results due to their use of numerous fundamental classifiers to improve accuracy. Consequently, CNN was selected as the best approach for addressing the SQL Injection categorization problem. Table shows the summary of literature review.

**Table 1**  
 Summary of literature review

Literature	Year	Description	Classification Algorithms	Accuracy rate
Betarte <i>et al.</i> , [4]	2017	Suggested machine learning framework of web application to detect and prevent SQL web application.	1- KNN 2- SVM 3- RF	97% 98% 95%
Sheykhkanloo <i>et al.</i> , [3]	2018	using Neural Network Based models was regarded as a suitable method for protecting systems against botnet attacks	1- ANN	95%
George <i>et al.</i> , [5]	2018	proposed the Token-based Detection framework is a novel solution for detecting and blocking code injections	1-ANN	99.23%
Kevin <i>et al.</i> , [6]	2019	Discusses a machine learning classifier that can detect SQL injection vulnerabilities in PHP code.	1-CNN	95%
Anandha <i>et al.</i> , [7]	2021	Proposed the Machine learning algorithms are used to solve the SQL Injection Detection Challenge.	1- Passive Aggressive 2- SVM 3- Logistic Regression 4- Nave Bayes	79% 79% 92% 95%

### 3. Background

#### 3.1 SQL Injection

SQL injection, often known as SQLI, is a common attack vector that involves injecting malicious SQL code into a backend database to access data that was not intended to be exposed. An attacker can exploit a SQL injection vulnerability to bypass a web application's authentication and authorization controls, potentially retrieving the contents of an entire database if the conditions are met [7]. Figure 2 illustrates this concept.



Fig. 2. SQL injection attack

SQL injection attacks are often categorized based on the attacker's motivations, such as data entry, data extraction, denial of service, and database schema manipulation. Examples of specific methods include Blind SQL Injection, Error-based SQL Injection, and Union-based SQL Injection queries.

### 3.1.1 Blind SQL injection

When a developer fails to fully safeguard a website against traditional SQL Injection, blind injections are often possible. Blind SQL Injection is a type of SQLI attack that uses true or false questions to infer information from the database based on the application's response. This attack is commonly used in practice to exploit generic error messages but does not protect code vulnerable to SQL injection. [8] Blind SQL Injection occurs when an application is vulnerable to SQL injection but does not disclose the results of the SQL query or any database errors in its HTTP responses.

Unfortunately, when blind SQL injection vulnerabilities exist, many techniques, such as UNION attacks, become ineffective because they rely on the ability to inspect the results of the injected query in the application's responses. Although blind SQL injection can still be used to gain unauthorized access to data, different tactics are required. Because blind SQL injections do not allow the use of common SQL injection methods like UNION, sub-query injection, or XPATH, they are more complex and time-consuming to exploit.

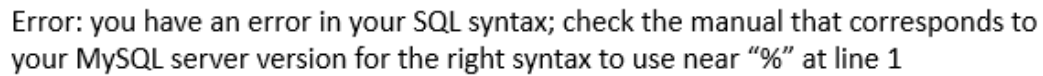
Nevertheless, the security risks are comparable. If a malicious query is successfully executed, the attacker gains control over the database server. This can lead to the theft of sensitive data, such as credit card numbers, and privilege escalation, potentially allowing a full takeover of the web server's operating system.

### 3.1.2 Error based SQL injection

In a SQL Injection attack focused on errors, inaccurate data is injected into a query, causing the database to generate errors. This is typically done by forcing a potentially error-prone database action. The attacker can then observe these database errors and use them to learn how to navigate the database using SQL queries [9]. In essence, the attacker uses the thrown error to refine the next payload. The first step in performing this test is to determine whether the application interacts with a database to access data. Typical scenarios where an application might access a database include:

- i. Authentication Forms: If a web form is used for authentication, there is a risk that the credentials are verified against a database containing every user's credential (or, more securely, password hashes).
- ii. Search Engines: An SQL query may be used to retrieve relevant entries from the database based on a user's search term.

In this scenario, the goal is to disrupt the query and generate an error by testing all input fields that could be used to form a SQL query, including hidden fields in POST requests. It is also important to consider HTTP headers and cookies. Add a single quotation mark (as a string terminator) or a semicolon (used to terminate an SQL query) to the field or parameter under test. If the input is not properly filtered, this is likely to result in an error. The output from a vulnerable field is shown in Figure 3.



```
Error: you have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "%" at line 1
```

**Fig. 3.** The output of a vulnerable field

Penetration testers can gain valuable insights from detailed error messages, such as those shown in the examples. These messages can help them plan an effective attack.

### 3.1.3 Union based SQL injection attack

In a UNION query attack, the attacker uses the UNION operator to combine a malicious query with the original query. This allows the attacker to link the results of the original query with those from the malicious query, potentially accessing values from columns in other tables [9]. When an application is vulnerable to SQL injection and the results of a query are returned in the program's responses, the UNION keyword can be exploited, resulting in a SQL injection UNION attack. There are two critical prerequisites for a UNION query to be successful:

- The number of columns returned by each query must be the same.
- Each column's data type must be compatible with the corresponding columns in the queries.

### 3.2 Prevent SQL Injection

**Utilize Secure Coding Techniques:** Regardless of the programming language, ensure that secure coding principles are followed. Common web development platforms (PHP, ASP.NET, Java, Python, and Ruby) offer solutions for preventing SQL injections, including Blind SQL Injections. Avoid dynamic SQL at all costs. The optimal approach is to use prepared queries, also known as parameterized statements. Additionally, stored procedures, supported by most SQL databases (PostgreSQL, Oracle, MySQL, and MS SQL Server), can be used. All user data inputs should be escaped or filtered for special characters (such as single quotation marks used in traditional SQL injections).

**Utilize Automated Testing Services:** Regular automated scans will help uncover any new vulnerabilities that may have been introduced since the last scan or that may have appeared with new releases. This process often involves machine learning (ML) to automatically identify data inputs and determine if they contain SQL queries.

The next section will detail the strategy used to develop a model capable of accurately classifying input types. Before that, we will provide an overview of machine learning and some of the algorithms used to compare the findings.

### 3.3 Machine Learning

Machine learning is a type of computer algorithm that improves its performance over time through automated learning. It refers to a system's ability to learn from problem-specific training

data to automate the process of building analytical models and performing related tasks using various machine learning techniques.

### 3.3.1 Machine learning algorithms

#### 3.3.1.1 Random Forest

Random Forest is a supervised machine learning algorithm used for both classification and regression problems. It employs majority voting for classification and averaging for regression by building decision trees from multiple samples. One of the key features of the Random Forest algorithm is its ability to handle datasets with both continuous and categorical variables, making it effective for both regression and classification tasks. It generally provides more accurate results for classification problems. The Random Forest method uses a large number of decision trees to solve a problem. The algorithm creates a 'forest' that can be trained using either bagging or bootstrap aggregation techniques.

#### 3.3.1.2 SVM

As shown in Figure 4 below, Support Vector Machines (SVMs) are a set of supervised learning techniques used for classification, prediction, and outlier detection. They can address both linear and non-linear problems and are applicable to a wide range of tasks. SVMs create a line or hyperplane that separates the data into different categories. In other words, SVM is an algorithm that takes data and generates a line or hyperplane to divide the classes, if possible.

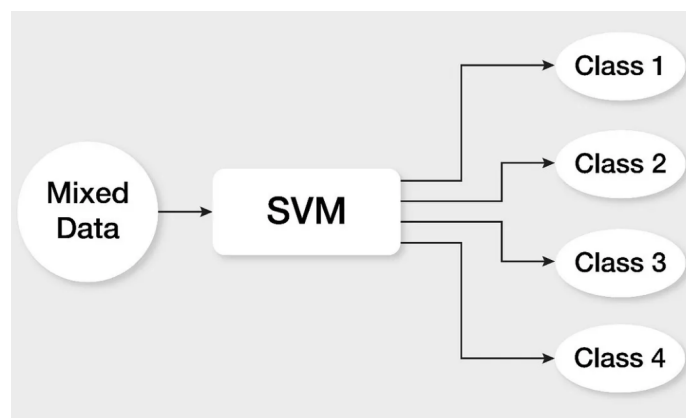


Fig. 4. SVM Principle

#### 3.3.1.3 RNN

Recurrent Neural Networks (RNNs) are a powerful and robust type of neural network and are among the most promising algorithms in use due to their internal memory capability. An RNN operates on the principle of retaining a layer's output and feeding it back into the input to predict future outcomes. By processing both current and previously received inputs, an RNN can handle sequential data effectively. The internal memory of RNNs allows them to remember previous inputs.

#### 3.3.1.4 GBC

The primary idea behind this approach is to build models sequentially, with each model aiming to correct the errors of the previous model. But how do we achieve this? How can we minimize the possibility of error? This is accomplished by basing a new model on the mistakes or residuals of the prior model.

## 4. Methodology

As an initial step towards creating measurable work processes, there are basic steps to begin knowledge exploration: (1) collecting data, (2) pre-processing, and (3) training and testing performance.

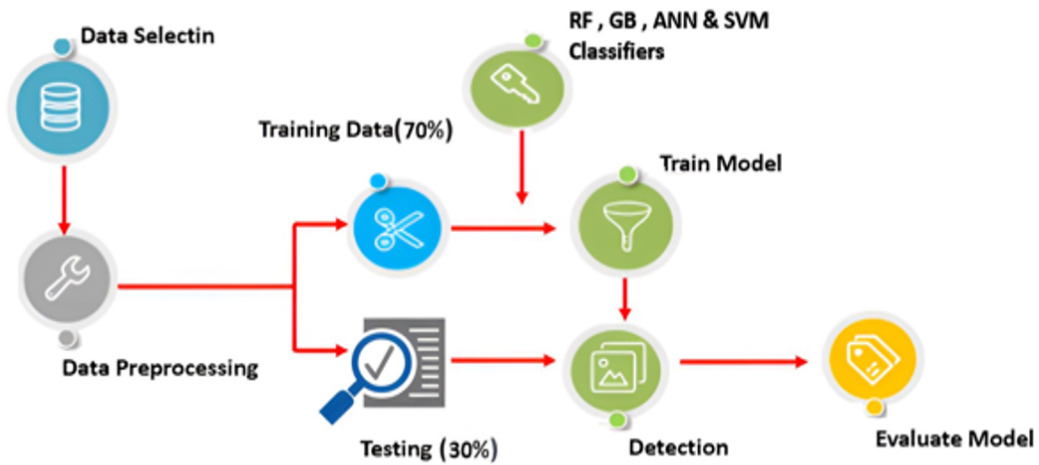


Fig. 5. The methodology

The process begins by constructing a decision stump and applying equal weights to all data points. Weights are then increased for misclassified points and decreased for correctly classified or easily classified points. A new decision stump is created based on these weighted data points, with the goal of improving upon the predictions made by the first stump. Several gradient-boosting libraries are available, including XGBoost, H2O, and LightGBM. The main differences among them lie in the tree structure, feature engineering, and handling of sparse data.

### 4.1 Collecting Data

Data collection and preparation are the first steps in designing the model. For this study, the SQL\_Injection Dataset 2021 was collected. What distinguishes this dataset is that it is both real and comprehensive, consisting of SQL Injection queries and a total of 30,920 records. The testing environment prepared for this study was as follows:

- i. Device Specifications:
  - o Processor: Intel Core i5
  - o RAM: 16 GB
  - o Windows Edition: Windows 10
- ii. Software:
  - o Anaconda Python 3.7 platform with Spyder, Scientific Python Development Environment, version 3.3.6.

### 4.2 Data Pre-processing

Selecting the optimal features to improve classifier performance is insufficient if the dataset is not well prepared. At this stage, the dataset was pre-processed as follows:

- i. Removing Redundant and Missing Values: Start by identifying and removing redundant rows to eliminate duplicate records. Next, replace missing values with the median of the corresponding values (to reduce computational overhead and training time) or zero, or

eliminate the entire row if necessary. This procedure addressed both redundant and missing values.

- ii. Normalization: To prevent the learning algorithm from being misled by variables with varying magnitudes, it is recommended to rescale features so that they have a mean between 0 and 1. Combining variables with large and small magnitudes can confuse the algorithm and lead it to undervalue the smaller magnitude variables.
- iii. Dataset Splitting: The dataset was split into 70% for the training set and 30% for the testing set.

### 4.3 Training and Testing

After pre-processing, the process involves two phases:

- i. The Training Phase: In this phase, the model learns to detect benign and malicious patterns by utilizing the sequence of statistical data in the 70% training set.
- ii. The Testing Phase: The performance of the Random Forest, SVM, RNN, and Gradient Boosting classifiers is evaluated using the remaining 30% of the dataset. In this phase, the expected and actual results are compared, as illustrated in the next section.

## 5. The Evaluation and Results

This section explains the majority of the experiments conducted and evaluates the results. It details our approach of applying four classifiers using Python to assess and compare the performance of different classifiers to determine which yields the best results. The system's objective was to detect malicious SQL Injection. The accuracy rates and times for each part of the experiment are depicted in the tables below.

By comparing the expected and actual values during the testing phase, four metrics are generated: False Positive (FP), True Negative (TN), True Positive (TP), and False Negative (FN). FP represents the number of times the classifier incorrectly identifies benign records as malicious, TN correctly identifies harmless records, TP correctly identifies malicious records, and FN represents benign records incorrectly identified as malicious. [10].

These metrics are used to compute evaluation measures such as Accuracy, Precision, F1 Score, and Recall. These measures, presented in Table 2, are used to assess the framework's performance.

**Table 2**

Confusion matrix

The Evaluation Matrix	The equation
Accuracy	$TP + TN$
Measures the classifier efficiency	$TP + TN + FP + FN$
Precision	$TP$
Measures the classifier efficiency in detecting the DDoS Attack records	$TP + FP$
F1	$2 (Precision * Recall)$
A criterion to balance FP and FN	$(Precision + Recall)$
Recall	$TP$
Measures the Classifier efficiency in detecting the benign records	$TP + FN$



**Table 3**  
 Evaluation result using ANN Classifier

ANN Classifier			
Execution time	181.3065710067749		
Accuracy	0.96 %		
Matrices	Precision	recall	f1-score
Class 0 (Benign)	0.95	0.99	0.97
Class 1 (SQL Injection attack)	0.98	0.91	0.94
Average	0.96	0.96	0.96

The results of the ML classifier were calculated after the testing phase using 30% of the dataset. These results are shown in Table 2, 3, 4, and 5, respectively.

**Table 4**  
 Evaluation result using Random Forest Classifier

Random Forest Classifier			
Execution time	507.38437604904175 sec.		
Accuracy	0.93 %		
Matrices	Precision	recall	f1-score
Class 0 (Benign)	0.93	0.95	0.94
Class 1 (SQL Injection attack)	0.91	0.89	0.90
Average	0.93	0.93	0.93

**Table 5**  
 Evaluation result using Gradient Boosting Classifier

Gradient Boosting Classifier			
Execution time	1530.8685019016266 sec.		
Accuracy	0.91 %		
Matrices	Precision	recall	f1-score
Class 0 (Benign)	0.89	0.98	0.93
Class 1 (SQL Injection attack)	0.96	0.79	0.86
Average	0.91	0.91	0.91

**Table 6**  
 Evaluation result using SVM Classifier

SVM Classifier			
Execution time	181.3065710067749 sec.		
Accuracy	0.94 %		
Matrices	Precision	recall	f1-score
Class 0 (Benign)	0.93	0.98	0.95
Class 1 (SQL Injection attack)	0.97	0.87	0.91
Average	0.94	0.94	0.94

The evaluation results show that the approach achieved an average accuracy rate of 96%, along with precision, recall, and F1 scores of 96%. This indicates that the framework can effectively distinguish between benign and SQL Injection attack classes without bias. In contrast, the ANN Classifier required less time during the testing phase and achieved a higher accuracy rate compared to other classifiers, as shown in Figure 6.

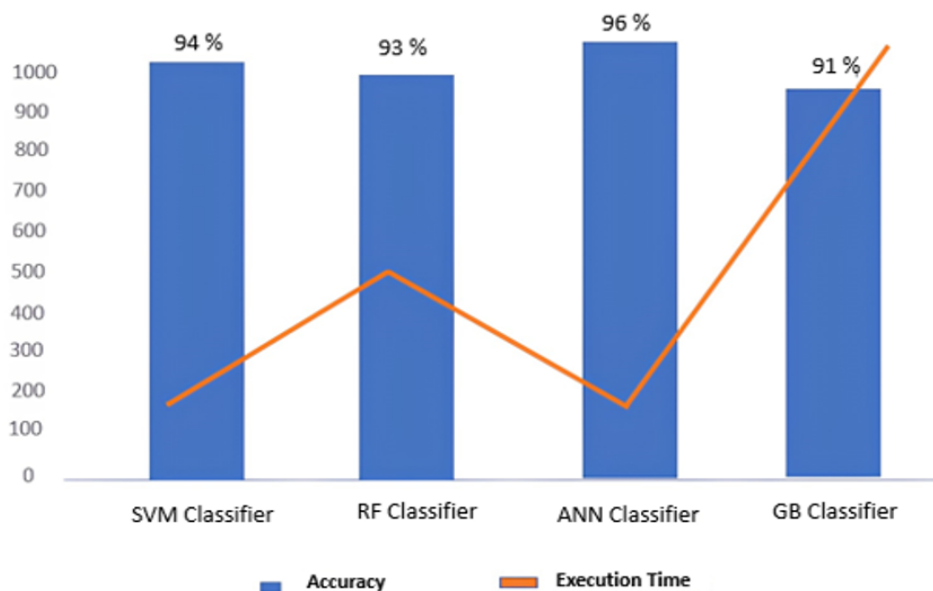
Table 7 compares our proposed method with those of other researchers in the same field who used the same classifiers. Our method demonstrates superior accuracy and a lower false positive (FP) rate. These improvements are attributed to the pre-processing, feature extraction, and dynamic adaptation of the classifier to each dataset. The proposed method, trained on the SQL Injection dataset, effectively understands various attacks and malicious activities.

**Table 7.**  
 Comparison results

Study	Accuracy %	Classifiers ALgorithms
Proposed method	96 %	ANN , SVM , RF , GB
Sheykhkanloo et al.[3]	95%	ANN
Hasan et al [11]	93.8%	SVM , KNN
Rfid et al [12]	93%	RF , Decision Tree, SVM , GB
Joshi et al. [13]	93.9%	Naive Bayes
Ingre e al. [14]	83.7%	Decision Tree

### 6. Conclusion and Future Work

The goal of this research is to develop a framework for detecting SQL Injection using machine learning and classification methods such as Random Forest Classifier, Gradient Boosting Classifier, SVM, and ANN. The results indicate that the ANN Classifier is the most suitable for achieving this study's goal, as it achieved a 96% accuracy rate and completed the testing phase more quickly, making the proposed framework effective. For future research, we will focus on applying deep learning techniques, specifically RNN and LSTM, to further enhance the framework.



**Fig. 6.** The results of all classifiers

## References

- [1] OWASP. "OWASP Top 10: 2021." OWASP, 2021. <https://owasp.org/Top10/>.
- [2] Positive Technologies. "Web Applications Vulnerabilities and Threats: Statistics for 2019." Ptsecurity.com. Positive Technologies, February 13, 2020. <https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/>.
- [3] Sheykhkanloo, Naghmeh Moradpoor. "A learning-based neural network model for the detection and classification of SQL injection attacks." *International Journal of Cyber Warfare and Terrorism (IJCWT)* 7, no. 2 (2017): 16-41. <https://doi.org/10.4018/IJCWT.2017040102>
- [4] Betarte, Gustavo, Álvaro Pardo, and Rodrigo Martínez. "Web application attacks detection using machine learning techniques." In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1065-1072. IEEE, 2018. <https://doi.org/10.1109/ICMLA.2018.00174>
- [5] George, Teresa K., K. Poulouse Jacob, and Rekha K. James. "Token based detection and neural network based reconstruction framework against code injection vulnerabilities." *Journal of Information Security and Applications* 41 (2018): 75-91. <https://doi.org/10.1016/j.jisa.2018.05.005>
- [6] Zhang, Kevin. "A machine learning based approach to identify SQL injection vulnerabilities." In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1286-1288. IEEE, 2019. <https://doi.org/10.1109/ASE.2019.00164>
- [7] Krishnan, SS Anandha, Adhil N. Sabu, Priya P. Sajan, and A. L. Sreedeeep. "SQL injection detection using machine learning." *Revista Geintec-Gestao Inovacao E Tecnologias* 11, no. 3 (2021): 300-310. <https://doi.org/10.47059/revistageintec.v11i3.1939>
- [8] Kareem, Fairoz Q., Siddeeq Y. Ameen, Azar Abid Salih, Dindar Mikaeel Ahmed, Shakir Fattah Kak, Hajar Maseeh Yasin, Ibrahim Mahmood Ibrahim, Awder Mohammed Ahmed, Zryan Najat Rashid, and Naaman Omar. "SQL injection attacks prevention system technology." *Asian Journal of Research in Computer Science* 6, no. 15 (2021): 13-32. <https://doi.org/10.9734/ajrcos/2021/v10i330242>
- [9] Jemal, Ines, Omar Cheikhrouhou, Habib Hamam, and Adel Mahfoudhi. "Sql injection attack detection and prevention techniques using machine learning." *International Journal of Applied Engineering Research* 15, no. 6 (2020): 569-580.
- [10] Ashi, Zein, Laila Aburashed, Mohammad Al-Fawa'reh, and Malek Qasaimeh. "Fast and reliable DDoS detection using dimensionality reduction and machine learning." In *2020 15th International Conference for Internet Technology and Secured Transactions (ICITST)*, pp. 1-10. IEEE, 2020. <https://doi.org/10.23919/ICITST51030.2020.9351347>
- [11] Alkahtani, Hasan, and Theyazn HH Aldhyani. "Botnet Attack Detection by Using CNN-LSTM Model for Internet of Things Applications." *Security and Communication Networks* 2021, no. 1 (2021): 3806459. <https://doi.org/10.1155/2021/3806459>
- [12] Islam, Md Rafid Ul, Md Saiful Islam, Zakaria Ahmed, Anindya Iqbal, and Rifat Shahriyar. "Automatic detection of NoSQL injection using supervised learning." In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, pp. 760-769. IEEE, 2019. <https://doi.org/10.1109/COMPSAC.2019.00113>
- [13] Joshi, Anamika, and V. Geetha. "SQL Injection detection using machine learning." In *2014 international conference on control, instrumentation, communication and computational technologies (ICCICCT)*, pp. 1111-1115. IEEE, 2014. <https://doi.org/10.1109/ICCICCT.2014.6993127>
- [14] Ingre, Bhupendra, Anamika Yadav, and Atul Kumar Soni. "Decision tree based intrusion detection system for NSL-KDD dataset." In *Information and Communication Technology for Intelligent Systems (ICTIS 2017)-Volume 2*, pp. 207-218. Springer International Publishing, 2018. [https://doi.org/10.1007/978-3-319-63645-0\\_23](https://doi.org/10.1007/978-3-319-63645-0_23)
- [15] Sivasangari, A., J. Jyotsna, and K. Pravalika. "SQL injection attack detection using machine learning algorithm." In *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1166-1169. IEEE, 2021. <https://doi.org/10.1109/ICOEI51242.2021.9452914>
- [16] Muhammad, Taseer, and Hamayoon Ghafory. "Sql injection attack detection using machine learning algorithm." *Mesopotamian journal of cybersecurity* 2022 (2022): 5-17. <https://doi.org/10.58496/MJCS/2022/002>

- [17] Farhan, Ammar Hatem, and Rehab Flaih Hasan. "Detection SQL injection attacks against web application by using support vector machine with principal component analysis." In *AIP Conference Proceedings*, vol. 3009, no. 1. AIP Publishing, 2024. <https://doi.org/10.1063/5.0190440>
- [18] Oudah, Mohammed AM, and Mohd Fadzli Marhusin. "SQL Injection Detection using Machine Learning: A Review." *Malaysian Journal of Science Health & Technology* 10, no. 1 (2024): 39-49. <https://doi.org/10.33102/mjosht.v10i1.368>
- [19] Abdullah, Hilmi Salih, and Adnan Mohsin Abdulazeez. "Detection of SQL Injection Attacks Based on Supervised Machine Learning Algorithms: A Review." *International Journal of Informatics, Information System and Computer Engineering (INJIISCOM)* 5, no. 2 (2024): 152-165. <https://doi.org/10.34010/injiiscom.v5i2.12731>
- [20] Kumar, Animesh, Sandip Dutta, and Prashant Pranav. "Analysis of SQL injection attacks in the cloud and in WEB applications." *Security and Privacy* 7, no. 3 (2024): e370. <https://doi.org/10.1002/spy2.370>