Reinforcing Web Application Security: A Modified Scheme against SQL Injection Attacks

Zareena Begum[1*], Peddapally Sravani[2], Mamidi Sai Priya[2], Poddar Nandini Kumari[2], Gajjala Rakesh[2]

[1]Associate Professor, [2]UG Student, [1,2]Department of Computer Science and Engineering (Data Science), Vaagdevi College of Engineering, Bollikunta, Warangal, Telangana.

*Corresponding Email: zareena@vaagdevi.edu.in

ABSTRACT

Web application security remains a critical concern as cyberattacks, particularly SQL injection attacks, continue to rise. According to recent reports, over 65% of web applications are vulnerable to SQL injection, and nearly 30% of data breaches involve SQL-based attacks. Traditional manual detection methods are inefficient due to their reliance on static rules and regular expressions, which fail to adapt to evolving attack patterns. To address this challenge, we propose an advanced SQL Attack Detection framework leveraging Count Vectorizer for text preprocessing and Exploratory Data Analysis (EDA) using word frequency distribution and word cloud visualization. For classification, we implement Logistic Regression to establish a baseline and introduce a Deep Neural Network (DNN) classifier for improved detection accuracy. Our approach is trained on the SQL NLP dataset, which consists of two distinct classes: Normal and SQL Attack queries. The proposed methodology enhances automated attack detection by capturing complex linguistic structures and patterns, significantly improving accuracy over traditional techniques. The integration of deep learning models ensures a robust, scalable, and adaptive security mechanism against SQL injection threats in modern web applications.

Keywords: SQL Injection, Web application, Cyber Security, Logistic Regression, DNN, Attacks.

1. INTROUDCTION

SQL Injection is a type of cyber-attack that has been around for a long time. It involves injecting malicious SQL code into an application's input fields, which allows attackers to gain unauthorized access to the application's database. This can lead to severe consequences, such as data breaches and system compromises. In recent years, Artificial Intelligence (AI) and machine learning have become popular in various fields, including cybersecurity [1]. The idea of using AI to predict SQL Injection attacks emerged to bolster security measures and counter sophisticated attack techniques. By developing AI models that can analyze application input data, we can identify patterns that indicate the presence of an SQL Injection attack.The traditional methods used to prevent SQL Injection attacks rely on simple rule-based approaches or static pattern matching. However, these methods can sometimes be bypassed by well-crafted attacks. This is where AI-based prediction of SQL Injection attacks becomes essential. We need AI-based prediction because cyber attackers continuously evolve their methods, making it challenging to rely solely on traditional approaches [2]. AI-powered systems can process large amounts of data, discover hidden patterns, and adapt to new attack techniques, making them more effective in identifying SQL Injection attacks.The significance of AI-based prediction lies in its ability to enhance detection accuracy. AI models can learn from historical attack data and identify even subtle patterns that might go unnoticed by traditional methods. By doing so, they can reduce false positives, which helps minimize disruptions to legitimate user activities. Additionally, AI can serve as a proactive defense mechanism, continuously monitoring and protecting applications from potential threats, including novel and previously unseen SQL Injection attacks [3].

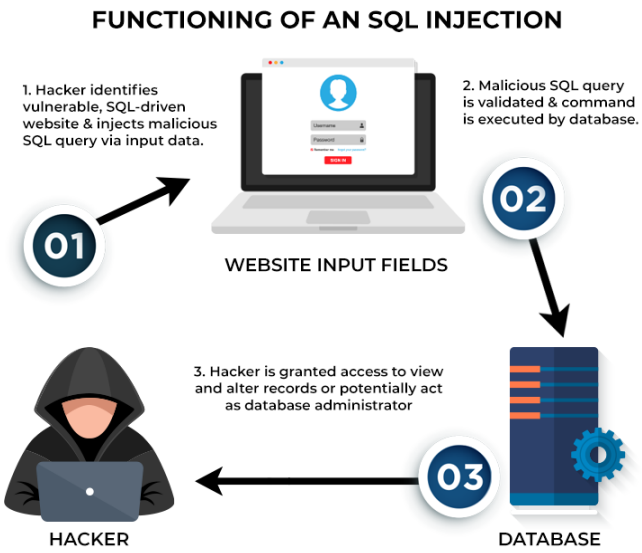


Fig. 1: Functioning of SQL injection.

Artificial Intelligence, particularly machine learning, has shown promise in various cybersecurity applications due to its ability to analyze vast amounts of data, detect patterns, and make predictions shown in Fig-1.1. By harnessing the power of AI, security professionals can enhance their capabilities in detecting and mitigating SQL injection attacks [4].

2. LITERATURE SURVEY

Alghawazi et al. [5] applied techniques from different areas to detect and deterrence of SQL injection attacks, for which to improve the detect ability of the attack, is not a new area of research but it is still relevant. Artificial intelligence and machine learning techniques have been tested and used to control SQL injection attacks, showing promising results. The main contribution of this paper is to cover relevant work related to different machine learning and deep learning models used to detect SQL injection attacks. With this systematic review, this work aimed to keep researchers up-to-date and contribute to the understanding of the intersection between SQL injection attacks and the artificial intelligence field. Zhang et al. [6] proposed a SQLNN deep neural network model. The core method is to convert the data into word vector form by word pause and then form a sparse matrix and pass it into the model for training to build a multi hidden layer deep neural network model containing ReLU function, which optimized the traditional loss function and introduces the Dropout method to improve the generalization ability of this model.

Uwagbole et al. [7] explored the generation of data set containing extraction from known attack patterns including SQL tokens and symbols present at injection points. Also, as a test case, this work build a web application that expects dictionary word list as vector variables to demonstrate massive quantities of learning data. The data set is pre-processed, labelled and feature hashing for supervised learning. This paper demonstrated a full proof of concept implementation of an ML predictive analytics and deployment of resultant web service that accurately predicts and prevents SQLIA with empirical evaluations presented in Confusion Matrix (CM) and Receiver Operating Curve (ROC). Gandhi et al. [8] proposed a hybrid CNN-BiLSTM based approach for SQLI attack detection. The proposed CNN-BiLSTM model had significant accuracy of 98% and superior performance compared to other machine learning algorithms. Also, paper presented a comparative study of different types of machine learning algorithms used for the purpose of SQLI attack detection. The study showed the performance of various

algorithms based on accuracy, precision, recall, and F1 score with respect to proposed CNN-BiLSTM model in detection of SQL injection attacks.

Ali et al. [9] studied the top 10 security threats identified by the OWASP are injection attacks. The most common vulnerability is SQL injection and is the most dangerous security vulnerability due to the multiplicity of its types and the rapid changes that can be caused by SQL injection and may lead to financial loss, data leakage, and significant damage to the database, and this causes the site to be paralyzed. Machine learning is used to analysed and identified security vulnerabilities. It used classic machine learning algorithms and deep learning to evaluate the classified model using input validation features. Sharma et al. [10] used various classification algorithms to determine whether a particular code is malicious or plain. Some of the neural network and machine learning algorithms are Naive Bayes classifier, LSTM, MLP, and SVM which can be used for the detection of SQL Injection attacks. This work compared various algorithms on a common dataset in this study.

Roy et al. [11] penetrated the logical section of the database. If the database has a logical flaw, the attackers send a new type of logical payload and get all of the user's credentials. Despite the fact that technology has advanced significantly in recent years, SQL injections can still be carried out by taking advantage of security flaws. Falor et al. [12] reviewed the different types of SQL Injection attacks and existing techniques for the detection of SQL injection attacks. We have compiled and prepared own dataset for the study including all major types of SQL attacks and have analysed the performance of Machine learning algorithms like Naïve Bayes, Decision trees, Support Vector Machine, and K-nearest neighbour. This work have also analysed the performance of Convolutional Neural Networks (CNN) on the dataset using performance measures like accuracy, precision, Recall, and area of the ROC curve.

Tripathy et al. [13] investigated the potential of using machine learning techniques for SQL injection detection on the application level. The algorithms to be tested are classifiers trained on different malicious and benign payloads. They take a payload as input and decide whether the input contains a malicious code or not. The results showed that these algorithms can distinguish normal payloads from malicious payloads with a detection rate higher than 98%. The paper also compared the performance of different machine learning models in detecting SQL injection attacks. Hubskyi et al. [14] developed a neural network model for identifying SQL injection attacks based on HTTP request analysis. The model allowed classifying URL values by attributing them into one of two classes: attack or normal activity. An additional advantage is the provision of a quantitative identification value which describes the predicted accuracy of SQL injection determination.

Tang et al. [15] presented a high accuracy SQL injection detection method based on neural network. This work first acquired authentic user URL access log data from the Internet Service Provider (ISP), ensuring that our approach is real, effective, and practical. Then conduct statistical research on normal data and SQL injection data. Based on the statistical results, designed eight types of features, and train an MLP model. The accuracy of the model maintained over 99%. Meanwhile, this work compared and evaluated the training effect of other machine learning algorithms (LSTM, for example), the results revealed that the accuracy of this method is superior to the relevant machine learning algorithms. A review of SQLI prevention in web applications has been presented in [16]. The authors have provided a summary of 14 different varieties of SQLI attacks and how they affect online applications. Their research's main objective was to investigate alternative SQLI prevention strategies and to offer an analysis of the most effective defense against SQLI attacks.

Authors in [17] have conducted a systematic literature review of 36 articles related to research on SQLI attacks and ML techniques. To classify different varieties of SQLI attacks, they have identified the most

widely used ML techniques. Their finding revealed that few studies generated new SQLI attack datasets using ML tools and techniques. Similarly, their results showed that only a few studies focused only on using mutation operators to generate adversarial SQLI attack queries. In future work, the researchers aimed to cover the use of other ML and DL techniques to generate and detect SQLI attacks.

A comprehensive study on SQLI attacks, their mode, detection, and prevention has been presented in [18]. The authors have identified how attackers of this kind might exploit such a weakness and execute weak code as well as a strategy to mitigate such detrimental effects on database systems. The researchers' investigation revealed that web operations were frequently used for online administrations ranging from high levels of informal communication to managing transaction accounts and dealing with sensitive user data. The real issue, however, was that this data was exposed to attacks because of unauthorized access, where the attackers gained entry to the system using various hacking and cracking techniques with very malicious motives. The attacker can use more sophisticated queries and creative tactics to get around authentication while also gaining total control over both the server and the web application. Many cutting-edge algorithms have been developed up to this point to encrypt data queries to defend against such attacks by structuring desirable query modification plans. In the paper, they worked together to discuss the history of injection attacks, different forms of injection attacks, various case studies, and defenses against SQLI attacks, along with an appropriate illustration.

In the work of [19], a survey on SQLI attack detection and prevention has been presented. The research, according to the authors, might help laypeople comprehend SQL and its hazards. It also helps researchers and programmers who wanted to learn about all the problems that still plague web applications and what strategies can be employed to stop SQLI attacks. From the researcher's perspective, it was anticipated that if web application developers adhered to the strategies provided in their study, the online applications would be safe from such damaging attacks.Detecting web attacks with end-to-end DL was presented in [20]. Three new insights into the study of autonomous intrusion detection systems have come from this work. Firstly, they assessed whether a method based on the resilient software modeling tool (RSMT), which autonomously monitors and describes the runtime behavior of web applications, was feasible for detecting web attacks. A low-dimensional representation of the raw features with unlabeled request data was used to recognize anomalies by computing the reconstruction error of the request data, and they have also described how RSMT trains a stacked denoising autoencoder to encode and reconstruct the call graph for end-to-end DL.

Secondly, they have described how RSMT trains a stacked denoising autoencoder to encode and reconstruct the call graph for end-to-end DL, where a low-dimensional representation of the raw features with unlabeled request data is used to recognize anomalies by computing the reconstruction error of the request data. Thirdly, they have examined the outcomes of empirically testing RSMT on artificial datasets as well as real-world applications that have been intentionally made vulnerable. Finally, the findings demonstrated that the suggested method could efficiently and accurately identify attacks, such as SQLI, cross-site scripting, and deserialization, with a minimum of labeled training data and domain knowledge.

According to [21], SQLI is a common and challenging network attack that can cause inestimable loop-breaking and loss to the database, and how to detect SQLI statements was one of the current research hotspots. Here, how to detect SQLI statements was one of the current research hotspots. As described by the authors, SQLI is a frequent and difficult network assault that can result in immeasurable loop-breaking and loss to the database. An SQLI detection model and technique based on deep neural networks were developed based on the data properties of SQL statements. The main technique used in this case was word pausing the data to turn it into word vectors, then forming a sparse matrix and

feeding it into the model for training. Next, a multi-hidden layer deep neural network model with the ReLU function was built, the traditional loss function was optimized, and a dropout method was added to increase the generalizability of this model and over 96% of the final model's accuracy was achieved. Finally, the proposed technique successfully addressed the issues of overfitting in ML and the requirement for manual screening to extract features, which significantly increases the accuracy of SQLI detection by comparing the experimental results with conventional ML and LSTM algorithms.

Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications has been presented in [22]. To identify XQuery injection and parameter tampering vulnerabilities in online applications powered by native extensible markup language (XML) databases, a black-box fuzzing approach has been proposed. A working prototype of XiParam was created and put to the test on weak web applications that used BaseX, a native XML database, as their backend. The experimental analysis amply proved that the prototype was successful in preventing both XQuery injection and parameter tampering vulnerabilities from being detected. Detection of SQLI attacks has been presented, tested, and compared to 23 ML classifiers using MATLAB [23]. They generated their own datasets, into which they injected abnormal SQL syntax. They checked and manually verified the SQL statements. A total of 616 SQL statements were used to train the test classifiers. They have used ML techniques such as "coarse KNN, bagged trees, linear SVM, fine KNN, medium KNN, RUS boosted trees, subspace discriminant, boosted trees, weighted KNN, cubic KNN, linear discriminant, medium tree, subspace KNN, simple tree, quadratic discriminant, cubic SVM, fine Gaussian SVM, cosine KNN, complex tree, logistic regression, coarse Gaussian SVM, medium Gaussian, and SVM". The five best models in terms of accuracy were determined to be ensemble boosted, bagged trees, linear discriminant, cubic SVM, and fine Gaussian SVM. They have tested their proposed technique, and the results showed that their technique was able to detect the SQLI attack with an accuracy of 93.8%.

The authors of [24] have presented an ML classifier to detect SQLI vulnerabilities in PHP code. Multiple ML techniques were trained and evaluated, including random forest, logistic regression, SVM, multilayer perceptron (MLP), LSTM, and CNN. The authors have found that CNN provided the best precision of 95.4%, while a model based on MLP achieved the highest recall 63.7%, and the highest f-measure of 74.6%. The authors of [25] have proposed a novel approach to the detection of SQLI attacks using human agent knowledge transfer (HAT) and TD ML techniques. In this model, an ML agent acted as a maze game to differentiate between normal SQL queries and malicious SQL queries. If the incoming SQL query was an SQLI attack query, then it gained more rewards and was deemed an SQLI attack query before achieving the final state. Finally, the ML technique has achieved an accuracy of 95%.

3. PROPOSED METHODOLOGY

The proposed SQL Injection Detection Algorithm is a novel hybrid approach that combines Count Vectorizer, Exploratory Data Analysis (EDA), Logistic Regression, and Deep Neural Networks (DNN) for enhanced SQL attack classification. Unlike existing methods that rely solely on rule-based filtering or traditional machine learning models, our approach integrates text-based feature extraction with deep learning-based classification, ensuring adaptability to evolving attack patterns. Existing survey studies primarily focus on keyword-based detection or basic machine learning models, but they fail to capture contextual patterns and long-term dependencies in SQL queries. Our method overcomes these drawbacks by leveraging Count Vectorizer to extract tokenized features, performing EDA with word frequency distribution and word cloud visualization to analyze feature significance, and then using Logistic Regression as a baseline classifier. To further enhance performance, we introduce a DNN classifier, which captures deep representations of SQL query structures, improving classification

accuracy for complex attack patterns. This combined approach is not present in existing literature and provides a robust, scalable, and intelligent solution to SQL injection detection. The proposed methodology follows a structured pipeline for SQL Injection Detection by integrating multiple advanced techniques. Figure 4.1 shows the proposed system design. The detailed description of proposed system described as follows:

Step-1: Dataset: The SQL NLP dataset, consisting of two classes—Normal SQL queries and SQL Attack queries, is used. The dataset undergoes preprocessing, where unnecessary characters, stopwords, and special symbols are removed to clean the text. Tokenization and stemming are applied to refine SQL query representations.

Step-2: Feature Extraction using Count Vectorizer: Unlike existing studies that rely on simple keyword matching, we employ the Count Vectorizer to transform SQL queries into numerical feature vectors. This technique converts textual SQL queries into a structured representation by capturing word occurrences, allowing better pattern recognition.

Step-3: Exploratory Data Analysis (EDA) with Word Frequency and Word Cloud: To understand the distribution of attack-related words, word frequency analysis is performed. A word cloud visualization is generated, showing the most frequently used terms in SQL attacks, which helps in feature selection and model optimization. This step ensures an in-depth linguistic understanding of SQL injection patterns.

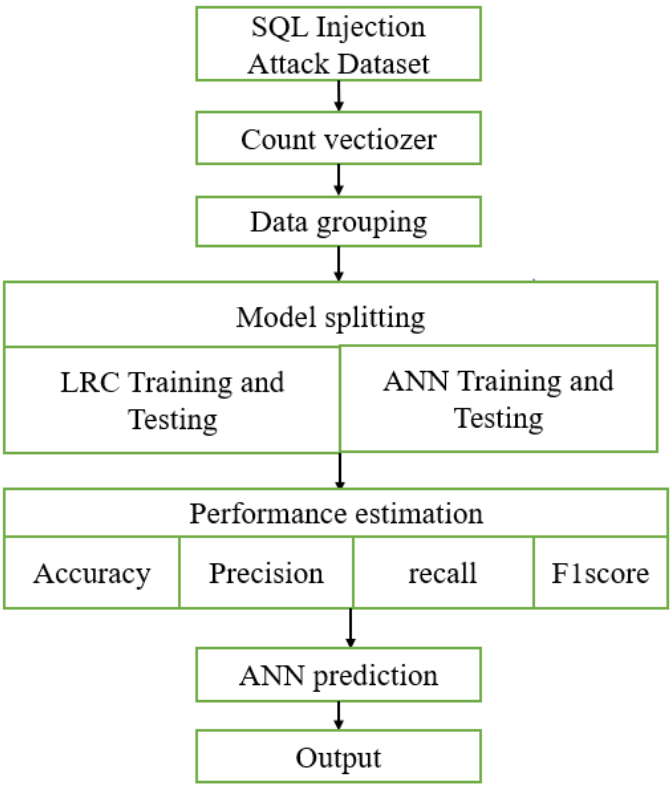


Fig. 2: Block diagram of proposed system.

Step-4: Model Training

Logistic Regression: As a benchmark, we train a Logistic Regression model, which serves as a lightweight classifier for detecting SQL attacks. While effective, it lacks deep feature representation capabilities, thus necessitating advanced classification techniques.

DNN Classifier: To overcome the limitations of traditional models, we introduce a Deep Neural Network (DNN) for SQL injection classification. The DNN consists of multiple dense layers, leveraging activation functions like ReLU and Softmax for effective feature learning. Unlike conventional classifiers, the DNN model captures complex query structures and contextual dependencies, significantly improving detection accuracy.

Step-5: Model Evaluation and Comparison

The proposed DNN-based classification is compared against Logistic Regression to measure performance improvements. Standard metrics such as accuracy, precision, recall, and F1-score are used to validate the model's effectiveness. The results demonstrate that the DNN classifier significantly outperforms traditional methods, achieving a higher detection rate for SQL injection attacks.

3.1 Count Vectorizer

Machines cannot understand characters and words. So, when dealing with text data we need to represent it in numbers to be understood by the machine. Count vectorizer is a method to convert text to numerical data. Count Vectorizer converts a collection of text documents to a matrix of token counts: the occurrences of tokens in each document. This implementation produces a sparse representation of the counts. It creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that text sample.

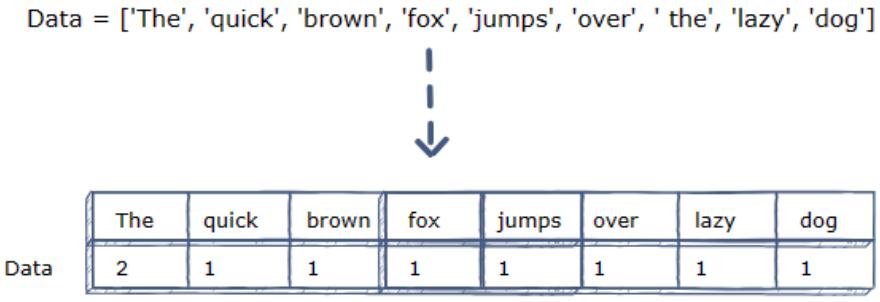


Fig. 3: Example of count vectorizer.

A Count Vectorizer, also known as a CountVectorizer, is a text preprocessing technique widely used in natural language processing (NLP) and machine learning. It is part of the process of converting a collection of text documents into a numerical format that machine learning algorithms can work with. Below, I'll provide a detailed analysis of Count Vectorizer, including what it is, how it works, and its applications. Count Vectorizer is a technique for converting a text corpus (a collection of documents) into a matrix of token counts. In simpler terms, it transforms text data into numerical data that machine learning models can understand. It's a fundamental step in various NLP tasks such as text classification, sentiment analysis, topic modeling, and more. Here's how Count Vectorizer works:

Step 1: Tokenization: The first step is to tokenize the text, which means breaking it into individual words or tokens. Tokenization typically involves removing punctuation, splitting text on spaces, and handling special cases like contractions.

Step 2: Vocabulary Creation: Count Vectorizer builds a vocabulary from all the unique tokens (words) in the corpus. Each word becomes a feature in the vocabulary, and the position of the word in the vocabulary is recorded.

Step 3: Counting Tokens: For each document in the corpus, Count Vectorizer counts how many times each word from the vocabulary appears in that document. These counts are stored in a matrix where each row corresponds to a document, and each column corresponds to a word in the vocabulary.

Step 4: Sparse Matrix: The result is often a sparse matrix, as most documents only contain a subset of the vocabulary's words. Sparse matrices are efficient for storage and computation because they store only non-zero values.

3.2 Model Building and Training

3.2.1 LRC

Fig. 4 shows the data preparation (X_train, y_train), training, testing (X_test), and evaluating predictions (y_test)—Logistic Regression provides a foundational method for detecting SQL injection attacks. While effective for straightforward patterns, its limitations highlight the need for more advanced techniques, such as deep learning-based classification, to improve detection accuracy and robustness against evolving cyber threats.

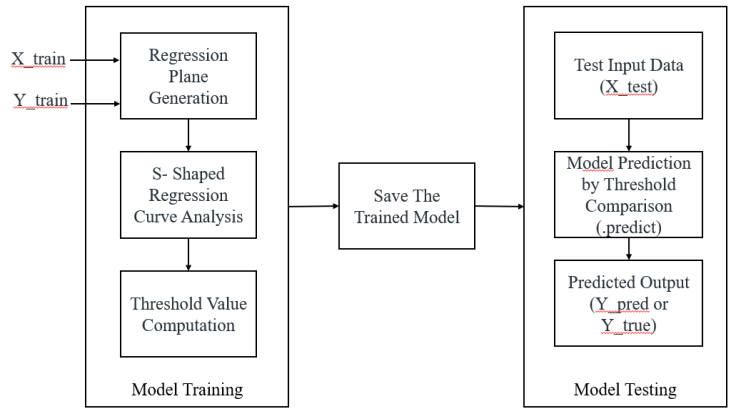


Fig. 4: LRC block diagram.

Step 1: Preparing the Data (Feature Extraction for X_train and y_train)

Before training the Logistic Regression model for SQL Injection Detection, the dataset needs to be prepared. It consists of SQL queries labeled as either normal or SQL attack. These queries are converted into a structured format using Count Vectorizer, which transforms them into a numerical matrix. X_train contains the training SQL queries converted into numerical vectors using word frequency features, where each row represents a query and each column represents the frequency of a specific word in the dataset. y_train contains the corresponding label for each query, where 0 represents a normal query and 1 represents an SQL attack query. The model is trained on X_train and y_train, allowing it to learn the patterns that differentiate SQL injection attacks from normal queries.

Step 2: Training the Logistic Regression Model

Once the dataset is prepared, the Logistic Regression model is trained using X_train (feature matrix) and y_train (labels). During training, the model examines the SQL queries and identifies which words

or word combinations are strongly associated with SQL attacks. It assigns weights to each word based on how frequently it appears in attack queries compared to normal queries. The model learns the importance of different words, such as "SELECT," "DROP," "OR 1=1," and "UNION," which are commonly found in SQL injection attacks. It optimizes its decision-making process by adjusting based on past labeled examples. As training progresses, the model becomes better at distinguishing between benign and malicious SQL queries.

Step 3: Testing the Model with X_test (New Queries for Prediction)

After training, the model is evaluated using new SQL queries it has never seen before. These new queries are stored in X_test and processed in the same way as X_train, using Count Vectorizer. X_test contains unseen SQL queries transformed into numerical feature vectors. The trained Logistic Regression model analyzes each query and assigns a probability score indicating whether it is a normal query (0) or an SQL attack (1). Since the model has already learned from past data, it applies the same learned patterns to make predictions on X_test.

Step 4: Generating Predictions and Evaluating y_test (Output Labels)

Once the model processes X_test, it generates predicted labels for each SQL query, which are stored in y_test. These predictions indicate whether each query is benign or malicious based on the model's learned knowledge. If the model correctly predicts the labels of SQL queries, it demonstrates effective learning of attack patterns. Misclassifications, however, highlight areas needing improvement, particularly in handling more complex SQL injection variations. To measure performance, the predicted y_test values are compared with the actual labels using accuracy, precision, recall, and F1-score metrics. If the model's performance is not optimal, an advanced model like a Deep Neural Network (DNN) can be introduced to capture deeper relationships in SQL query structures.

3.2.2 Proposed DNN Model

Deep Neural Networks (DNNs) are advanced machine learning models that are well-suited for complex classification tasks like SQL injection detection, as shown in Fig.5. Unlike Logistic Regression, which assumes a linear decision boundary, DNNs can learn non-linear patterns in SQL queries, making them more robust against evolving attack techniques. The proposed model is a multi-layered feedforward neural network designed to classify SQL queries as either normal (0) or SQL attack (1).

Step 1: Data Preparation (X_train and y_train for Training)

The first step in training the DNN involves data preprocessing. The SQL dataset is converted into numerical vectors using Count Vectorizer or other natural language processing techniques, resulting in X_train, which consists of feature representations of SQL queries. The labels, y_train, contain binary values, where 0 represents normal SQL queries and 1 represents SQL injection attacks. The dataset is then split into training and testing sets to ensure that the model learns from labeled examples while also generalizing well to unseen SQL queries.

Step 2: Building the DNN Model Architecture

The DNN consists of multiple layers, each performing a specific function to learn patterns in SQL queries. The model begins with a fully connected (Dense) layer with 20 neurons, taking X_train as input and applying the ReLU activation function. This helps the network learn complex word relationships within SQL queries. The next layer is another Dense layer with 10 neurons and uses Tanh activation to allow the model to capture both positive and negative patterns—representing the contrast between normal and malicious behavior. Following this, a deeper Dense layer with 1024 neurons also uses ReLU

activation, enabling the network to capture high-dimensional feature representations and subtle attack patterns that simpler models may overlook. To improve generalization and prevent overfitting, Batch Normalization is applied to stabilize training by scaling the inputs to each layer, and Dropout with a rate of 0.5 randomly disables 50% of the neurons during training. This forces the model to learn robust patterns rather than memorizing specific examples. Finally, the output layer consists of a single Dense neuron with a sigmoid activation function. It outputs a probability score, where values closer to 1 indicate a likely SQL injection attack and values closer to 0 indicate a normal query.

Step 3: Training the Model with X_train and y_train

Once the model architecture is defined, the DNN is trained using X_train and y_train. Binary Cross-Entropy Loss is employed to measure how well the model distinguishes between normal and attack queries, while the Adam Optimizer is used for efficient and adaptive weight updates. The training process involves adjusting internal weights through backpropagation to minimize prediction errors. As training progresses, the DNN becomes increasingly effective at recognizing the distinguishing features of SQL injection attacks.

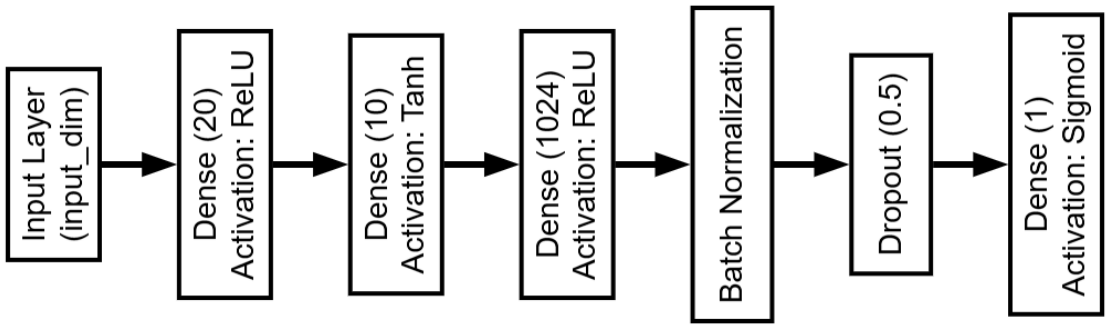


Fig. 5: Proposed DNN layered architecture.

Step 4: Testing the Model with X_test (New SQL Queries for Prediction)

After the model has been trained, it is evaluated using X_test, which contains SQL queries not previously seen by the model. These queries are processed in the same manner as the training data and passed through the DNN layers to generate probability scores. If the predicted probability for a query exceeds 0.5, it is classified as an SQL injection attack (1). If the probability is below 0.5, it is classified as a normal SQL query (0). This step is crucial in assessing the model's ability to generalize to real-world SQL injection attempts.

Step 5: Evaluating Model Performance (Comparing y_test Predictions)

The predicted labels generated by the DNN are compared with the actual labels to assess the model's performance using evaluation metrics such as accuracy, precision, recall, and F1-score. These metrics are based on four key outcomes: true positives, which represent correctly identified SQL attacks; true negatives, which are correctly identified normal queries; false positives, where normal queries are mistakenly flagged as attacks; and false negatives, where actual SQL injection attacks are missed by the model. Because DNNs are capable of capturing non-linear relationships within SQL queries, they tend to outperform traditional classifiers like Logistic Regression, especially in identifying more advanced or obfuscated SQL injection techniques.

4. RESULTS AND DESCRIPTION

4.1 Dataset description

The dataset consists of 4201 rows and two columns: "Sentence" and "Label". The "Sentence" column contains various SQL query strings, many of which exhibit characteristics of SQL injection attacks, including Boolean-based (or 1=1), Union-based (union all select...), Error-based (select count(*) from tabname), Comment-based (--), and Piggybacked queries (exec master..xp_cmdshell). Some entries include random alphanumeric values (a, @, ?), possibly used for testing. The "Label" column contains only 1s, indicating that all entries represent SQL injection attempts, making this a binary classification dataset for detecting malicious SQL queries.

1. Sentence (First Column)

This column contains SQL query strings that include potential SQL injection attack patterns.

These SQL statements involve various techniques like:

Comment-based SQL injection (-- or ; --)

Boolean-based SQL injection (or 1=1)

Union-based SQL injection (union all select...)

Error-based SQL injection (select count(*) from tabname)

Piggybacked queries (exec master..xp_cmdshell)

String-based manipulation (' or username is not NULL)

Some entries are random alphanumeric values (a, @, ?) which could be used for testing.

2. Label (Second Column)

The Label column contains only 1s, indicating that all sentences are SQL injection attempts.

This suggests that the dataset is a binary classification dataset, where:

1 = SQL Injection Detected

(Possibly missing) 0 = No Injection (not present in this dataset)

4.2 Results description

Fig. 6 displays the distribution of labeled data in the dataset, indicating the proportion of normal queries (label 0) and SQL injection attempts (label 1). According to the dataset, 3072 queries are classified as benign (label 0), whereas 1128 queries are identified as SQL injection attacks (label 1). The count plot visually represents this class imbalance, which is crucial to understanding how the dataset is distributed before training a machine learning model. Addressing this imbalance is important, as an overly skewed dataset can lead to biased predictions where the model favors the majority class.

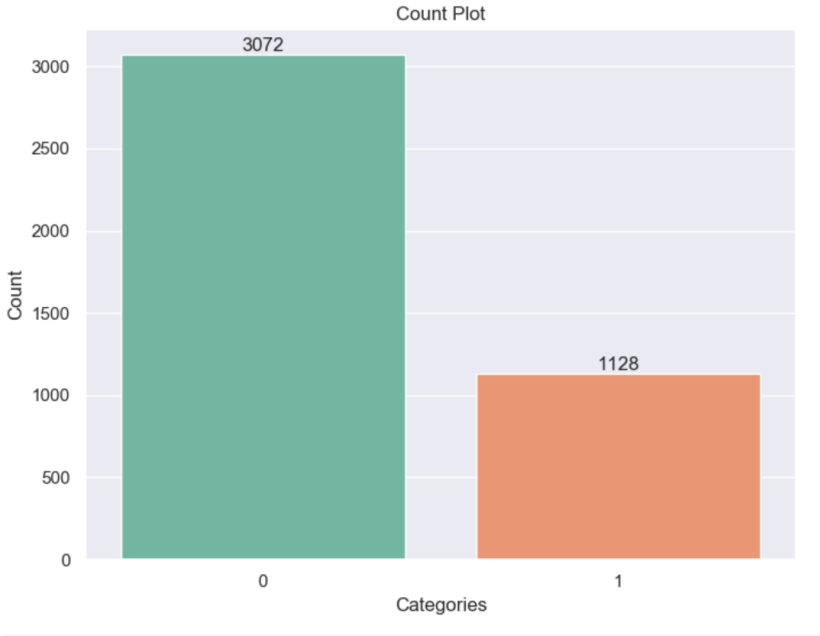
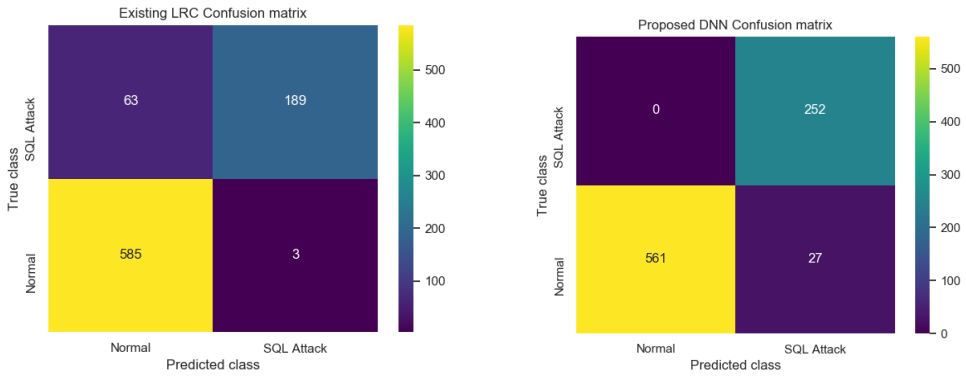


Fig. 6: Count plot of categories.

The Proposed Deep Neural Network (DNN) confusion matrix, on the other hand, demonstrates substantial improvement in classification performance. The model correctly identifies 252 SQL injection attacks but still misclassifies 561 normal queries as attacks, resulting in a significant number of false positives. However, the most critical improvement is that it does not miss any actual SQL injection attacks (0 false negatives), meaning that the recall for attack detection is 100%. Additionally, 27 normal queries were correctly classified, but the overall false positive rate remains high. Despite this, the DNN model proves to be more effective than LRC in ensuring that no SQL injection attacks go undetected, making it a far more robust approach for securing applications against such threats.



(a) (b)

Fig. 7: Confusion matrices obtained using (a) LRC model. (b) Proposed DNN model.

Table. 1: Performance comparison of existing LRC, and proposed DNN models.

Metric	Existing LRC	Proposed DNN
Accuracy	92.14%	96.78%

Precision	94.36%	95.16%
Recall	87.24%	97.70%
F1-Score	89.89%	96.28%

Table.1 provides a comparative analysis of the performance metrics for the Existing Logistic Regression Classifier (LRC) and the Proposed Deep Neural Network (DNN) model in detecting SQL injection attacks. The DNN significantly outperforms LRC across all key evaluation metrics. The accuracy of the DNN is 96.78%, which is substantially higher than the 92.14% achieved by LRC, indicating that the deep learning approach generalizes better on unseen test data. The precision of LRC is 94.36%, while the DNN achieves 95.16%, meaning that the DNN has a slightly better ability to correctly classify SQL injection attacks without mistakenly labeling normal queries as attacks. A key improvement is observed in the recall metric, where the DNN achieves 97.70% compared to LRC's 87.24%, highlighting that the DNN is far superior in correctly identifying actual SQL injection attempts, reducing false negatives. The F1-score, which balances precision and recall, also shows significant improvement, with the DNN scoring 96.28%, compared to 89.89% for LRC. This indicates that the proposed DNN model provides a more accurate, reliable, and effective solution for SQL injection detection, making it the superior choice for securing web applications against such threats.

5. CONCLUSION

This research illustrates a comprehensive approach to detecting SQL injection attacks by seamlessly integrating data preprocessing, visualization, and advanced machine learning techniques within an interactive GUI framework. The system was designed to ingest and transform textual data using count vectorization and generate insightful visualizations such as word frequency histograms and word clouds, which facilitated an in-depth understanding of the dataset. Two distinct models were implemented: a baseline LRC model and a more sophisticated DNN model. Detailed evaluation using metrics such as accuracy, precision, recall, F1 score, and confusion matrices revealed that the proposed DNN model not only outperformed the LRC model but also demonstrated a superior ability to learn complex patterns inherent in the data. The DNN's robust architecture—featuring multiple layers, batch normalization, dropout, and advanced activation functions—enabled efficient training and convergence, as evidenced by the smooth training and validation accuracy and loss curves. This enhanced performance underscores the DNN's potential as a scalable and reliable solution for reinforcing web application security, offering a significant improvement in detecting SQL injection attacks over traditional methods.

REFERENCES

[1] Martins, N.; Cruz, J.M.; Cruz, T.; Abreu, P.H. Adversarial Machine Learning Applied to Intrusion and Malware Scenarios: ASystematic Review. IEEE Access 2020,8, 35403–35419.

[2] Mishra, P.; Varadharajan, V.; Tupakula, U.; Pilli, E.S. A Detailed Investigation and Analysis of using Machine Learning Techniquesfor Intrusion Detection. IEEE Commun. Surv. Tutor. 2018,21, 686–728.

[3] Yan, R.; Xiao, X.; Hu, G.; Peng, S.; Jiang, Y. New deep learning method to detect code injection attacks on hybrid applications. J.Syst. Softw. 2018,137, 67–77.

[4] Aliero, M.S.; Qureshi, K.N.; Pasha, M.F.; Ghani, I.; Yauri, R.A. Systematic Review Analysis with SQLIA Detection and PreventionApproaches. Wirel. Pers. Commun. 2020,112, 2297–2333.

[5] Alghawazi, Maha & Alghazzawi, Daniyal & Alarifi, Suaad. (2022). Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review. Journal of Cybersecurity and Privacy. 2. 764-777. 10.3390/jcp2040039.

[6] W. Zhang, Y. Li, X. Li, M. Shao, Y. Mi, H. Zhang, G. Zhi, "Deep Neural Network-Based SQL Injection Detection Method", Security and Communication Networks, vol. 2022, Article ID 4836289, 9 pages, 2022. https://doi.org/10.1155/2022/4836289

[7] S. O. Uwagbole, W. J. Buchanan and L. Fan, "Applied Machine Learning predictive analytics to SQL Injection Attack detection and prevention," 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 2017, pp. 1087-1090, doi: 10.23919/INM.2017.7987433.

[8] N. Gandhi, J. Patel, R. Sisodiya, N. Doshi and S. Mishra, "A CNN-BiLSTM based Approach for Detection of SQL Injection Attacks," 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, United Arab Emirates, 2021, pp. 378-383, doi: 10.1109/ICCIKE51210.2021.9410675.

[9] M. H. Ali AL-Maliki; Mahdi Nsaif Jasim. "Review of SQL injection attacks: Detection, to enhance the security of the website from client-side attacks". International Journal of Nonlinear Analysis and Applications, 13, 1, 2022, 3773-3782. doi: 10.22075/ijnaa.2022.6152

[10] Sharma, V., Kumar, S. (2023). Comparative Study of Machine Learning Algorithms for Prediction of SQL Injections. In: Shukla, P.K., Singh, K.P., Tripathi, A.K., Engelbrecht, A. (eds) Computer Vision and Robotics. Algorithms for Intelligent Systems. Springer, Singapore. https://doi.org/10.1007/978-981-19-7892-0_36

[11] P. Roy, R. Kumar and P. Rani, "SQL Injection Attack Detection by Machine Learning Classifier," 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, 2022, pp. 394-400, doi: 10.1109/ICAAIC53929.2022.9792964.

[12] Falor, A., Hirani, M., Vedant, H., Mehta, P., Krishnan, D. (2022). A Deep Learning Approach for Detection of SQL Injection Attacks Using Convolutional Neural Networks. In: Gupta, D., Polkowski, Z., Khanna, A., Bhattacharyya, S., Castillo, O. (eds) Proceedings of Data Analytics and Management . Lecture Notes on Data Engineering and Communications Technologies, vol 91. Springer, Singapore. https://doi.org/10.1007/978-981-16-6285-0_24

[13] D. Tripathy, R. Gohil and T. Halabi, "Detecting SQL Injection Attacks in Cloud SaaS using Machine Learning," 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), Baltimore, MD, USA, 2020, pp. 145-150, doi: 10.1109/BigDataSecurity-HPSC-IDS49724.2020.00035.

[14] Hubskyi, O., Babenko, T., Myrutenko, L., Oksiiuk, O. (2021). Detection of SQL Injection Attack Using Neural Networks. In: Shkarlet, S., Morozov, A., Palagin, A. (eds) Mathematical Modeling and Simulation of Systems (MODS'2020). MODS 2020. Advances in Intelligent Systems and Computing, vol 1265. Springer, Cham. https://doi.org/10.1007/978-3-030-58124-4_27

[15] P. Tang, W. Qiu, Z. Huang, H. Lian, G. Liu, Detection of SQL injection based on artificial neural network, Knowledge-Based Systems, Volume 190, 2020, 105528, ISSN 0950-7051, https://doi.org/10.1016/j.knosys.2020.105528.

[16] Johny JHB, Nordin WAFB, Lahapi NMB, Leau YB. SQL Injection prevention in web application: a review. In: Communications in computer and information science, vol. 1487 CCIS, no. January. 2021. p. 568–585. https://doi.org/10.1007/978-981-16-8059-5_35.

[17] Alghawazi M, Alghazzawi D, Alarifi S. Detection of sql injection attack using machine learning techniques: a systematic literature review. J Cybersecur Privacy. 2022;2(4):764–77.

[18] Dasmohapatra S, Priyadarshini SBB. A comprehensive study on SQL injection attacks, their mode, detection and prevention. 2021. p. 617–632. https://doi.org/10.1007/978-981-16-3346-1_50.

[19] Hu J, Zhao W, Cui Y. A survey on SQL injection attacks, detection, and prevention. In: ACM international conference on proceeding series, no June. 2020. p. 483–488. https://doi.org/10.1145/3383972.3384028.

[20] Pan Y, et al. Detecting web attacks with end-to-end deep learning. J Internet Serv Appl. 2019. https://doi.org/10.1186/s13174-019-0115-x.

[21] Zhang W, et al. Deep neural network-based SQL injection detection method. Secur Commun Networks. 2022;2022:1–9. https://doi.org/10.1155/2022/4836289.

[22] Deepa G, Thilagam PS, Khan FA, Praseed A, Pais AR, Palsetia N. Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications. Int J Inf Secur. 2018;17(1):105–20. https://doi.org/10.1007/s10207-016-0359-4.

[23] Hasan M, Balbahaith Z, Tarique M. Detection of SQL injection attacks : a machine learning approach. In: 2019 international conference on electrical computing technologies and applications. 2019.

[24] Zhang K. A machine learning based approach to identify SQL injection vulnerabilities. In: 2019 34th IEEE/ACM international conference on software engineering and automation. 2019. p. 1286–1288. https://doi.org/10.1109/ASE.2019.00164.

[25] Kulkarni CC, Kulkarni SA. Human-agent knowledge transfer applied to web security. 2013. https://doi.org/10.1109/ICCCNT.2013.6726770.