

A Hybrid Approach for Detecting SQL-Injection Using Machine Learning Techniques

Hari Krishna¹, Jared Oluoch¹ ^a and Junghwan Kim²

¹*Department of Electrical Engineering & Computer Science, University of Toledo,
2801 W. Bancroft Street, Toledo OH, U.S.A.*

²*Department of Engineering Technology, University of Toledo, 2801 W. Bancroft Street, Toledo OH, U.S.A.*

Keywords: SQL Injection Detection, Machine Learning, Naive Bayes Algorithm, Long Short-term Memory, Random Forest classifier, Cybersecurity, Algorithm Integration.

Abstract: SQL injection is a common web hacking technique that allows hackers to gain unauthorized access to a database. These database breaches may have far-reaching financial consequences to individuals, organizations, and the society. This paper introduces an innovative approach that combines Naive Bayes, Long Short-Term Memory (LSTM), and Random Forest to enhance the detection and mitigation of SQL injections. By extracting and analyzing data through the sequential application of Naive Bayes and LSTM algorithms, the proposed methodology uniquely synthesizes their outputs to inform a Random Forest classifier, aiming to optimize accuracy in identifying potential threats. The efficacy of this approach is validated through comprehensive testing, yielding a significant improvement in detection accuracy compared to conventional methods. Findings demonstrate the potential of integrating diverse machine learning techniques for cybersecurity applications and pave the way for future advancements in the automated detection of SQL injection and other similar cyber threats. The implications of this research extend to developing more secure web environments, ultimately contributing to the broader field of information security.


1 INTRODUCTION

Web applications have become part of nearly every aspect of modern life, ranging from business operations to personal data management. This proliferation of web-based applications comes with the need for enhanced security to protect the confidentiality, integrity, and availability of data. One of the most common security threats for web applications is Structured Query Language Injection (SQLi). These SQLi threats stand out due to their frequency, simplicity of execution, and potential for severe impact. More often, they lead to unauthorized access to and/or destruction of data.

SQLi attacks exploit vulnerabilities in web applications that use SQL databases, allowing attackers to execute malicious SQL code through improperly sanitized input fields. Traditional defenses mechanisms against such attacks involve signature-based detection systems and manual coding practices. However, attackers continuously evolve strategies to bypass these measures, leaving systems vulnerable to exploitation.

Machine learning (ML) has emerged as a powerful tool in cybersecurity, offering the ability to learn from and adapt to new data patterns and anomalies. Leveraging ML for the detection and prevention of SQLi presents a promising solution to the problem of constantly evolving attack vectors. By analyzing patterns within requests, ML-based systems can identify malicious behavior that deviates from the norm, including sophisticated attacks that would go unnoticed by traditional defenses.

This manuscript explores the efficacy of ML techniques in detecting SQLi attacks. It begins by examining the nature of SQLi attacks and existing detection methods. It then proposes a framework that employs a range of feature extraction methods and ML classifiers to differentiate between benign and malicious SQL queries. This paper's methodology focuses on detection accuracy and the system's ability to generalize, thus maintaining high performance in the face of new, previously unseen attack patterns. The paper hypothesizes that an ML-based approach can outperform traditional SQLi detection methods and provide enhanced security for web applications. To test this

^a  <https://orcid.org/0000-0002-9840-8180>

hypothesis, we implemented a series of experiments using various ML algorithms, creating a benchmark with a vast dataset (SHAH,) of almost 30,000 malicious and benign SQL queries. We rigorously evaluated the performance of the proposed method to check for its reliability and validity.

The novelty of our work is its combination of Naive Bayes, Long Short-Term Memory (LSTM), and Random Forest algorithms to detect SQLi. Specifically, it leverages the strengths of each algorithm - Naive Bayes for its efficiency and ability to handle large datasets, LSTM for its prowess in processing sequential data, and Random Forest for its accuracy in classification tasks. This proposed hybrid model significantly enhances detection accuracy. The proposed approach not only contributes to the theoretical understanding of ML applications in cybersecurity, but also provides a practical framework for developing more resilient web applications against SQLi attacks.

1.1 Motivation and Problem Definition

Developing machine learning-based defenses against SQLi attacks is important. Traditional security mechanisms are proving insufficient against the sophistication and evolving nature of SQLi, which remains a top threat in the Open Worldwide Application Security Project (OWASP) top 10 list for web application security risks (OWASP,). The potential damage from these attacks is substantial, including data breaches, loss of customer trust, financial liabilities, and, in extreme cases, complete operational shutdown. Figure 1 shows an SQLi workflow.

1.1.1 SQLi Attacks Overview

SQLi is a code injection technique that exploits a security vulnerability in an application's database layer. The vulnerability occurs when user inputs are either incorrectly filtered for string literal escape characters embedded in SQL statements or are not strongly typed and thereby unexpectedly executed. This can allow attackers to execute arbitrary SQL code on the database, leading to unauthorized access or manipulation of data. The basic concept of SQL injection revolves around the attacker's ability to insert or "inject" a malicious SQL query via the input data from the client to the application. A successful injection can lead to data leakage, deletion, or modification, among other impacts. Below is a simple example to illustrate how an SQL injection can occur: Suppose a web application uses the following SQL query to authenticate users:

```
SELECT * FROM users WHERE username =
'$username' AND password = '$password'; (1)
```

In this scenario 1, \$username and \$password are placeholders for user inputs. An attacker can inject SQL code if the application does not properly sanitize the user input.

Ex: 1- Username: admin' – Password: [left blank]
Resulting SQL Query:

```
SELECT * FROM users WHERE username
= 'admin' – ' AND password = ' '; (2)
```

The attacker inputs admin' – in the username field. The apostrophe (') ends the username string, and the double hyphen (–) comments out the rest of the SQL statement 2. This manipulation effectively removes the password check from the SQL query because everything after the – is considered a comment. If the admin username exists, the database processes the query as a legitimate request for the user named "admin" without verifying the password. Ex: 2- Username: 'OR '1'='1 Password: [not required for this injection] Resulting SQL Query:

```
SELECT * FROM users WHERE
username = ' ' OR '1' = '1'; (3)
```

The attacker's input ends the username criterion and adds a new condition that always evaluates to true ('1'='1') 3. Because the OR operator is used, the query will return true for every row, effectively bypassing any need for specific user credentials (Sadeghian et al., 2013).

Furthermore, the automation of attacks and the emergence of SQLi-as-a-Service offerings on the dark web have made these types of attacks accessible to non-skilled individuals, exacerbating the problem. Therefore, enhancing SQLi detection has technical significance and a broad socio-economic impact. Automating machine learning into SQLi detection is motivated by the need for a dynamic, robust, scalable solution that can adapt over time and detect even the most cunning attacks. The core problem addressed in this research is detecting SQLi attacks in web applications with greater accuracy and efficiency than current methods. Traditional pattern-matching and signature-based systems are limited by the need for constant updates and their inability to detect novel or obfuscated attacks. Moreover, they often suffer from high false-positive rates, causing unnecessary disruption to legitimate users and consuming valuable human and computational resources.

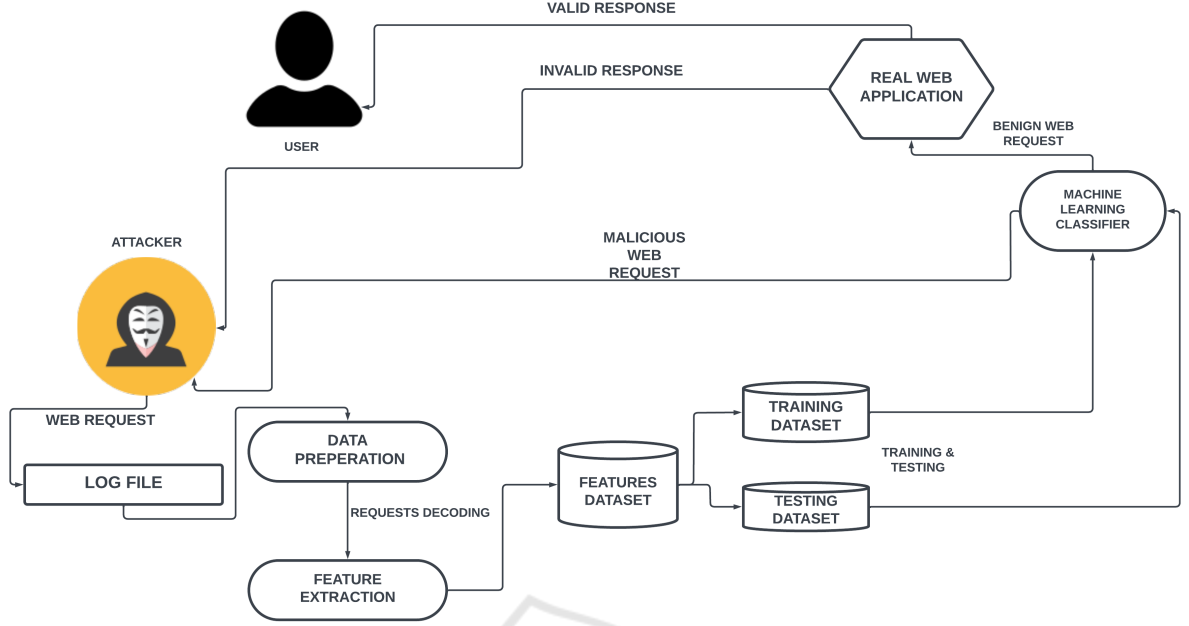


Figure 1: SQLi Workflow.

1.1.2 Research Questions

This paper addresses the following research questions.

- How can machine learning algorithms be effectively trained to distinguish between benign and malicious SQL queries with high accuracy?
- What feature extraction techniques can best capture the characteristics of SQLi to facilitate this classification?
- Can a machine learning-based system generalize from known attacks to detect zero-day SQLi attacks?
- How can such a system be designed to minimize false positives while maintaining an optimal detection rate?

This research seeks to define a machine learning-based approach that can autonomously adapt to the evolving landscape of SQLi threats without frequent rule updates or manual intervention. It uses ML's pattern recognition and generalization capabilities to create a more resilient web application infrastructure.

The rest of this paper is organized as follows. Section 2 discusses literature that closely relates to our work. Section 3 presents the system design. Section 4 outlines the performance metrics. Section 5 discusses results of our work. Finally, we conclude our work in Section 6.

2 RELATED WORK

SQLi attacks pose a persistent threat to the security of web applications, necessitating ongoing research into effective detection and mitigation strategies. Over the years, various approaches have been explored, ranging from traditional signature-based detection to more recent machine learning (ML) techniques. In Natural Language Processing (NLP), various feature engineering methods exist, yet for detecting SQLi attacks, Word Level term frequency-inverse document frequency (TF-IDF) vectors stand out as particularly effective. TF-IDF plays a crucial role in search and relevance determination of specific words within a document. Term Frequency measures the frequency of a word's appearance in a single document, whereas Document Frequency assesses the prevalence of a word across a collection of documents(Oudah et al., 2022). The primary benefit of using TF-IDF is its assumption that documents are merely collections of individual words without any interrelation. This simplicity yet effectiveness is especially suitable for our scenario because SQL lacks the grammatical and tense structures found in natural languages(Krishnan et al., 2021).

In their work (Alghawazi et al., 2022) conducted a systematic review, emphasizing the role of machine learning and deep learning models in detecting SQL injection attacks. Their paper highlights the promising results AI and ML techniques have shown in controlling SQLi, underscoring the intersection between

artificial intelligence fields and cybersecurity measures against SQLi attacks (Alghawazi et al., 2022).

SQL injection attacks fall into seven distinct categories: tautologies, illegal or logically incorrect queries, piggy-backed queries, stored queries, inference, and alternate encodings. In such attacks, a harmful script is inserted into a web application with weak security through an entry point, which is then relayed to the database at the back end (Farooq, 2021).

Attackers target vulnerabilities in management APIs, which, if exploited, can lead to successful attacks and compromise an organization's assets. Subsequently, attackers may use the compromised cloud to launch additional attacks on other cloud users. Exploiting vulnerabilities in the systems, software, or applications that facilitate multi-tenancy in cloud infrastructure can disrupt the separation between tenants. This disruption allows an attacker to access one organization's resources and potentially reach another user's or organization's data. The nature of multi-tenancy expands the potential attack surface, raising the likelihood of data leakage if separation controls are inadequate (Tripathy et al., 2020).

Most existing countermeasures against SQL injection rely on syntax-based detection methods or a set of pre-defined rules to identify such attacks. While these solutions may be effective against basic forms of SQLi, they are less effective against more advanced and sophisticated attacks. This vulnerability arises because attackers can devise new strategies to bypass detection, leveraging their understanding of how conventional detection mechanisms, which primarily focus on analyzing SQL syntax, operate (Abdulmalik, 2021).

A lot of research has been done on Semantic Learning-Based Detection Model. A study introduced synBERT, a semantic learning-based model for SQLi attack detection. This model embeds sentence-level semantic information from SQL statements into embedding vectors, which can be mapped to SQL syntax tree structures (Lu et al., 2023a). The research showcased synBERT's capability to outperform previous models, demonstrating over 90% accuracy in detecting SQLi on a wide range of datasets (Lu et al., 2023b).

The application of deep learning technologies has been explored to address the challenges traditional SQLi detection methods face. One framework involves offline training and online testing stages, processing samples through encoding, generalization, and tokenization before training a classifier that can efficiently identify SQLi attacks (Sun et al., 2023).

Other research work has been done in Probabilis-

tic Neural Networks (PNN) in SQLi Detection. For instance, Fawaz Khaled Alarfaj and Nayeem Ahmad Khan proposed using a PNN optimized by the BAT algorithm for detecting SQLi attacks. By extracting features from SQL queries and employing Chi-Square testing for feature selection (Alarfaj and Khan, 2023), their PNN model achieved an accuracy of 99.19%, demonstrating the effectiveness of deep learning and optimization algorithms in SQLi detection.

A deep neural network-based model, SQLNN, has been designed to detect SQL injection statements effectively. This model utilizes TF-IDF for data processing, highlighting the importance of filtering out common words to focus on significant terms for SQLi detection (Zhang et al., 2022).

Building on these insights, our work introduces a hybrid approach that combines the strengths of Naive Bayes, LSTM, and Random Forest algorithms. This combination seeks to address the individual limitations of each method—leveraging Naive Bayes for its efficiency with large datasets, LSTM for its deep learning capabilities in recognizing complex patterns, and Random Forest for its robustness and accuracy in classification tasks. To the best of our knowledge, this is the first study to explore such an integrated approach for SQLi detection, promising enhanced accuracy and greater adaptability to the evolving landscape of SQLi threats.

3 SYSTEM DESIGN

Various datasets are used to train an algorithm for detecting SQLi attacks. These datasets typically consist of a mix of normal and malicious SQL queries, allowing the algorithm to learn patterns associated with SQL injection attacks.

1. **Annotated Data:** The queries are usually labeled as normal or malicious. This annotation is crucial for supervised learning methods, where the model learns from labeled examples.
2. **Diversity of SQL Queries:** The dataset includes a wide range of SQL queries, both legitimate and malicious. This variety helps the model to differentiate between normal operations and SQLi attacks.
3. **Malicious SQL Samples:** These include typical SQL injection patterns like tautologies, illegal/logically incorrect queries, union queries, piggy-backed queries, and stored procedures exploitation.
4. **Normal SQL Samples:** These are regular, non-malicious SQL queries that an application would

typically process. Including these helps to reduce false positives, where legitimate queries are incorrectly flagged as SQLi.

5. **Parameterized Data:** Datasets often include queries with various parameters and structures to mimic real-world scenarios where inputs vary significantly.

These datasets are vital for machine learning models to effectively learn and distinguish between normal behavior and potential security threats from SQLi attacks. They are usually sourced from public repositories or cybersecurity organizations or generated through controlled penetration testing on the web.

This study proposes a novel methodology for detecting SQLi attacks by harnessing the strengths of Naive Bayes, Long Short-Term Memory (LSTM) networks, and Random Forest classifiers. Our approach is designed to optimize detection accuracy while mitigating common limitations associated with each algorithm when used in isolation. Below, we detail the dataset preparation, feature selection, individual algorithm implementation, and the integration strategy that forms the core of our detection framework. Algorithm 1 present the algorithm for our framework.

```

Data Preparation df ← load_csv("dataset")
queries ← df["query"] labels ← df["label"]
TF-IDF Vectorization vectorizer ←
create_tfidf_vectorizer() X_tfidf ←
vectorizer.fit_transform(queries)
while there are more training data do
  Train Naive Bayes nb_model ←
  create_multinomial_nb()
  nb_model.fit(X_tfidf, labels)
  Train LSTM lstm_model ←
  create_lstm_model(tokenizer,
  max_sequence_length)
  lstm_model.fit(X_train_lstm, y_train_lstm)
  Combine Naive Bayes and LSTM
  Features combined_features_train ←
  concatenate(nb_probs_train,
  lstm_features_train)
  Train Random Forest on Combined
  Features rf_model ←
  create_random_forest()
  rf_model.fit(combined_features_train,
  labels)
  Evaluate Random Forest predictions_rf ←
  rf_model.predict(combined_features_test)
  accuracy ← calculate_accuracy(y_test_nb,
  predictions_rf)
end

```

Algorithm 1: Proposed Model Algorithm.

3.1 Dataset Preparation and Pre-Processing

Our methodology consists of a comprehensive dataset derived from (SHAH,), with legitimate and malicious SQL query samples. We pre-process this data to ensure it is suitable for machine learning analysis. The dataset is then divided into training and testing sets, with 70% allocated for training and 30% reserved for validation purposes. The process begins with transforming the text data (SQL queries) into a numerical format that machine learning algorithms can process. This is done using the TF-IDF vectorizer.

- **Term Frequency (TF)** measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear many more times in long documents than in shorter ones. Thus, the term frequency is often divided by the document length (the total number of terms in the document) as a way of normalization:

$$TF(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}} \quad (4)$$

- **Inverse Document Frequency (IDF)** measures how important a term is. While computing TF, all terms are considered equally important. However, certain terms, such as "is," "of," and "that," may appear many times but have little importance. Thus, we need to weigh down the frequent terms while scaling up the rare ones by computing:

$$IDF(t) = \frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}} \quad (5)$$

each word or term is then represented by TF-IDF score, which is the multiplication of TF and IDF. Figure 2 shows the workflow of the model.

3.2 Feature Selection and Extraction

Given the diverse nature of SQLi attacks, selecting relevant features is crucial for effective detection. We employ selection techniques, such as TF-IDF and word embeddings, to extract features that capture syntactical and semantic nuances of SQL queries. This process enhances our models' learning efficiency and reduces false positive rates.

Figure 2 represents the overall workflow of our approach, showcasing each step from initial data collection to the final analysis.

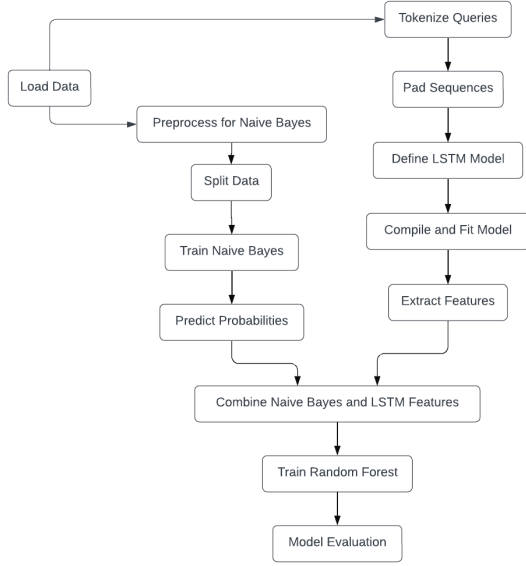


Figure 2: Workflow.

3.3 Model Training and Feature Integration

Naive Bayes Model. The MultinomialNB classifier was trained on the TF-IDF-transformed dataset 45. The classifier calculates the probability of each class given a set of inputs:

$$P(y | X) = \frac{P(X | y) \times P(y)}{P(X)} \quad (6)$$

where:

y is the class variable,

X represents features,

$P(y | X)$ is the posterior probability,

$P(X | y)$ is the likelihood,

$P(y)$ is the class prior probability, and

$P(X)$ is the predictor prior probability.

These probabilities represent how likely each query is thought to be malicious or regular, based on the frequency and distribution of words within the text, adjusted by the overall importance of these words in the dataset. The output probabilities, P_{NB} , served as one component of the feature set for the integrated model.

LSTM Model. The LSTM (Long Short-term Memory) model is a type of recurrent neural network (RNN) that is particularly designed for learning from sequences, such as time-series data or text. In our study, a bidirectional LSTM architecture is utilized. A bidirectional LSTM processes data in both forward and reverse directions, effectively increasing the information available to the network and improving the context for each point in the sequence.

The model is enhanced with dropout and L2 regularization—methods used to prevent overfitting when a model learns the training data too well, including the noise, and performs poorly on new data. Dropout works by randomly setting a fraction of the output units of the layer to 0 at each update during training time, which helps to prevent overfitting by making the network’s cells less sensitive to the weights of other cells. L2 regularization, also known as weight decay, adds a penalty term to the loss function proportional to the sum of the squares of the weights, which encourages the model weights to be small and, in turn, simplifies the model.

$$\text{Regularization Loss} = \lambda \sum_{i=1}^n w_i^2 \quad (7)$$

where λ is the regularization factor (0.001 here) and w_i are the weights of the kernel in the LSTM layer.

EarlyStopping is configured to monitor the validation loss (val_loss) in this setup. The patience parameter is set to 5, which means training will continue until the validation loss fails to improve for five consecutive epochs. When this condition is met, training stops, and to restore_best_weights=True, the model weights are rolled back to the point where the validation loss was at its minimum.

The bidirectional LSTM model is trained on the dataset to recognize patterns indicative of SQL injection. As it processes the input sequences (i.e., the tokenized SQL queries), it builds up a state that captures information about the sequences seen so far. After training, the model can extract feature representations from sequences, essentially high-level data abstractions.

The features are extracted from the final dense layer of the LSTM model. In neural networks, the penultimate layer is just before the final output layer. This layer captures the input data’s most informative and discriminative representations, which are crucial for the subsequent classification task.

Random Forest Classifier. The Random Forest model integrated the Naive Bayes probabilities and LSTM-derived features, creating a combined feature set, $F_{\text{combined}} = [P_{NB} \oplus F_{\text{LSTM}}]$, where \oplus denotes concatenation. When constructing each decision tree within the Random Forest, the algorithm iteratively chooses the best feature to split the data at each node. The “best” feature is determined based on which feature split maximizes the Information Gain. The algorithm compares the Information Gain of splits using different features and chooses the feature that provides the highest gain. This process is repeated for each node in each tree until a stopping criterion is met.

(e.g., maximum tree depth, minimum node size).

$$IG(D_p, f) = I(D_p) - \left(\frac{N_{left}}{N} I(D_{left}) + \frac{N_{right}}{N} I(D_{right}) \right) \quad (8)$$

where:

IG is information gain

I is the impurity measure,

D_p, D_{left}, D_{right} are the datasets of the parent and two child nodes,

N is the total number of samples,

N_{left} and N_{right} are the number of samples in the left

and right child nodes,

f is the feature to split on.

In Random Forest, not all features are considered for splitting at each node. Instead, a random subset of features is selected. This adds diversity to the model, crucial for improving generalization over fitting individual decision trees. The ensemble's final prediction is typically more robust and less prone to overfitting than a single decision tree. Each tree votes for a class, and the class with the most votes becomes the model's prediction. The final prediction is based on the combined features by constructing multiple decision trees during training and outputs the class, which is the mode of the classes (classification) of the individual trees.

This ensemble approach leverages the predictive capabilities of both algorithms, aiming to harness their complementary strengths for enhanced detection accuracy.

4 PERFORMANCE METRICS

The performance of the integrated model was evaluated using accuracy, precision, recall, and F-1 score to provide a comprehensive view of its effectiveness.

Accuracy is defined as the proportion of true results (both true positives and true negatives) among the total number of cases examined. Formally, it is represented as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

Where *TP*, *TN*, *FP*, and *FN* represent the true positives, true negatives, false positives, and false negatives, respectively.

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It is also known as the Positive Predictive Value.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \quad (10)$$

Recall, also known as Sensitivity or True Positive Rate, is the ratio of correctly predicted positive observations to all observations in an actual class.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \quad (11)$$

The F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (12)$$

5 RESULTS

The evaluation of our integrated model for SQL injection detection, combining Naive Bayes, LSTM, and Random Forest algorithms, demonstrates a significant advancement in detection accuracy. As shown in Table 1, our combined approach achieved better results compared to existing work. The LSTM model, tailored for sequence processing and augmented with bidirectional layers and regularization techniques, reached a training accuracy of 99.83% and a validation accuracy of 99.07% by the final epoch. This high accuracy on the validation set underscores the model's ability to generalize well to unseen data, a critical aspect of effective SQL injection detection.

Upon integrating the outputs from the Naive Bayes and LSTM models with the Random Forest classifier, our system attained a final detection accuracy of 99.89% on the test set. This performance marks a substantial improvement over our initial benchmarks, where accuracy hovered around 96%. The accuracy progression indicates the synergistic effect of combining these diverse machine-learning strategies.

The enhancement in detection accuracy from approximately 96% to 99.89% underscores our hybrid approach's effectiveness. By leveraging the probabilistic outputs of Naive Bayes, the sequential data processing capability of LSTM, and the ensemble decision-making power of Random Forest, our model effectively captures the complex patterns and anomalies characteristic of SQL injection attacks. This improvement validates the potential of integrating multiple machine learning paradigms and highlights the adaptability and robustness of our detection system against a wide array of attack vectors. These results underscore the potential of our hybrid approach to adapt and respond to the evolving dynamics of SQL injection threats, outperforming models reliant on singular methodologies.

Table 1: Performance metrics of the methods on the individual test runs.

Method	Accuracy	Precision	Recall	F1	TP	TN	FP	FN
Naïve Bayes	0.8619	0.9028	0.8122	0.892	1950	2054	210	432
SVM	0.9723	0.96218	0.9951	0.9807	1956	2088	40	75
LSTM	0.9925	0.991	0.9878	0.9865	2045	2548	15	19
Random Forest	0.9723	0.9621	0.9527	0.9807	2156	3545	55	107
Paper(Tasdemir et al., 2023)	0.9986	0.9996	0.9966	0.9981	2259	3854	1	8
Paper(Lu et al., 2023a)	0.9974	0.9968	0.9952	0.9960				
Combined Approach	0.9987	0.9991	0.9973	0.9981	2241	3896	2	6

5.1 Discussion

The integrated model's high accuracy rate highlights the efficacy of combining probabilistic, sequential, and ensemble learning techniques in detecting SQLi attacks. By leveraging the distinct advantages of Naive Bayes, LSTM, and Random Forest classifiers, our approach addresses the limitations inherent in using these algorithms in isolation. Furthermore, the evaluation underscores the importance of feature extraction and selection in enhancing the model's detection capabilities, as evidenced by the significant role played by TF-IDF vectorization and LSTM-derived features.

6 CONCLUSION AND FUTURE WORK

The study of machine learning algorithms in the context of SQLi detection has yielded promising results. Decision Trees and SVMs offer decent accuracy and balance between precision and recall, making them suitable for scenarios where computational efficiency is crucial. In contrast, Random Forests and Neural Networks demonstrate superior performance in accuracy and F1-score, indicating their effectiveness in complex SQLi detection scenarios. These results highlight the potential of machine learning in enhancing cybersecurity measures against SQLi attacks.

However, the performance of these algorithms can be influenced by factors such as the dataset's quality, feature selection, and algorithm configuration. Considering that the dynamic and evolving nature of cyber threats like SQLi requires continuous adaptation, and the improvement of these models is crucial.

Future work should focus on utilizing more diverse and comprehensive datasets, including the latest types of SQLi attacks. Data augmentation techniques can also be explored to enhance the model's

generalizability and performance in real-world scenarios. There is scope for improving the algorithms, especially in reducing false positives and negatives. Advanced machine learning and deep learning techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), could be explored. Implementing these algorithms in real-time SQLi detection systems would be a significant step forward. This includes integrating machine learning models into existing database management systems or web application frameworks.

As machine learning models become more complex, ensuring the explainability and interpretability of these models is crucial. This is important for trust and accountability, especially in security-critical applications. Investigating hybrid models that combine the strengths of different machine learning algorithms could improve performance in detecting SQLi attacks. Developing models that can continuously learn and adapt to new types of SQLi attacks over time would be invaluable, ensuring that the detection mechanisms remain effective as attack patterns evolve.

By pursuing these avenues, we can enhance the effectiveness of machine learning in cybersecurity, particularly in the crucial area of SQLi detection, thereby making digital spaces more secure against these pervasive threats.

REFERENCES

- Abdulmalik, Y. (2021). An improved sql injection attack detection model using machine learning techniques. *International Journal of Innovative Computing*, 11(1):53–57.
- Alarfaj, F. K. and Khan, N. A. (2023). Enhancing the performance of sql injection attack detection through probabilistic neural networks. *Applied Sciences*, 13(7).
- Alghawazi, M., Alghazzawi, D., and Alarifi, S. (2022). Detection of SQL injection attack using machine learn-

- ing techniques: A systematic literature review. *Journal of Cybersecurity and Privacy*, 2(4):764–777.
- Farooq, U. (2021). Ensemble machine learning approaches for detection of sql injection attack. *Tehnički glasnik*, 15(1):112–120.
- Krishnan, S. A., Sabu, A. N., Sajan, P. P., and Sreedeeep, A. (2021). Sql injection detection using machine learning. *Revista Geintec-Gestao Inovacao E Tecnologias*, 11(3):300–310.
- Lu, D., Fei, J., and Liu, L. (2023a). A semantic Learning-Based SQL injection attack detection technology. *Electronics*, 12(6):1344.
- Lu, D., Fei, J., and Liu, L. (2023b). A semantic learning-based sql injection attack detection technology. *Electronics*, 12(6).
- Oudah, M. A., Marhusin, M. F., and Narzullaev, A. (2022). Sql injection detection using machine learning with different tf-idf feature extraction approaches.
- OWASP. Owasp top 10:2021 a03:2021 - injection. https://owasp.org/Top10/A03_2021-Injection. Accessed: August 20 2023.
- Sadeghian, A., Zamani, M., and Abdullah, S. M. (2013). A taxonomy of sql injection attacks. In *2013 International Conference on Informatics and Creative Multimedia*, pages 269–273.
- SHAH, S. S. H. Sql injection dataset, kaggle. <https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset>. Accessed: Jan 20 2023.
- Sun, H., Du, Y., and Li, Q. (2023). Deep learning-based detection technology for sql injection research and implementation. *Applied Sciences*, 13(16).
- Tasdemir, K., Khan, R., Siddiqui, F., Sezer, S., Kurugollu, F., Yengec-Tasdemir, S. B., and Bolat, A. (2023). Advancing sql injection detection for high-speed data centers: A novel approach using cascaded nlp.
- Tripathy, D., Gohil, R., and Halabi, T. (2020). Detecting sql injection attacks in cloud saas using machine learning. In *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pages 145–150.
- Zhang, W., Li, Y., Li, X., Shao, M., Mi, Y., Zhang, H., and Zhi, G. (2022). Deep neural Network-Based SQL injection detection method. *Security and Communication Networks*, 2022.