# A Structured Query Language (SQL) Injection Attack Detection Based on Probabilistic Neural Network

Jawaher Alzaidi [1,*] and Jehan Janbi [2]

[1] Department of Information Technology, College of Computer and Information Technology, Taif University, Taif, Saudi Arabia
[2] Department of Computer Science, College of Computer and Information Technology, Taif University, Taif, Saudi Arabia
Email: jawaher52477@gmail.com (J.A.); j.gonbi@tu.edu.sa (J.J.)
*Corresponding author

***Abstract*—Structured Query Language (SQL) injection attacks remain one of the most critical web application security threats. Deep Learning (DL) models such as Convolutional Neural Networks (CNN) often require high computational resources, large numbers of parameters, and long training times, which limit their suitability for lightweight detection systems. This study proposes Probabilistic Neural Network (PNN)-based models for SQL injection detection to improve detection efficiency while maintaining low computational complexity. Three PNN-based approaches were investigated: standard PNN, Bayesian Optimization-based PNN (PNN-BO), and Enhanced-PNN with additional dense, batch normalization, and dropout layers. The proposed models were compared with Artificial Neural Network (ANN) and CNN models using a publicly available SQL injection dataset. Experimental results show that the Enhanced-PNN achieved 99.22% accuracy, 99.24% precision, and 99.22% recall while maintaining significantly lower model complexity and training time compared with other DL models. In addition, PNN-BO improved the standard PNN performance from 97.45% to 98.66% accuracy through bandwidth optimization. The findings demonstrate that PNN-based models can provide accurate and computationally efficient solutions for SQL injection attack detection.**

***Keywords*—Structured Query Language (SQL) injection, machine learning, deep learning, Probabilistic Neural Network (PNN), Bayesian Optimization (BO)**

## I. Introduction

Structured Query Language (SQL) is a widely used language for managing and interacting with relational database systems. Web applications rely on Database Management Systems (DBMSs) such as MySQL, PostgreSQL, and Oracle [1]. However, SQL is vulnerable to several security threats. Attackers exploit these vulnerabilities by injecting malicious SQL statements into the user input.

This often occurs because of insufficient input validation in web applications, allowing attackers to bypass authentication and gain unauthorized access, as shown in Fig. 1 [2]. According to the Open Web Application Security Project (OWASP), SQL injection attacks pose a critical threat to web applications by undermining core security pillars such as confidentiality, authentication, authorization, and data integrity. Such vulnerabilities can lead to severe repercussions, including unauthorized access to sensitive information (e.g., usernames, passwords, and personal data), unauthorized manipulation (insertion, deletion, or modification) of database records, and even the execution of arbitrary operating system commands.

SQL injection attacks can be classified into several types, as listed in Table I [3]. Each type of SQL injection exploits defects in the construction or handling of SQL queries, which can lead to the unauthorized disclosure, modification, or execution of sensitive data or the execution of malicious commands.

Deep Learning (DL) models, such as Convolutional Neural Networks (CNN), have been widely used for intrusion and attack detection. However, these models often require high computational resources, numerous parameters, and long training times, which may limit their suitability for lightweight cybersecurity applications. In addition, deep architectures may suffer from overfitting when they are trained on limited datasets.

Probabilistic Neural Networks (PNNs) provide an alternative approach to classification tasks owing to their rapid training process, simple architecture, and efficient probabilistic decision-making [4]. PNN models are less computationally expensive than many DL models and have demonstrated effective performance in several classification applications.

Manuscript received January 14, 2026; revised April 3, 2026; accepted June 3, 2026; published September 14, 2026.

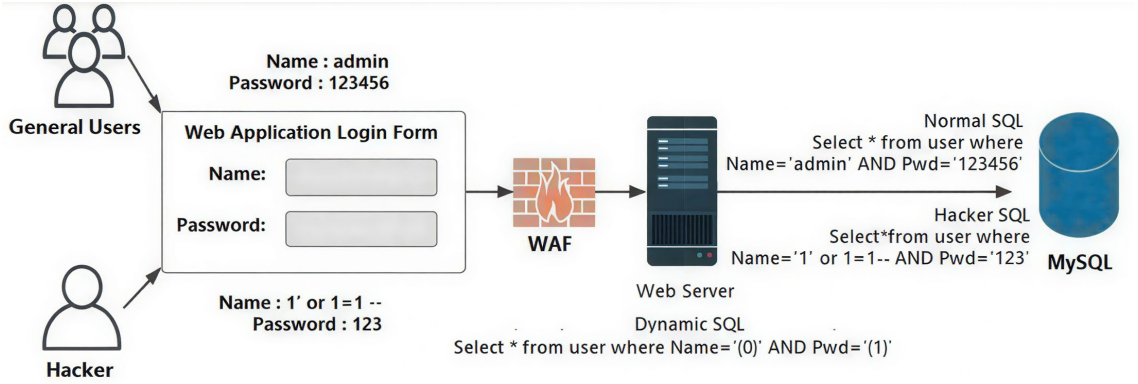


Fig. 1. SQL injection attack process (adopted from [2]).

TABLE I. SQL INJECTION QUERY TYPES (ADAPTED FROM [3])

| SQL Injection Type | Example |
|---|---|
| Illegal/Logically incorrect query SQL | SELECT * FROM products WHERE product_name='; SELECT * FROM users; |
| Tautology SQLI | SELECT * FROM users WHERE name ='user' AND password = 'password' OR '1' = '1'; |
| Piggybacked Query SQLI | SELECT book_name FROM booktable WHERE book_id='book1'AND book_name=0;DROP booktable; |
| Blind SQLI | 'admin'; IF SYSTEM_USER='sa' SELECT 1/0 ELSE SELECT 5 |
| Union query SQLI | SELECT*FROM accountable WHERE user_login='user' UNION SELECT * FROM accountTable WHERE No=10,232 |
| Stored Procedure SQLI | SELECT * FROM accountable WHERE userlogin='user' AND passwd='pass'; SHUTDOWN;--; |
| Timing SQLI | SELECT price FROM products WHERE id=' or 1=1--';sleep (1) ; |
| Alternate Encoding SQLI | SELECT*FROM accountable WHERE user='user' AND pin='pass'; EXEC ('44524f50204 44154414241534520'); SHUTDOWN; |

Although previous studies have applied DL techniques to SQL injection detection, limited attention has been paid to improving PNN-based architecture for lightweight and efficient attack detection. In addition, the optimization of PNN parameters and enhancement of its architecture for improving classification performance remain insufficiently explored.

The main contributions of this study are as follows:

- Proposed and evaluated PNN-based models for SQL injection-attack detection.
- Applied BO to improve the bandwidth parameter of the PNN model.
- Developed an Enhanced-PNN architecture by integrating dense, batch normalization, and dropout layers to improve performance and reduce overfitting.
- Compared the proposed models with Artificial Neural Network (ANN) and CNN models using standard evaluation metrics and confidence interval analysis.
- Demonstrated that the proposed PNN-based approaches achieve competitive performance while maintaining a lower computational complexity.

The comparison includes evaluation metrics such as accuracy, precision, recall, and F1-Score, as well as parameter count and training time analysis. The model was trained and tested using an SQL injection dataset. This paper is organized as follows: Section II presents the literature review, Section III presents the materials and methods, Section IV presents the results and discussion, and Section V presents the conclusion.

## II. LITERATURE REVIEW

This section reviews previous studies related to intrusion and SQL injection detection using Machine Learning (ML) and DL techniques. This review focuses on different methodological approaches, including ANN, CNN, Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), hybrid DL models, and PNN.

Several studies have applied ML and DL techniques for intrusion and SQL injection detection. DL approaches, particularly CNN and recurrent-based architectures, have demonstrated strong performance in cybersecurity applications.

Studies [5–13] investigated DL techniques for intrusion detection using datasets such as NSL-KDD [14], ISCXIDS2012 [15], CIC-IDS2017 [16], UNSW-NB15 [17], and CICIDS2018 [18]. These studies explored architectures, including CNN, RNN, Gated Recurrent Unit (GRU), Bidirectional LSTM (BiLSTM), and hybrid DL models. The reported results demonstrate that DL models can effectively learn complex attack patterns and achieve high detection accuracy.

For SQL injection detection, several studies have applied DL models, such as CNN-BiLSTM [19], RNN-autoencoder [2], TextCNN-LSTM [20], Deep Neural Network (DNN) [21], and multi-view CNN-LSTM [22]. These studies achieved high classification performance on private and public SQL injection datasets.

In addition to DL approaches, traditional ML methods have been explored for SQL injection detection. Irungu *et al.* [23] applied supervised ML models, including Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Random Forest (RF), while Peralta-Garcia *et al.* [3] evaluated RF, Decision Tree (DT), and SVM models on a Kaggle SQL injection dataset [24]. These studies reported competitive detection accuracy

with lower computational complexity than some DL approaches. ANN-based intrusion detection models were also investigated, which demonstrated that ANN architectures can achieve high classification accuracy in cybersecurity applications [25, 26]. Recent studies have explored lightweight neural architectures, such as PNN. Alarfaj and Khan [4] applied the PNN model for SQL injection detection and reported a high classification performance. Table II summarizes the most relevant studies on SQL injection and intrusion detection using ML and DL techniques.

Although previous studies achieved high accuracy, many DL architectures require high computational resources, large parameter sizes and longer training times. In addition, limited research has focused on improving PNN architecture and optimizing their parameters for efficient SQL injection detection. Therefore, this study focuses on enhancing PNN performance using the BO algorithm and an improved architecture, while maintaining computational efficiency.

TABLE II. SQL INJECTION DETECTION STUDIES

| No. | Reference | Year | Dataset | Method | Accuracy |
|---|---|---|---|---|---|
| 1 | [19] | 2021 | Private dataset | CNN-BiLSTM | 98% |
| 2 | [21] | 2022 | Private dataset | DNN | 96% |
| 3 | [2] | 2023 | SQL dataset | RNN-autoencoder | 94% |
| 4 | [23] | 2023 | Private dataset | SVM<br>KNN<br>RF | SVM 98.3605%;<br>KNN 96.296%;<br>RF 97.530% |
| 5 | [4] | 2023 | Private dataset | PNN | 99.19% |
| 6 | [20] | 2023 | GitHub SQL injection dataset [27] | TextCNN- LSTM | 99.57% |
| 7 | [25] | 2024 | Private dataset | ANN-GWO | 99.68% |
| 8 | [3] | 2024 | Kaggle SQL dataset [24] | RF<br>DT<br>SVM | RF 99%;<br>DT 97%;<br>SVM 98% |
| 9 | [22] | 2024 | Private dataset | LSTM-CNN | 99.96% |

## III. MATERIALS AND METHODS

This section describes the methodology used in this study to detect SQL injection attacks. It includes dataset collection, preprocessing steps, feature extraction using Term Frequency-Inverse Document Frequency (TF-IDF), and model development. In addition, both the proposed and comparison models (ANN and CNN) are presented, followed by the evaluation methods used to assess their performance. Fig. 2 illustrates the workflow of the steps followed in this study, starting from data collection and ending with the performance evaluation.

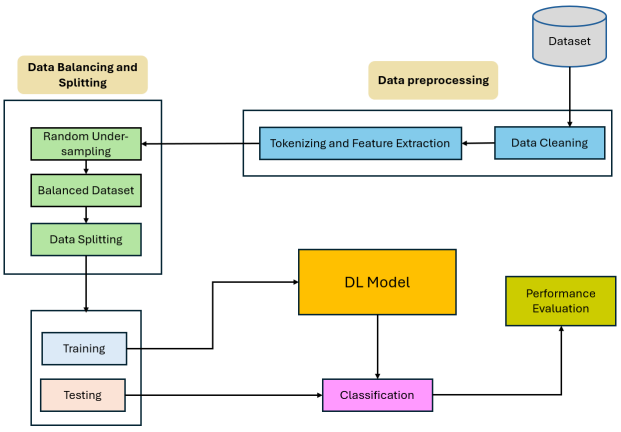


Fig. 2. Workflow of the proposed methodology.

### A. Dataset Collection

This study used a publicly available SQL injection dataset from Kaggle [24]. The dataset contained 30,905 SQL queries, including 11,382 benign and 19,523 malicious queries. Each record consists of a query and a corresponding label, where 0 represents a normal query, and 1 indicates a SQL injection attack. This dataset is widely used to evaluate ML models for SQL injection detection.

### B. Data Preprocessing

The preprocessing steps applied to the dataset are summarized below:

- Removal of missing values: All records with missing values were removed from the dataset.
- Removal of duplicate records: Duplicate queries were eliminated to ensure data quality.
- Text normalization: All SQL queries were converted to lowercase to ensure consistency

### C. Tokenization and Feature Extraction

Tokenization was executed using a word-based approach that segmented SQL queries along whitespace and word boundaries. Feature extraction was subsequently performed using the TF-IDF technique. To optimize the dimensionality of the feature space and mitigate the computational overhead during training, the TF-IDF vectorizer was limited to a maximum of 1000 features (max_features = 1000). This threshold was strategically selected to retain the most informative text features associated with malicious SQL injection patterns while minimizing the training latency. To prevent data leakage, the vectorizer was fitted exclusively to the training partition and subsequently transformed across the test set.

### D. Data Balancing

The raw dataset exhibited a notable class imbalance between malicious and benign queries, which could inherently bias the classifier toward the majority class.

Random under sampling was applied to match the majority and minority classes to achieve statistical equilibrium. Although oversampling techniques were initially explored, they induced severe overfitting owing to sample duplication and failed to yield performance improvements. Following the under sampling process, a perfectly balanced dataset consisting of 9327 queries per class was established to ensure an unbiased representation during model training and evaluation.

### E. Data Splitting

The dataset was split into training and testing sets using an 80/20 ratio with a fixed random seed (random_state = 50) to ensure the reproducibility of the results. The training data was further divided into training and validation sets using an 80/20 split (random_state = 42) to monitor the model performance during training. The 80/20 split ratio was selected because it is commonly used in ML studies to provide sufficient training data while preserving an independent testing set for the evaluation.

### F. Model Training

In this study, a standard PNN was employed as the baseline model. To increase its performance, two modifications are proposed: the first uses BO to tune the model parameters (PNN-BO), and the second introduces an enhanced PNN architecture with additional layers (Enhanced PNN). These modifications are the primary contributions of this study. In addition to the PNN models, ANN and CNN models were used for comparative evaluation. To ensure a fair comparison, all the models were trained on the same dataset under identical training conditions. Table III summarizes the architecture and configurations of all models used in this study.

All models were trained using the Adam optimizer, binary cross-entropy loss, a batch size of 32, and 100 epochs each. These settings were selected to ensure stable training and a fair comparison across all the evaluated models.

#### 1) PNN

The PNN model used in this study is based on a Gaussian kernel function that measures the similarities between the input features. The model consists of a kernel-based layer, followed by a summation layer and a sigmoid output layer for binary classification. This structure enables efficient classification while maintaining a low computational complexity.

#### 2) PNN-BO

The PNN-BO model extends the standard PNN by applying BO to tune the bandwidth parameter of a Gaussian kernel. The optimization process aimed to improve the classification accuracy by automatically selecting an appropriate bandwidth value. The search space for the bandwidth parameter was defined between 0.1 and 2.0, whereas the learning rate search space ranged from 0.0001 to 0.1. The objective of the optimization was to maximize the classification accuracy during training. Bayesian Optimization was performed using five initial random evaluations, followed by ten optimization iterations. This approach enables efficient parameter exploration with fewer evaluations than traditional search methods. The optimal bandwidth obtained through BO was approximately 1.411. Fig. 3 illustrates the workflow of the proposed PNN-BO model.

TABLE III. MODEL ARCHITECTURE AND CONFIGURATION

| Model | Architecture | Activation Functions | Regularization |
|---|---|---|---|
| ANN | Dense (128) → Dense (64) → Dense (32) → Dense (16) → Dense (1) | ReLU (hidden), Sigmoid (output) | **None** |
| CNN | Conv1D (64, $k$ = 3) → MaxPooling → Conv1D (128, $k$ = 3) → MaxPooling → Flatten → Dense (128) → Dense (1) | ReLU (hidden), Sigmoid (output) | Dropout (0.5) |
| PNN | PNN Layer (Gaussian Kernel) → Summation Layer → Dense (1) | Gaussian kernel, Sigmoid | None |
| PNN-BO | PNN Layer (optimized bandwidth) → Summation Layer → Dense (1) | Gaussian kernel, Sigmoid | None |
| Enhanced-PNN | PNN Layer → Summation → Dense (64) → BatchNorm → Dropout (0.3) → Dense (1) | ReLU, Sigmoid | BatchNorm + Dropout |

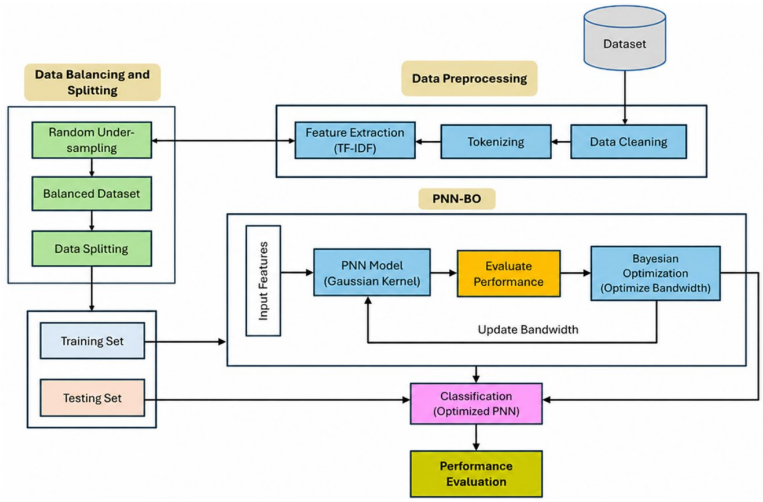


Fig. 3. PNN-BO model architecture.

*3) Enhanced PNN*

The Enhanced-PNN extends the standard PNN by adding additional layers after the summation layer to improve the learning capability and generalization performance. A dense layer with 64 neurons and Rectified Linear Unit (ReLU) activation was introduced to improve the feature representation and nonlinear learning. Batch normalization was applied to stabilize the training and accelerate convergence by reducing internal distribution shifts. In addition, a dropout layer with a rate of 0.3 was used to reduce overfitting, while preserving sufficient learning capacity. The final output layer uses a sigmoid activation function for binary classification. Fig. 4 illustrates the workflow of the proposed Enhanced-PNN layers.

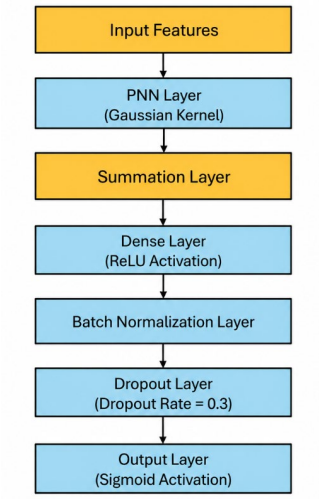


Fig 4. Enhanced-PNN layers workflow.

*4) ANN*

The ANN model consists of numerous fully connected layers that process the input features and learn complex correlations in the data. The output layer uses a sigmoid activation function for binary classifications. This structure allows the model to capture nonlinear patterns for accurate categorization.

*5) CNN*

The CNN model consists of one-dimensional convolutional layers, followed by max-pooling layers for feature extraction and fully connected layers for classification. Dropout was used to reduce overfitting.

*G. Model Evaluation*

The evaluation metrics were calculated based on the confusion matrix. Accuracy was used to measure the overall classification performance of the models. Precision, recall, and F1-score were obtained from the classification report using macro-averaging, which assigns equal importance to both the normal and malicious classes. This evaluation strategy was adopted because the dataset was balanced using random undersampling, ensuring that both classes contributed equally to the reported performance. In addition, recall is particularly important in SQL injection detection because it reflects the model's ability to correctly identify malicious queries, thereby reducing false negatives that could lead to serious security risks. The following equations demonstrate the evaluation metrics.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (4)$$

where:

- $TP$ = True Positive.
- $TN$ = True Negative.
- $FP$ = False Positive.
- $FN$ = False Negative.

## IV. Result and Discussion

This section presents the experimental results obtained from the models evaluated. The performance was analyzed using standard evaluation metrics, including accuracy, precision, recall, and F1-Score. In addition, the training time and model complexity were considered to provide a comprehensive comparison. Table IV highlights the performance of all evaluated models.

TABLE IV. Overall Performance Evaluation Results of the Models

| Model | Accuracy (%) | Precision (%) | Recall (%) | F1-Score (%) |
|---|---|---|---|---|
| ANN | 99.28 | 99.29 | 99.27 | 99.28 |
| CNN | 99.20 | 99.20 | 99.19 | 99.20 |
| PNN | 97.45 | 97.47 | 97.46 | 97.45 |
| PNN+BO | 98.66 | 98.68 | 98.65 | 98.66 |
| Enhanced-PNN | 99.22 | 99.24 | 99.22 | 99.22 |

Note: Precision, Recall and F1-score are reported as macro-average values.

As shown in Table IV, ANN, CNN, and Enhanced-PNN performed similarly in terms of accuracy and other evaluation metrics. The Enhanced-PNN achieved an accuracy of 99.22%, which is comparable to that of ANN (99.28%) and CNN (99.20%). The strong performance of the Enhanced-PNN can be attributed to the additional dense, batch normalization, and dropout layers. The dense layer improves nonlinear feature learning and representation capability, whereas batch normalization stabilizes training and accelerates convergence. In addition, the dropout layer reduces overfitting and improves the model's ability to generalize to previously unseen queries. These enhancements contributed to

improved classification performance compared to the standard PNN model. The PNN-BO model achieved a higher performance than the standard PNN because of the optimization of the Gaussian kernel bandwidth parameter using BO. Selecting an appropriate bandwidth improves the separation between normal and malicious query distributions, leading to better classification accuracy. The optimal bandwidth obtained through BO was approximately 1.411. The optimized model improved the accuracy from 97.45% to 98.66%, representing an improvement of approximately 1.21%. Although seemingly small, this 1.21% increase is highly significant, as it reduces the model's overall error rate by nearly half, making it crucial for critical cybersecurity environments.

To further analyze the model performance, the confusion matrix values are presented in Table V.

TABLE V. CONFUSION MATRIX VALUES FOR ALL MODELS

| Model | TP | TN | FP | FN |
|---|---|---|---|---|
| ANN | 1881 | 1823 | 1 | 26 |
| CNN | 1875 | 1826 | 7 | 23 |
| PNN | 1815 | 1821 | 67 | 28 |
| PNN-BO | 1872 | 1809 | 10 | 40 |
| Enhanced-PNN | 1881 | 1821 | 1 | 28 |

The low FN number indicates that the model effectively detects SQL injection attacks, which is crucial for cybersecurity applications. For the SQL injection (malicious) class, Enhanced-PNN achieved a class-specific recall of 98.49%, indicating that only 28 malicious queries were misclassified as benign. The overall recall reported in Table IV (99.22%) corresponds to macro-average recall calculated across both classes. In addition to the performance metrics, computational efficiency was measured by the training time and number of parameters, as shown in Table VI.

TABLE VI. MODEL COMPLEXITY AND TRAINING TIME

| Model | Parameters | Time (s) |
|---|---|---|
| ANN | 139,009 | 2–5 |
| CNN | 41,601 | 2 |
| PNN | 2002 | 1–4 |
| PNN-BO | 2002 | 1–3 |
| Enhanced-PNN | 2321 | 1–3 |

The results, as shown in Table VI, demonstrate that the PNN models have fewer parameters and require less training time than other DL models, such as CNN. Although ANN and CNN achieved comparable classification performance, the proposed PNN-based models required fewer trainable parameters and a shorter training time, making them more suitable for lightweight cybersecurity applications. This demonstrates the efficiency of the proposed approach.

The 95% confidence intervals were calculated using the Wilson method for the accuracy, precision, and recall of each model. The results, presented in Table VII, show that the confidence intervals for ANN, CNN, and Enhanced-PNN overlap across several evaluation metrics, suggesting that the observed performance differences may not be statistically significant. Therefore, the superiority of one model over the other cannot be conclusively determined. Nevertheless, the proposed PNN-based models achieved competitive performance while maintaining lower computational complexity and faster training, making them suitable for lightweight cybersecurity applications.

TABLE VII. PERFORMANCE METRICS WITH 95% CONFIDENCE INTERVAL

| Model | Accuracy | Precision | Recall |
|---|---|---|---|
| ANN | 99.28% [99.01–99.55] | 99.29% [99.00–99.58] | 99.27% [98.98–99.56] |
| CNN | 99.20% [98.92–99.48] | 99.20% [98.91–99.49] | 99.19% [98.90–99.48] |
| PNN | 97.45% [96.94–97.96] | 97.47% [96.91–98.03] | 97.46% [96.90–98.02] |
| PNN-BO | 98.66% [98.29–99.03] | 98.68% [98.30–99.06] | 98.65% [98.27–99.03] |
| Enhanced-PNN | 99.22% [98.94–99.50] | 99.24% [98.96–99.52] | 99.22% [98.94–99.50] |

## V. CONCLUSION

This study presents a method for detecting SQL injection attacks using PNN-based models. In addition to the standard PNN, two improved versions are presented, including the PNN-BO and Enhanced-PNN architectures. The experimental results show that the proposed models achieve competitive performance while maintaining computational efficiency in terms of training time and model complexity. However, owing to overlapping confidence intervals among some models, the observed performance differences may be statistically insignificant and should be evaluated with caution. Compared to DL models, PNN-based techniques require less training time and fewer parameters.

However, the use of a single dataset limited the generalizability of the study. Future work will focus on evaluating the proposed models on larger and more diversified datasets containing more sophisticated attributes and a wider range of attack patterns. In addition, further optimization strategies will be investigated to improve the model's performance.

## CONFLICT OF INTEREST

The authors declare no conflict of interest.

## AUTHOR CONTRIBUTIONS

J.A. conducted the research, designed and implemented the proposed model, performed the experiments, analyzed the results, wrote the original manuscript, and reviewed and validated the experimental results. J.J. supervised the research, provided methodological guidance, reviewed the manuscript, and contributed to improving the logical flow and clarity of ideas. All the authors have reviewed and approved the final manuscript.

## REFERENCES


[1] S. T. Echeverri, "Evaluation of SQL injection (SQLi) attack detection strategies in web applications using machine learning," Undergraduate thesis, Engineering Faculty, Universidad de Antioquia, Colombia, 2024.

[2] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Deep learning architecture for detecting SQL injection attacks based on RNN autoencoder model," *Mathematics*, vol. 11, no. 15, 3286, Aug. 2023.

[3] E. Peralta-Garcia, J. Quevedo-Monsalbe, V. Tuesta-Monteza, and J. Arcila-Diaz, "Detecting structured query language injections in web microservices using machine learning," *Informatics*, vol. 11, no. 2, 15, Jun. 2024. doi: 10.3390/informatics11020015

[4] F. K. Alarfaj and N. A. Khan, "Enhancing the performance of SQL injection attack detection through probabilistic neural networks," *Applied Sciences*, vol. 13, no. 7, 4365, Apr. 2023.

[5] S. M. Kasongo and Y. Sun, "A deep learning method with filter based feature engineering for wireless intrusion detection system," *IEEE Access*, vol. 7, pp. 38597–38607, 2019.

[6] S. Al-Emadi, A. Al-Mohannadi, and F. Al-Senaid, "Using deep learning techniques for network intrusion detection," in *Proc. 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, Feb. 2020, pp. 171–176. doi: 10.1109/ICIoT48696.2020.9089524

[7] S. Saraeian and M. M. Golchi, "Application of deep learning technique in an intrusion detection system," *Int. J. Comput. Intell. Appl.*, vol. 19, no. 2, 2050016, Jun. 2020. doi: 10.1142/S1469026820500169

[8] C. Liu, Z. Gu, and J. Wang, "A hybrid intrusion detection system based on scalable k-means+ random forest and deep learning," *IEEE Access*, vol. 9, pp. 75729–75740, 2021. doi: 10.1109/ACCESS.2021.3082147

[9] L. Ashiku and C. Dagli, "Network intrusion detection system using deep learning," *Procedia Computer Science*, vol. 185, pp. 239–247, 2021. doi: 10.1016/j.procs.2021.05.025

[10] A. Aldallal, "Toward efficient intrusion detection system using hybrid deep learning approach," *Symmetry (Basel).*, vol. 14, no. 9, 1916, Sep. 2022. doi: 10.3390/sym14091916

[11] S. M. Kasongo, "A deep learning technique for intrusion detection system using a Recurrent Neural Networks based framework," *Comput. Commun.*, vol. 199, pp. 113–125, Feb. 2023. doi: 10.1016/j.comcom.2022.12.010

[12] A. Henry *et al.*, "Composition of hybrid deep learning model and feature optimization for intrusion detection system," *Sensors*, vol. 23, no. 2, 890, Jan. 2023. doi: 10.3390/s23020890

[13] W. H. Aljuaid and Alshamrani, "A deep learning approach for intrusion detection systems in cloud computing environments," *Appl. Sci. (Switzerland)*, vol. 14, no. 13, 5381, Jul. 2024.

[14] University of New Brunswick. (2026). ISCX NSL-KDD dataset. [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html

[15] University of New Brunswick. (2026). Intrusion detection evaluation dataset (ISCXIDS2012). [Online]. Available: https://www.unb.ca/cic/datasets/ids.html

[16] Canadian Institute for Cybersecurity. (2026). Intrusion detection evaluation dataset (CIC-IDS2017). [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html

[17] N. Moustafa and J. Slay. (2026). UNSW-NB15 Dataset. *Canadian Institute for Cybersecurity*. [Online]. Available: https://research.unsw.edu.au/projects/unsw-nb15-dataset

[18] Canadian Institute for Cybersecurity. (2026). CSE-CIC-IDS2018 Dataset. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2018.html

[19] N. Gandhi, J. Patel, R. Sisodiya, N. Doshi, and S. Mishra, "A CNN-BiLSTM based approach for detection of SQL injection attacks," in *Proc. 2nd IEEE International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, Mar. 2021, pp. 378–383. doi: 10.1109/ICCIKE51210.2021.9410675

[20] H. Sun, Y. Du, and Q. Li, "Deep learning-based detection technology for SQL injection research and implementation," *Applied Sciences (Switzerland)*, vol. 13, no. 16, 9466, Aug. 2023. doi: 10.3390/app13169466

[21] W. Zhang *et al.*, "Deep neural network-based SQL injection detection method," *Security and Communication Networks*, vol. 2022, no. 1, 4836289, 2022. doi: 10.1155/2022/4836289

[22] A. G. Kakisim, "A deep learning approach based on multi-view consensus for SQL injection detection," *Int. J. Inf. Secur.*, vol. 23, no. 2, pp. 1541–1556, Apr. 2024.

[23] J. Irungu, S. Graham, A. Girma, and T. Kacem, "Artificial Intelligence Techniques for SQL Injection Attack Detection," in *Proc. 2023 8th International Conference on Intelligent Information Technology*, Feb. 2023, pp. 38–45. doi: 10.1145/3591569.3591576

[24] Kaggle. (2026). SQL injection dataset. [Online]. Available: https://www.kaggle.com/datasets/sajid576/sql-injection-dataset

[25] B. Arasteh, B. Aghaei, B. Farzad, K. Arasteh, F. Kiani, and M. Torkamanian-Afshar, "Detecting SQL injection attacks by binary gray wolf optimizer and machine learning algorithms," *Neural Comput. Appl.*, vol. 36, no. 12, pp. 6771–6792, Apr. 2024. doi: 10.1007/s00521-024-09429-z

[26] N. Alotibi and M. Alshammari, "Deep Learning-based Intrusion detection: A novel approach for identifying brute-force attacks on FTP and SSH protocol," *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 6, 2023. doi: 10.14569/IJACSA.2023.0140612

[27] R. Pahalavan. (2026). NIDS Datasets. *GitHub*. [Online]. Available: https://github.com/rdpahalavan/nids-datasets