The Journal of Supercomputing (2025) 82:31
https://doi.org/10.1007/s11227-025-08165-y

# A cybersecurity method to detect SQL injection attacks using heuristic-driven feature selection and machine learning algorithms

**Bahman Arasteh[1,2,3] · Mohammadbagher Karimi[4] · Huseyin Kusetogullari[5,6] · Keyvan Arasteh[1] · Farzad Kiani[7]**

Received: 16 April 2025 / Accepted: 10 December 2025 / Published online: 26 December 2025


**Abstract**
SQL injection is a serious security risk that allows attackers to access application databases. SQL injection attacks can be identified using various methods, including machine learning algorithms. Finding the top-performing features in the training dataset is a combinatorial optimization problem known to be NP-complete. Finding the dataset's most effective and significant features is the goal of feature selection. This study aims to optimize the sensitivity, specificity, and accuracy of the SQL injection detection method. The first stage of the suggested method involved creating a unique training dataset with 13 characteristics. A binary form of the Whale Optimization Algorithm was suggested to find the most effective features in the dataset. An effective SQL injection detection system was developed by combining the whale algorithm as a feature selector with various machine learning techniques. The suggested SQL injection detector achieved 98.88% accuracy, 99.35% sensitivity, and a 98.83% F1-score using an artificial neural network and the whale optimizer. Using the proposed strategy to select about 31% of the features improved the performance of the attack detectors.

**Keywords** Cybersecurity · SQL injection · Optimal feature selection · Binary whale algorithm · Machine learning algorithms

## 1 Introduction

One essential factor in determining the quality of software products is software security [1]. Furthermore, researchers strongly advise against SQL injection (SQLi) in software development practices, as they view it as a severe software security vulnerability. A web vulnerability known as SQL injection enables attackers to modify the database queries a program generates. Malicious queries,

Extended author information available on the last page of the article

**Table 1** SQL injection queries

| |
|---|
| SELECT name, price FROM products WHERE PRDid = '1' **UNION SELECT user, pass FROM users–**'; |
| SELECT * FROM students WHERE STDId = 200 OR 1 = 1; |
| SELECT * FROM customers WHERE custom_id = (SELECT custom_id FROM customers WHERE username = johndoe' **UNION SELECT custom_id FROM customers WHERE 'b' = 'b**); |
| SELECT * FROM employee WHERE user = 'admin' –' AND pass = ''; |

known as SQLi attacks, modify SQL statements to execute malicious code. Such queries enable unlimited data access and unlawful database intrusion, as attackers can use them to pull data directly from the database. According to the Open Web Application Security Project (OWASP), SQLi attacks rank among the top ten web application security threats and are among the most dangerous web application vulnerabilities [1–3]. Table 1 presents examples of SQL injection queries.

The majority of existing security techniques are unable to defend web application databases against SQLi attacks consistently. Some SQLi detectors are built using machine learning (ML) methods, and the classifiers produced may distinguish between harmful and normal SQL queries [4]. Various real-world datasets are used by ML methods to train and construct SQLi detectors. The effectiveness and performance of the final classifier are highly influenced by the features of the dataset used during the training phase. Finding a dataset's most interesting features is considered an NP-complete issue. Feature selection improves the built SQLi detector's performance, sensitivity, and accuracy by identifying the most influential and minimal features in the training dataset. The best features in a dataset are found using a variety of metaheuristic approaches. Low success rate and convergence, selection of a large number of features (which harms the performance of the SQLi detector), and inefficient accuracy and precision of the created SQLi detectors are the primary drawbacks of the current feature selection methods. Regarding the shortcomings of the existing SQLi detection techniques, the objectives of this study are:

- Increasing the success rate and reliability of SQLi detectors.
- Reducing the false alarm rate of SQLi detectors.
- Improving the accuracy and precision of the SQLi detectors.
- Determining the SQLi dataset's most compelling features.

This work proposed an effective method for identifying SQL injection (SQLi) attacks using a small set of compelling features derived from training datasets. Initially, a real-world dataset including 12 features was utilized. The Whale Optimization Algorithm (WOA) was subsequently designed and implemented in a binary form to determine which features were optimal for the SQLi detectors. Afterward, different ML algorithms were used to create efficient SQLi detectors based on the improved dataset chosen by binary WOA (BWOA). Lastly, the

effectiveness of the suggested feature selection method was verified using the test dataset. The study's primary conclusions are:

- Developing a binary version of the Whale Optimization Algorithm (BWOA) as the selector of features in the SQLi dataset
- Determining the minimal subset of SQLi features that significantly influences the SQLi detector's accuracy and precision.
- Producing an effective and reliable SQLi detector by using only 31% of the most compelling features of the training dataset.

The remaining sections are arranged as follows: Sect. 2 provides an overview and discussion of SQL injection approaches in the relevant literature. Section 3 shows and explains the suggested selector of features and SQLi detection technique. Section 4 presents the experimental setup and performance of the developed SQLi detector. Chapter 5 concludes with conclusions and recommendations for further study.

## 2 Related works

SQLi attacks can be identified in various ways. In this part, we discuss several state-of-the-art works. Unauthorized objects are blocked using blacklists in traditional SQLi approaches. Numerous approaches have been proposed in the relevant research in this area. Reference [5] presents a SQLi detection approach that addresses the limitations of traditional methods, which often fail to identify complex or mixed attack patterns. The authors use a content-matching technique to extract features while preserving important attack tokens that may be lost during grammar parsing or standard word embedding. Using these features, they train several deep learning models, including CNN, CNN-RNN, and CNN-LSTM, and report that the CNN-LSTM model performs best by capturing both structural and sequential patterns. The proposed system achieves a high detection accuracy of 98.35%, demonstrating strong performance on the tested SQLi and XSS data. While the method offers clear advantages, such as preserving key information, improving detection accuracy, and using enhanced deep learning models, it also has limitations, including the possible lack of dataset diversity, limited evaluation against obfuscated attacks, the absence of comparisons with modern Transformer-based models, the potential risk of overfitting, and no assessment of real-time deployment performance.

In [6], the growing challenge of web application attacks, driven by the huge amount of online data and the increasing activity of cybercriminals, has been explained. The paper introduces two SQLi detection approaches: a deep learning model using LSTM and several traditional machine learning classifiers. A major issue addressed is data imbalance, which can cause models to favor benign samples. The authors apply several preprocessing and resampling techniques to reduce this bias and improve classifier performance. Their experiments show that although preprocessing helps, the LSTM model performs far better than all machine learning models. This highlights the model's robustness and

adaptability. The study uses multiple validation methods, including holdout and five-fold cross-validation, to confirm consistency. Various resampling methods (ROS, SMOTE, ADASYN, RUS, and Tomek Links) are tested, but none outperform the LSTM. The paper also explains that ML/DL methods complement existing rule-based systems. These methods are most useful in large-scale, high-risk environments. The authors propose future steps, including optimizing the model for real-time detection, integrating it with WAFs via APIs, improving model explainability, and developing a continuous learning framework.

The study in [7] focuses on detecting SQL injection (SQLi) attacks using lightweight flow-based network protocols rather than full packet inspection. SQLi remains a major threat and is ranked in the OWASP Top 3. Traditional detection methods often require analyzing every packet in a network, which is impractical for high-traffic routers and large-scale environments. To solve this problem, the authors show that SQLi detection is possible using NetFlow-based flow data rather than full packet payloads. They create and publish two datasets containing NetFlow v5 flow records generated from SQLi attacks on the most widely used database engines. Using these datasets, they train several machine learning models. The Logistic Regression and Perceptron + SGD models achieve the best results, both exceeding 96% accuracy and achieving false alarm rates below 1%, with Logistic Regression achieving a false alarm rate below 0.07%. The ensemble model performs moderately well, with 85.6% accuracy and an 89% detection rate. The authors note that their work serves as an initial step toward SQLIA detection using flow data and that the proposed models are suitable for real-world deployment to generate alerts and improve security.

An automated penetration testing tool designed to detect web vulnerabilities such as SQLi, XSS, CSRF, command injection, and directory traversal is suggested in [8]. The tool uses a mix of automated and manual testing, supported by Python scripts, APIs, and integration with OWASP ZAP. It effectively identifies vulnerabilities, improves scanning speed by 20%, and provides clear visual reports for prioritizing risks based on CVSS scores. The study also highlights challenges for automated tools, such as handling obfuscated inputs, multi-step authentication, and complex attacks, such as buffer overflows and XXE. The tool's modular, scalable design helps security teams focus on analysis rather than manual scanning. Future enhancements will include AI-based threat detection and real-time monitoring.

Reference [9] introduces an approach to SQLi detection using graph neural networks. The authors represent each SQL statement as a graph, with statements as nodes and their relationships as edges. They propose three graph CNN models, including a two-layer GCN, a non-local graph convolution model, and a modified non-local model that replaces $1 \times 1$ convolutions with GCN layers. Compared to traditional machine learning and deep learning models, the graph models offer better efficiency, fewer parameters, and the ability to process input sequences of any length. While Transformer-based models and LSTMs sometimes achieve higher accuracy, they require far more computational resources. In contrast, the proposed graph models balance accuracy and speed, making them highly suitable for real-world cybersecurity systems. Overall, this work provides a comprehensive comparison of prior

detection models and introduces a lightweight, highly effective graph-based method for SQL injection detection.

A comprehensive solution for SQLi detection, classification, prioritization, and prevention is proposed in reference [10]. The proposed system, called SQLR34P3R, treats SQLi not as a simple binary problem but as a multi-class, multi-vector threat. The authors collect large datasets of web and network traffic and test several machine learning and deep learning models. Their hybrid CNN-LSTM model achieves an average F1-score of 97% for detecting and classifying SQL injection traffic. A key contribution of SQLR34P3R is its risk analysis module, which uses CVE information and scoring systems to help organizations prioritize vulnerability remediation based on exploitability. The system is tested in real-world settings, including DVWA and DNS exfiltration attacks generated with SQLMap, showing strong performance in real-time detection. SQLR34P3R also functions as a command-line proxy that filters web traffic and blocks SQLi attempts. Its three-part design (multi-source detection, multi-class classification, and advanced risk assessment) provides broader coverage than existing solutions. A drawback of the study is that it does not cover several important areas. It lacks evidence-based vulnerability prioritization, stronger preventive measures beyond IP blocking, real-time traffic detection, and the use of contextual frameworks for better vulnerability scoring. These gaps are left for future work.

Reference [11] presents SQLGuardNet, a new deep learning model that achieves very high accuracy in detecting malicious SQL queries. The model uses a multi-branch architecture combining three techniques: a Transformer block with multi-head attention, a 1D CNN, and an LSTM-based RNN. These branches capture distinct patterns in SQL queries and are merged to form a robust classification system. SQLGuardNet is lightweight (3.31M parameters) yet powerful, achieving high accuracy. Soft voting further improves performance when combined with other top models. The paper also analyzes unusual model behaviors, such as high precision but low recall in some models, and demonstrates that SQLGuardNet maintains stable performance even against complex injection types, such as nested and piggy-backed SQLi. The study notes key limitations, including reliance on Kaggle datasets that may not reflect all real-world attack patterns and the architecture's complexity, which may be challenging for practitioners without deep learning expertise. Overall, SQLGuardNet proves to be an efficient, and robust solution for SQL injection detection.

A lightweight SQLi detection model (SQLLS) built using a bidirectional LSTM is suggested in [12]. The system first converts SQL statements into numerical vectors using TF-IDF, then applies a combined feature selection method (GFC) that uses gradient boosting, Fisher score, and Chi-square tests to keep only the most important features. Mixed-precision training and model pruning are used to reduce memory use and speed up computation. Experiments show that SQLLS achieves high accuracy, a very low false alarm rate (0.154%), and faster running time than existing models. The paper also discusses major limitations of related solutions such as SQLR34P3R. These include analyzing only HTTP requests, omitting other data sources (IP addresses, responses, protocol-level queries), lacking multi-class NetFlow datasets, having imbalanced class sizes, and lacking a feedback loop between

deployed models and training data. IP blocking is also considered a weak prevention method because blocking a single attacker could mistakenly block many legitimate users behind the same public IP address. Collecting limited SQLi datasets, non-real-time traffic detection, and a lack of environmental and business context are the drawbacks of the method when scoring vulnerabilities.

Reference [13] presents MVC-BiCNN, a deep learning system that achieves very high accuracy in detecting SQLi attacks. The method converts each SQL query into three different semantic views using 21 SQL tags. These views help the model better understand the meaning and role of SQL terms. The system then uses a hybrid architecture that combines a bidirectional LSTM to capture long-term dependencies and a CNN to extract local patterns. Each view is processed separately, and the system uses a consensus function to make the final decision. This makes the detection process more robust and reliable. The model achieves a higher detection rate and shows fewer false positives and false negatives than other models. The main strengths of the approach are its high accuracy, rich multi-view representations, and strong hybrid architecture. However, the system has some limitations. It uses complex preprocessing, requires more computation, and focuses solely on SQL injection attacks. It does not yet support other attack types, such as XSS, and lacks dynamic monitoring to adapt to new threats. The authors suggest future work to expand the system to detect additional code injection attacks and to add dynamic monitoring to improve adaptability. This work proposed an efficient approach for using various ML algorithms to detect SQL injection attacks. A specialized training dataset with a limited set of important attributes was used to train the machine learning algorithms. The suggested hybrid machine learning approach mitigates the significant shortcomings of the prior approach, achieving higher success rates and lower errors.

## 3 Suggested method

In this work, we devise and introduce a practical technique for identifying SQL injection attacks. Initially, a unique training dataset with thirteen features was employed. The best features are then chosen in the second stage using a binary WOA termed BWOA. ANN (Neural Network), SVM (Support Vector Machine), DT (Decision Tree), and K-Nearest Neighbor (KNN) approaches have been used to construct an integer classification model for SQLi matching. During the final testing phase, the effectiveness of the proposed feature selection approach was evaluated using a test dataset. The steps of the suggested strategy are depicted in Fig. 1.

### 3.1 Dataset creation

The datasets needed to train a supervised ML algorithm are covered in this subsection. Using SQL Query Profiler, the SQLi dataset was created. Furthermore, conventional requests and SQL injection (SQLi) attacks were generated using SQL injection traffic-generating tools such as SqlMap. The original datasets were processed, and their correlations were calculated to form a single dataset [14].

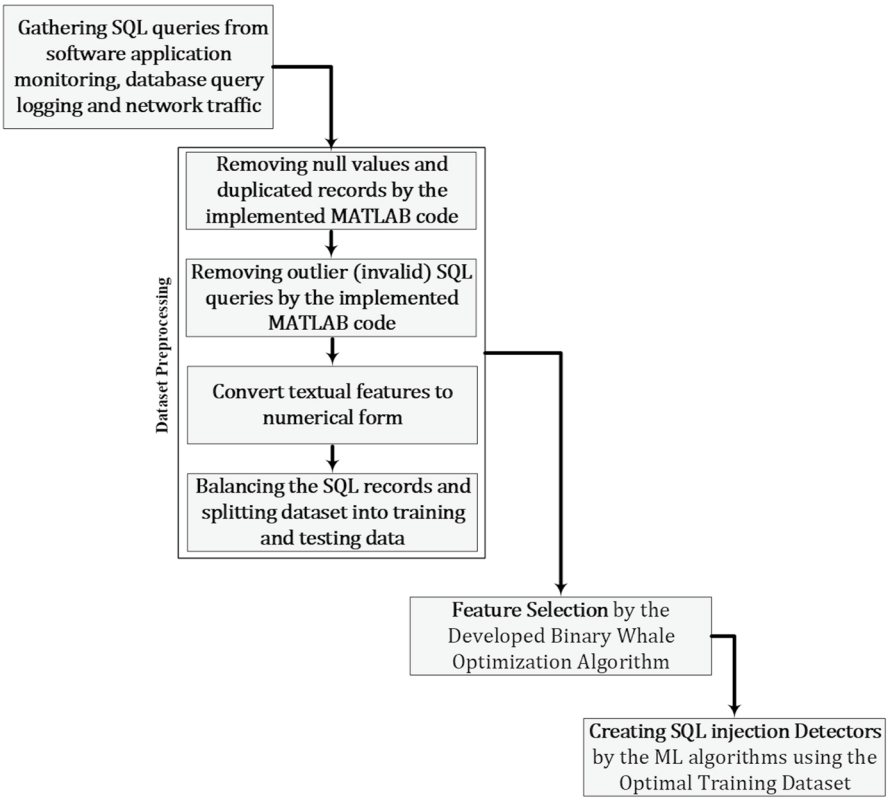


**Fig. 1** Suggested method's workflow

The dataset is expanded to include attack and regular SQL queries for a particular online application. Different SQL complexities and formats can be used to run queries. Relational databases allow for both basic and complex queries. A standard SQL query or a SQL injection (SQLi) attack is represented by each row (record) in the database. Features from SQL queries, including single quotes, arguments, comments, and dots, are included in the dataset. Table 2 presents the features of the dataset used. To ensure objectivity and reproducibility, the extraction followed a consistent rule-based transformation process rather than manual selection. After extraction, all features were converted to numerical values, yielding a homogeneous, ML-friendly dataset that enhances both training efficiency and predictive performance of the applied models. The final dataset provides a balanced and well-structured basis for supervised learning.

The original dataset is further filtered and modified to create a single dataset containing numerical features. There are 1027 entries in this dataset, each representing a normal or malicious SQL query. Thirteen attributes are taken from the SQL query code and included in each record. The dataset's features are shown in Table 2. The dataset contained 473 valid SQL queries and 554 invalid entries

**Table 2** Features of the SQLi attacks dataset [14]

| | |
|---|---|
| 1 | "Length of SQL query" |
| 2 | "Nesting Level Query" |
| 3 | "Num. of union Operator in the SQL Query" |
| 4 | "Num. of Constant Value in the SQL Query" |
| 5 | "Num. of OR Operator in the SQL Query" |
| 6 | "Num. of Specific Character in the SQL Query" |
| 7 | "Type of SQL Query in the SQL Query" |
| 8 | "Num. of AND Operator in the SQL Query" |
| 9 | "Num. of double quotation Character (") in the SQL Query" |
| 10 | "Num. of double quotation Character (') in the SQL Query" |
| 11 | "Num. of Null Value in the SQL Query" |
| 12 | "Num. Parenthesis in the SQL Query" |
| 13 | "Class (Attack or Not Attack") |

(resulting from SQL injection attacks). The filtered and rearranged dataset containing 12 numerical features is utilized in this study to train the ML algorithm [14]. ML algorithms perform more accurately and precisely when the features in the dataset have similar data types. To establish a homogeneous dataset, the attributes of the primary dataset (see Table 2) were converted to numeric values. The attributes used in this research are all numeric and have the same data type. Twelve of the dataset's features are independent, and the thirteenth feature (which determines if the query is valid or an attack) is dependent. The label of the normal query (property 13) is zero, and the label of an attack query is set to one. The numerical properties of the resulting dataset are displayed in Table 3.

Table 3 lists the features extracted from the raw SQLi payloads. These features were derived from string-based SQL queries containing both normal and attack patterns. During preprocessing, unnecessary elements were removed, leaving only meaningful features. Each extracted feature was then converted into a numerical

**Table 3** Feature of the scalar dataset [14]

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 1 | 1 | 2 | 0 | 3 | 2 | 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 16 | 1 | 0 | 1 | 0 | 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 11 | 1 | 0 | 1 | 0 | 4 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | 17 | 1 | 0 | 1 | 0 | 10 | 1 | 0 | 0 | 1 | 0 | 2 | 1 |
| 5 | 28 | 2 | 1 | 7 | 0 | 13 | 2 | 0 | 0 | 2 | 0 | 0 | 1 |
| 6 | 17 | 1 | 0 | 2 | 0 | 10 | 1 | 0 | 0 | 1 | 0 | 2 | 1 |
| 7 | 15 | 2 | 1 | 1 | 0 | 5 | 1 | 0 | 1 | 4 | 1 | 0 | 1 |
| 8 | 14 | 1 | 0 | 3 | 0 | 4 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 9 | 31 | 3 | 0 | 5 | 0 | 15 | 1 | 0 | 0 | 2 | 0 | 4 | 1 |
| 10 | 18 | 1 | 0 | 3 | 0 | 8 | 1 | 1 | 0 | 4 | 0 | 0 | 0 |
| 11 | 18 | 1 | 0 | 3 | 0 | 8 | 1 | 1 | 0 | 4 | 0 | 0 | 0 |

format, making the dataset homogeneous and compatible with machine learning algorithms. This conversion ensures that all features share a consistent type, improving model efficiency and stability. The selected features are sufficient to capture the essential differences between normal and malicious queries. Consequently, the resulting dataset is compact, discriminative, and efficient for use in feature selection and ML-based SQLi detection.

### 3.2 WOA as feature selection

This work introduces a binary WOA for feature selection. Originally, WOA was developed as a metaheuristic for optimization [15]. Enhanced WOA's distinctive local search method uses advanced mutation and crossover operators. Once the dataset is prepared, BWOA, a binary variation in WOA, is created. This algorithm's objective is feature selection, in which the best characteristics of the SQLi dataset are identified. The feature selection procedure is carried out before the ML algorithms to produce the appropriate classification model. BWOA is a swarm-based approach that uses local and global search strategies. WOA is a bio-inspired algorithm that mimics the bubble-net hunting technique of humpback whales. WOA is based on how humpback whales hunt, especially their use of bubble nets to feed. To capture their meal, the whales construct a bubble net around them. Algorithm 1 displays the pseudocode of the developed WOA for the feature selection.

**Algorithm 1** Pseudocode of the WOA which was developed as a feature selector [15]

```
1. Initialize the population of whales and the parameters a, A, C, l, and p
3. Calculate the fitness of each whale in the population
4. X_best = The best whale (solution) based on fitness
5. while (termination condition not met) do
6.     for each whale i (i = 1, 2, ..., n) do
               //Update the parameters A, C, and l:
7.             A = 2a * r1 − a;
8.             C = 2 * r2;
9.             l = random number in [-1, 1];
10             p = random number in [0, 1];
11.            if (p < 0.5) then
12.                if (|A| < 1) then
                   //Update the position of the whale using the best solution
13.                   D = |C * Xbest - Xi|;
14.                   Xi = Xbest - A * D;
15.                 else if (|A| ≥ 1) then
                   //Select a random whale and update the position by a random whale
16.                      D = |C * Xrand - Xi|;
17.                      Xi = Xrand - A * D;
18.                end if
19.            else if (p ≥ 0.5) then
                  //Update the position in a spiral shape around the best solution
20.                  D = |Xbest - Xi|;
21.                  Xi = D * exp(b * l) * cos(2π * l) + Xbest;
               end if
22.     calculate the fitness of each whale;
23.     Update X_best;
 end while
24. Return the best solution (X_best)
```

Using BWOA, which efficiently identifies the most significant features, the performance of the ML algorithms and, consequently, the classification model was enhanced. This method uses whale intelligence to select the best features by balancing exploration and exploitation. Each whale in the system is represented by a 13-dimensional vector, which also corresponds to the number of features in the dataset. A whale vector, which indicates the population matrix, consists of *n* rows and *13* columns. A whale's structure is displayed as a feature vector in Figure 2. This length corresponds to the quantity of features in the training dataset labeled F0 to F12. This binary vector's bits correspond to distinct features in the training dataset. The related feature is activated when the bit is set to 1; when it is set to 0, the feature is disabled during training.

The whale movement in the WOA is modeled using mathematical equations that simulate the behavior of real whales during the bubble-chasing strategy. The whale's movement in the WOA is as follows:

- Updating the position of the whale using the best whale
- Updating the position of the whale using the select a random whale
- Updating the position of the whale in a spiral shape around the best whale

In the first form of whale movement in WOA, the whales locate the prey (optimal solution) and approach it. This motion is modeled by Eq. (3) and Eq. (4). The

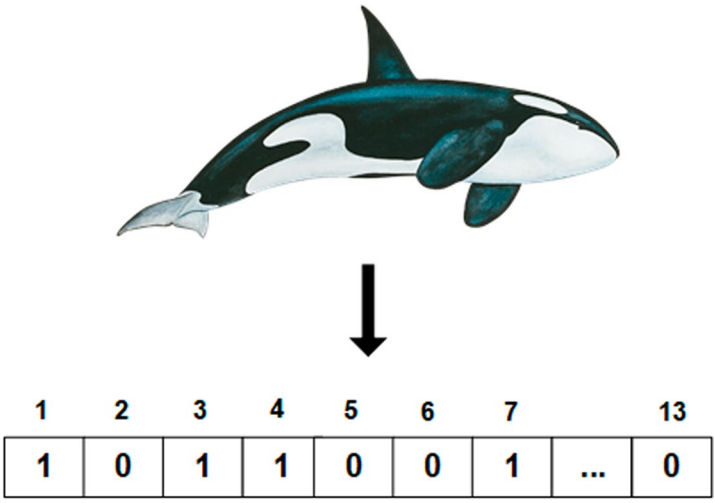


**Fig. 2** Implemented binary vector for each whale in the population

whale's current location in iteration t is shown by X (t). The whale's modified position at iteration $t+1$ is represented by $X(t+1)$. The position of the best solution (prey) at iteration $t$ is indicated by $X_{best}\,(t)$. A and C are coefficient vectors calculated by Eq. (1) and Eq. (2). *r1* and *r2* are random numbers in the range [0,1].

$$A = 2a * r1 - a\ D \tag{1}$$

$$C = 2 * r2 \tag{2}$$

$$D = |C \cdot Xbest(t) - X(t)| \tag{3}$$

$$X(t+1) = Xbest(t) - A \cdot D \tag{4}$$

When $|A| \geq 1$, the whales use exploration by randomly selecting a whale in the population and moving toward it. This prevents the local optima by encouraging global exploration. In this form of movement (the second form of whale movement in the WOA), the position of the whale is updated using the position of a randomly selected whale. Equation (5) and Eq. (6) are used to model this type of movement in the whale algorithm.

$$D = |C * X_{rand} - X_t| \tag{5}$$

$$X(t+1) = X_{rand} - A * D; \tag{6}$$

In the third form of movement, two basic methods are used to simulate the bubble-net hunting strategy of whales: spiral updating of position and a shrinking-encircling

mechanism. The whale has a probability $p$ that will determine which of these two techniques it will pick. When $p < 0.5$ and $|A| < 1$, the whales move closer to the best solution, exploiting the search space around it. This is modeled using Eq. (3) and Eq. (4). If $p \geq 0.5$, the whale follows a logarithmic spiral path toward the prey, simulating a bubble-net attack. This type of movement is modeled by Eq. (7) and Eq. (8); in these equations, $b$ is a constant that defines the shape of the logarithmic spiral and $l$ is a random number in the range $[-1, 1]$ that controls the spiral movement.

```
D' = |Xbest - Xi|;
```
(7)

```
Xi = D' * exp(b * l) * cos(2π * l) + Xbest;
```
(8)

### 3.3 Binary WOA

The sigmoid function was utilized to create a new binary WOA. Every whale (solution) in this method has a binary number. To select or exclude features, the binary response can take only 0 or 1 values. A feature is chosen for the new dataset if its value is 1. The feature is not selected if its value is 0. Values from a population matrix were converted to binary space using the sigmoid function. The sigmoid transfer function returns the value of 1 or 0. By applying the arbitrary threshold in Eq. (9) to transform an element into a binary solution, the sigmoid function chooses a feature. The suggested BWOA population is converted to the binary search space using Eq. (9) and Eq. (10).

$$SG\left(X_i^d(t)\right) = \frac{1}{1 + e^{-X_i^d(t)}} \tag{9}$$

$$X_i^d(t+1) = \begin{cases} 0 if rand < SG\left(X_i^d(t)\right) \\ 1 if rand \geq SG\left(X_i^d(t)\right) \end{cases} \tag{10}$$

The utilized fitness function is computed using Eq. (11). In Eq. (11), $\gamma_R(D)$ indicates the quality of the selected features by the BWOA. $C$ indicates the total number of features. The variables $\alpha$ and $\beta$ are considered as coefficients ($\alpha \in [0,1]$ and $\beta = 1\text{-}\alpha$). Variable $R$ shows the number of features selected by BWOA. The percentage of unselected features is indicated by $|C\text{-}R|/C$. The fitness function is used to enhance the quality of classification ($\gamma_R(D)$.) To transform the fitness function into a function that minimizes it, the error rate of classification is used instead of the fitness function. Furthermore, the fitness function utilizes the selected features' number instead of unselected features. In Eq. (12), $E_R(D)$ displays the error rate; $|R|$ shows the quantity of features identified by BWOA; also, $|C|$ indicates to total number of features. The values of $\alpha$ and $\beta$ are the coefficient parameters in Eq. (12) ($\alpha \in [0,1], \beta = 1 - \alpha$).

$$\text{Fitness} = \alpha\gamma_R(D) + \beta\frac{|C - R|}{|C|} \tag{11}$$

**Table 4** Best values of BWOA parameters

| Parameter | Optimal Value |
|---|---|
| Population size | 50 |
| B | 1 |
| P | random number in [0, 1] |
| l | random number in [−1, 1]; |
| Max iterations | 100 |

$$\text{Fitness} = \alpha E_R(D) + \beta \frac{|R|}{|C|} \tag{12}$$

## 4 Experiments

### 4.1 Platforms

The BWOA was implemented as a feature selector in MATLAB. Four ML classification methods were implemented to create the Artificial Neural Network (ANN), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Decision Tree (DT)-based classifiers. Two varieties of SQLi classifiers were constructed; in the first experiments, the raw dataset (all features) was used to train ML models. For the second kind, ML techniques were used to construct the SQLi classifier after the best feature was chosen using the built-in feature selector (BWOA). Two distinct SQLi classification models were created throughout the trials. ML techniques used the dataset's features to train the SQLi classifier. In the second modification, the best features were found using BWOA. The experiments were carried out on a PC running Windows 11, equipped with an Intel Core i7 3.4 GHz processor, 32 GB of RAM, and other specifications. MATLAB version 2020b was used to implement the suggested plan. Sensitivity, accuracy, F1-score, and precision are the classification metrics (criteria) that are applied to assess the recommended approach. In the BWOA, the fitness function is a function of the error rate and the quantity (number) of selected features. The BWOA parameters were experimentally calibrated following the criteria. Table 4 lists the BWOA parameters that have the highest values.

The efficacy of SQLi detectors constructed using ANN, KNN, DT, and SVM, with and without the suggested features selector, is assessed using the methodology described in this research. The chosen features, sensitivity, F1, accuracy, and precision were considered when evaluating the suggested method. The following research questions have been thoroughly investigated and tested:

*RQ1:* How efficient is the proposed feature selection method in terms of success rate and convergence behavior?

*RQ2:* How much does the proposed feature selector reduce the number of required features for training?

**Table 5** Description of the dataset

| Dataset | # Whole Records | # Normal Records | # Attack Records |
|---|---|---|---|
| # Queries Record | 1027 | 473 | 554 |

*RQ3:* Does the proposed BWOA-based feature selection method significantly improve classification performance (F1-score, sensitivity, accuracy, and precision) compared to existing SQLi detection approaches?

*RQ4:* How stable and consistent is the selected feature subset across different runs or datasets?

In this study, two sets of ML tests were conducted. The dataset containing 12 features was first utilized to train the ANN, KNN, DT, and SVM. In the second experiment, ML techniques were used to build a SQLi detector based on the features selected by BWOA. The results were examined and validated to assess how well the proposed feature selection method performed on the SQLi dataset. The classifiers trained with ANN, KNN, DT, and SVM algorithms were evaluated on the same SQLi dataset. The model was trained using just 70% of the available data. Furthermore, 30% of the total dataset is used for testing. Training and test datasets with uniform distributions and carefully selected data include both standard SQL queries and SQLi attacks. The dataset's structure is outlined in Table 3. This study considered several variables, including the length of the query, the number of parentheses, the number of AND operators, the number of UNION statements, the number of special characters, and other features. Numerical features should be used when training ML algorithms to improve classification model performance. Consequently, the original dataset's nominal features were transformed into numerical features. The data format is displayed in Tables 2 and 3. Each row in the dataset shows a selection of each query's features (Table 2). Additionally, whether the related record is a regular SQL query or a SQLi query is indicated by the last attribute (the 13th attribute). Table 5 provides information about the dataset used.

## 4.2 Results

### 4.2.1 Convergence and success rate

Numerous experiments were carried out to evaluate the convergence behavior and effectiveness of the proposed approach. The overall convergence of the BWOA in different configurations is presented in Figs. 3, 4, 5, 6, and 7. These results collectively show how BWOA successfully converges toward selecting the optimal feature subset for the SQLi dataset. Each figure illustrates the fitness function's progression, which forms the basis for assessing convergence quality. Figure 3 presents the results of the first experiment, in which the convergence behavior of SQLi detectors created using BWOA and different ML algorithms (ANN, DT, SVM, and KNN) was monitored over 100 iterations. The results demonstrate that BWOA consistently converges without falling into local optima, showing its ability to locate the most informative features in the

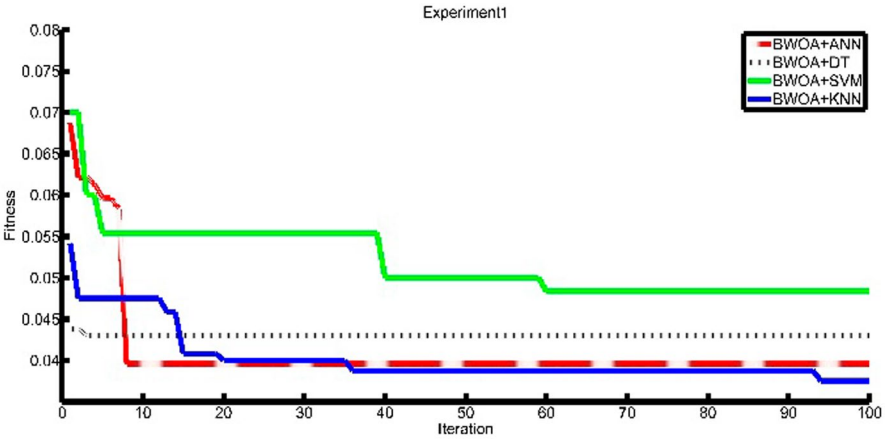


**Fig. 3** Convergence rate of BWOA + ML in experiment 1

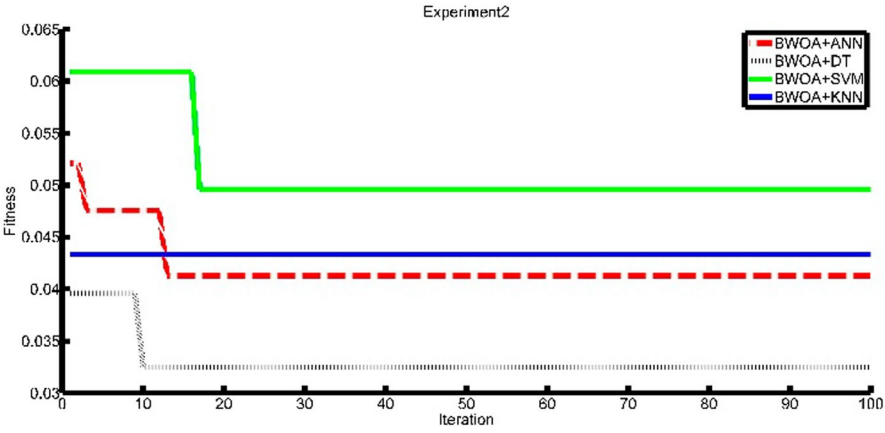


**Fig. 4** Convergence rate of BWOA + ML in experiment 2

dataset. Figure 4 displays the results of the second experiment, highlighting the performance differences among SQLi detectors trained using different ML algorithms. These findings confirm that BWOA adapts uniquely to each ML method, producing distinct convergence curves while still steering each classifier toward a near-optimal feature subset.

Figure 5 further reinforces these observations by comparing the fitness values obtained across multiple feature selection runs. In this figure, BWOA + DT and BWOA + KNN achieve the lowest (optimal) fitness values, indicating the lowest detection error and the fewest selected features. In contrast, BWOA + SVM shows slightly weaker performance, though still benefiting from BWOA's optimization. Figures 6 and 7 illustrate additional repetitions and stability analyses, demonstrating that the variation in fitness curves across runs remains small. This confirms that the

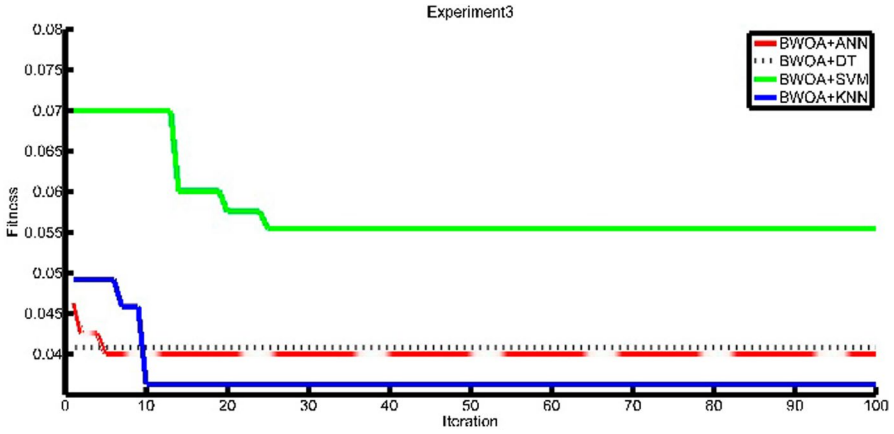


**Fig. 5** Convergence rate of BWOA + ML in experiment 3

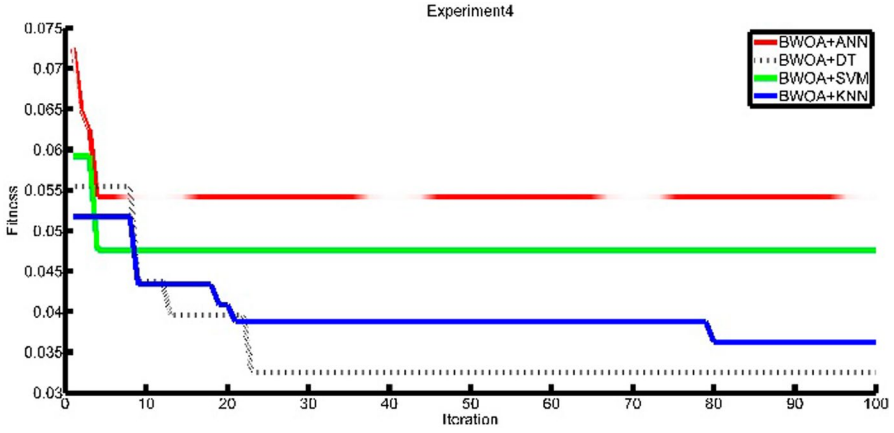


**Fig. 6** Convergence rate of BWOA + ML in experiment 4

proposed BWOA-based feature selection method is both stable and reliable. Moreover, the SQLi detectors constructed using BWOA + ML consistently outperform those trained on the raw dataset without feature selection. During the evaluation phase, the effectiveness of the selected features was validated using accuracy, precision, and sensitivity metrics. Since the original training dataset contains 12 features, the substantial improvement achieved by applying BWOA confirms that removing irrelevant or redundant features leads to more accurate and robust SQLi detection models.

Table 6 displays the fitness values of the BWOA and various ML algorithms after 100 iterations. The results indicate that the fitness values provided by the BWOA across different runs are similar. The lower standard deviation in the fitness values

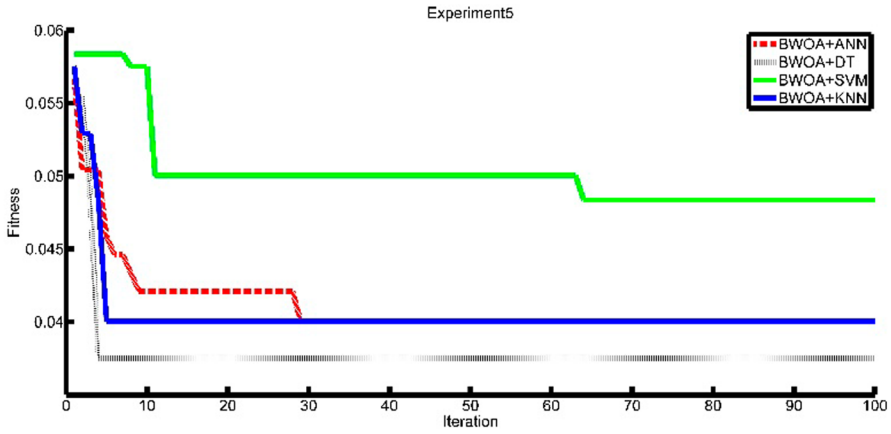


**Fig. 7** Convergence rate of BWOA + ML in experiment 5

obtained across different BWOA executions indicates greater stability and a higher success rate.

### 4.2.2 The suggested SQLi detector's accuracy

The proposed feature selection method was used with classification algorithms (ANN, KNN, DT, and SVM) in the initial set of tests to build the SQLi classifier. Five runs of each experiment were conducted. The execution results of five distinct ML algorithms are displayed in Fig. 8. Every feature in the dataset was initially applied during the ML algorithms' training phase. Second, the ML algorithms' training phase made use of the features that BWOA had chosen. When BWOA is not used as a feature selector, KNN and ANN achieve higher accuracy in SQLi classification than DT and SVM. On the other hand, when BWOA was absent, the SQLi detector produced by DT and SVM did not function well.

Several experiments were conducted using feature selection to assess the performance of the proposed SQLi detector. The suggested BWOA technique was used to choose the best features from the SQLi dataset. Various ML techniques were trained on the dataset filtered by BWOA. Next, the filtered SQLi dataset was used to test the SQLi detectors created. Figure 8 shows that the SQLi detectors generated with the BWOA + ML approaches have an average accuracy higher than those developed with the ANN, SVM, and KNN methods without feature selection. The findings show that the SQLi detector's accuracy (created by ANN) is around 97.66%, while this figure for the SQLi detector constructed under BWOA + ANN is approximately 98.08%. The SQLi detector's accuracy improved significantly when BWOA was used as a feature selector. The average accuracy of BWOA + DT's SQLi detectors is 96.29%. By comparison, the SQLi detector produced by DT without feature selection is approximately 92.01%. The accuracy of the SQLi detector (produced by the ML algorithms) without feature selection is 95.73%, whereas the suggested SQLi detector (BWOA + ML) achieves approximately 97.80%. Regarding the box

**Table 6** Fitness values obtained by BWOA, along with different ML algorithms in five runs

| | Run1 | Run2 | Run3 | Run4 | Run 5 | AVG | STDV |
|---|---|---|---|---|---|---|---|
| ANN | 0.03959 | 0.03959 | 0.04334 | 0.05418 | 0.04002 | 0.04062 | 0.00625 |
| DT | 0.04295 | 0.03251 | 0.04084 | 0.03251 | 0.03751 | 0.04371 | 0.00475 |
| SVM | 0.04835 | 0.04960 | 0.05543 | 0.04753 | 0.04835 | 0.04429 | 0.00320 |
| KNN | 0.03751 | 0.04334 | 0.03626 | 0.03626 | 0.04002 | 0.03834 | 0.00302 |

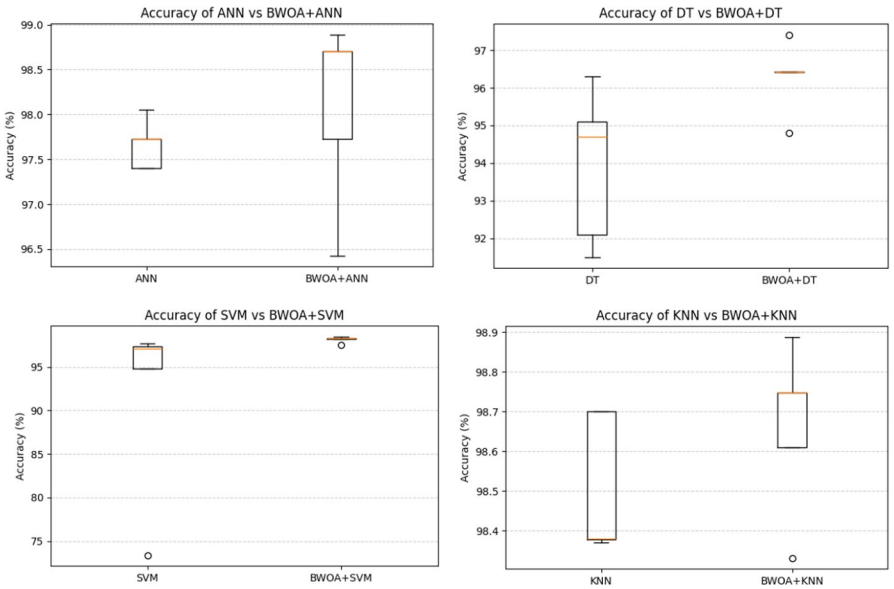


**Fig. 8** Accuracy values provided by the SQLi detectors with and without feature selection

diagram shown in Fig. 8, the average standard deviation of the accuracy values provided by different SQLi detectors created by BWOA + DT, BWOA + SVM, and BWOA + KNN was reduced due to the use of BWOA.

### 4.2.3 Sensitivity of the suggested SQLi detector

The sensitivity of the generated SQLi classifiers, both with and without feature selection, is displayed in Fig. 9. The SQLi detector built with BWOA + ML has an average sensitivity of roughly 97.75%. Inefficient features in the training dataset may affect the SQLi classifier's performance. The findings verify that the chosen features significantly affect SQLi detection sensitivity. The suggested feature selector extracts the most beneficial characteristics from the training dataset and eliminates the ineffective ones. As shown in Fig. 9, the average sensitivity of SQLi detectors built with an ANN, without feature selection (on the raw dataset), is approximately

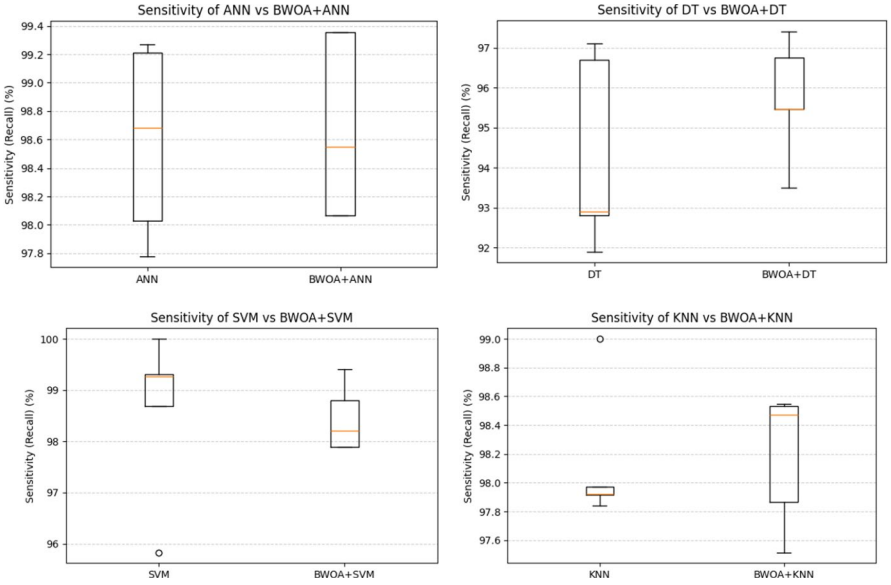


**Fig. 9** Sensitivity of the created SQLi detector with and without the feature selection

98.59%. Conversely, the ANN-based SQLi detector with feature selection has an average sensitivity of approximately 98.67%. Similarly, the average sensitivity of the SQLi detector proposed by DT, with and without BWOA, was 94.28% and 95.71%, respectively. With feature selection, the average recommended sensitivity of the SQLi detector grew from 97.40 to 97.75%. Compared with detectors trained on the raw dataset, detectors trained with feature selection achieve higher average sensitivity.

### 4.2.4 The suggested SQLi detector's precision

Figure 10 displays the precision of different SQLi detectors constructed during five runs using various ML techniques. According to the results shown in Fig. 10, the suggested feature selection significantly improves the precision of the SQLi detectors. Using the BWOA-filtered dataset, multiple machine learning algorithms were trained to generate an effective SQLi classifier. Figure 10 displays the box plot of the average precision of the SQLi detectors executed across five runs. According to the findings, the precision of the created SQLi detector for DT is around 92.76%, while the precision of BWOA + DT is approximately 96.64%. While the precision of BWOA + KNN is roughly 98.91%, that of the detector that was designed for KNN is approximately 98.50%. On average, the developed SQLi detector created by ML without the feature choice is about 94.03%, whereas the BWOA + ML has a precision of about 97.73%. Furthermore, the standard deviation of the precision of the suggested SQLi detector during different executions is considerably reduced. The lower the standard deviation, the higher the stability and success rate of the proposed method compared to the existing methods.

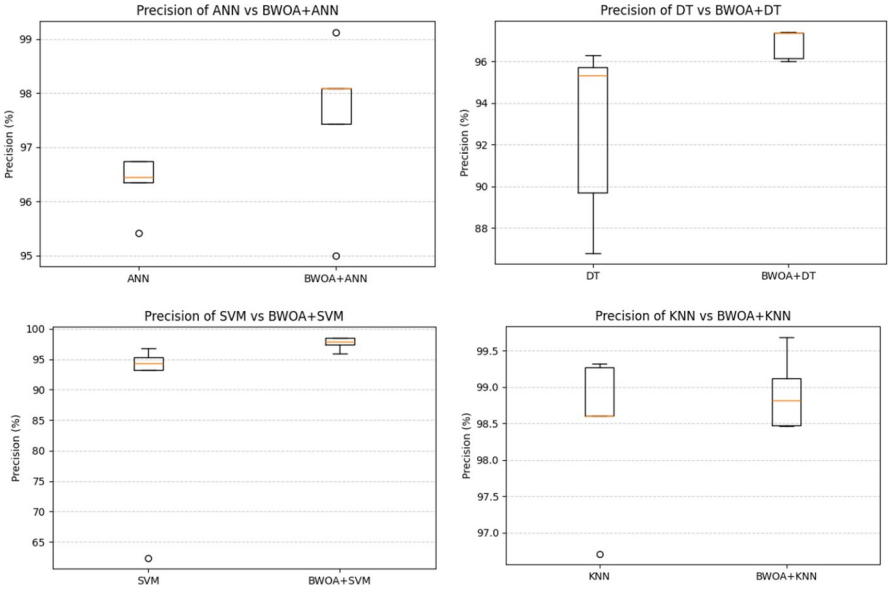


**Fig. 10** Precision of the produced SQLi detector with and without the feature choice

### 4.2.5 F1 value of the suggested SQLi detector

The F1 measure represents the harmonic mean of recall and precision for SQLi classifiers. Figure 11 shows the F1 value for the SQLi detectors generated by the BWOA + ML algorithms. The best results are achieved with ML algorithms that leverage unique features. The F1 values of the suggested SQLi detectors produced by BWOA + ANN and BWOA + KNN are 98.10% and 98.54%, respectively. The F1 values for SQLi detectors based on BWOA + SVM are 98.02%, and the F1 value for SQLi detectors generated by SVM in the absence of feature choice is around 92.62%. As shown in Fig. 11, the SQLi detector generated by BWOA + ML has an F1 score of 97.73%. The F1 score of the SQLi detector increases significantly with the proposed feature selection.

### 4.2.6 Stability of the suggested SQLi detector

The impact of the suggested feature selection method on the functionality of the generated SQLi detectors, using various ML algorithms, is shown in Fig. 12. The results indicate the average performance (accuracy, sensitivity, precision, and F1) of the SQLi detectors generated by ML algorithms with and without the suggested BWOA. Without feature selection, the average accuracy across the five runs of the SQLi detectors (ANN, DT, SVM, and KNN) is approximately 95.75%. Before ML training, this percentage rose to 97.80% when the suggested feature selection was applied. A comparable enhancement was achieved for the criterion of sensitivity. Using ML methods without a feature picker, the average sensitivity across five runs

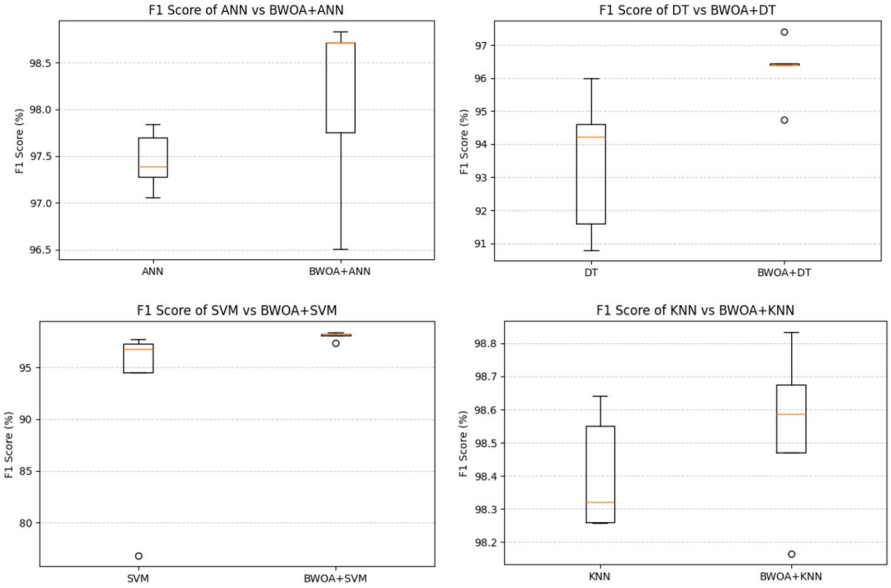


**Fig. 11** F1 value of the produced SQLi detector with and without the feature choice

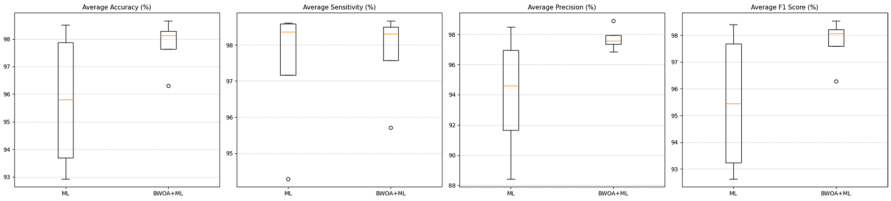


**Fig. 12** Standard deviation among the average performance of the created SQLi detectors with and without features selector during five executions

of SQLi detectors was approximately 97.40%; the suggested selector raised this to 97.75%. For the precision criterion, a similar improvement was obtained. Five runs of SQLi detectors with ML techniques, without feature selection, yield an average precision of about 94.03%. This percentage increased to 97.73% with the recommended feature choice. Regarding the findings shown in Fig. 12, the built-in SQLi detectors exhibit superior accuracy, sensitivity, precision, and F1 scores. Overall, the suggested feature choice enhances the performance of SQLi detectors generated by ML algorithms.

The stability of the feature choice algorithm is the subject of one of the research questions. The variance and distribution of results from SQLi detectors made by several ML algorithms are displayed in Fig. 12. The findings demonstrate considerable differences in the performance (accuracy, sensitivity, and precision) of SQLi detectors generated by various ML methods. Nonetheless, thanks to the BWOA, the SQLi

**Table 7** Selection number of each feature by the BWOA for five executions using various ML algorithms

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BWO + ANN | 1 | 5 | 0 | 5 | 1 | 3 | 0 | 0 | 0 | 0 | 1 | 4 |
| BWO + DT | 4 | 0 | 2 | 5 | 0 | 3 | 1 | 0 | 0 | 1 | 0 | 1 |
| BWO + SVM | 0 | 4 | 1 | 5 | 0 | 3 | 0 | 1 | 2 | 0 | 1 | 4 |
| BWO + KNN | 3 | 2 | 0 | 5 | 1 | 2 | 0 | 0 | 1 | 0 | 0 | 3 |

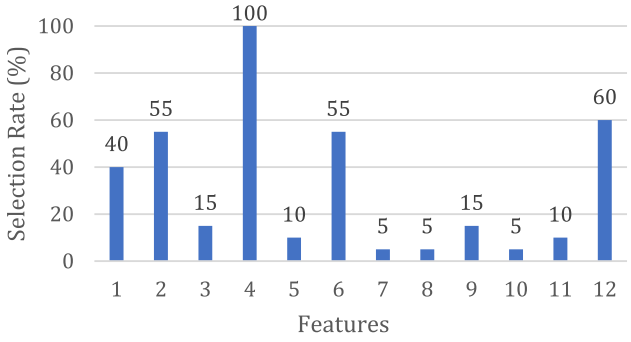


**Fig. 13** Selection probability of each feature by the BWOA in different ML algorithms

detectors produced using different ML algorithms yield comparable results. The accuracy distribution of SQLi detectors created by ML and BWOA + ML is 2.74 and 1.03, respectively. Indeed, the variation in the SQLi detector's accuracy was reduced considerably. The variation among the sensitivity of the SQLi detectors generated by ML and BWOA + ML is 2.04 and 1.37, respectively. Thanks to the suggested feature selection method, the standard deviation among the precision of the SQLi detectors during different executions is reduced from 4.42 to 0.86. The findings show significant dispersion in the results produced by SQLi detectors across different ML algorithms without the use of a BWOA, whereas the red boxes in Fig. 12 demonstrate a considerable reduction in dispersion in the results obtained by the created SQLi detectors across different BWOA + MLs. Indeed, the stability of the results from the ML + BWOA-created SQLi detectors is indicated by their lower standard deviation. Overall, the suggested SQLi detection method is more stable and consequently more reliable than the existing method.

#### 4.2.7 The selected features by the BWOA

Table 7 displays the number of times the BWOA selected each feature in five runs. Figure 13 shows the selection probability of each feature by the suggested BWOA. Regarding the results, feature 4 is the most compelling SQLi detection feature selected by the BWOA. Indeed, the "number of constant values" in the SQL queries is the most persuasive feature for SQLi detection. Feature 12 (number of parentheses in the SQL queries) is the second practical feature identified by the BWOA. The third effective feature is the query's nesting level (second feature). Feature 6 is

**Table 8** Characteristics that the BWOA selected for five executions of various ML algorithms

| | BWO + ANN | BWO + DT | BWO + SVM | BWO + KNN |
|---|---|---|---|---|
| Run1 | [2, 4, 6, 12] | [1, 4, 10] | [2, 4, 9, 12] | [1, 4, 12] |
| Run2 | [2, 4, 6, 12] | [1, 4, 6] | [3, 4, 6, 12] | [2, 4–6] |
| Run3 | [2, 4–6] | [1, 3, 4, 12] | [2, 4, 6, 11, 12] | [1, 4, 12] |
| Run4 | [1, 2, 4, 11, 12] | [1, 4, 6] | [2, 4, 6, 8] | [2, 4, 6, 9] |
| Run5 | [2, 4, 12] | [3, 4, 6, 7] | [2, 4, 9, 12] | [1, 4, 12] |

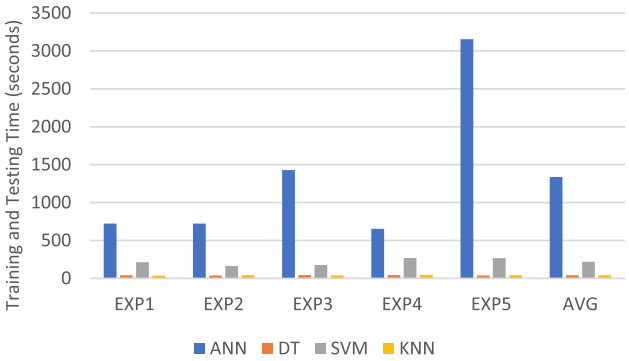


**Fig. 14** Required time for training and testing of different ML algorithms to create the SQLi classifier in five executions

another essential feature for detecting SQLi attacks. Feature 6 indicates the number of specific characters in the SQL query. The length of SQL queries (feature 1) is another effective feature for creating SQLi detectors. The number of union operators and the number of double quotations (“) in the SQL queries are the other essential features of the SQLi dataset used to train the ML algorithms. The BWOA selects an average of 4, 3.4, 4, and 4.2 features in ANN, DT, SVM, and KNN, respectively. From the dataset’s 12 features, the suggested BWOA selects, on average, 4 features. Furthermore, using only roughly 4 features, the suggested SQLi detectors, as illustrated in Fig. 13 and Table 7, offer noticeably improved accuracy, sensitivity, precision, and F1 scores.

Table 8 displays the feature set chosen by BWOA for each of the five runs of various ML algorithms. The suggested SQLi detector performs better when approximately 31% of the features are used. The SQLi classifier that results performs better when the ML algorithm is trained on a dataset with ideal features.

The time required to develop classifiers (training and testing) using various ML approaches is displayed in Fig. 14. The outcomes demonstrate that ANN needs more runtime than other techniques. KNN and DT have shorter average runtimes on the selective dataset.

The findings show that BWOA + ML improved the sensitivity, accuracy, F1, and precision of the SQLi detectors. The feature chosen by the suggested BWOA in various implementations is displayed in Table 8. By choosing 31% of the most

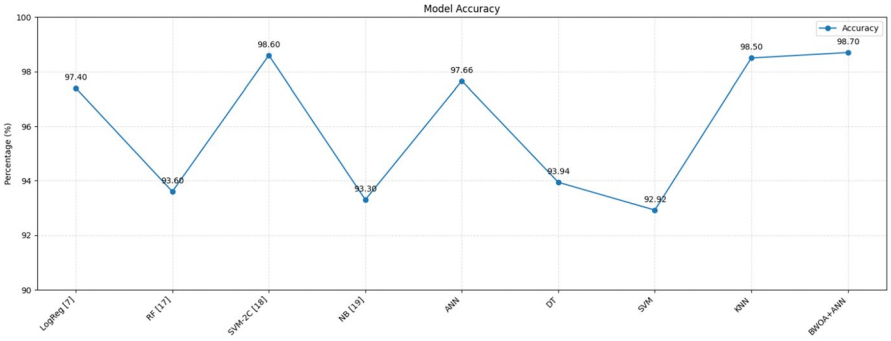


**Fig. 15** Accuracy of different SQLi detection methods using different ML algorithms

advantageous attributes, the proposed method enhanced the SQLi detection system's overall performance. The study found that the BWOA-based approach outperforms earlier attack-detection techniques in terms of accuracy, sensitivity, F1, and precision. Furthermore, since the necessary SQLi detector considers only 31% of the features, it runs faster than earlier techniques. Consequently, the suggested method drastically reduces the time needed to fold to identify SQL injection attacks. The accuracy, sensitivity, precision, and F1 values of the SQLi detectors produced by BWOA + ML exhibit lower standard deviations than those of alternative methods. According to the results shown in Figs. 4, 5, 6, and 7, the lower dispersion in accuracy, sensitivity, F1, and precision was attained when the training dataset was filtered using BWOA. Indeed, the suggested SQLi detector provides reliable and stable results thanks to the BWOA.

Figure 15 illustrates the accuracy comparison of various SQLi detection methods across multiple machine learning algorithms. As shown in the figure, the BWOA + ANN model achieves the highest accuracy (98.70%), outperforming all baseline approaches, including Two-Class SVM (98.60%), KNN (98.50%), and ANN (97.66%). Traditional classifiers such as Random Forest, Naïve Bayes, DT, and SVM achieve noticeably lower accuracy, ranging from 92 to 94%. These results demonstrate that the proposed BWOA + ML-based SQLi detector provides a more reliable and robust detection performance than existing machine learning models. The integration of BWOA optimization significantly improves feature selection and model learning, resulting in superior classification accuracy for SQL injection attacks.

Figure 16 presents the sensitivity performance of different SQLi detection algorithms and highlights how effectively each method identifies malicious SQL injection attempts. The results show that the proposed BWOA + ANN model achieves the highest sensitivity at 99.35%, demonstrating superior capability in detecting actual SQLi attacks compared to all other models. Two-Class SVM also performs strongly with 99.80%, followed closely by ANN, SVM, and KNN, all exceeding 98%. In contrast, traditional approaches such as Random Forest and Naïve Bayes exhibit significantly lower sensitivity, with Random Forest dropping to just 57.70%. These findings indicate that while several ML-based detectors perform reasonably well, the

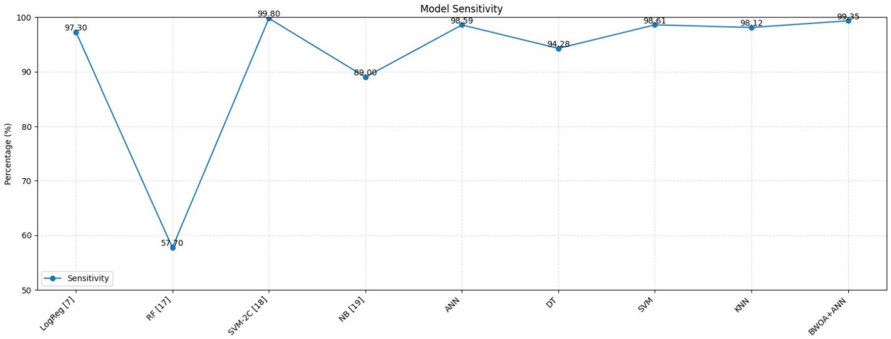


**Fig. 16** Sensitivity of different SQLi detection methods using different ML algorithms

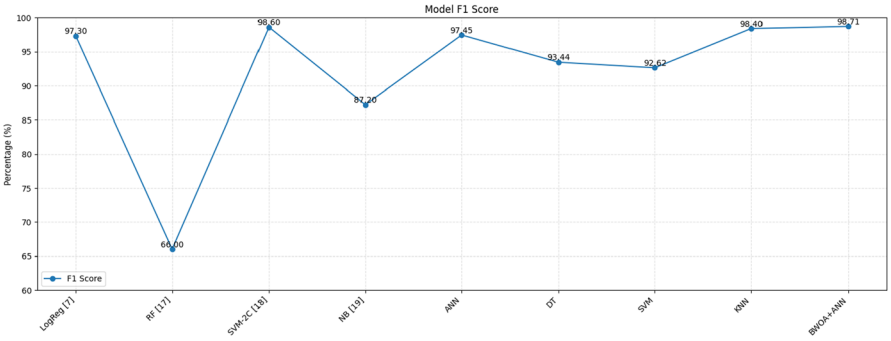


**Fig. 17** F1 value of different SQLi detection methods using different ML algorithms

BWOA-optimized model provides the most consistent and reliable detection of SQL injection attacks, minimizing false negatives and offering a more robust security solution.

Figure 17 illustrates the F1 performance of various SQLi detection algorithms, combining both precision and sensitivity to provide a balanced measure of detection effectiveness. The results show that the proposed BWOA + ANN model achieves the highest F1 score at 98.71%, confirming its strong and consistent classification capability. This is closely followed by Two-Class SVM (98.60%) and KNN (98.40%), both of which demonstrate high performance. Traditional algorithms such as Random Forest and Naïve Bayes produce noticeably lower F1 scores, dropping to 66.00% and 87.20%, respectively, indicating weaker overall detection reliability. The ANN, DT, and SVM models perform reasonably well, with F1 values above 92%. Overall, the results confirm that BWOA-enhanced learning significantly improves the balance between precision and recall, making it the most effective SQLi detection method among the compared machine learning techniques.

As shown in Table 9, the ANN classifier performed very well with all four feature selection methods. The whale optimizer achieved the best results, giving the highest accuracy (98.70%) and sensitivity (99.35%). The Gray Wolf and Bedbug optimizers

**Table 9** Predictive performance of the ANN classifier, along with different feature selectors

| | Whale optimizer | PSO optimizer | Gray Wolf optimizer | Bedbug optimizer |
|---|---|---|---|---|
| Accuracy | 98.70% | 97.40% | 98.37% | 98.50% |
| Sensitivity | 99.35% | 96.75 | 98.72% | 98.99% |

also performed well, with accuracy and sensitivity close to the top. PSO showed the lowest scores among the four methods but still performed well overall. These results show that the choice of feature selector affects ANN performance, and the whale optimizer gives the most effective feature set in this comparison.

### 4.3 Discussion

The results show that the proposed BWOA-based feature selection improves SQLi detection when combined with different ML algorithms. Figure 3 demonstrates that BWOA converges well and avoids local optima. Among the classifiers, BWOA + DT and BWOA + KNN achieved the lowest fitness values, indicating fewer errors and fewer selected features. The small variation in fitness values across runs shows that BWOA is stable and reliable. The accuracy of SQLi detectors increased with BWOA. The results obtained indicate that ANN's accuracy improved to 98.70%, and DT's to 97.40%. Across all ML methods, using BWOA reduced performance differences between runs. Sensitivity also improved with feature selection. For example, ANN's sensitivity increased to 99.35%, and DT's to 97.30%. This shows that BWOA effectively removes irrelevant or noisy features, reducing false negatives and enhancing the overall reliability of SQLi detection. The stability of the proposed method is apparent from the lower variance in performance metrics. Accuracy variation dropped from 2.74 to 1.03, and precision variation decreased from 4.42 to 0.86. This confirms that selecting the most essential features improves classifier performance and consistency.

The obtained results are not due to overfitting but to the efficient feature reduction achieved by the Binary Whale Optimization Algorithm (BWOA). The proposed method eliminates redundant and irrelevant features, reducing the feature set to only 31% of the original size. This reduction simplifies the model structure, reduces training time, and improves the detector's generalization capability. Although the accuracy values are high, the improvement (0.10% in accuracy) over the best baseline (Two-Class SVM) is still meaningful, and 0.11% in F1-score, while using a much smaller feature subset. This demonstrates that the proposed method achieves comparable or superior detection capability while requiring significantly less computational effort. Moreover, because the suggested model effectively learns from a limited number of informative features, it is highly suitable for data-limited applications where large-scale training data are unavailable.

All the ML algorithms used in our study (namely ANN, SVM, KNN, and Decision Tree) have been implemented in the same programming language (MATLAB) and have been executed on the same hardware and software platform, ensuring

consistency and fairness in the comparison. The BWOA was applied as a feature selector for all these ML classifiers under identical conditions, allowing a reliable evaluation of its effectiveness. Extending the comparison to include additional ensemble-based models would require implementation on the same platform and conducting a large number of additional experiments to maintain comparability. This represents a substantial increase in the scope of the experiment. Therefore, comparing the results of the suggested feature selector with those of other machine learning and deep learning algorithms is indeed valuable; we have proposed this as a future study. This approach will allow a rigorous and fair evaluation of the BWOA-based feature selection in combination with a broader range of classifiers, including deep learning methods.

Finally, while this study focused on classical ML algorithms, evaluating the BWOA with ensemble or deep learning models is suggested for future work. This would require implementing these algorithms on the same platform and running more experiments to ensure a fair comparison. Although the baseline models already achieved relatively high predictive performance (around 95%), the integration of the Whale Optimization Algorithm (WOA) was not solely aimed at improving accuracy, precision, and sensitivity. The primary motivation was to enhance training efficiency and model scalability by identifying and removing ineffective or redundant features. The proposed BWOA-based feature selector significantly reduced the feature space (retaining only 31% of the most informative features) while improving detection accuracy. This reduction in feature dimensionality not only reduces training time but also improves the model's generalizability to other datasets with limited samples or attributes. Therefore, WOA was essential in achieving a more efficient, compact, and robust SQLi detection framework, rather than simply boosting already high accuracy scores.

## 5 Conclusion

This study presented a feature selection-based method for identifying SQLi attacks using the Binary Whale Optimization Algorithm (BWOA). The proposed approach aims to remove ineffective features from the SQLi dataset, improving both the efficiency and performance of intrusion detection systems. Experimental results demonstrate that the proposed SQLi detection (BWOA + ANN) method achieves high performance (98.88% accuracy, 99.35% sensitivity, and a 98.83% F1-score) while utilizing only 31% of the most significant features. Reducing the number of features not only enhances computational efficiency but also contributes to more stable and reliable ML-based SQLi detectors.

While the results are promising, the study has several limitations. First, the evaluation was conducted using classical machine learning algorithms (ANN, DT, SVM, KNN) on a specific hardware and software platform. Extending the analysis to ensemble-based methods and deep learning models would require additional experiments to ensure fair comparisons. Second, the current study focused on a single heuristic algorithm as a feature selection method. Exploring other metaheuristic or hybrid approaches may reveal further improvements. Finally, the experiments were

performed on a particular SQLi dataset; assessing the technique on larger or more diverse datasets would strengthen the generalizability of the findings.

Several directions for future research are suggested. Integrating chaotic maps or other advanced optimization strategies into BWOA could enhance feature selection efficiency [16]. Effective SQLi detection models can be created using a combination of machine learning and heuristics. Additionally, combining BWOA with ensemble-based models may offer a balance between detection performance and computational cost. Overall, the study demonstrates the potential of combining heuristic feature selection with machine learning to develop robust, accurate, and efficient SQLi detection systems; the suggested feature selector can be extended for software fault prediction applications [20, 21].

**Author contributions** B. Arasteh proposed the research procedure. B. Arasteh and K. Arasteh implemented and coded the designed attack detection algorithm. B. Arasteh, K. Arasteh, and H. Kusetogullari benchmarked the implemented attack detector. B. Arasteh, M. Karimi, and H. Kusetogullari carried out the analysis of the data and findings. B. Arasteh and f. Kiani wrote the manuscript.

**Data availability** The related data to the current study are available on [22] and can be freely accessed on request. The data used in this research do not belong to any third parties or individuals and were produced by the researchers. The data can be accessible to other researchers.

## Declarations

**Competing interests** The authors declare no competing interests.

## References

1. Pillai S, Sharma A (2023) Hybrid unsupervised web-attack detection and classification—a deep learning approach. Comput Stand Interfaces 86:103738. https://doi.org/10.1016/j.csi.2023.103738
2. Bhattacharya T, Peddi AV, Ponaganti S et al (2025) A survey on various security protocols of edge computing. J Supercomput 81:310. https://doi.org/10.1007/s11227-024-06678-6
3. Ibarra-Fiallos S, Higuera JB, Intriago-Pazmino M, Higuera JRB, Montalvo JAS, Cubo J (2021) Effective filter for common injection attacks in online web applications. IEEE Access 9:10378–10391. https://doi.org/10.1109/ACCESS.2021.3050566
4. Hashemi Jouybari SM, Janbaz A, Esfandiari A (2025) A multivariate filter feature selection and stacking-based ensemble learning algorithm for network intrusion detection. J Supercomput 81:1129. https://doi.org/10.1007/s11227-025-07609-9
5. Chen Y, Liang G, Wang Q (2025) Research on SQL injection detection technology based on content matching and deep learning. Comput Mater Contin 84(1):1145–1167. https://doi.org/10.32604/cmc.2025.063319
6. Fathi KS, Barakat S, Rezk A (2025) An effective SQL injection detection model using LSTM for imbalanced datasets. Comput Secur 153:104391. https://doi.org/10.1016/j.cose.2025.104391
7. Crespo-Martínez IS, Campazas-Vega A, Guerrero-Higueras ÁM, Riego-DelCastillo V, Álvarez-Aparicio C, Fernández-Llamas C (2023) SQL injection attack detection in network flow data. Comput Secur 127:103093. https://doi.org/10.1016/j.cose.2023.103093
8. Kaya MO, Dagdogen HA, Ozdem M, Das R (2026) An effective new penetration test approach to detect web attacks on web applications. Expert Syst Appl 298:129623. https://doi.org/10.1016/j.eswa.2025.129623

9. Nguyen D-C, Ha M-H, Do M-T, Chen OT-C (2025) Towards lightweight model using non-local-based graph convolution neural network for SQL injection detection. Egypt Inform J 30:100684. https://doi.org/10.1016/j.eij.2025.100684
10. Paul A, Sharma V, Olukoya O (2024) SQL injection attack: detection, prioritization & prevention. J Inf Secur Appl 85:103871. https://doi.org/10.1016/j.jisa.2024.103871
11. Smrity TA, Muntaqim MZ (2025) SQLGuardNet: deep learning adaptive processing-based pattern recognition for enhanced SQL injection detection. Signal Image Video Process 19:1152. https://doi.org/10.1007/s11760-025-04642-2
12. Qin Q, Li Y, Mi Y et al (2025) SQL injection detection algorithm based on Bi-LSTM and integrated feature selection. J Supercomput 81:608. https://doi.org/10.1007/s11227-025-07109-w
13. Kakisim AG (2024) A deep learning approach based on multi-view consensus for SQL injection detection. Int J Inf Secur 23:1541–1556. https://doi.org/10.1007/s10207-023
14. Arasteh B, Aghaei B, Farzad B et al (2024) Detecting SQL injection attacks by binary gray wolf optimizer and machine learning algorithms. Neural Comput Appl 36:6771–6792. https://doi.org/10.1007/s00521-024-09429-z
15. Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67. https://doi.org/10.1016/j.advengsoft.2016.01.008
16. Arasteh B (2022) Clustered design-model generation from a program source code using chaos-based metaheuristic algorithms. Neural Comput Appl. https://doi.org/10.1007/s00521-022-07781-6
17. Gao H, Zhu J, Liu L, Xu J, Wu Y, Liu A (2019) Detecting SQL injection attacks using grammar pattern recognition and access behavior mining. In: Proceedings of the IEEE International Conference on Energy Internet (ICEI), Nanjing, China
18. Zhang K, Dataset AT (2019) A machine learning-based approach to identify SQL injection vulnerabilities. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA0, pp 2019–2021
19. Zhang H, Zhao J, Zhao B, Yan X, Yuan H, Li F (2019) SQL injection detection based on deep belief network. In: Proceedings of the CSAE 2019: 3rd International Conference on Computer Science and Application Engineering, Sanya, China
20. Arasteh B, Arasteh K, Ghaffari A et al (2024) A new binary chaos-based metaheuristic algorithm for software defect prediction. Cluster Comput 27:10093–10123. https://doi.org/10.1007/s10586-024-04486-4
21. Wang H, Arasteh B, Arasteh K, Gharehchopogh FS, Rouhi A (2024) A software defect prediction method using binary gray wolf optimizer and machine learning algorithms. Comput Electr Eng 118:109336. https://doi.org/10.1016/j.compeleceng.2024.109336
22. https://github.com/bahmanarasteh/SQLi-WOA.git




## Authors and Affiliations

**Bahman Arasteh[1,2,3] · Mohammadbagher Karimi[4] · Huseyin Kusetogullari[5,6] · Keyvan Arasteh[1] · Farzad Kiani[7]**

✉ Bahman Arasteh
Bahman.arasteh@istinye.edu.tr

[1] Department of Software Engineering, Faculty of Engineering and Natural Science, Istinye University, Istanbul, Turkey

[2] Department of Computer Science, Khazar University, Baku, Azerbaijan

[3] Applied Science Research Center, Applied Science Private University, Amman, Jordan

[4] Department of Software Development, Faculty of Computer and Information Technology, Cappadocia University, Nevsehir, Turkey

[5] Department of Computer Science, Blekinge Institute of Technology, 37179 Karlskrona, Sweden

[6] Artificial Intelligence Research Center (AIRC), College of Engineering and Information Technology, Ajman University, Ajman, UAE

[7] Data Science Application and Research Center (VEBIM), Fatih Sultan Mehmet Vakif University, Istanbul, Turkey