informatics

MDPI

Article

Detecting Structured Query Language Injections in Web Microservices Using Machine Learning

Edwin Peralta-Garcia, Juan Quevedo-Monsalbe, Victor Tuesta-Monteza and Juan Arcila-Diaz *

Escuela de Ingeniería de Sistemas, Universidad Señor de Sipán, Chiclayo 14000, Peru; pgarciaedwinjah@uss.edu.pe (E.P.-G.); qmonsalvejuanca@uss.edu.pe (J.Q.-M.); vtuesta@uss.edu.pe (V.T.-M.)
* Correspondence: diarcilaju@uss.edu.pe

Abstract: Structured Query Language (SQL) injections pose a constant threat to web services, highlighting the need for efficient detection to address this vulnerability. This study compares machine learning algorithms for detecting SQL injections in web microservices trained using a public dataset of 22,764 records. Additionally, a software architecture based on the microservices approach was implemented, in which trained models and the web application were deployed to validate requests and detect attacks. A literature review was conducted to identify types of SQL injections and machine learning algorithms. The results of random forest, decision tree, and support vector machine were compared for detecting SQL injections. The findings show that random forest outperforms with a precision and accuracy of 99%, a recall of 97%, and an F1 score of 98%. In contrast, decision tree achieved a precision of 92%, a recall of 86%, and an F1 score of 97%. Support Vector Machine (SVM) presented an accuracy, precision, and F1 score of 98%, with a recall of 97%.

Keywords: SQL injection; machine learning; web applications; microservices; detection

Citation: Peralta-Garcia, E.; Quevedo-Monsalbe, J.; Tuesta-Monteza, V.; Arcila-Diaz, J. Detecting Structured Query Language Injections in Web Microservices Using Machine Learning. *Informatics* **2024**, *11*, 15. https://doi.org/10.3390/informatics11020015

Academic Editors: Jiang Bian and Olga Kurasova

Received: 15 January 2024
Revised: 20 March 2024
Accepted: 21 March 2024
Published: 2 April 2024

1. Introduction

Microservices is a software development style that has gained importance in recent years. It is based on decomposing an application into independent services, each responsible for a specific function, providing scalability, flexibility, and ease of implementation [1]; however, it presents challenges in terms of data management, particularly in databases that contain the important and confidential information of an organization, facing threats such as SQL injections (SQLI) [2]. This type of attack, which occurs in various organizations, is caused by users who intend to obtain sensitive information related to the database. They manipulate the data, generating harmful effects that affect organizations worldwide [3]. Preventing these types of attacks has become a priority for organizations and software development industries. Early and accurate detection is crucial to avoid harmful effects [4,5]. Historically, preventing these types of attacks involved validating data entry and examining it for special characters associated with common attacks. While this practice still persists today, it is no longer effective in countering other types of attacks [6]. Due to constant technological advancements, organizations and developers are actively seeking new solutions to address this problem, which continues to emerge.

The aim of this study is to compare machine learning algorithms for detecting SQL injections in web microservices using public SQLI datasets. The effectiveness of the machine learning algorithms in detecting this type of vulnerability in web microservices will be evaluated. The research question that arises from this is: what is the most effective machine learning algorithm for detecting SQLI in web microservices?

Machine learning-based tools for detecting SQLI attacks are impacting software development industries and organizations. Various machine learning methods have been developed to detect SQLI. In the study conducted by [7], security auditing tools were utilized to detect SQL injections (SQLIs) and identify common patterns and attack vectors.

The collected data were used to develop two classifier systems: one based on naive Bayes (NB) and another on decision tree (DT). The results showed that the NB classifier achieved a performance of 97% with an accuracy of 98%. In reference [8], a text vectorization model was employed to detect SQLI. The model employed an algorithm called ITFIDF and was improved by using support vector machine (SVM), K-nearest neighbors (KNNs), and DT. The results of the first algorithm showed a precision of 99.08% and a recall of 99%, while the results show that the combination of term frequency–inverse document frequency (TF-IDF) and support vector machines (SVMs) yields the best results, with K-nearest neighbors (KNNs) and decision trees (DTs) achieving slightly lower scores. Specifically, TF-IDF + SVM achieved a precision of 98.22%, a recall of 98.12%, and an F score of 98.17%, while KNN achieved a precision of 98.18%, a recall of 97.98%, and an F1 score of 98.08%; the overall accuracy was 34% with an F score of 99.21%. In reference [9], authentic SQLI data were collected at the enterprise level and combined with an equal number of normal queries, these data were used to train a neural network based on multi-layer perceptron (MLP) and long short term memory (LSTM). The results indicate that the MLP model with a hidden layer is the most optimal, achieving a precision of 96.24%, a recall of 100%, an accuracy of 99.67%, and a false positive rate of 0.36%; the processing time was 84.1 milliseconds. The study [10] constructed a dataset by extracting labeled samples from the web and incorporating an sql-injection-payloadlist. A Keras sequential MLP neural network model was trained using this dataset and designed to exploit the WHERE clause in databases. This approach yielded remarkable results, with a precision of 85%, a recall of 100%, an F1 score of 92%, and an accuracy of 94.4%. In the work [11], researchers collected malicious SQL queries using approaches such as bag-of-words (BOW) and word2vec, queries that were used as a training set for multi-layer perceptron (MLP) models, convolutional neural networks (CNNs), and long short-term memory (LSTM); when using BOW, the precision levels achieved were 91.0%, 95.4%, and 91.9%, with word2vec, the percentages were 76.3%, 76.7%, and 81.6%, respectively. In reference [12], Ku-bAnomaly, a system that utilizes neural network and supervised learning techniques, was developed to detect multiple anomalies in websites; the results showed an exceptional accuracy rate of 99.6%, a recall rate of 99.6%, and an F1 score of 99.6% for SQL injections (SQLI).

The present research chose the machine learning approach due to its demonstrated capacity to address complex database attack problems. Different machine learning techniques were implemented to train a model capable of accurately detecting SQLI. The aim is to provide a reliable and effective algorithm for the software development industry.

2. Data and Materials

2.1. Characterization of SQLI Attacks

The purpose of characterizing SQLI attacks on databases was initially to understand their types and execution methods. Eight types of SQL injections were identified as being widely used by attackers. Subsequently, the specific characteristics of each type of attack were analyzed to obtain a greater understanding of its structure and application. Table 1 presents eight injection types, their definitions, normal and abnormal queries, injectable parameters, and associated descriptions.

Table 1. Properties of SQLI Types.

Attack Type	Definition	Normal Consultation	Injectable Parameter	Abnormal Query	Description
Tautology SQLI	This type of attack attempts to use a conditional question argument to test the validity of the SQL query, this is achieved by using the WHERE clause where the attacker injects the condition and transforms it into a tautology that is always valid [13,14].	SELECT * FROM users WHERE name = 'user' AND password = 'password';	OR '1' = '1';	SELECT * FROM users WHERE name = 'user' AND password = 'password' OR '1' = '1';	It becomes an anomalous query by adding the statement 'OR '1' = '1';' to the SQL query. This causes the table fields to always evaluate to true, resulting in data being returned from the table, regardless of the actual conditions [14,15].

Table 1. *Cont.*

Attack Type	Definition	Normal Consultation	Injectable Parameter	Abnormal Query	Description
Ilegal/Logically incorrect query SQLI	Injection of this type occurs due to errors, lack of validations or the inputs are invalid from a logical point of view, since during the development period it helps programmers to correct their programs [16].	SELECT * FROM products WHERE product_name = 'name';	; SELECT * FROM users; --';	SELECT * FROM products WHERE product_name = '; SELECT * FROM users; --';	This injection attempts to execute two SQL queries at the same time, the first closes prematurely and the second selects all records in the users table, potentially revealing sensitive information. The ";" indicates the completion of a query in SQL. The "--" initiates a comment, which means that everything after it in the line is considered descriptive text.
Union query SQLI	The injection of this type consists mainly of the word UNION, which simulates a single query consisting of the SELECT clause. For this query to be applied, the tables must be related and the correct names of the tables along with their fields should be known [13,17].	SELECT * FROM accountTable WHERE user_login = 'user';	UNION	SELECT * FROM accountTable WHERE user_login = 'user' UNION SELECT * FROM accountTable WHERE No = 10,232	The query can point to an SQL injection from a login because it attempts to join two tables with a single query, thus attempting to return all users who have the queried username or the value 10,232 in the query column [13].
Piggy-Backed Query SQLI	This type of injection executes two requests simultaneously, the first should be correct and the second should be controlled for data extraction, addition, modification, remote execution, or denial of service [16,18].	SELECT book_name FROM booktable WHERE book_id = 'book1' AND book_name = 0	;	SELECT book_name FROM booktable WHERE book_id = 'book1' AND book_name = 0; DROP booktable;	This query is characterized by the fact that a real query, which is executed normally, is added with malicious queries [17].
Blind SQLI	It consists of the formation of a series of true/false queries, thus collecting valuable data by inferring from the responses of the consulted website [13,18]	SELECT 5 WHERE username = :username	1/0 ELSE SELECT 5	'admin'; IF SYSTEM_USER = 'sa' SELECT 1/0 ELSE SELECT 5	The query consists of two parts, before and after the semicolon, if the value is correct it will return an error message or a shortcut, but if the answer is not obtained one will arrive at '5' as the answer.
Timing SQLI	This injection manages response times in which the attacker records the responses generated by the database, based on the incoming and outgoing data transfer technique [16].	SELECT price FROM products WHERE id = 1;	or 1 = 1 --'; sleep (1)	SELECT price FROM products WHERE id = ' or 1 = 1 --'; sleep (1);	In this type of injection, the success of the attack is evaluated by entering an input, where the first statement corresponds to the ID. If the injection is successful, the response time will be observed within the period programmed in the malicious query.
Store Procedure SQLI	A type of database vulnerability that allows an attacker to inject malicious code into a cached SQL query. This type of attack can lead to the execution of arbitrary commands in the database, which can compromise the entire database and its records [13,14].	SELECT * FROM accountTable WHERE user = 'user' AND passwd = 'pass';	SHUTDOWN;--;	SELECT * FROM accountTable WHERE user login = 'user' AND passwd = 'pass'; SHUTDOWN;--;	The query has the last part that is considered injectable, this being a forced way of stopping or shutting down the database management system, plus the last signs indicate that everything after that is considered a query.
Alternate Encoding SQLI	This type of injection is applied to bypass special character validations, thus using alternative encodings such as hexadecimal, ASCII or Unicode [18].	SELECT * FROM accountTable WHERE user = 'user' AND pin = 'pass';	44524f50204 44154414241 534520	SELECT * FROM accountTable WHERE user = 'user' AND pin = 'pass'; EXEC('44524f50204 44154414241534520'); SHUTDOWN;	This injection seeks to hide certain characters or reserved words from the databases in order to make them pass the validations and be executed, in this case there is an encryption based on HEXADECIMAL.

The "*" in a SELECT clause indicates that all columns in the table specified in the query will be selected. A ":username" parameter is referenced. The ":" indicates a placeholder for a value to be provided at query execution time.

2.2. Dataset

In order to implement the machine learning algorithms, it was crucial to identify the appropriate dataset. A search was conducted on various sources, including Kaggle and Github, resulting in the identification of approximately 30 datasets. The most suitable dataset was then selected, taking into consideration the following criteria:

- To effectively train machine learning algorithms aimed at detecting SQL injections, research suggests that a dataset ranging between 20,000 and 40,000 registers is required [9,11,19];
- The dataset must come from a reliable source and be found in known repositories;

- Precise labeling and classification are necessary, with values assigned to indicate whether a query is abnormal or normal;
- Normal data should also be included for better training;
- The selected dataset must be legally permitted for use in research.

Thus, we selected a dataset that meets the aforementioned criteria, specifically the 'SQL Injection Detection by Machine Learning' dataset [20], for this study.

To ensure a fair ratio between abnormal and normal queries, we performed balancing through subsampling, which involves removing data. The original dataset contained 30,905 records, with 19,537 anomalous and 11,382 normal queries. To enhance the training process of machine learning algorithms, subsampling was performed, resulting in a final dataset of 22,764 records, with 11,382 data points for both anomalous and normal queries. This procedure ensures more robust and fair results during algorithm training.

2.3. Algorithm Selection

To select the machine learning algorithms, a systematic literature review was performed following the guidelines of the international PRISMA statement. For the selection of scientific and academic articles addressing machine learning uses for SQLI detection, we chose to use the Web of Science (WOS), Scopus, and IEEE Xplore databases. A total of 320 articles related to the topic were extracted, but after applying inclusion and exclusion criteria, 29 relevant articles were identified that provided substantial information on machine learning algorithms used to detect SQL injections. Subsequently, the algorithms with the best performance were selected based on precision, accuracy, recall, and F1 score criteria. The decision tree, SVM, and random forest algorithms were found to have the best results. Table 2 provides further details on the criteria implemented by various authors.

Table 2. Top 10 algorithms with the best performance.

Algorithm	Accuracy	Precision	Recall	F1-Score
Light Gradient Boosting Machine (LGBM) [17]	0.9933	0.9933	0.9933	0.9933
Gradient Boosting Machine (GBM) [17]	0.9904	0.9898	0.9903	0.991
Artificial Neural Network (ANN) [13,18]	0.9893	0.9870	0.9913	0.99
AdaBoost (AB) [17,21]	0.9808	0.9559	0.9592	0.9561
Decision Tree (DT) [16,18,22,23]	0.9668	0.9315	0.88955	0.9164
Random Forest (RF) [18,22,23]	0.9634	0.9247	0.8947	0.9149
Support Vector Machine (SVM) [18,22,23]	0.9546	0.9706	0.9085	0.9395
Logistic Regression (LR) [4]	0.9503	0.9737	0.9089	0.9653
Naive Bayes (NB) [18,24]	0.9074	0.8966	0.7985	0.9010
KNN (K-Nearest Neighbors) [21]	0.8920	0.9143	0.8931	0.8853

Furthermore, the choice of these algorithms is justified for the following reasons. The decision tree (DT) algorithm is simple to interpret and allows for the identification of characteristics relevant to the detection of SQL injections. Random forest (RF), being an ensemble of decision trees, offers robustness and resistance to overfitting by using multiple models to make predictions, making it easier to detect injections, and it can also handle unbalanced datasets more effectively. Finally, support vector machines (SVMs) are effective in high-dimensional feature spaces and are less prone to overfitting compared to other machine learning algorithms. In the context of SQL injection detection, SVMs can separate legitimate SQL queries from malicious ones in a feature space defined by relevant characteristics.

Similarly, when selecting the DT, RF, and SVM algorithms, it was deemed essential to evaluate their applicability in scientific research. The systematic review revealed that the DT algorithm was used in 9 studies, SVM in 10, and RF in 9, indicating their relevance and adoption in the scientific and industry fields.

3. Method

After analyzing SQL injections to identify their various types, we selected a dataset that included both normal and anomalous injections. Next, we implemented three machine learning algorithms known for their high performance. These algorithms were trained using the selected dataset to detect SQL injections in requests directed to an application developed under the microservices approach.

3.1. Machine Learning for SQLI Detection

The Google Collaboratory work environment implemented SVM, decision tree, and random forest algorithms using the Python programming language.

3.1.1. Support Vector Machine

The Support Vector Machine (SVM) algorithm is a type of supervised learning used for data classification. Its objective is to identify an optimal hyperplane that maximizes the separation between classes [25]. In order to detect SQL injections, the algorithm identifies SQLI patterns in the input data, learning to distinguish between normal and abnormal queries.

The objective of the SVM algorithm is to maximize the margin by minimizing classification errors. The margin is defined as the distance between the decision boundary (hyperplane) and the closest data points of each class [25]. Additionally, support vectors have a significant impact on the classification accuracy of the SVM. They are the points that are closest to the hyperplane limit and determine the position and orientation of the decision.

The algorithm procedure spans from the initial data preparation phase to the prediction stage. Initially, load the set of SQL queries (dataset) and assign labels, where the value '1' represents an abnormal query, while '0' indicates a normal query. This is represented in class 1 and 2, respectively, as shown in Figure 1. Next, preprocess the data by eliminating rows that contain null values. Subsequently, the data are divided using the 80/20 Pareto rule, allocating 80% for algorithm training and 20% for testing. The SVM algorithm utilizes 8-bit unicode transformation format (UTF-8) encoding to handle natural language texts that may include special characters. To ensure the integrity of the dataset, the dropna() function was used during data processing to remove rows with missing or null values. The algorithm utilizes the TF-IDF technique for vectorization, which assigns weights to words based on their importance in the text content. This approach enables a weighted count of words in queries, facilitating the determination of an SQL injection. Additionally, a linear kernel is used to classify queries as an SQL injection or not, without requiring the non-linear separation of classes.

3.1.2. Decision Tree

The decision tree algorithm is a supervised learning model used in the field of artificial intelligence and machine learning. Its structure is reflected in a sequence of decisions and their respective branches [26]. The setting can be adjusted depending on the desired depth for the task or the optimal scan level. The decision tree consists of a root node from which child nodes emerge, making up the branches, and leaves that contain the final predictions.

In the context of detecting SQL injection, the algorithm processes labeled data from the root node and uses information gain to generate additional branches (see Figure 2). Each branch divides the dataset into smaller subsets, creating child nodes. Feature selection is performed after each split, and the process is repeated iteratively until a stopping criterion is reached, such as the maximum depth of the tree or information gain that does not exceed a predetermined threshold. In addition, the algorithm generates terminal nodes (leaves) that represent final predictions. These predictions are either the most common class in the subset for classification problems or the mean for regression problems. This approach is notable for its capacity to adapt to new data and its interpretability, which is essential in security applications such as SQL injection detection. The decision tree architecture is modified by

adjusting its depth. The shapes of the branches and the creation of leaves are determined internally based on whether the processed data are recognized as SQL injections. The algorithm uses UTF-8 encoding to ensure the integrity of special character handling. The dropna() function is used to discard rows with null values, thus maintaining the quality of the data used to train the model. Regarding text processing, the natural language toolkit (NLTK) library was used to apply lemmatization and tokenization techniques. This resulted in effective normalization and cleaning of SQL queries, which improved the accuracy of the model in detecting SQL injections. The pre-processed text was transformed into numerical vectors using the CountVectorizer method, making it interpretable by the machine learning algorithm. This facilitated the representation of the text characteristics and their use in the decision tree model. Additionally, the model underwent specific adjustments, including setting a maximum tree depth of 30 (max_depth = 30) and utilizing the ccp_alpha parameter to prevent overfitting by penalizing and pruning the tree. These adjustments facilitated effective generalization of the model on unseen data and improved its predictive capacity.

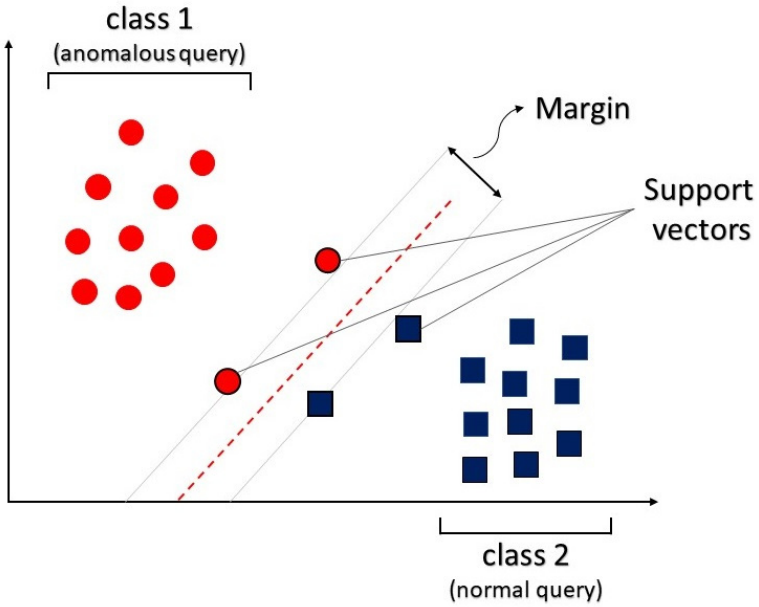

Figure 1. SVM architecture.

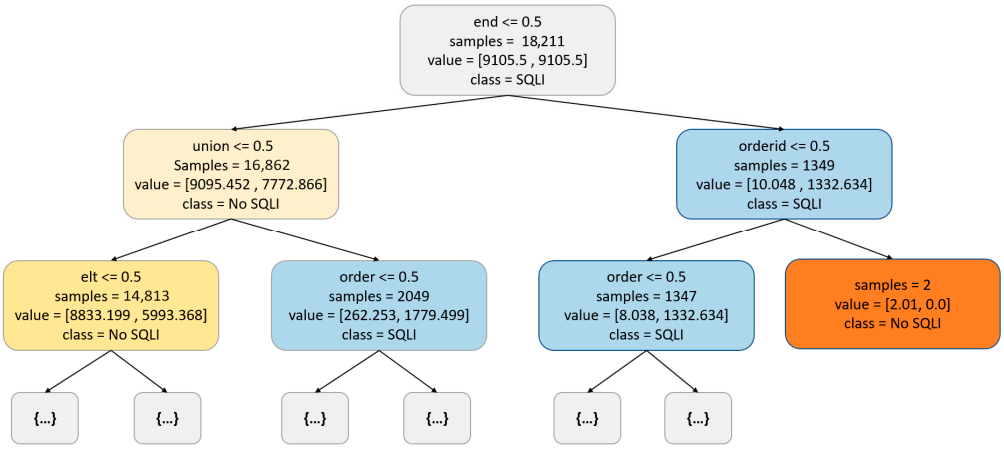

Figure 2. Algorithm architecture of decision tree.

3.1.3. Random Forest

The random forest algorithm is a type of supervised learning that constructs multiple decision trees and combines their results to improve precision and mitigate overfitting.

This algorithm creates a set of decision trees, each trained on a random subsample of the training dataset [27]. In SQL injection detection, the random forest algorithm captures more complex patterns and improves model generalization (Figure 3).

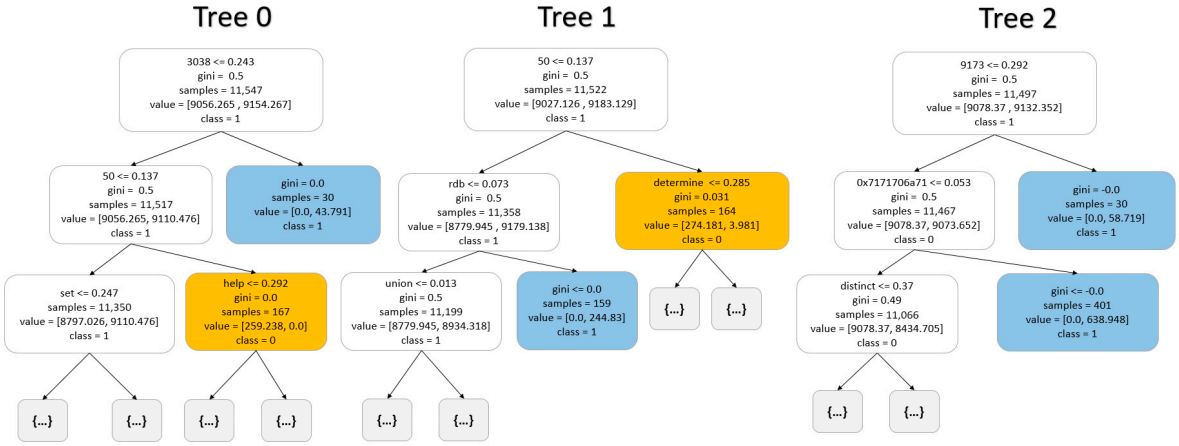

Figure 3. Algorithm architecture of random forest.

The process starts with data preparation, which involves removing null values, dividing the data according to Pareto's law (80/20), and vectorizing the queries using TF-IDF. The algorithm is trained using specific criteria, such as the number of estimators (n_estimators), the maximum tree depth (max_depth), the weight of the classes (class_weight), and the random state (random_state).

3.1.4. Performance Evaluations

SQLI detection can be classified as either true positive (TP) or true negative (TN) when accurate detections are made in attacks. Conversely, it is classified as false positive (FP) or false negative (FN) when detections are incorrect.

Precision (P) is a metric used to measure the quality of predictions, minimizing false positives and maximizing the number of correctly classified true positives. It can be calculated using Formula (1).

$$P = \frac{TP}{TP + FP} \tag{1}$$

Recall (R): Assesses the classification accuracy of all elements within a given class.

$$\boldsymbol{R = \frac{TP}{TP + FN}} \tag{2}$$

F1 Score (F): This indicator provides a balance between precision and recall, allowing for a better comparison of combined performance.

$$F = 2 * \frac{(P * R)}{(P + R)} \tag{3}$$

Accuracy (Acc): Evaluate the prediction that the algorithm makes correctly and have an accurate classification.

$$\boldsymbol{Acc = \frac{TP + TN}{TP + TN + FP + FN}} \tag{4}$$

3.2. Microservices-Based Software Architecture

The system developed for the detection of SQL injection attacks is based on a software architecture based on microservices, an approach recognized for its modular and distributed structure, made up of multiple interrelated and scalable components. The software architecture developed in this research is composed of three independent services

with specific functionalities that communicate with each other through a well-defined application programming interface (API).

One service was developed with a front-end application that exposed the necessary interface for the user to make requests in the information system. This application was developed using Angular, a framework based on TypeScript, which facilitates user interaction and sends requests to the service where the business logic is located.

Another service that makes up the software architecture contains the business logic; this service was developed using the Spring Framework in Java. This component, which exposes its communication API, also acts as an intermediary between the user interface and the SQLI recognition service. Its main task is to manage incoming requests, validate them and forward them to the appropriate service for processing. The MySQL database management system has been used to store the information relevant to the operation of the application, and it is the component that receives the SQL query if it is not considered a SQLI attack.

To handle SQL injections, a dedicated service was created to host machine learning models trained for SQLI detection. This service, developed in Python, exposes an API developed with the FastAPI framework that allows interaction with models based on the SVM, decision tree, and random forest algorithms. These models have the capacity to analyze the received queries and determine if they exhibit patterns associated with SQL injection attacks.

Figure 4 shows the workflow diagram of the application based on the microservices architecture. The process begins when a user submits a request through the web interface, which is received by the business logic for initial validation. The request is then sent to the SQLI detection service, where the machine learning model is applied to analyze its content. If a SQL injection attack is detected, the user receives a notification that a possible SQLI has been detected; otherwise, the request follows its normal processing flow, is sent to the database, and the response is processed and presented to the user through the user interface.

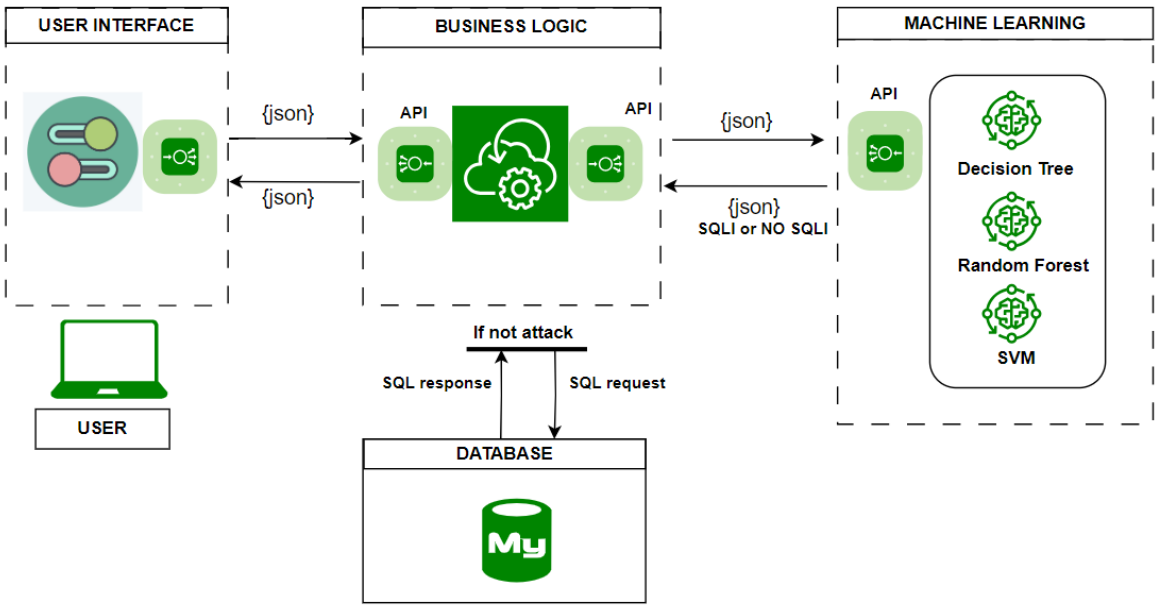

Figure 4. Application architecture based on microservices.

4. Results

4.1. Training Time

To execute the algorithms, Google Collaboratory service was utilized and worked in an environment with two Intel(R) Xeon(R) 2.20 GHz processors, 12 Gi of RAM.

Table 3 shows the training time (in seconds) and the RAM memory usage of the algorithms used, random forest, SVM, and decision tree, in the execution environment. The times varied depending on the specific structure and architecture of each algorithm. In the case of random forest, it uses a set of 200 decision trees with 15 branches each, achieving an estimated training time of 24 s. On the other hand, the SVM model requires a longer training time, reaching 33 s, due to the complexity of its architecture and the processing necessary to address the injections. Similarly, decision tree, an algorithm composed of a single tree to generate predictions, has a training time of 6 s.

Table 3. Training time and RAM memory usage of the evaluated machine learning algorithms.

Machine Learning Algorithms	Training Time (s)	RAM Memory Usage
Random forest	24	1.5 Gi
SVM	33	1.7 Gi
Decision tree	6	1.4 Gi

4.2. Performance

To train the SQL injection detection algorithms, a dataset of 22,764 records was used, with 11,382 records of normal queries and the same number of anomalous queries. A strategy based on the Pareto law was used, where 80% of the data was used for training and 20%, or 4553 records, were used for testing.

Figure 5 shows the confusion matrix of the SVM algorithm for SQL injection detection, which shows that there were 2298 cases of true positives (TP), 2176 cases of true negatives (TN), 57 cases of false positives (FP), and 22 cases of false negatives (FN). In comparison, the false negative rate was 0.48%, meaning that 0.48% of cases were incorrectly identified as negative.

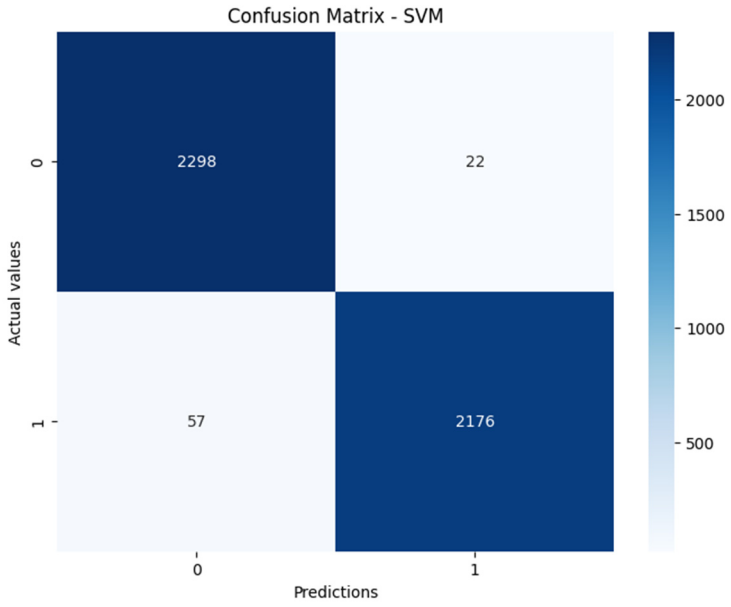

Figure 5. SVM confusion matrix.

The confusion matrix of the random forest algorithm training is shown in Figure 6; it can be seen that 2313 cases of true positives (TP), 2168 cases of true negatives (TN), 65 cases of false positives (FP) and 7 cases of false negatives (FN) were identified. In comparison, the false negative rate was 0.15%, which would be the percentage of cases that were incorrectly identified as negative.

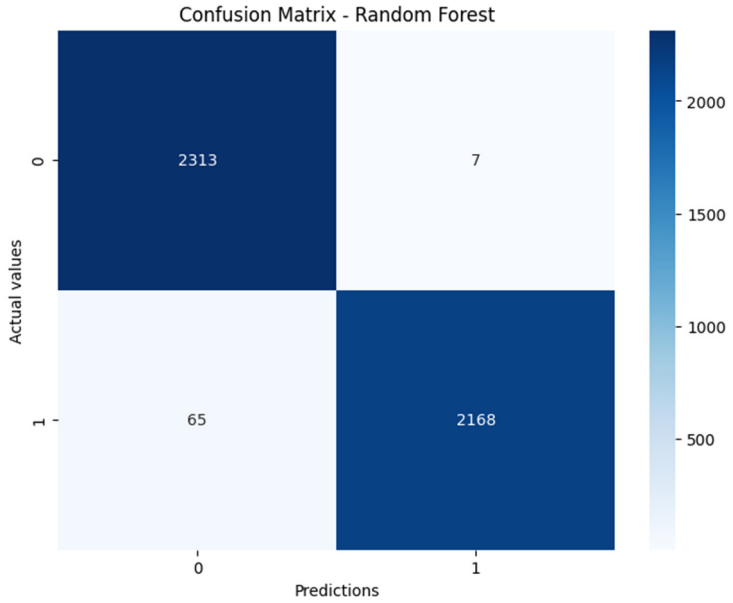

Figure 6. Random forest confusion matrix.

Figure 7 shows the confusion matrix of the decision tree algorithm for detecting SQL injections, which shows that there were 2276 true positives (TP), 1932 true negatives (TN), 301 false positives (FP), and 44 false negatives (FN). In comparison, the false negative rate was 0.48%, indicating that 0.97% of cases were falsely identified as negative.

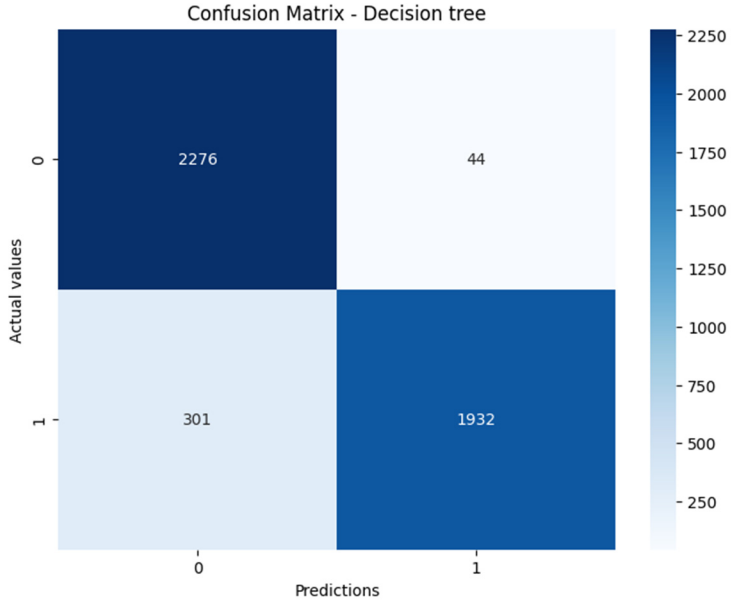

Figure 7. Decision tree confusion matrix.

Table 4 shows the performance indicators of the three algorithms selected during the training phase to detect SQLI in the web microservice. In this context, the algorithms were trained using different tokenization and vectorization models, which allowed to effectively process the chains that make up both injections and non-injections. The results show that the random forest algorithm stands out for its excellent performance in the detection of SQLI, achieving a precision and accuracy of 99%, a recall of 97%, and an F1 score of 98%. Similarly, the SVM algorithm shows excellent performance with a precision, accuracy, and F1 score of 98% and a recall of 97%. Conversely, the decision tree algorithm does not show

optimal results for the detection of SQLI, with a precision of 97%, a recall of 86%, an F1 score of 91%, and an accuracy of 92%.

Table 4. Performance of evaluated machine learning algorithms.

Algorithm	Accuracy	Precision	Recall	F1-Score
Random Forest	99%	97%	98%	99%
SVM	98%	97%	98%	98%
Decision tree	97%	86%	91%	92%

5. Discussion

The results obtained in this study, which focuses on the comparison of machine learning algorithms for the detection of different types of SQL injections in web microservices using a set of data classified as injections and non-injections, have significant implications for the field of cybersecurity. It is important to clarify that, unlike previous studies that report on the detection of SQLI in web applications based on monolithic architectures, the present research focuses on the specific context of web microservices [28], where the existing literature has not yet delved deeper. The software architecture based on the microservices approach is composed of three independent services with specific functionalities: a service that provides a user interface to make SQL requests, which can be normal requests or attacks; a service that receives these requests and sends them to the processing service that evaluates the request using models trained with three machine learning algorithms (SVM, RF, and DT).

In the SVM and RF algorithms, vectorization using TF-IDF was used to detect anomalous patterns and handle high query loads, taking advantage of the versatility of the vectorizer. However, in the third algorithm, the vectorizer CountVectorizer was used, a choice that differs from the approaches presented in previous studies [22]. CountVectorizer generates a term count matrix, where each row stores a term uniquely extracted from the dataset, which facilitates the effective identification of SQL injections. Furthermore, it is crucial to highlight that gradient boosting (GB) [29] was implemented in the RF algorithm; this strategy has not been addressed in previous studies; it is used to improve the performance of the model and achieve a more accurate detection of SQL injections.

However, some limitations were identified during the development of this study. In particular, the lack of detailed information on attacks on web microservices was highlighted, an architecture that companies are currently evaluating and considering to improve their information systems due to its benefits in terms of redundancy, scalability and flexibility, this may be because currently this type of attack is mitigated since the data access layers of web systems are developed using object-relational mapper (ORM) frameworks. However, the databases work by receiving SQL requests as well, so that it is still possible to carry out these types of attacks. In addition, it has been found that the processing of SQL and SQLI queries presents significant challenges due to the composition of these queries, which involve aspects of natural language processing (NLP), special characters, signs, letters, and other elements. The complexity inherent in these elements has added difficulties to the efficient handling of queries and, therefore, represents an important consideration for future research and improvements in the implementation of solutions. So, we propose the use of parameterized queries and the development of more advanced techniques to handle the inherent complexity of these queries.

The evaluated metrics, including precision, recall, F1 score, and accuracy, reflect encouraging results in terms of effectiveness in detecting SQLI in web microservices. Among the machine learning models applied, the analysis shows that the random forest algorithm, with an average performance of 98% based on the evaluated metrics, positions itself as the most feasible and efficient option for the detection of SQLI in web microservices, whose use could represent a significant contribution to the field of cybersecurity. The random forest algorithm has proven to obtain superior metrics thanks to the characteristics and structure it has for classifying injections, allowing specific patterns to be identified and

accurate predictions to be made, consolidating itself as a key tool in the prevention and detection of security threats in web microservices environments.

Comparing the results with other studies, it can be observed that they focus on the detection of SQLI in web applications based on monolithic architectures. In this context, the present research stands out by applying machine learning algorithms specifically to web applications based on architectures composed of independent services, an approach known as microservices. Thus, in the research of [30], it is observed that conversions of input data (dataset) were performed, converting them into numerical values and data verification was performed to avoid empty columns or rows in the training, reaching six algorithms, among which linear regression and perceptron + SGD stand out, obtaining a precision greater than 96%, results somewhat similar to the values obtained in the models used in the present study. On the other hand, in [31], a model for the detection of SQL injection attacks based on semantic learning and deep learning is proposed and applied using natural language processing NLP techniques (TF-IDF and Word2Vec); techniques also used here. The results obtained in this study show that the synBERT model reached 99.74% accuracy, 99.68% precision, a recall of 99.52%, and an F1 score of 99.60%. On the other hand, in [32], the RNN automatic encoder is proposed, which manages to achieve a precision of 95%, an accuracy of 94%, a recall of 90%, and an F1 score of 92%.

The results obtained suggest that machine learning algorithms could be highly effective in detecting SQL injections in applications based on web microservices. This perspective aims to significantly improve the level of security by enabling early detection of possible computer attacks. However, it is important to mention that the possibility of applying algorithms in web microservices is challenging due to the complexity of the architecture and data request components. The demonstrated effectiveness of the random forest algorithm can make a significant contribution to the field of cybersecurity, opening opportunities for future research and development in the detection of SQLI in web microservice environments.

6. Conclusions

The characterization of the types of SQL attacks has made it possible to identify the eight types most commonly used by attackers. This analysis has provided a detailed understanding of their definitions and the structure of the anomalous queries corresponding to each type, which in turn has allowed specific defensive measures to be taken against these attacks.

The selection of the dataset from recognized public repositories, such as Kaggle and Github, and the application of subsampling for data balancing, has allowed us to obtain an optimized dataset to strengthen the training and execution of the machine learning algorithms for detection of SQLI in web microservices.

For the selection of the machine learning algorithms, a systematic review of the literature was carried out; the result was that the decision tree, SVM and random forest algorithms presented better results considering the precision criteria: accuracy, recall and F1 score.

The entire software architecture was developed based on the microservices approach, considering three services: a service that exposes the user interface to capture SQL requests, a service that receives web requests and forwards them to the evaluation of the request to determine if it is an attack. In addition, the MySQL database management system has been used to store the relevant information for the operation of the application, and this is the component that receives the SQL query if it is not considered a SQLI attack.

To train the selected algorithms, the process was carried out using Python in the Google Collaboratory virtual environment. The dataset consisted of 22,764 records, with 11,382 records of normal queries and the same number of anomalous queries. Pareto's law was applied to divide the dataset into 80% for training and 20% for testing.

The analysis of the results confirms that the algorithm with the best performance to detect SQLI in web microservices is the random forest algorithm with a precision and accuracy of 99%, a recall of 97%, and an F1 score of 98%.

Author Contributions: Conceptualization, J.Q.-M. and E.P.-G.; methodology, J.A.-D.; software, J.Q.-M. and E.P.-G.; validation, J.Q.-M. and E.P.-G.; formal analysis, J.A.-D.; investigation, J.Q.-M., E.P.-G., V.T.-M. and J.A.-D.; data curation, J.Q.-M. and E.P.-G.; writing—original draft preparation, J.Q.-M., E.P.-G., V.T.-M. and J.A.-D.; writing—review and editing, J.A.-D.; supervision, V.T.-M. and J.A.-D. All authors have read and agreed to the published version of the manuscript.

Funding: The APC was funded by Universidad Señor de Sipán (Perú).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data may be provided free of charge to interested readers by requesting the correspondence author's email.

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

References

1. Arcila-Diaz, J.C.; Valdivia, C. A microservice-based software architecture for improving the availability of dental health records. *Int. J. Comput.* **2022**, *21*, 475–481. [CrossRef]
2. Mocean, L.; Vlad, M.-P. Database security in RDF terms. *Sci. Bull.* **2023**, *28*, 55–65. [CrossRef]
3. Wang, Y.-C.; Zhang, G.-L.; Zhang, Y.-L. Analysis of SQL injection based on Petri net in wireless network. *J. Inf. Sci. Eng.* **2023**, *39*, 167–181.
4. Shagari, S.M.; Gabi, D.; Dankolo, N.M.; Gana, N.N. Countermeasure to structured query language injection attack for web applications using hybrid logistic regression technique. *J. Niger. Soc. Phys. Sci.* **2022**, *4*, 832. [CrossRef]
5. Furhad, M.H.; Chakrabortty, R.K.; Ryan, M.J.; Uddin, J.; Sarker, I.H. A hybrid framework for detecting structured query language injection attacks in web-based applications. *Int. J. Electr. Comput. Eng.* **2022**, *12*, 5405–5414. [CrossRef]
6. Marashdih, A.W.; Zaaba, Z.F.; Suwais, K. Predicting input validation vulnerabilities based on minimal SSA features and machine learning. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 9311–9331. [CrossRef]
7. Lodeiro-Santiago, M.; Caballero-Gil, C.; Caballero-Gil, P. Collaborative SQL-injections detection system with machine learning. In Proceedings of the 1st International Conference on Internet of Things and Machine Learning (IML '17), New York, NY, USA, 17–18 October 2017; pp. 1–5.
8. Li, Y.; Zhang, B. Detection of SQL injection attacks based on improved TFIDF algorithm. *J. Phys. Conf. Ser.* **2019**, *1395*, 012013. [CrossRef]
9. Tang, P.; Qiu, W.; Huang, Z.; Lian, H.; Liu, G. Detection of SQL injection based on artificial neural network. *Knowl. Based Syst.* **2020**, *190*, 105528. [CrossRef]
10. Begum, A.M.; Arock, M. Efficient detection Of SQL injection attack(SQLIA) using pattern-based neural network model. In Proceedings of the IEEE 2021 International Conference on Computing, Communication, and Intelligent Systems, ICCCIS 2021, Greater Noida, India, 19–20 February 2021; pp. 343–347.
11. Zhang, K. A machine learning based approach to identify SQL injection vulnerabilities. In Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering, ASE, San Diego, CA, USA, 11–15 November 2019; pp. 1286–1288.
12. Tien, C.W.; Huang, T.Y.; Tien, C.W.; Huang, T.C.; Kuo, S.Y. KubAnomaly: Anomaly detection for the Docker orchestration platform with neural network approaches. *Eng. Rep.* **2019**, *1*, e12080. [CrossRef]
13. Deriba, F.G.; Salau, A.O.; Mohammed, S.H.; Kassa, T.M.; Demilie, W.B. Development of a compressive framework using machine learning approaches for SQL injection attacks. *Prz. Elektrotech.* **2022**, *98*, 181–187. [CrossRef]
14. Kasim, Ö. An ensemble classification-based approach to detect attack level of SQL injections. *J. Inf. Secur. Appl.* **2021**, *59*, 102852. [CrossRef]
15. Padma, N.J.; Ravishankar, N.; Raju, M.B.; Ravi, N.C. Surgical striking SQL injection attacks using LSTM. *Indian J. Comput. Sci. Eng.* **2022**, *13*, 208–220. [CrossRef]
16. Alkhathami, J.M.; Alzahrani, S.M. Detection of SQL injection attacks using machine learning in cloud computing platform. *J. Theor. Appl. Inf. Technol.* **2022**, *100*, 5446–5459.
17. Farooq, U. Ensemble machine learning approaches for detection of SQL injection attack. *Teh. Glas.* **2021**, *15*, 112–120. [CrossRef]
18. Demilie, W.B.; Deriba, F.G. Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques. *J. Big Data* **2022**, *9*, 181–187. [CrossRef]
19. Gandhi, N.; Patel, J.; Sisodiya, R.; Doshi, N.; Mishra, S. A CNN-BiLSTM based approach for detection of SQL injection attacks. In Proceedings of the 2nd IEEE International Conference on Computational Intelligence and Knowledge Economy, ICCIKE 2021, Dubai, United Arab Emirates, 17–18 March 2021; pp. 378–383.

20. Sanshui. SQL Injection Detection by Machine Learning. Available online: https://www.kaggle.com/code/sanshui123/sql-injection-detection-by-machine-learning/input?select=Modified_SQL_Dataset.csv (accessed on 20 December 2023).
21. Devalla, V.; Raghavan, S.S.; Maste, S.; Kotian, J.D.; Annapurna, D.D. mURLi: A tool for detection of malicious URLs and injection attacks. *Procedia Comput. Sci.* **2022**, *215*, 662–676. [CrossRef]
22. Ashlam, A.A.; Badii, A.; Stahl, F. A novel approach exploiting machine learning to detect SQLi attacks. In Proceedings of the 2022 5th International Conference on Advanced Systems and Emergent Technologies, IC_ASET 2022, Hammamet, Tunisia, 22–25 March 2022; pp. 513–517.
23. Fu, H.; Guo, C.; Jiang, C.; Ping, Y.; Lv, X. SDSIOT: An SQL injection attack detection and stage identification method based on outbound traffic. *Electronics* **2023**, *12*, 2472. [CrossRef]
24. Zhao, C.; Si, S.; Tu, T.; Shi, Y.; Qin, S. Deep-Learning Based Injection Attacks Detection Method for HTTP. *Mathematics* **2022**, *10*, 2914. [CrossRef]
25. Kecman, V. Support Vector Machines–An Introduction. In *Support Vector Machines: Theory and Applications*; Wang, L., Ed.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1–47. [CrossRef]
26. Suthaharan, S. Decision Tree Learning. In *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning*; Suthaharan, S., Ed.; Springer: Boston, MA, USA, 2016; pp. 237–269. [CrossRef]
27. Altman, N.; Krzywinski, M. Ensemble methods: Bagging and random forests. *Nat. Methods* **2017**, *14*, 933. [CrossRef]
28. Baškarada, S.; Nguyen, V.; Koronios, A. Architecting microservices: Practical opportunities and challenges. *J. Comput. Inf. Syst.* **2020**, *60*, 428–436. [CrossRef]
29. Callens, A.; Morichon, D.; Abadie, S.; Delpey, M.; Liquet, B. Using random forest and gradient boosting trees to improve wave forecast at a specific location. *Appl. Ocean Res.* **2020**, *104*, 102339. [CrossRef]
30. Crespo-Martínez, I.S.; Campazas-Vega, A.; Guerrero-Higueras, Á.M.; Riego-DelCastillo, V.; Álvarez-Aparicio, C.; Fernández-Llamas, C. SQL injection attack detection in network flow data. *Comput. Secur.* **2023**, *127*, 103093. [CrossRef]
31. Lu, D.; Fei, J.; Liu, L. A semantic learning-based SQL injection attack detection technology. *Electronics* **2023**, *12*, 1344. [CrossRef]
32. Alghawazi, M.; Alghazzawi, D.; Alarifi, S. Deep learning architecture for detecting SQL injection attacks based on RNN autoencoder model. *Mathematics* **2023**, *11*, 3286. [CrossRef]