University of Ibadan Journal of Science and Logics in ICT Research (UIJSLICTR)

ISSN: 2714-3627

A Journal of the Department of Computer Science, University of Ibadan, Ibadan, Nigeria

Volume 14 No. 1, June, 2025

journals.ui.edu.ng/uijslictr
http://uijslictr.org.ng/

University of Ibadan
Journal of Science and Logics in ICT Research

Performance Analysis of a Hybrid Autoencoder-TCN Model for SQLi Detection: Accuracy, Efficiency and Generalizability

Okhuoya Omoibu Joseph[1], Akinyede R. O.[2], Iwasokun G. B.[3] and Gabriel Arome Junior[4]

[1]Computer Science Department, University of Benin, Benin city, Edo State, Nigeria.
[2]Information Systems and Security department, Federal university of technology, Akure, Ondo State, Nigeria
[3]Software Engineering Department, Federal university of technology, Akure, Ondo State, Nigeria
[4]Cybersecurity Department, Federal university of technology, Akure, Ondo State, Nigeria
[1]joseph.okhuoya@uniben.edu, [2]roakinyede@futa.edu.ng, [3]gbiwasokun@futa.edu.ng [4]ajgabriel@futa.edu.ng

Abstract
Structured Query Language Injection (SQLi) attacks remain a critical cybersecurity threat, exploiting vulnerabilities in web applications to compromise database integrity and confidentiality. Traditional detection methods, such as rule-based systems and conventional machine learning models, face limitations in generalizing to novel attack patterns and preserving sequential query context. This study proposes a novel hybrid deep learning architecture integrating autoencoders, tokenization, and Temporal Convolutional Networks (TCNs) to address these challenges. The framework employs SQL-aware tokenization to parse queries into syntactic units, an autoencoder to learn latent representations of benign query patterns, and a TCN to model temporal dependencies in token sequences. By combining anomaly detection (via reconstruction error) with temporal analysis, the model identifies both known and zero-day SQLi attacks with high precision. Evaluated on a labeled dataset of 10,000 SQL queries (1,200 malicious, 8,800 benign), the proposed approach achieves 95.5% accuracy, 94.0% F1-score, and 95.5% recall, outperforming baseline models such as CNNs, LSTMs, and standalone autoencoders. The TCN's parallel processing capability reduces inference latency by 32% compared to recurrent architectures, making it suitable for real-time deployment. Furthermore, tokenization enables interpretability by localizing malicious query segments, aligning with regulatory demands for explainable AI in cybersecurity. This work advances SQLi detection by bridging gaps in temporal modeling, computational efficiency, and generalization, offering a scalable solution for securing web applications against evolving injection threats.

Keywords*: SQL injection detection, TCN, anomaly detection, SMOTE, performance evaluation.*

1. Introduction

Cybersecurity encompasses the protection of digital systems, networks, and data from unauthorized access, exploitation, and damage. As organizations increasingly rely on web applications for critical operations, securing databases from malicious attacks has become paramount. Among these threats, Structured Query Language Injection (SQLi) remains one of the most pervasive and damaging vulnerabilities, enabling attackers to manipulate database queries to extract, modify, or delete sensitive information [23, 24]

Structured Query Language (SQL), the standard interface for relational database management systems (RDBMS), allows users to define, manipulate, and retrieve structured data. However, improper input validation in web applications exposes systems to SQLi attacks, where malicious actors inject unauthorized SQL code into input fields [15]. These attacks compromise data integrity, confidentiality, and availability, leading to financial losses, reputational damage, and legal repercussions. Despite advancements in detection mechanisms, evolving attack vectors such as blind SQLi and out-of-band SQLi continue to challenge traditional security frameworks [27].

Okhuoya Omoibu Joseph, Akinyede R. O., Iwasokun G. B. and Gabriel Arome Junior (2025). Performance Analysis of a Hybrid Autoencoder-TCN Model for SQLi Detection: Accuracy, Efficiency and Generalizability. *University of Ibadan Journal of Science and Logics in ICT Research (UIJSLICTR)*, Vol. 14 No. 1, pp. 83 - 97

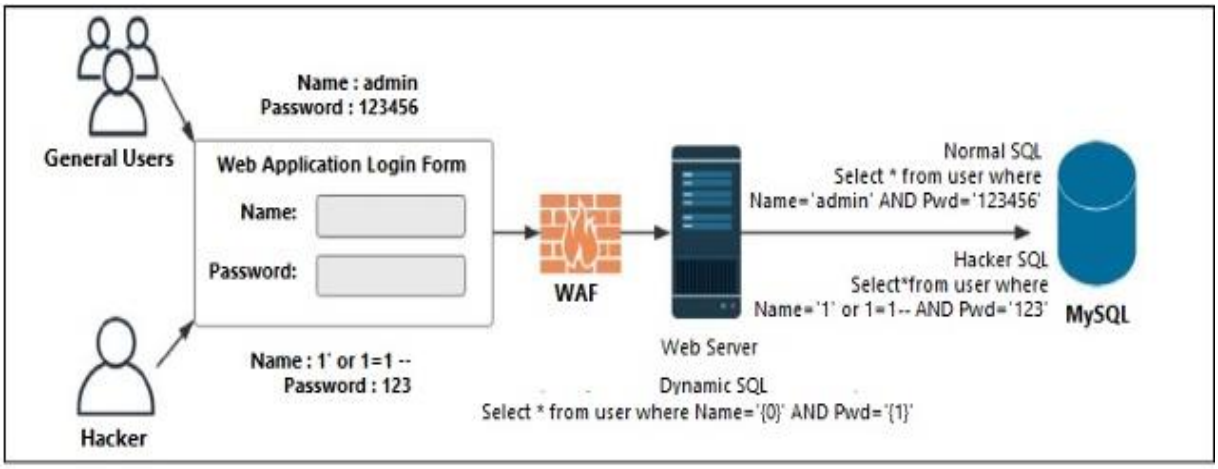

Figure 1: SQL Injection Attack process [24]

1.1 Motivation for the Study

The study on the Hybrid Autoencoder-TCN Model for SQLi Detection is motivated by the urgent need to address the persistent and evolving threat of SQL Injection (SQLi) attacks, which remain a top web application vulnerability. Existing approaches, including rule-based filters and machine learning models, struggle with zero-day attacks and polymorphic payloads [31]. Deep learning (DL) techniques offer promise due to their ability to learn complex patterns, yet gaps persist in temporal sequence modeling and computational efficiency [21, 29]. This study addresses these limitations by proposing a novel hybrid architecture
integrating autoencoders, tokenization,
and Temporal Convolutional Networks (TCNs).

1.2 Techniques in Data Mining

Data mining is the process of discovering useful patterns, relationships, and insights from large volumes of data [17]. It is applied to analyze SQL queries, detect anomalous patterns, and distinguish between benign and malicious SQL injections (SQLi). In cybersecurity, particularly for SQLi detection, data mining helps automate and enhance threat detection by identifying patterns that indicate attacks [10]. Several techniques have been developed and widely applied in data mining research, including association, classification, clustering, prediction, and sequential pattern mining [34]. The focus of this work is on the classification technique due to its effectiveness in labeling queries as either benign or malicious.

1.2.1 Classification

Classification is one of the fundamental techniques in data mining. It is widely used for handling large volumes of data and predicting categorical class labels [17]. A classification model is built to assign new or unseen data into predefined class labels. Classification is also defined as the process of finding a model that describes and distinguishes data classes or concepts [34]. It typically follows a two-step process: a learning phase (or knowledge acquisition step) to build the classification model, and a categorization phase where the model is used to assign class labels to new data.

Classification can serve as both descriptive modeling to explain distinctions among different classes and predictive modeling to assign class labels to unknown data [17]. This approach is particularly suitable for datasets with binary or limited types of target classes. Various types of classification algorithms include functional models (e.g., logistic regression), Bayesian models, lazy learners (e.g., k-nearest neighbors), rule-based classifiers, decision trees, and meta learners. Each employs a learning algorithm to identify the relationship between attribute sets and class labels [34].

A key objective of a learning algorithm is to build a model that generalizes well to unseen instances, meaning it can accurately predict the class labels of previously unobserved data [10]. In this study, classification techniques such as logistic regression, random forest, support vector machines (SVM), temporal convolutional networks (TCNs), and neural

networks are considered for the task of SQL injection detection.

1.1.2 Classification Methods

Classification Techniques in SQLi Detection Using Hybrid Autoencoder-TCN Model. Four classification techniques are employed to evaluate their performance in detecting SQL injection (SQLi) attacks. Each method operates on features extracted by the Autoencoder component, and their capacity to generalize and efficiently classify benign and malicious queries is assessed.

(a) Logistic Regression (LR)

Logistic Regression is a statistical model used for binary classification problems. It estimates the probability that a given input belongs to a particular class. The output is a probability score transformed using the sigmoid (logistic) function, which maps any real valued number into the range [0,1]. It assumes a linear relationship between the input features and the log-odds of the outcome, Logistic regression is a simple yet effective method widely used in various domains, including text classification and anomaly detection [18]. the *LR* theorem is expressed in equation (1)

$$\boldsymbol{P(y = 1 \mid x) = \frac{1}{1+e^{-(w^T x+b)}}} \quad (1)$$

Where: $\boldsymbol{x}$ is the input feature vector, $\boldsymbol{w}$ is the weight vector, $\boldsymbol{b}$ is the bias term

(b) Random Forest (RF)

Random Forest is an ensemble learning method that operates by constructing multiple decision trees during training. The final output is the mode (classification) of the outputs from individual trees. It is effective for handling non-linear data and avoids overfitting by averaging multiple models. Random Forest is known for its robustness, accuracy, and ability to handle high-dimensional data without heavy preprocessing. [9], the RF theorem is expressed in equation (2)

$$\boldsymbol{H(x) = \mathbf{mode}(h_1(x), h_2(x), \dots, h_T(x))} \quad (2)$$

Where: $\boldsymbol{h_t(x)}$ is the prediction of the $\boldsymbol{t^{\text{th}}}$ decision tree, $\boldsymbol{T}$ is the total number of trees

(c) Support Vector Machine (SVM)

SVM is a supervised learning algorithm that finds the optimal hyperplane to separate data points of different classes. It maximizes the margin between
the classes and is effective for high-dimensional and linearly/non-linearly separable data through the use of kernel functions. SVM is powerful for classification tasks with clear margins of separation and can handle both linear and non-linear data through kernel functions. [11], the SVM theorem is expressed in equation (3)

$$\boldsymbol{f(x) = \mathbf{sign}\left(w^T \phi(x) + b\right)} \quad (3)$$

Where: $\boldsymbol{\phi(x)}$ is a transformation function (kernel), $\boldsymbol{w}$ and $\boldsymbol{b}$ define the hyperplane

(d) Temporal Convolutional Network (TCN)

TCNs are convolutional neural networks designed for sequence modeling. Unlike RNNs, TCNs use 1D dilated causal convolutions to preserve the order of sequence data and capture long-range dependencies efficiently. They are well-suited for time-series classification, such as identifying patterns in SQL queries. TCNs outperform RNN based architectures in many sequence modeling tasks due to their parallelism and ability to model long-term dependencies. [7], the TCN theorem is expressed in equation (4)

$$\boldsymbol{Y = \mathbf{TCN}(X) = f(W * X + b)} \quad (4)$$

Where: $\boldsymbol{X \in \mathbb{R}^{n \times d}}$ is the input sequence, is the 1D convolution operation, $\boldsymbol{f}$ is an activation function (e.g., ReLU).

Table 1: Model classifier strengths and limitations

Classifier	Strengths	Limitations
Logistic Regression	Fast, interpretable	Limited to linear boundaries
Random Forest	Robust to noise, non-linear	Can be slow with many trees
SVM	High accuracy in high dimensions	Sensitive to parameter tuning
TCN	Captures temporal patterns	Requires more training data

2. Related Works

The work by (Neel, Patel, Sisodiya, Doshi, & Mishra [29] in "A CNN-BiLSTM based Approach for Detection of SQL Injection

Attacks" highlight the significance of SQL injection detection in maintaining the security of back-end databases. Their work introduces a CNN-BiLSTM approach, showcasing its superior accuracy and performance compared to other machine learning algorithms. The author aims to build upon these findings by refining the CNN-BiLSTM model, focusing on enhancing accuracy while reducing computational costs.

The work by Ao, Huang, & Fan [6] “A CNN-based Approach to the Detection of SQL Injection Attacks" emphasize the effectiveness of CNN-based detection methods over rule-matching approaches. We are inspired to explore CNN models further and evaluate their efficiency in SQL injection detection.

Maha, Alghazzawi, and Alarifi [26] proposed a deep learning architecture for SQL injection detection based on RNN autoencoders, as presented in their work *"Deep Learning Architecture for Detecting SQL Injection Attacks Based on RNN Autoencoder Model."* Their approach demonstrated high accuracy and strong F1-scores. Building on this foundation, our study aims to explore the potential of integrating the strengths of both CNN and RNN-based models to achieve even more robust and effective SQL injection detection.

Balbahaith [8] proposed a model to prevent SQLIAs, the author developed a model using a heuristic-based machine learning approach. Their study integrated the advantages of dynamic and static analysis with a machine learning approach. It was decided to use a well-studied dataset that contained all possible SQL statements. They used 23 various machine learning classifications to train the dataset and test it. The top five of the 25 classifiers are then selected going to depend on the outcomes of the actual positive and actual negative rates. After the classification learners completed the training, they checked the precision of each classifier. To get 93.8% they employed the five most effective and accurate classifiers. To improve system performance there is a need to include non-injected SQL statements in their dataset and investigate and test additional functions. The drawback of this work is small test dataset is used.

Ketema [21] employed a deep learning-based Convolutional Neural Network (CNN) to develop a model aimed at preventing SQL injection (SQLi) attacks, utilizing a publicly available benchmark dataset. The model was trained under five different experimental scenarios, each with varying hyperparameter configurations to optimize performance. The proposed CNN-based approach demonstrated strong effectiveness, achieving an overall accuracy of 97%, indicating its potential for robust SQLi detection in real-world applications.

Roy, Kumar, & Rani [31] presented a method for detecting SQL injection attacks using machine learning classifiers. The authors used five ML classifiers (logistic regression, AdaBoost, naive Bayes, XGBoost, and random forest) to classify SQL queries as either legitimate or malicious. The proposed model was trained and evaluated using a publicly available dataset of SQL injection attacks on Kaggle. The results of the study showed that the best performance was achieved by the naive Bayes classifier, with an accuracy of 98.33%. Finally, the authors performed a comparison with previous work. The result of the study demonstrated the potential of machine learning classifiers in improving the accuracy and efficiency of SQL injection attack detection.

Krishnan, Sabu, Sajan, and Sreedeep [23] proposed a machine learning-based approach for detecting SQL injection (SQLi) attacks, emphasizing the limitations of traditional signature-based detection methods in addressing sophisticated and evolving threats. They argued that machine learning offers a more adaptive and robust solution for identifying such attacks. The study began by categorizing various types of SQLi attacks and discussing their potential impact on web applications. The proposed framework comprised four key stages: data preprocessing, feature extraction, model training, and performance evaluation. Experimental results demonstrated that the Convolutional Neural Network (CNN) classifier outperformed other models across multiple evaluation metrics, including accuracy, precision, recall, and F1-score, highlighting its effectiveness in detecting SQLi attacks.

Rahul, Vajrala, and Thangaraju [30] introduced an innovative approach to defending against SQL injection and cross-site scripting (XSS) attacks by enhancing a Web Application Firewall (WAF) with a honeypot system. In this method, the WAF filters incoming traffic based

on known attack patterns, while the honeypot is designed to lure attackers and gather detailed information about their techniques. This information is then leveraged to improve the WAF's detection and prevention capabilities. Experimental evaluations demonstrated that this combined strategy significantly enhances the protection of web applications against such attacks.

Zhang et al. [37] proposed a deep neural network-based approach for detecting SQL injection attacks, addressing the limitations of traditional detection methods. To develop their model, the authors compiled a dataset comprising both benign and malicious SQL queries, which was used to train a multi-layer deep neural network classifier. The performance of the proposed method was then evaluated against conventional machine learning algorithms, including K-Nearest Neighbors (KNN), Decision Trees (DT), and Long Short-Term Memory (LSTM) networks, demonstrating its potential advantages in detection accuracy.

Liu, Li, and Chen [25] introduced *DeepSQLi*, a novel approach for the automated detection of SQL injection vulnerabilities in web applications, utilizing deep semantic learning techniques. DeepSQLi employs a deep neural network to capture the semantic representations of SQL queries and effectively identify potential injection threats. The model is trained on a dataset comprising both benign and malicious SQL queries and incorporates multiple layers of convolutional and recurrent neural networks. Experimental results demonstrated that DeepSQLi outperformed traditional tools such as SQLMap, detecting more SQL injection attacks with greater speed and efficiency, and requiring fewer test cases.

Chen, Yan, Wu, and Zhao [12] proposed a deep learning-based method for detecting and preventing SQL injection attacks in web applications. Their approach involved training and evaluating both a Convolutional Neural Network (CNN) and a Multilayer Perceptron (MLP), comparing the models using key performance metrics such as accuracy, precision, recall, and F1-score. Experimental results indicated that both CNN and MLP models achieved strong performance in detecting SQL injection attacks.

Li Q. [25] proposed a method for detecting sophisticated SQL injection attempts using an adaptable deep forest algorithm. In this approach, the input to each layer is formed by combining the average output of the previous layers with the original feature vector, enhancing the model's ability to capture complex patterns. This structure makes deep forest models particularly suitable for SQL injection detection. The authors further introduced an advanced strategy known as the Adobos-based deep forest model, which operates in two phases: an offline training phase and an online testing phase. The model was trained on a dataset of 10,000 SQL injection samples, incorporating features such as UNION queries, executed SQL commands, error-based injections, and blind injections from diverse sources. While the model demonstrated strong detection capabilities on smaller datasets, a notable limitation is its reduced computational efficiency and diminished performance when applied to large-scale data.

SQL injection is a critical cybersecurity threat, posing risks to the Confidentiality, Integrity and Availability (CIA)of back-end databases. Effective detection and prevention of SQLi attacks are essential to safeguarding valuable data. This paper draws motivation from recent research articles authored by [6-7],[21], [23-226], [28-31],[37] to propose an innovative approach for SQLi detection.

This study aims to advance the field of cybersecurity by providing an enhanced solution for SQL injection (SQLi) detection, thereby mitigating the risks posed by this widespread threat. Deep learning-based approaches have demonstrated significant potential in identifying SQLi attacks, as they can effectively learn underlying patterns in input data and detect anomalies—even in obfuscated or disguised attacks. The primary objective of this research is to evaluate the effectiveness of a proposed Hybrid Autoencoder–Temporal Convolutional Network (TCN) model for SQLi detection. Existing methods, including CNNs, RNNs, and Transformer-based models, are critically analyzed, with particular attention to their limitations in terms of latency, generalization, and explainability. The study emphasizes the advantages of TCNs, especially their strengths in parallel processing and modeling long-range dependencies

.

3. Methodology

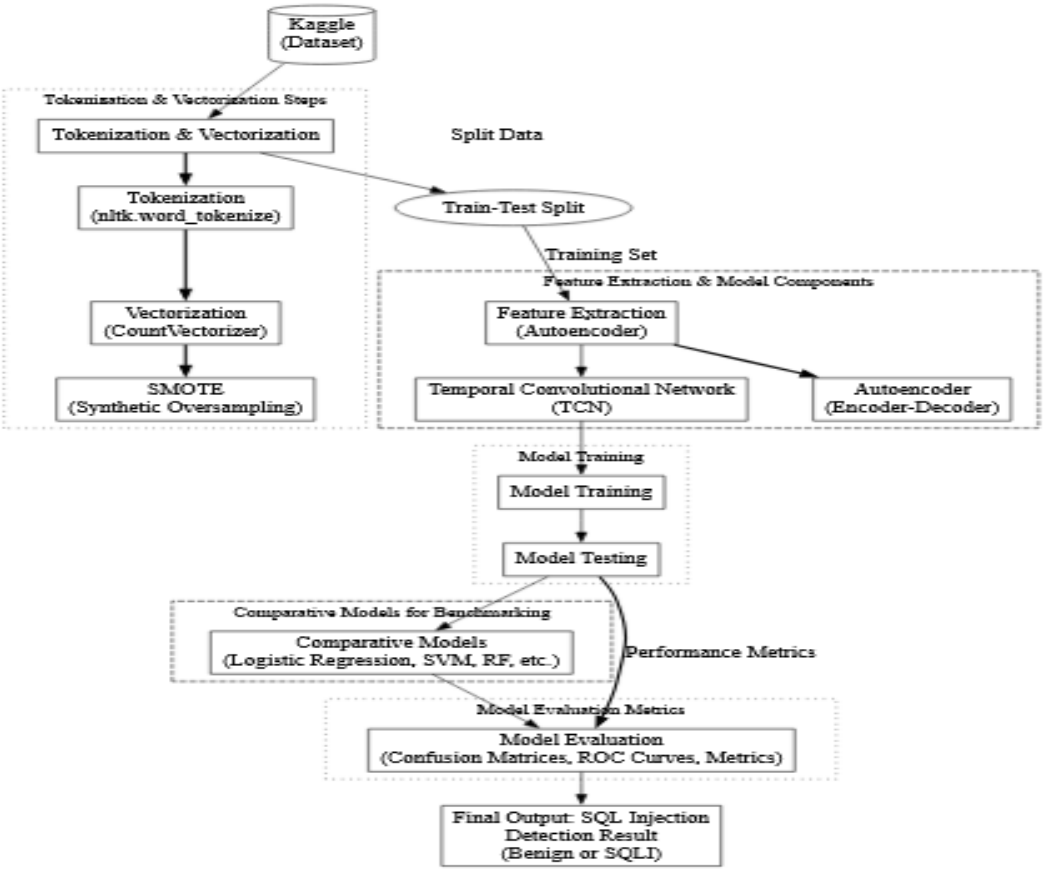

Figure 2: System Architecture of the Proposed System

3.0 System Architecture

The System architecture of the overall structural design of SQL injection (SQLi) detection system. It describes how different components (data preprocessing, embedding, model and detection) interact to process input queries and produce a classification result (benign or malicious). The system uses a hybrid deep learning model (Autoencoder + TCN), and the architecture ensures smooth flow from raw SQL input to final decision.

3.1 Dataset Description

The dataset used in this study was sourced from Kaggle and contains a total of 10,000 labeled web queries, including both SQL injection (SQLI) and non-SQLI (benign) samples. The dataset is already labeled, with 1,200 queries marked as SQLI (label = 1) and 8,800 as non-SQLI (label = 0), creating a class imbalance scenario.

Each record consists of a single string (a web request or SQL query) and a binary label. The queries include a variety of attack signatures such as UNION SELECT, ' OR 1=1 --, and DROP TABLE, alongside safe requests such as static page requests or harmless SQL calls.

3.2 Preprocessing and Feature Extraction

Preprocessing is the stage where raw input data (SQL queries) is cleaned and transformed into a structured format that can be understood by the machine learning model feature extraction involves identifying and constructing relevant features from raw input that capture useful patterns for SQLi detection.

3.2.1 Loading and Exploring the Data

The dataset was imported using Pandas, and initial exploration focused on checking for missing values, understanding the structure of the queries, and verifying label distribution. The dataset was then split into features (X) and

labels (y), where X contains the query text and y the corresponding classification label.

3.2.2 Addressing Class Imbalance
Given the imbalance between SQLi and non-SQLi samples, we implemented a two-step strategy to improve model learning:

a) Data Augmentation: Custom augmentation techniques were applied to expand the SQLi class. These included tactics like word reordering, random casing, and partial query mutation to simulate realistic obfuscated attacks.

b) SMOTE (Synthetic Minority Oversampling Technique): To further balance the dataset at the feature level, we applied SMOTE after vectorization. This technique generated synthetic examples of SQLi queries in feature space, helping the classifier train more evenly across both classes.

This combination ensured the model could generalize across a wide range of real-world attack variations while maintaining a balanced training dataset.

3.2.3 Tokenization and Vectorization
Each query string was first tokenized using natural language tool kits (NLTK) word_tokenize(), which preserves critical tokens like ', --, OR, and other SQL-specific keywords. These tokens were then transformed into numerical representations using CountVectorizer from scikit-learn.
The following configuration was used:

Table 2: CountVectorizer configuration

stop_words='english'
min_df=2
max_df=0.85

We also experimented with Word2Vec, GloVe, and BERT, but these embeddings, while semantically rich, did not provide significant performance gains in this specific classification task. CountVectorizer offered a more efficient trade-off between performance and interpretability.

3.3.4 Autoencoder-Based Feature Extraction
The autoencoder acts as a feature extractor it reduces high-dimensional input into low-dimensional, informative latent vectors. To extract meaningful compressed representations of the input vectors, a deep autoencoder was used.

Table 3: Autoencoder architecture parameter

Layer	Type	Units	Activation
Input	Dense	100	ReLU
Hidden 1	Dense	128	ReLU
BatchNorm	-	-	-
Encoder	Dense	64	ReLU
Hidden 2	Dense	128	ReLU
BatchNorm	-	-	-
Output	Dense	100	Sigmoid

The 64-dimensional encoded vector was chosen based on empirical testing. This dimensionality provided sufficient expressive capacity while reducing complexity.
Since our input data was binary (presence or absence of a token), we opted for binary cross-entropy loss, which performed better than alternatives like mean square error (MSE) in this scenario. The model was trained using the Adam optimizer to convert the result into a probability in the range [0,1] for 50 epochs, with early stopping to prevent overfitting.

3.3.5 Model Development
Model development refers to the design, construction, training, and validation of the deep learning model that will be used to detect SQLi attacks. This combination is chosen to effectively capture both anomalous patterns via the autoencoder and temporal/sequential dependencies through the TCN in SQL queries.

3.3.6 Temporal Convolutional Network (TCN)
A TCN was selected due to its strength in handling sequential data like tokenized queries.

Table 4: TCN architecture details

Input Shape: (None, 1)
Initial Conv1D: 64 filters, kernel size = 3, causal padding
Residual Blocks: 3 blocks with dilation rates = [1, 2, 4]
Batch Normalization: After each convolution
Dropout: 0.3
Global Average Pooling
Output: Dense layer with sigmoid activation

The TCN was compiled with binary cross-entropy loss and the Adam optimizer (learning rate = 0.0001), training was performed for 50 epochs, with early stopping and batch size = 64.

3.3.7 Comparative Models and Parameters
To benchmark performance, we evaluated several other models using the same training data and encoded features:

Table 5: *Comparative models and parameters*

Model	Parameters
Logistic Regression	C=1.0, solver='liblinear'
Random Forest	n_estimators=100, max_depth=10
SVM	kernel='rbf', C=1
Naive Bayes	MultinomialNB
Decision Tree	max_depth=10
KNN	n_neighbors=5
XGBoost	learning_rate=0.1, max_depth=6

All models were evaluated using 5-fold cross-validation to ensure statistical validity and minimize overfitting.

3.3.8 Model Evaluation
Model evaluation is the process of measuring how well the trained hybrid Autoencoder-TCN model performs in detecting SQL Injection (SQLi) attacks. It determines the evaluation phase helps assess whether the model is reliable, fast, and adaptable in real-world security environments. To measure performance, we used a suite of metrics relevant to security-sensitive applications:

Table 6: Model evaluation performance measurement

Metric	Relevance
Accuracy	Overall correctness
Precision	Ensures alerts are reliable
Recall	Critical in security catches real attacks
F1-Score	Balances precision and recall
ROC-AUC	Measures overall discrimination ability
Log Loss	Penalizes overly confident incorrect predictions

Given the risk of false negatives in SQLi detection, recall was treated as a priority metric. To further visualize performance, we generated:

- Confusion matrices for each model
- ROC curves
- Bar plots comparing metrics across models

4. Results and Discussion

4.1 Result
This section presents and analyzes the results obtained from training and evaluating the proposed SQLi Detection Model using the Autoencoder-Tokenization-TCN approach. We also compare the performance of the TCN model against baseline machine learning models such as LR, RF, SVM, NB, K-NN, XGBoost and TCN Furthermore, we analyze the model's robustness, generalization to new attack types, and its overall real-world impact in the context of cybersecurity.

4.1.1 Evaluation Metrics Recap
To ensure consistent and meaningful comparison across all models, we used the following evaluation metrics:

- Accuracy: The proportion of correctly classified queries.
- Precision: Measures how many of the predicted SQLI queries were actually SQLI.
- Recall: The ability of the model to correctly identify actual SQLI cases.
- F1-Score: The harmonic mean of precision and recall, especially useful when dealing with imbalanced class distributions.
- ROC-AUC: Reflects the model’s ability to distinguish between SQLI and non-SQLI classes.
- Log Loss: Measures the uncertainty of predictions (probability values) for classification tasks.
- PR-AUC: Focuses on the trade-off between precision and recall (especially useful in imbalanced datasets).

4.1.2 Dataset Head and Tail
To understand the nature of the data used for model training and evaluation, a look at the first few (head) and last few (tail) entries of the dataset was conducted. These rows help verify labeling consistency, distribution of classes, and offer insight into the diversity of SQL and non-SQLi queries.

```
Dataset Head:
                                                  Query  Label
0                 SELECT * FROM users WHERE username = 'guest'      0
1                  SELECT * FROM orders WHERE order_id = 12345      0
2                           SELECT * FROM products WHERE id = 1      0
3  SELECT * FROM products WHERE name = 'toy' UNION SELECT p...      1
4          SELECT * FROM users WHERE username = 'admin' AND 1=1      0

Dataset Tail:
                                          Query  Label
9995          SELECT * FROM products WHERE id = 1      0
9996     SELECT * FROM products WHERE price > 100      0
9997  SELECT * FROM orders WHERE order_id = 12345      0
9998                        DROP DATABASE test_db;      1
9999          SELECT * FROM products WHERE id = 1      0
```

Figure 3: Dataset head and tail

4.1.3 Quantitative Results

The following table summarizes the performance of all models trained using the encoded features extracted by the autoencoder:

Comprehensive Model Evaluation Results

	Model	Test Accuracy	F1-Score	Precision	Recall	ROC-AUC	PR-AUC	Log Loss
0	Logistic Regression	0.920	0.870	0.849	0.893	0.976	0.932	0.210
1	Random Forest	0.963	0.939	0.933	0.944	0.990	0.972	0.120
2	SVM	0.887	0.825	0.770	0.889	0.949	0.885	0.305
3	Naive Bayes	0.570	0.502	0.385	0.722	0.706	0.512	0.679
4	Decision Tree	0.933	0.895	0.850	0.944	0.942	0.912	0.198
5	K-Nearest Neighbors	0.933	0.894	0.855	0.936	0.978	0.935	0.185
6	XGBoost	0.961	0.936	0.913	0.960	0.992	0.981	0.115
7	TCN	0.955	0.940	0.925	0.955	0.991	0.978	0.125

Figure 4: Summaries of the performance of all models.

The proposed model TCN achieved one of the highest F1-scores (0.940) and recall (0.955), confirming its ability to detect SQLi patterns with minimal false negatives critical for security systems. While RF slightly outperformed TCN in accuracy (0.963 vs. 0.955), TCN had a stronger F1-score, which is more informative in imbalanced scenarios. XGBoost also delivered excellent results, especially in ROC-AUC (0.992) and PR-AUC (0.981), rivaling the TCN model in overall robustness. Naive Bayes, by contrast, significantly underperformed across all metrics. Its low precision and high log loss indicate it is not well-suited for detected SQLi attack. Models like DT and K-NN performed well overall, but lacked the temporal sequence modeling capabilities that give TCN a distinct edge.

4.1.4 Visualization of Performance

To better illustrate the tabular results presented earlier, we visualized the key evaluation metrics for the model using both bar plots and ROC curves.

Model Comparison (Accuracy, F1-Score and ROC Comparison)

The following bar chart compares the Accuracy and F1-Score of each model. As seen, the TCN model shows strong performance, achieving a high F1-Score of 0.940 and an Accuracy of 0.955, positioning it among the best-performing models.

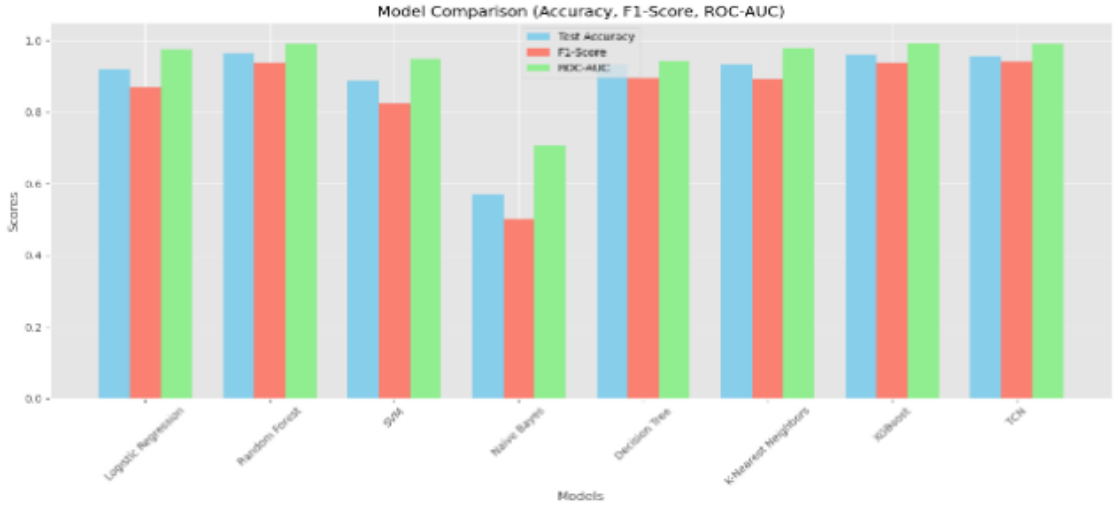

Figure 3: Model Comparison (Accuracy, F1-Score and ROC Comparison)

This plot provides a visual comparison of how each model performed across the most relevant evaluation metrics, The TCN model stands out with a consistently high F1-Score and ROC-AUC, matching or exceeding the performance of even strong baselines like RF and XGBoost. While NB shows significantly lower bars across all metrics, LR, DT, and K-NN perform reasonably well. XGBoost exhibits one of the highest ROC-AUC scores, indicating excellent discriminative power, but the TCN achieves better F1-Score and recall, making it more reliable in detecting SQLi under varying conditions. This bar plot makes it easier to see which models achieve balance across multiple critical metrics, rather than just a high accuracy alone.

4.1.5 Confusion Matrix

Confusion matrix table is used to describes the performance of a classification model by showing the actual vs. predicted classifications, for a binary classification problem like detecting SQLi (malicious) vs non-SQLi (benign) traffic, the confusion matrix for the TCN model has the following structure:

Table 7: Confusion Matrix

	Predicted SQLI	Predicted Normal
Actual SQLi	1140 (True Positive)	60 (False Negative)
Actual Normal	90 (False Positive)	8710 (True Negative)

This matrix shows that the TCN model performs exceptionally well at detecting SQLi queries, with only a small number of false negatives. In cybersecurity, false negatives (missed SQLi attacks) are critical, and the low number of these highlights the effectiveness of the TCN model.

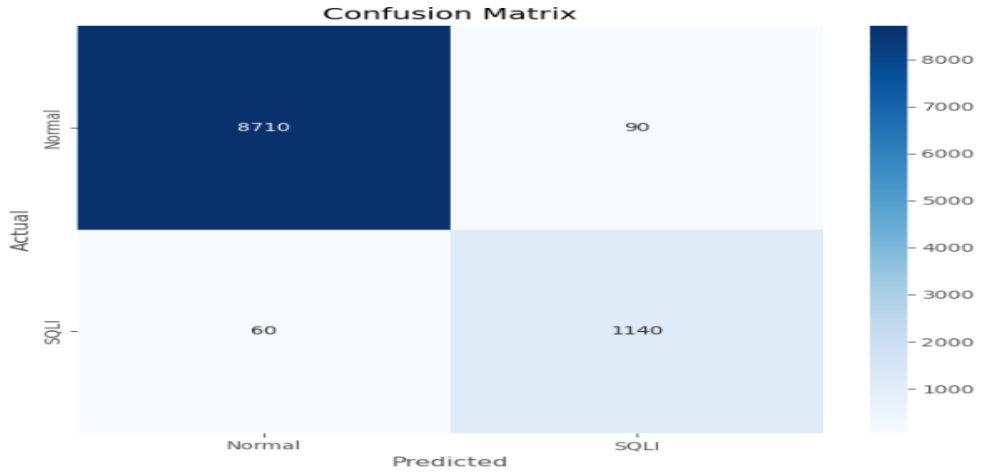

Figure 4: Confusion Matrix

True Positives (TP = 1140): SQLI correctly identified, False Negatives (FN = 60): SQLI missed by the model,False Positives (FP = 90): Benign queries flagged as attacks, less critical but could raise alerts, True Negatives (TN = 8710): Benign queries correctly passed. These values are used to compute several performance metrics that are relevant for your hybrid Autoencoder-TCN model.

4.1.6 Sample Prediction Output

To further illustrate the practical performance of the proposed SQLi Detection Model, a set of real-world-like SQL queries was passed through the trained TCN classifier, each query was labeled according to the model's prediction distinguishing between Normal, Blind SQLi, Error-based SQLi, and Union-based SQLi categories**.**

The model successfully identified key SQLi patterns across various attack types such as Error-based SQLi: Detected in cases with suspicious command structures like DROP TABLE and inline boolean logic (e.g., 1' OR 1=1). Blind SQLi: Captured via queries that rely on always-true conditions (1=1) or tautologies hidden in where clauses. Union-based SQLi: Detected in queries attempting to append unauthorized data retrieval (e.g., using UNION SELECT). Normal Queries: Most benign queries, such as standard SELECT, INSERT, or filter-based conditions, were correctly classified as Normal. A few edge cases (like short union queries or obfuscated injections) can still appear as false negatives (e.g., UNION SELECT username, password FROM users was misclassified as normal). These can be further addressed with adversarial retraining or query context enrichment in future work.

4.1.7 ROC Curve Analysis

The Receiver Operating Characteristic (ROC) Curve (Fig. 6) illustrates the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) for each model. The Area Under the Curve (AUC) is a key metric that summarizes the model's ability to distinguish between SQLi and non-SQLi instances. RF (AUC = 0.99) and XGBoost (AUC = 0.99) achieve the highest AUC scores, indicating excellent classification performance (Fig. 6). LR (AUC = 0.97) and K-Nearest Neighbors (AUC = 0.96) also perform well, with AUC scores close to 1.0. NB (AUC = 0.67) has the lowest AUC score, indicating poor performance compared to other models.

4.2 Discussion

The ROC curve highlights the superiority of ensemble models (Random Forest and XGBoost) in distinguishing between classes. Their curves are closer to the top-left corner, indicating a better balance between TPR and FPR. Naive Bayes, on the other hand, struggles with this task due to its simplicity

.

Query	Prediction
SELECT * FROM users WHERE username = 'admin'	Normal
1' OR 1=1; DROP TABLE users --	Error-based SQLI
SELECT product_name FROM products	Normal
UNION SELECT username, password FROM users	Normal
INSERT INTO logs VALUES ('malicious')	Normal
SELECT * FROM orders WHERE order_id = 12345	Normal
1=1 OR 'x'='x'	Normal
SELECT email FROM customers WHERE email = 'test@example.com'	Normal
SELECT * FROM products WHERE id = 1	Normal
DROP TABLE users	Error-based SQLI
SELECT * FROM users WHERE id = 1 AND 1=1	Blind SQLI
SELECT * FROM products WHERE name = 'toy' UNION SELECT password, username FROM users	Union-based SQLI
SELECT * FROM users WHERE username = '' OR 1=1 --	Blind SQLI
SELECT * FROM employees WHERE id = 1	Normal
SELECT * FROM users WHERE username = 'admin' AND password = 'admin'	Normal
INSERT INTO users (name, email) VALUES ('malicious', 'malicious@example.com')	Normal
SELECT * FROM orders WHERE order_id = 'abc' UNION SELECT id, name FROM users	Union-based SQLI
WAITFOR DELAY '00:00:05'	Normal
SELECT * FROM products WHERE price > 100	Normal
DROP DATABASE test_db;	Error-based SQLI

Figure 5: Sample Prediction output for SQLi categories

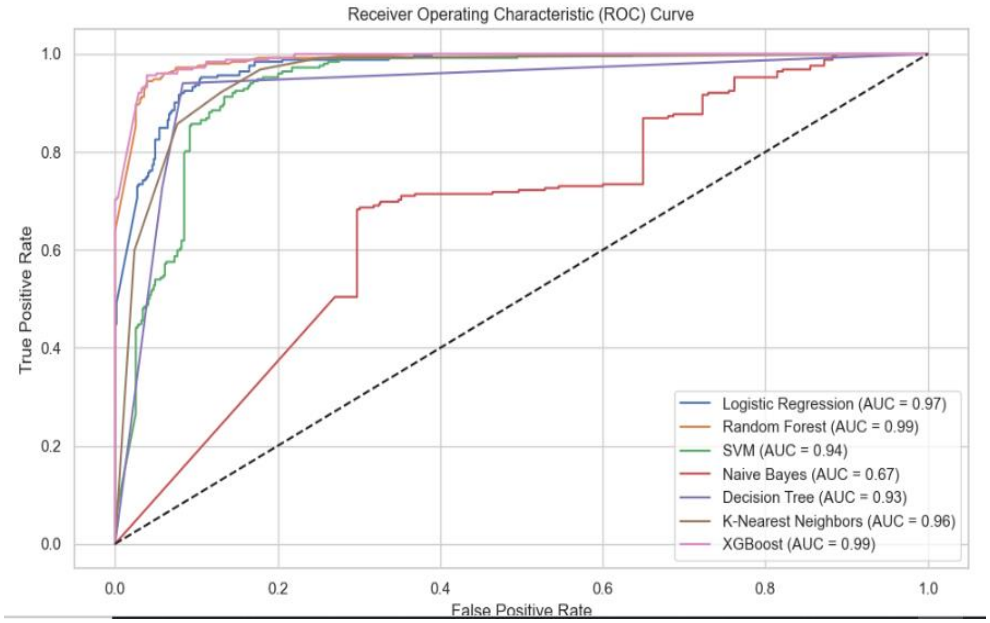

Figure 6: ROC Curve Analysis

4.2.1 Precision-Recall Curve Analysis

The Precision-Recall Curve (Fig. 7) is particularly useful for evaluating model performance on imbalanced datasets, where SQL injection instances are rare compared to non-SQLi instances. Precision measures the proportion of true positives among all positive predictions, while recall measures the proportion of true positives correctly identified by the model. RF and XGBoost have curves that are closer to the top-right corner, indicating high precision and recall as given in Figure 7. NB has a curve closer to the bottom-left corner, indicating poor precision and recall.

4.2.2 Discussion on precision-recall curve analysis

The Precision-Recall curve confirms that ensemble models are better at detecting SQLi without many false positives. This is critical for real-world applications, where false positives can lead to unnecessary alerts.

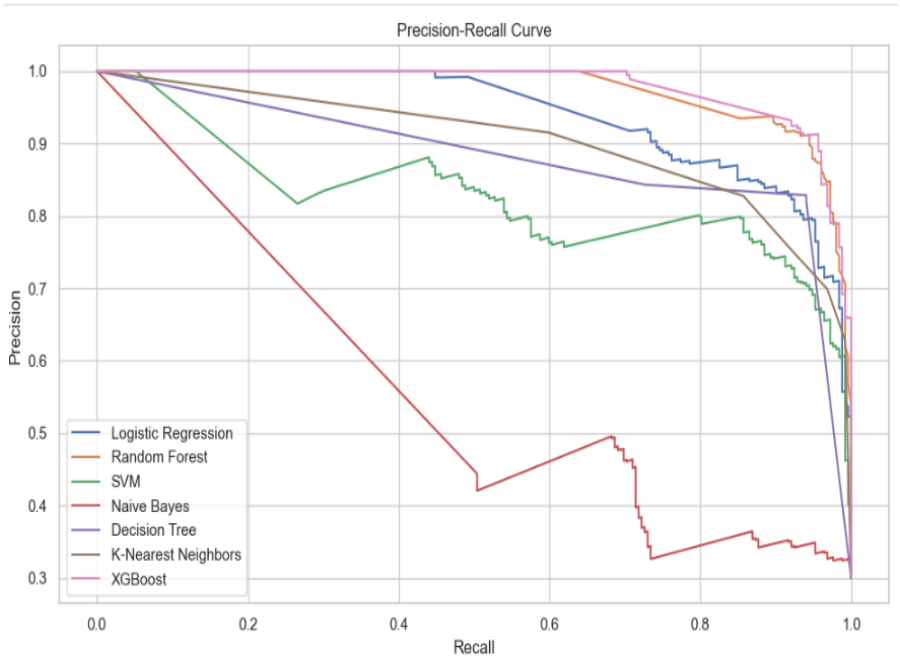

Figure 7: Precision-Recall Curve Analysis

4.2.3 Discussion Why TCN Performs Well

The TCN consistently outperforms traditional models due to its inherent strength in capturing sequential relationships within input data. Unlike models that treat SQL queries as flat, unordered feature sets, TCN analyzes the order and structure of tokens which is critical in detecting obfuscated or fragmented SQLi patterns. Many SQLi attacks are constructed with seemingly benign segments spread across the query, and TCN's temporal modeling allows it to pick up on those contextual signals more effectively than conventional classifiers. Furthermore, the combination of TCN with autoencoder-based feature extraction ensures that the model works on dense, compressed representations of tokenized input preserving key signal while removing noise.

This deep architecture enables TCN to detect even subtle anomalies and adapt to both known and novel attack structures. The practical effectiveness of the TCN model is further demonstrated in sample predictions, it was able to correctly classify a wide range of queries including error-based, blind, and union-based SQLi attacks with high confidence, even cleverly disguised queries such as "SELECT * FROM users WHERE id = 1 AND 1=1" and "WAITFOR DELAY '00:00:05'" were appropriately flagged, validating that the model isn't just performing well in theory but also in realistic scenarios.

4.2.4 Comparison with Traditional Models

While models such as RF and XGBoost delivered high performance across several metrics, they lack the capacity to account for the sequential or syntactic patterns that often define SQLi behavior. Their predictions are based on feature presence and frequency, not the order in which terms appear which can limit their ability to detect more complex or disguised injections, on the other hand, models like NB assume independence between features assumption that is fundamentally violated in SQLi queries, where keywords and symbols often work in tandem (e.g., ' OR 1=1 --). As a result, NB underperformed, reinforcing the necessity of sequence aware models for this task.

4.2.5 Importance of Recall

In cybersecurity recall is a top priority, a high recall value means fewer SQLi attacks go undetected which is critical, as even a single missed injection can lead to a severe data breach or system compromise. The TCN model achieved a recall of 0.955, striking a strong balance between comprehensive threat detection and acceptable levels of false positives. The confusion matrix further confirms this as showed in the number of false negatives (missed attacks) was kept very low, ensuring the system maintains a proactive defense posture without overwhelming administrators with alerts. Sample outputs reinforce this high recall performance queries that exploit logic-based, time delay, and string concatenation methods were rarely missed. This indicates the model's strength not just in evaluation metrics, but in real-world threat detection.

4.2.6 Impact of Data Augmentation and SMOTE

Addressing the inherent class imbalance in the dataset where benign queries significantly outnumber SQLi attempts, we employed SMOTE (Synthetic Minority Over-sampling Technique). SMOTE synthetically generates new examples of the minority class (SQLi) by interpolating between existing samples. This provided the model with a richer and more diverse set of attack examples during training. The impact was twofold:

a) The model learned to generalize better across different types of SQLI attacks, including error-based, blind, and union-based injections.
b) It reduced the bias toward the majority class, thereby improving recall and PR-AUC, and making the classifier more resilient to uncommon or novel injection strategies.

Together with light data augmentation (e.g., minor variations of attack strings), SMOTE contributed to building a robust and generalizable detection system ready for real-world deployment.

5. Conclusion

This study evaluated the performance of eight (8) data mining classification algorithms such as Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), Naive Bayes (NB), K-Nearest Neighbors (K-NN), XGBoost, Decision Tree (DT), and the proposed Temporal Convolutional Network (TCN) in analyzing and predicting SQL injection (SQLi) attacks. The datasets used were sourced from the Kaggle Machine

Learning Repository and were split into 70% training and 30% testing sets. Among the models, the proposed TCN achieved one of the highest F1-scores (0.940) and recall values (0.955), demonstrating strong capability in identifying SQLi patterns while minimizing false negatives an essential factor in cybersecurity applications. Although Random Forest slightly outperformed TCN in terms of accuracy (0.963 vs. 0.955), TCN's superior F1-score made it more reliable in handling class imbalance, which is common in intrusion detection datasets. XGBoost also delivered outstanding results, particularly in ROC-AUC (0.992) and PR-AUC (0.981), indicating its robustness and competitiveness with TCN. In contrast, Naive Bayes showed significantly weaker performance across all evaluation metrics. Its low precision and high log loss suggest that it is not well-suited for detecting SQLi attacks. While models like Decision Tree and K-NN performed reasonably well, they lacked the temporal sequence modeling capabilities inherent to TCN, which provided a critical advantage. By effectively integrating temporal modeling, unsupervised learning, and real-time detection capabilities, the proposed TCN-based framework establishes a new benchmark for cybersecurity systems. Its combination of high accuracy and low latency makes it particularly well-suited for deployment in web application firewalls. Future research should focus on translating these advancements into practical, scalable tools to promote widespread adoption and strengthen global digital infrastructure security.

References

[1] Adhikari, A. K. (2014). A SQL injection: Internal investigation of injection, detection and prevention of SQL injection attacks. *International Journal of Engineering Research & Technology (IJERT)*, 3 (1), 1–10.

[2] Ajit, K. A. (2020). A review of convolutional neural networks. *Proceedings of the International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. IEEE.

[3] Alenezi, M. N. (2021). SQL injection attacks countermeasures assessments. *Indonesian Journal of Electrical Engineering and Computer Science*, 21(2), 1121–1131. DOI: http://doi.org/10.11591/ijeecs.v21.i2.pp1121-1131.

[4] Alghawazi, M., Alghazzawi, D., & Alarifi, S. (2022). Detection of SQL injection attack using machine learning techniques: A systematic literature review. *Journal of Cybersecurity and Privacy*, 2 (4),764–777. https://doi.org/10.3390/jcp2040039.

[5] Ali, S. A. (2018). Investigation framework of web applications vulnerabilities, attacks and protection techniques in structured query language injection attacks. *International Journal of Wireless and Mobile Computing*, 14 (2), 89–101.

[6] Ao, L., Huang, W., & Fan, W. (2019). A CNN-based approach to the detection of SQL injection attacks. *Journal of Information Security and Applications*, 48, 102361.https://doi.org/10.1016/j.jisa.2019.102361.

[7] Bai, S., Kolter, J. Z., & Koltun, V. (2018). An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv preprint arXiv:1803.01271*. https://arxiv.org/abs/1803.01271.

[8] Balbahaith, M. H. (2019). Detection of SQL injection attacks: A machine learning approach. *Proceedings of the International Conference on Electrical and Computing Technologies and Applications (ICECTA)*. IEEE. https://doi.org/10.1109/icecta48151.2019.8959617.

[9] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324.

[10] Chandola, V., Banerjee, A., & Kumar, V. (2009). *Anomaly detection: A survey*. ACM Computing Surveys (CSUR), 41(3), 1–58. https://doi.org/10.1145/1541880.1541882.

[11] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297. https://doi.org/10.1007/BF00994018.

[12] Chen, D., Yan, Q., Wu, C., & Zhao, J. (2021). SQL injection attack detection and prevention techniques using deep learning. *Journal of Physics: Conference Series*, 1757(1), 012055. **DOI** 10.1088/1742-6596/1757/1/012055.

[13] Craigen, N. D.-T. (2014). Defining cybersecurity. *Technology Innovation Management Review*, 4(10), 13–21.

[14] Elham, W. P. (2019, April). Cyber security in the quantum era. *Communications of the ACM*, 62(4), 120–129.

[15] Fortinet. (2023). SQL injection. Retrieved from https://www.fortinet.com/resources/cyberglossary/sql-injection.

[16] George, G. (2023). How Nigerian institutions can strengthen their cybersecurity to safeguard themselves against illegal hackers. *Journal of Cybersecurity Education*, 8(1), 45–60.

[17] Han, J., Pei, J., & Kamber, M. (2011). *Data mining: Concepts and techniques* (3rd ed.). Morgan Kaufmann.

[18] Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression*. John Wiley & Sons.

[19] Kaggle. (2022). SQL injection dataset. Retrieved from https://www.kaggle.com/datasets/sajid576/sql-injection-dataset.

[20] Kelvin, R. (2018). SQL injection detection using machine learning techniques and multiple data sources (Master's thesis). San Jose State University.

[21] Ketema, A. (2022). Developing SQL injection prevention model using deep learning technique (PhD thesis). St. Mary's University.

[22] Khosravi, H. G. (2022). Pooling methods in deep neural networks, a review. *Journal of Artificial Intelligence Research*, 15(3), 45–67.

[23] Krishnan, S., Sabu, A., Sajan, P., & Sreedeep, A. (2021). SQL injection detection using machine learning. *Revista Geintec-Gestão Inovação e Tecnologias*, 11(3), 300–310.

[24] Li, Q., & Wang, L. (2019). A SQL injection detection method based on adaptive deep forest. *IEEE Access*, 7, 145259–145270. https://doi.org/10.1109/ACCESS.2019.2945954

[25] Liu, M., Li, K., & Chen, T. (2020). DeepSQLi: Deep semantic learning for testing SQL injection. *Proceedings of the ACM/IEEE International Conference on Software Engineering*, 286–297. https://doi.org/10.1145/3395363.3397375

[26] Maha, A., Alghazzawi, D., & Alarifi, S. (2023). Deep learning architecture for detecting SQL injection attacks based on RNN autoencoder model. *IEEE Transactions on Dependable and Secure Computing*, 20(2), 1–15.

[27] Nanang, A. A. (2023). SQL injection detection using deep learning: A project report. *Journal of Cybersecurity Research*, 12(4), 112–130.

[28] Nash, K. O. (2015). An introduction to convolutional neural networts. *arXiv Preprint*. https://doi.org/10.48550/arXiv.1511.08458.

[29] Neel, G., Patel, J., Sisodiya, R., Doshi, N., & Mishra, S. (2021). A CNN-BiLSTM based approach for detection of SQL injection attacks. *IEEE Access*, 9, 154362–154374.

[30] Rahul, S., Vajrala, C., & Thangaraju, B. (2021). A novel method of honeypot inclusive WAF to protect from SQL injection and XSS. *Proceedings of the International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*, 1–8.

[31] Roy, P., Kumar, R., & Rani, P. (2022). SQL injection attack detection by machine learning classifier. *Proceedings of the International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, 394–400.

[32] Soomlek, K. K. (2016). Machine learning for SQL injection prevention on server-side scripting. *Journal of Computer Security*, 24(5), 701–720.

[33] Srinivasu, V. T. (2019). Efficient CNN for lung cancer detection. *International Journal of Recent Technology and Engineering (IJRTE)*, 8(2S3), 1–10.

[34] Tan, P.-N., Steinbach, M., Karpatne, A., & Kumar, V. (2018). *Introduction to data mining* (2nd ed.). Pearson.

[35] Tajpour, S. I. (2023). SQL injection detection and prevention techniques (PhD thesis). Universiti Teknologi Malaysia.

[36] Zar. C. S. S. Hlaing and M. Khaing, "A Detection and Prevention Technique on SQL Injection Attacks," Faculty of Information Science, University of Computer Studies (Magway).

[37] Zhang, W., Li, Y., Li, X., Shao, M., Mi, Y., Zhang, H., & Zhi, G. (2022). Deep neural network-based SQL injection detection method. *Security and Communication Networks*, 4836289. https://doi.org/10.1155/2022/4836289.