Real Time SQL Injection Detection Using Hybrid CNN-LSTM Deep Learning Approach.

MSc Programme: Cyber Security

Student Name: Saqib Sattar

Student Matriculation Number: S2360784

Supervisor Pius owoh

Table of Contents

Table of Figures

Table of Tables

1. Introduction

SQL Injection attacks are among the most persistent and critical cybersecurity threats faced by modern web applications. These attacks exploit vulnerabilities in user input fields to manipulate backend databases, often resulting in unauthorized data access, breaches of sensitive information, and, in severe cases, complete system compromise. As web applications have become integral to business operations and everyday life, the consequences of SQL Injection attacks have grown exponentially, emphasizing the need for effective detection and prevention mechanisms [1]. Traditional SQL Injection detection methods, such as rule-based systems, rely on predefined patterns to identify malicious inputs. While effective for known attack signatures, these systems are inherently limited in adapting to novel or evolving attack vectors. The static nature of rule-based approaches often leads to high false positive and false negative rates, reducing their reliability in real-world scenarios [2]. methods focusing solely on query structure may fail to recognize obfuscated attacks designed to bypass such checks [3]. To address these limitations, hybrid detection methods have been developed. By combining static analysis with dynamic runtime monitoring, hybrid approaches aim to capture a broader range of SQL Injection behaviours. However, these methods often face scalability challenges and are resource-intensive, particularly in high-traffic environments [5].

The emergence of machine learning and deep learning techniques has introduced a paradigm shift in SQL Injection detection. Unlike traditional methods, machine learning-based approaches can learn patterns and behaviours from large datasets, enabling them to detect even previously unseen attacks. Models such as Logistic Regression, Support Vector Machines, and Neural Networks have shown remarkable accuracy in classifying SQL Injection attempts. These models analyse input features such as query structure, syntax patterns, and statistical behaviours to distinguish between benign and malicious queries [8]. The study has demonstrated accuracy rates exceeding 99% using machine learning techniques, highlighting their potential for real-world applications [9].

Furthermore, advanced hybrid frameworks that integrate multiple machine learning techniques, such as combining Artificial Neural Networks with Support Vector Machines, have achieved even greater precision. These frameworks leverage the strengths of different

algorithms, addressing the weaknesses of standalone models. The Study on Artificial Neural Networks excel at capturing complex data patterns, Support Vector Machines provide robustness in handling outliers, making such hybrid approaches particularly effective in SQL Injection detection [11]. However, the development and deployment of machine learning-based systems are not without challenges. Issues such as dataset quality, feature selection, computational complexity, and real-time applicability remain significant hurdles [13]. High-quality datasets are essential for training models capable of generalizing across diverse attack scenarios, yet obtaining comprehensive datasets that include all potential SQL Injection variants is a persistent challenge [16].

Considering these advancements and challenges, this study targets to explore application of machine learning and deep learning techniques for SQL Injection detection, evaluate their effectiveness, and propose strategies to address existing limitations. By leveraging existing research and developing novel frameworks, this research aims to contribute to the advancement of adaptive and reliable SQL Injection detection mechanisms. Such systems are essential for safeguarding modern web applications against increasingly sophisticated cyber threats, ensuring data integrity, confidentiality, and availability in an evolving digital landscape [18].

1.1. Background

Web application security has become a critical concern due to the increasing reliance on digital platforms for personal and organizational purposes. Among the numerous cybersecurity threats, SQL Injection attacks stand out as one of the most severe. These attacks exploit vulnerabilities in user input fields to execute malicious SQL code, allowing attackers to manipulate backend databases, gain unauthorized access to sensitive information, or disrupt services. Such attacks pose significant risks to confidentiality, integrity, and data availability, making them a persistent challenge in cybersecurity [1, 2].

Traditional SQL Injection detection methods relied heavily on rule-based systems, which used predefined patterns to identify malicious inputs. While effective for known attack signatures, these systems often failed to detect new or obfuscated attacks. For instance, static analysis techniques, which focus on query structure, were unable to handle runtime behaviours or complex attack vectors. Dynamic approaches, on the other hand, monitored interactions

during execution but faced challenges such as high false positive rates and significant computational overhead [3, 5].

SQL Injection attacks utilize techniques such as tautology-based attacks, time-based blind attacks, and union-based methods to exploit vulnerabilities. For example, tautology-based attacks manipulate conditional statements to always evaluate as true, while union-based attacks use the UNION operator to extract additional data from the database. These sophisticated methods underscore the limitations of traditional rule-based approaches, highlighting the need for more adaptive and robust detection systems [9, 13].

The application of machine learning has revolutionized SQL Injection detection. Models such as Logistic Regression, Decision Trees, and Support Vector Machines analyse features like query syntax and statistical patterns to classify inputs as benign or malicious. For instance, Logistic Regression has demonstrated high accuracy in distinguishing normal and malicious inputs, making it a reliable choice for SQL Injection detection [5, 8]. Additionally, the use of feature engineering has further improved model performance by focusing on context-aware query analysis [11, 16]. The figure given blow visualizes the impact of input query patterns on detection accuracy, highlighting how features influence model effectiveness [16].

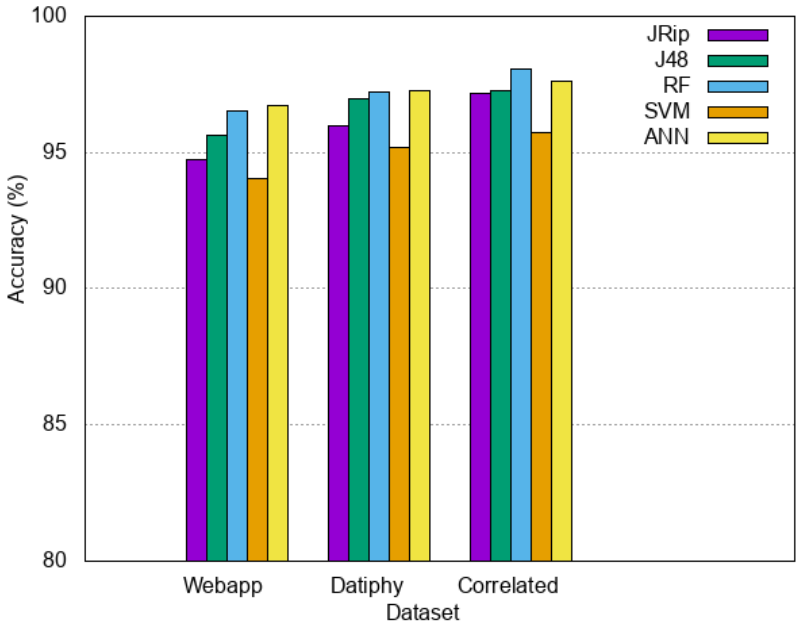


Figure 1 Classification Accuracy [16]

Despite these advancements, challenges persist. Collecting diverse and representative datasets is critical for training machine learning models, yet it remains difficult to cover all SQL Injection scenarios. Additionally, the computational demands of advanced models limit their deployment in real-time applications. Addressing these challenges is essential for the development of scalable and accurate SQL Injection detection systems [9, 11, 18].

1.2. Problem Statement

SQL Injection attacks continue to threaten web application security, exploiting vulnerabilities in user input fields to access or manipulate sensitive data. Despite advancements in detection methods, traditional approaches, such as rule-based and static analysis, struggle to adapt to evolving attack techniques, leading to high false positives and limited scalability. While machine learning offers promising solutions, challenges persist in achieving real-time detection, dataset diversity, and computational efficiency.

1.2.1. Research Questions

1. How can deep learning techniques be optimized to improve the accuracy?

2. What real time SQL injection detection techniques can be used in web-based applications?

3. How can real time SQL Injection detection models be evaluated

1.2.2. Research Objectives

1. To develop a hybridised deep learning model for improved accuracy in detecting SQL injection attacks in web-based application

2. To propose a real time SQL injection detection models

3. To evaluate deep learning framework for SQL Injection detection that minimizes false positives and achieves real-time applicability by leveraging advanced feature engineering and diverse datasets.

2. Literature Review

2.0.1. SQL Injection Detection Techniques

SQL Injection (Sql injection) attacks remain a significant security threat, requiring robust detection mechanisms. Fitsum Gizachew Deriba et al. [9] introduced a hybrid framework that combines statistical and dynamic techniques with machine learning (ML). This framework addresses the limitations of static and dynamic methods, such as high false positives and poor adaptability. Algorithms like Naive Bayes, Support Vector Machines (SVM), Decision Trees, and Artificial Neural Networks (ANN) were tested, with ANN achieving the highest accuracy. Naive Bayes proved more efficient in reducing web loading times, demonstrating the framework's potential for real-world applications. Luo et al. [16] proposed an integrated approach combining statistical and dynamic analysis to detect query anomalies. This comprehensive detection mechanism improves SQL Injection detection but faces scalability and real-time adaptability challenges. Similarly, Kevin Ross [5] explored the correlation of data from web applications and database layers, showing that Decision Trees and rule-based models can achieve comparable accuracy to neural networks while being more computationally efficient. Another study by Musaab Hasan et al. [3] evaluated 23 machine learning classifiers, integrating the top five into a GUI-based system. By combining static and dynamic analysis, the system achieved an accuracy of 93.8% and minimized false positives and negatives.

The use of multi-source datasets has also been investigated to enhance SQL Injection detection. For example, in [6], the author demonstrated that decision trees and rule-based models achieve similar accuracy to neural networks while being more efficient by correlating traffic from both web applications and databases. This methodology bridges the gap between traditional signature-based detection and machine learning techniques. Moreover, Venkatramulu et al. [13] explored word embeddings like Count Vectorizer and TF-IDF to improve model performance in SQL Injection detection. Algorithms such as Logistic Regression and XGBoost achieved high accuracy in malicious query classification, underscoring the importance of feature engineering in overcoming traditional detection methods' shortcomings.

2.0.2. Machine Learning Techniques for SQL Injection Detection

Machine learning (ML) has significantly enhanced SQL Injection detection, enabling automated feature extraction and accurate classification of malicious queries. Taseer Muhammad et al. [8] evaluated various ML algorithms for detecting SQL Injection attacks, focusing on models like Logistic Regression, Multilayer Perceptron (MLP), and Sequential Minimal Optimization (SMO). These models achieved accuracy rates of 98.7% through cross-validation, with the Instance-Based Learner (IBK) emerging as the fastest and most effective. The study emphasized the need for evaluation metrics beyond accuracy to address the growing complexity of SQL Injection attacks. Adeyinka Mustapha et al. [19] reviewed machine learning models used in e-commerce for SQL Injection detection, including SVM, Decision Trees, and Neural Networks. While highlighting the importance of dataset quality and feature engineering, the study identified scalability and real-time detection as key challenges requiring further exploration.

Machine learning approaches have also employed advanced techniques like ensemble learning and hybrid frameworks. Auninda Alam et al. [15] introduced SCAMM, a machine learning framework that uses algorithms like K-Nearest Neighbors (KNN), Random Forest, and Logistic Regression. The study identified Random Forest as the most accurate algorithm and highlighted the adaptability of machine learning in mitigating diverse SQL Injection forms. Uwagbole et al. [12] applied Symbolic Finite Automata (SFA) with SVM and Logistic Regression to improve detection accuracy. However, the study noted challenges in generalizing across diverse datasets, emphasizing the need for comprehensive training data. Ahmed et al. [18] combined Natural Language Processing (NLP) techniques with ensemble learning models, such as Random Forest, achieving improved accuracy. However, limitations in handling complex queries using Bag-of-Words representation were noted, calling for more advanced feature representations in SQL Injection detection.

2.0.3. Deep Learning Techniques for SQL Injection Detection

Deep learning (DL) techniques have revolutionized SQL Injection detection by automating feature extraction and learning complex patterns in malicious queries. Kevin Zhang [12] explored deep learning models like Convolutional Neural Networks (CNN) and Multilayer Perceptron (MLP) for detecting SQL Injection vulnerabilities in PHP code. By integrating input

validation, sanitization features, and word2vec embeddings, the study achieved high precision and recall, significantly advancing static code analysis for precise vulnerability detection. Ding Chen et al. [4] proposed a CNN-based SQL Injection detection method using Word2Vector embeddings. The approach eliminated reliance on predefined rules, allowing adaptation to zero-day attacks while achieving high accuracy and reducing false alarms.

Hybrid deep learning models have also gained attention for their potential in SQL Injection detection. Gandhi et al. [14] introduced a hybrid deep learning model combining CNN with Bidirectional Long Short-Term Memory (BiLSTM). This model achieved 98% accuracy and effectively detected complex query patterns but faced challenges in real-time applications due to high computational demands. Yazeed Abdulmalik [10] leveraged semantic feature extraction to enhance SQL Injection detection by analyzing query structures. This approach addressed the limitations of traditional syntax-based detection methods, improving detection accuracy and adaptability.

Comparative studies have also highlighted the strengths and weaknesses of deep learning models. Wibowo et al. [17] compared Support Vector Machine (SVM) and Random Forest for SQL Injection detection, with Random Forest outperforming SVM in accuracy and classification robustness. However, imbalanced datasets caused generalization issues, emphasizing the importance of using diverse and balanced training data. These studies demonstrate the significant advancements brought by deep learning while highlighting the need for further exploration into real-time adaptability and computational efficiency.

2.1. Related Work

Ao Luo et al. [21] proposed a CNN-based method for SQL Injection detection, utilizing Convolutional Neural Networks (CNNs) to identify malicious payloads in network traffic. The approach processed SQL injection data and normal traffic from sources like UNSW-NB15, KDD99, and simulated attacks using SQL map and DVWA. The CNN architecture consisted of convolutional layers for feature extraction, pooling layers for dimensionality reduction, and a fully connected layer with dropout for regularization. Evaluated against ModSecurity, the CNN model achieved 99.50% accuracy, 100% recall, and a 99.49% F1-score, outperforming ModSecurity's 96.89% accuracy. Limitations include reliance on predefined datasets and potential challenges with unseen attack patterns. Andri Setiyaji et al. [22] proposed a hybrid

approach for detecting SQL Injection attacks using machine learning and Convolutional Neural Networks (CNN). The authors used the largest SQL Injection dataset available on Kaggle, containing 148,326 samples (77,750 SQL Injection payloads and 70,576 benign queries). The methodology included preprocessing the data using CountVectorizer for feature extraction and splitting it into 80% training and 20% testing datasets. The CNN model architecture comprised convolutional layers for feature extraction, pooling layers for dimensionality reduction, and dense layers for classification, with Adam optimizer employed for training. The CNN achieved an accuracy of 91.04%, outperforming Naive Bayes, Logistic Regression, and Decision Tree models in overall performance metrics such as precision, recall, and F1-score. While the CNN demonstrated superior accuracy and robustness, the authors highlighted the need for further improvement in handling unseen attack patterns and enhancing the model's generalization. Ruhua Lu et al. [23] proposed a SQL Injection detection model based on Convolutional Neural Networks (CNN) to address the widespread and high-risk security vulnerabilities in web applications caused by SQL Injection attacks. The proposed framework includes two key stages: a model training phase and a classification detection phase. SQL Injection statements were pre-processed to remove noise, standardized, and vectorized using Word2Vec, with the Skip-Gram model selected for its ability to capture semantic relationships in SQL queries. The CNN model architecture consists of convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for classification. The dataset used for training and testing was synthetically generated, with the results showing that the CNN model effectively detected SQL Injection attacks with high accuracy. The study highlighted the robustness of CNN in feature extraction and classification. However, the limitations include reliance on synthetic datasets and potential challenges in generalizing to real-world, unseen attack patterns.

Manav Hirani et al. [24] proposed a deep learning approach for detecting SQL Injection attacks using Convolutional Neural Networks (CNN). The authors prepared a comprehensive dataset containing various SQL Injection payloads and legitimate queries, sourced from open libraries like Kaggle and extended with manually generated attack scenarios, including blind, union-based, and error-based SQL Injection techniques. The dataset underwent rigorous preprocessing steps such as de-duplication, noise removal, and label correction to ensure its quality. The study compared the performance of multiple algorithms, including CNN, Naive

Bayes, Decision Trees, Support Vector Machines (SVM), and K-Nearest Neighbors (KNN), using metrics like accuracy, precision, recall, and ROC-AUC. The CNN model outperformed all other algorithms, achieving an accuracy of 94.84%, precision of 85.67%, and recall of 96.56%, demonstrating its effectiveness in identifying malicious queries. However, the paper acknowledged limitations, including a relatively small dataset and challenges in generalizing the model to unseen attack types.

Neel Gandhi et al. [25] proposed a hybrid SQL Injection detection framework using Convolutional Neural Networks (CNN) and Bidirectional Long Short-Term Memory (BiLSTM) networks. The authors addressed SQL Injection vulnerabilities, a significant threat to web application security, by leveraging the combined strengths of CNN for feature extraction and BiLSTM for capturing long-term dependencies. The dataset, comprising 4,200 SQL queries (3,072 malicious and 1,128 benign), underwent preprocessing steps like tokenization and embedding using Word2Vec. The proposed hybrid model achieved an accuracy of 98%, outperforming traditional models such as Logistic Regression, Decision Trees, Random Forest, and standalone deep learning models like CNN and LSTM. Comparative evaluation based on metrics like precision, recall, and F1-score validated the superior performance of the CNN-BiLSTM model. However, the reliance on a relatively small and synthetic dataset limits the model's generalizability to real-world attack patterns.

Fitsum Gizachew Deriba et al. [9] introduces a hybrid SQL Injection detection framework combining statistical and dynamic analysis with machine learning techniques. Algorithms such as Naive Bayes, Support Vector Machines (SVM), Decision Trees, and Artificial Neural Networks (ANN) were evaluated. Using synthetic datasets, ANN achieved the highest accuracy, while Naive Bayes excelled in computational efficiency by reducing query response times. This method effectively addresses SQL Injection threats in web application security.

However, scalability issues and a lack of adapt ability to obfuscated attacks were notable limitations. Additionally, more diverse datasets were recommended to enhance generalizability for real-world applications. The author Taseer Muhammad et al. [8] evaluates machine learning algorithms such as Logistic Regression, Multilayer Perceptron (MLP), and Sequential Minimal Optimization (SMO) for SQL Injection detection. Simulated datasets were used, achieving 98.7% accuracy. The study highlighted Instance-Based Learner (IBK) as the fastest and most efficient model. Despite its strong performance, the framework lacked real-

time detection capabilities, restricting its applicability in dynamic environments. Furthermore, reliance on static datasets posed challenges for adaptability to novel SQL Injection techniques This study by Kevin Ross [6] investigates SQL Injection detection using multi-source datasets from both web application and database layers. Decision Trees and Neural Networks were evaluated, with Decision Trees demonstrating comparable accuracy but significantly better computational efficiency. Synthetic and real-world SQL Injection queries were used. The approach improved detection accuracy by leveraging correlated datasets, offering a robust alternative to traditional systems. However, predefined rules limited adaptability to advanced attack patterns, and scalability was insufficiently explored [6]. Auninda Alam et al. [15] present SCAMM, a machine learning-based framework for detecting SQL Injection attacks. The study evaluates models such as K-Nearest Neighbors (KNN), Random Forest, and Logistic Regression, identifying Random Forest as the most accurate in detecting various SQL Injection patterns. The framework relies on synthetic datasets comprising SQL Injection patterns and benign queries. While effective, challenges included false negatives in complex queries and dependence on curated datasets, limiting scalability in real-world applications.

Kevin Zhang [12] employs Convolutional Neural Networks (CNN) and Multilayer Perceptron (MLP) with word2vec embeddings for detecting SQL Injection vulnerabilities in PHP applications. Real-world PHP scripts and synthetic SQL Injection attacks were used as the dataset. CNN achieved superior precision and recall compared to traditional methods. However, the model's computational demands and focus on static code analysis restricted its effectiveness in dynamic environments, necessitating optimization for real-time use. Yazeed Abdulmalik [10] enhances SQL Injection detection through semantic feature extraction, employing models like Support Vector Machines (SVM) and Decision Trees. By analyzing query structures, the model addresses limitations of syntax-based methods. High detection accuracy was achieved using datasets of real-world and synthetic SQL Injection patterns. However, heavily obfuscated queries posed challenges, and the study highlighted the need for more diverse datasets to improve robustness in real-world applications.

The study by Ding Chen et al. [4] introduces a deep learning framework using Word2Vector embeddings and CNN for SQL Injection detection, targeting vulnerabilities in e-commerce platforms. Synthetic datasets containing SQL Injection patterns and benign queries were utilized, achieving high accuracy and reducing false alarms. The approach demonstrated

adaptability to zero-day attacks. However, its computational intensity limited deployment in resource-constrained environments. Adeyinka Mustapha et al. [19] review machine learning models for SQL Injection detection, focusing on techniques such as SVM, Decision Trees, and Neural Networks. The dataset consisted of traffic logs from e-commerce applications containing SQL Injection attempts and benign activities. The study underscores the importance of feature engineering and dataset quality while identifying scalability and real-time detection capabilities as key limitations.

The study by Musaab Hasan et al. [3] evaluates 23 machine learning classifiers for SQL Injection detection, integrating the top five into a GUI-based detection system. Random Forest and Gradient Boosting achieved an accuracy of 93.8%. The dataset included static and dynamic SQL Injection scenarios. While balanced datasets minimized false positives, reliance on curated datasets limited the system's ability to generalize to unseen attacks. Real-time performance was not addressed.

Luo et al. [16] propose a SQL Injection detection framework combining statistical and dynamic analysis. Logistic Regression and Decision Trees were evaluated using synthetic and real-world datasets. The framework demonstrated improved detection accuracy but faced scalability challenges in high-traffic environments. Real-time detection capabilities remained untested, limiting practical deployment. Gandhi et al. [14] combine Convolutional Neural Networks (CNN) with Bidirectional Long Short-Term Memory (BiLSTM) networks for SQL Injection detection. Synthetic datasets containing SQL Injection queries and benign inputs were used, achieving 98% accuracy. The model effectively detected complex query patterns but was computationally intensive and relied on large datasets, limiting scalability for real-world applications.

Ahmed et al. [18] apply Natural Language Processing (NLP) techniques and ensemble learning, including Random Forest and Bag-of-Words representation, for SQL Injection detection. Synthetic datasets demonstrated improved detection accuracy. However, the model’s reliance on Bag-of-Words representation restricted adaptability to obfuscated queries, and advanced feature extraction techniques were recommended for broader applicability.

Wibowo et al. [17] compare Support Vector Machines (SVM) and Random Forest for SQL Injection detection. The dataset consisted of synthetic benign and malicious queries. Random

Forest demonstrated superior accuracy and robustness. However, dataset imbalance limited the model's ability to generalize, prompting the study to recommend diverse and balanced training data.

The study by Moissinac et al. [10] combines SQL and HTTP data for SQL Injection detection using supervised machine learning models, including Decision Trees and SVM. Correlated datasets improved detection accuracy and reduced computational overhead. However, scalability and dependence on specific datasets limited the system's adaptability, necessitating further research. Venkatramulu et al. [13] explore the application of word embeddings like TF-IDF and Count Vectorizer with XGBoost and Logistic Regression for SQL Injection detection. The dataset comprised real-world traffic logs. High accuracy was achieved, but the model struggled with advanced attack patterns and relied heavily on feature engineering. Further optimization was recommended to handle evolving attack vectors.

Pathak et al. [20] use Progressive Neural Networks to detect SQL Injection patterns with a focus on real-time adaptability. Synthetic datasets helped the model achieve 95% accuracy. However, the study identified performance challenges in high-traffic scenarios, recommending further refinement for complex attack patterns.

The study by Mishra et al. [11] applies Naive Bayes, Gradient Boosting, and Decision Trees for SQL Injection detection. Synthetic queries were used, with Decision Trees demonstrating superior accuracy. However, testing in real-world scenarios was limited, raising concerns about the system's practical deployment.

Sharfuddin et al. [18] employ ensemble models like Random Forest and Bag-of-Words for SQL Injection detection. Synthetic datasets enabled high accuracy. However, adaptability to evolving attack patterns was limited, and reliance on Bag-of-Words restricted the robustness of the model.

Sethi et al. [19] review machine learning techniques such as Logistic Regression and Artificial Neural Networks (ANN) for SQL Injection detection in e-commerce applications. High accuracy was achieved, but dataset quality and balance were highlighted as major challenges. Scalability and adaptability to diverse attack patterns also posed limitations.

Uwagbole et al. [12] employ Symbolic Finite Automata (SFA) with machine learning models, including SVM and Logistic Regression, for SQL Injection detection. Synthetic datasets demonstrated high precision, but challenges in dataset diversity and scalability were noted. The study recommended broader datasets to improve adaptability.

3. Methods

3.1. Research Methodology

Adopting the Experimental Method as the primary research method, the results of this Study will be systematically organized, analysed and tested using a variety of approaches. The results will then be thoroughly reviewed and evaluated to draw meaningful conclusions.

The experimental method's ability to provide controlled environments, combined with its ability to manipulate the variables systematically, makes it especially suitable for detecting SQL injection attacks. In this research, we use a simulated web application environment to evaluate whether various strategies of SQL injection detection are successful in detection systems task performance.

3.2. Development Environment

3.2.1. Computational Resources

The development environment is critical for this development project, as the choice of tools and frameworks directly influences efficiency and outcomes. In the context of deep learning and advanced computational tasks, selecting the right software ecosystem is essential for managing the complexity of modern challenges. This study uses a set of comprehensive tools and frameworks designed for machine learning and data analysis to facilitate the development and evaluation of the proposed system

The tests were performed on Google Colab, utilizing the cloud-based resources given in the parameters of the experimental implementation. Allocated environment: 83.5 GB system RAM, 40 GB GPU RAM, 235.7 GB disk storage This configuration also afforded a respectable compute resource alongside adequate memory to alleviate any bottlenecks from data-heavy workloads and enabled uninterrupted experiments without hardware concerns.

The outcome was achieved mainly through Python 3.10.9 as the primary programming language, TensorFlow 2.13.0 and Keras 2.13.1 as underlying deep learning frameworks which offered strong enough functionality for building new models or training existing ones. We also leaned on Scikit-learn, one of the most widely used data science packages in Python, and several other Python libraries (NumPy, Pandas, Matplotlib and Seaborn) for additional features such as data preprocessing, statistical analysis visualization or comparative assessment. Taken as a whole, these components create a tranquil yet efficient working environment which is perfect for running complex computational tasks with absolute precision.

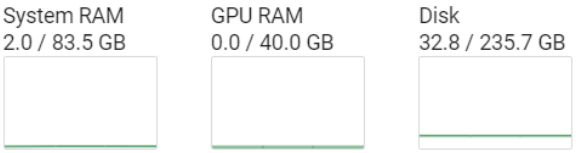


Figure 2 Computational Resource

3.2.2. PyCharm for Coding

PyCharm, an Integrated Development Environment (IDE) specifically designed for Python, was utilized extensively throughout the development of this project. Its robust set of features, including intelligent code completion, debugging tools, and seamless integration with version control systems, made it an ideal choice for implementing and testing the SQL injection detection models. The IDE's support for virtual environments ensured proper dependency management, while its built-in tools for visualization and profiling allowed for efficient debugging and performance optimization. PyCharm's user-friendly interface and advanced editor features significantly enhanced productivity, enabling the rapid iteration and refinement of the codebase during model development and evaluation. This comprehensive IDE provided a stable and efficient platform for executing the intricate workflows of the project.

3.3. Dataset Selection

In our analysis we will use the SQL Injection Dataset published on Kaggle, which was created by AlexTrinity [27]. The dataset comprises 95,933 entries that are carefully annotated to help learners achieve SQL Injection detection. Entries are classified into either benign and malicious, with (50,510) 52.56% of the queries labelled as benign (label 0), (45,846) 47.80% as malicious (label 1) and 11,578 more benign values merged into the dataset from the dataset provided by Syed Saqlain Hussain Shah[26] which makes total benign value (61,988) 57.66% and percentage of malicious value 42.34% of total dataset value. This specific labelling permits a realistic depiction of both valid SQL statements and attack types. This dataset contains multiple types of SQL queries with a whole range of safe statements, general SQL statements to advanced attack pattern (SQL Injection) that will help to train a good deep learning model. Introducing different query structures and different types of attacks increases the robustness of the dataset and allows models learned on it to generalize well to real-world cases and new threats. This research proposes using this dataset to build and test deep learning models to generally detect SQL Injection attacks. This helps to enhance web application security through early detection and prevention of possible threats.

3.4. Experimental Design

The proposed method for SQL injection detection is outlined in Figure 1, which illustrates a systematic process for model development and evaluation. The first phase is where datasets are pre-processed with division into training sets and test sets. For model architecture design we employ three different approaches: CNN, LSTM or CNN-LSTM type. Each architecture has such core layers as input, embedding layer, dense layer (as well as its variant), convolutional and LSTM respectively, depending on the concrete model type. Based on the respective models, we train them for assessment according to preset performance metrics. After evaluating the training results of these models if the performance of any of them was not up to mark, modifications were made and re-trained models until desired performance was achieved through training. This iterative process ensures the models are optimized for accurate SQL injection detection. The figure [3] encapsulates a comprehensive pipeline for designing and refining robust Deep learning models.

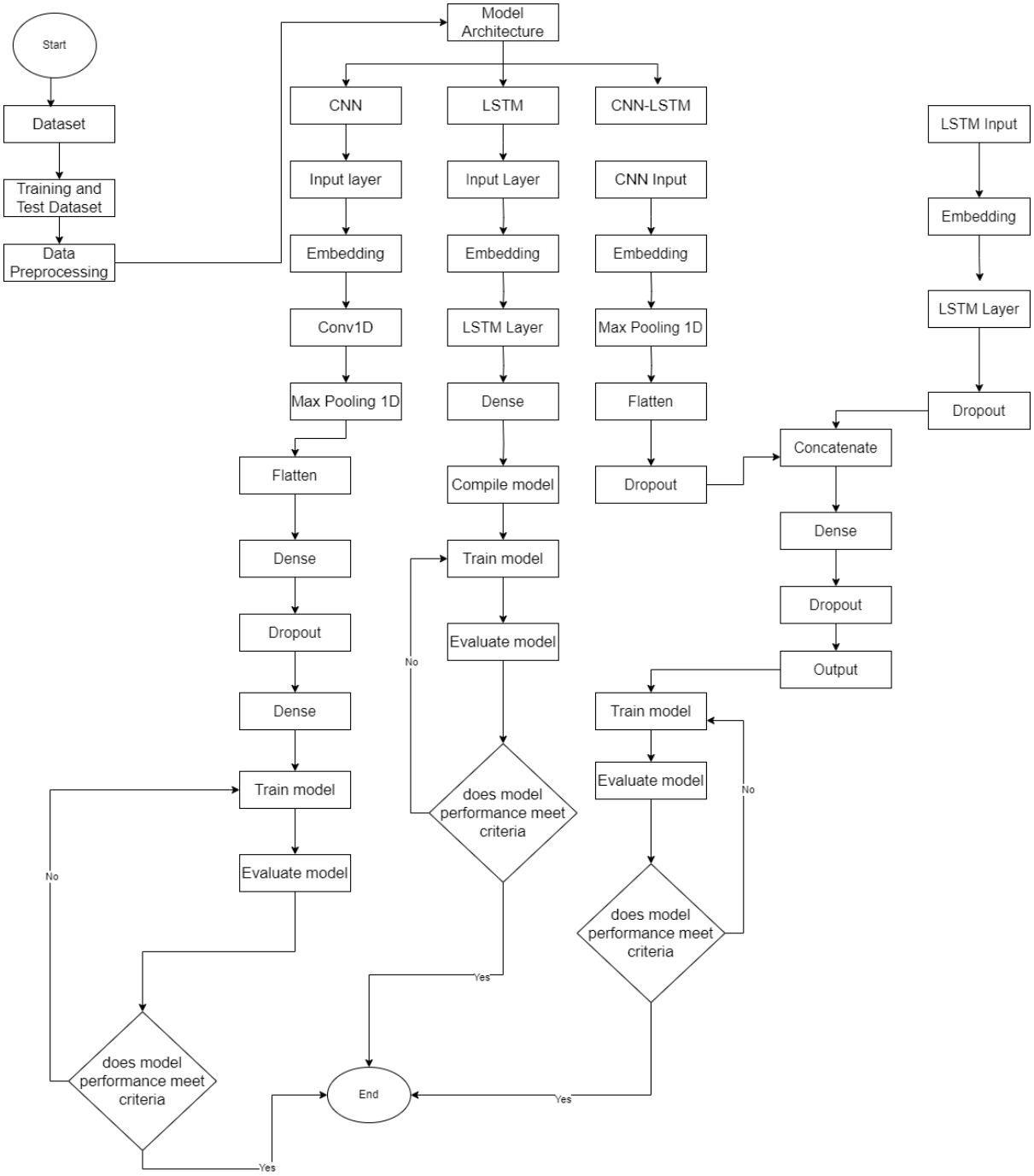


Figure 3 Flowchart of proposed models

3.5. Data Preprocessing

3.5.1. Dataset Cleaning by Applying Different Techniques

This pre-processing pipeline is outlined in the following table, which demonstrates how the data is prepared for training with deep neural networks, to ensure that it is clean, consistent, and ready to be used. The first step is to validate the two required Features, which are 'label'

and 'query', necessary for any deep learning task. The individual columns with missing values are then dealt with in a systematic manner: rows that have missing labels are dropped altogether, whereas empty strings are introduced to fill missing text in the 'query' column specifically. We convert the 'label' column to numeric format and remove any rows that were not able to convert, ensuring us that we have a consistent data. After dealing with missing values, the dataset is split into features - 'query' and labels - 'label' for further processing.

3.5.2. Tokenization

A Tokenizer processes the text data in the 'query' column, transforming it into numerical sequences. When it comes to tokenization, it allows us to separate the text data into a form that is more fitting for computational use. This preprocessing is crucial for allowing the deep learning architecture to process and identify patterns within the text inputs.

3.5.3. Padding

All tokenized sequences are padded to ensure all the input lengths are equal to max length, which is the longest sequence in the dataset. Padding augments shorter sequences with zeros so that every input to the model will have the same dimension. This uniformity is key when going through training and evaluation processes smoothly, as deep learning models often demand fixed sizes for their inputs. Taken together, this powerful preprocessing pipeline prepares the dataset for the training of accurate and reliable SQL injection detection models.

3.5.4. Dataset Balancing

As shown in Figure 1, the dataset was initially imbalanced in terms of class distribution. Now for the dataset before cleaning, the benign class (label 0) was found in 57.66% of the data, with malicious class (label 1) taking up 42.34% of the data. This class imbalance was problematic for training machine learning models because they would tend to be biased towards the majority class, leading to poor detection performance of the minority class. Hence, we removed the entries in the "Other" category to prepare and clean the data for a binary classification problem. Nonetheless, this was still imbalanced class, which was also to also be made sure that there was no pull back the performance of the model. To do so, we used Synthetic Minority Oversampling Technique (SMOTE) to balance the class distribution. Synthetic Minority Over-sampling Technique (SMOTE) has been used to create additional

samples for the minority (in this case, the attack samples) by generating synthetic samples where the new samples are generated by interpolating between the existing samples which balances the number of samples for both the benign class (label 0) and the malicious class (label 1).

As shown in Figure 3, applying SMOTE balanced the dataset, with equal records for both classes. Additionally, this solution ensured that deep learning models had a stronger basis to learn how to identify malicious SQL queries from legitimate ones, without an inherent bias towards the majority class. This ultimately enhances the model's generalization capabilities to real-world scenarios and increases the predictive performance of the model for SQL injection attack detection.

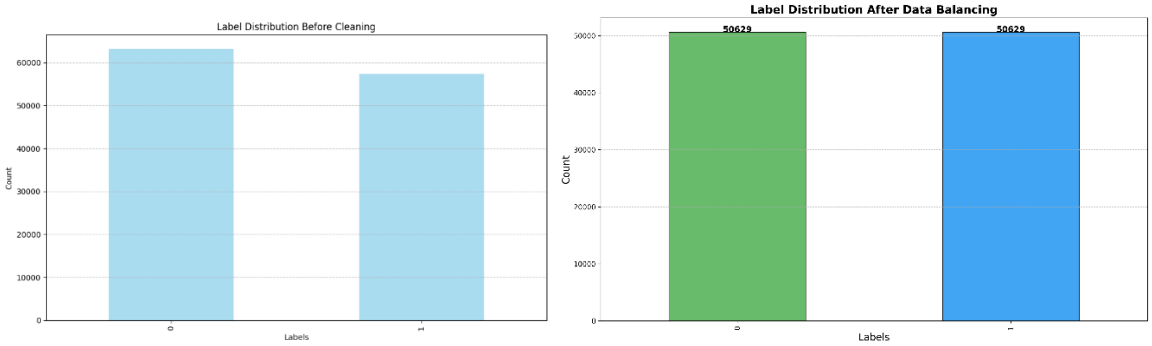


Figure 4 Data distribution

3.5.5. Data Splitting

One of the crucial stages of preparing the data for the purposes of model training involved the division of the dataset into two separate subsets: a training set and a testing set. In this experiment, the dataset was divided into 80% training and 20% testing which means an 80-20 split ratio was used in this study.

3.5.6. Data Splitting Objectives

Effective Model Training: The training set was necessarily larger to expose the model to the maximum number of examples. It exposed the model to a diverse set of examples, enabling it to learn complex patterns and relationships in the data.

Unbiased Testing: By using the testing set (20% of the dataset), we were able to give an independent and unbiased evaluation of the performance of model. This way, the assessment was a reflection of if the model was able to generalize to unseen data.

3.5.7. Why 80-20 Split Ratio?

Several key benefits motivated the adoption of the 80-20 split ratio:

- Large Training Dataset: The training subset was large enough to obtain sufficient examples for the model to learn how to recognize patterns and able to predict its performance.
- Reliable Testing Benchmark: Utilizing a separate dedicated testing subset placed a solid foundation for an accurate and independent assessment of the model's performance in scenarios showcasing real-world data while maintaining an untainted outline in terms of training data influences.
- Effective Use of Resources: It optimized the utilization of the dataset for training while ensuring a significant portion remained for testing, balancing the requirement for thorough learning with the need for sound assessment.

3.6. Model Selection

The models chosen for this study are Convolutional Neural Networks (CNN) and Long Short-Term Memory Networks (LSTM). SQL INJECTION detections can be simply improved by integrating these models in a sense which could be overlapped both in terms of sequential and unstructured data analysis.

3.6.1. Convolutional Neural Networks

Like Convolutional Neural Networks that were built for images but can also work very well with text data. For SQL Injection detection, CNNs examine the tokenized configuration of SQL queries to recognize local patterns and relationships. Convolutional layers are used to extract features, and pooling layers are used to reduce the dimensionality while keeping the most essential information. CNN approaches work well for identifying structural patterns in query tokens, such as specific keywords, operators, or symbols that indicate malicious activity (e.g. UNION, --, 1=1). When well optimised for computation they are suitable for real-time

detection thanks to their architectural efficiency for inference. The local feature extraction mechanism introduces robustness to small changes in the structure of the query. Nonetheless, CNNs fall short when it comes to capturing long-range dependencies or sequential relationships, which are crucial for detecting complex SQL Injection attacks. If your query structures are likely to be static, CNNs will work well for identifying obfuscated or zero-day SQL Injection patterns.

3.6.2. Long Short-Term Memory Networks

Long Short-Term Memory Networks (LSTMs) – a kind of RNN that can learn long-term dependencies between autoregressive types of sequential data. SQL queries are by nature sequential, which means that the positioning of specific keywords and operators is key to discerning malfeasance. LSTMs utilize memory cells and various gates (input, forget, and output gates) to store pertinent information for an extended period, making them well-suited for recognizing attack patterns based on the sequential arrangement of the different components of a query. LSTMs are proficient in learning context and association within a sequence that can help to identify the purpose of a sophisticated query (' OR 1=1 --) or multi-line attacks. They are a good match for detecting temporal dependencies and modelling the behaviour of attack payloads. While LSTM can attain high accuracy, their training time is computationally expensive, and they are better suited for batch analysis than for real-time explicit detection.

3.6.3. Hybrid Model

To overcome the limitations of individual models, a Hybrid CNN-LSTM model is used. This new model provides a detailed framework for SQL Injection detection by combining the spatial pattern recognition capability of CNNs with the sequential understanding of LSTMs. The hybrid model has two main components, the embedding layer plays the role of turning the tokenized SQL queries into a dense vector where each token was represented and the semantic relationship between them was captured. The convolutional layer recognizes spatial patterns in the split input, for example there can be found co-occur keywords and symbols. The pooling layer reduces the dimensionality of the feature map and retains important information. SQL is composed out of tokens which have a clear order that the model needs to learn sequential dependencies this is where the sequence LSTM layer comes in. The output

of the CNN and LSTM layers are merged in a fully connected layer that is used for classification, and the probability score obtained through the sigmoid activation indicates whether the sample used for the model inference is malicious or benign. Such a hybrid approach is capable to view both the structural and sequential properties of SQL queries and can achieve high accuracy for detecting obfuscated and complex attacks. It improves prediction accuracy of model, thereby aiding in reducing false positive and false negative and ultimately providing a balanced solution for sql pattern handling.

3.7. Model Training

After preprocessing the data, the second stage in a deep learning pipeline is model training. The data is divided into two parts: the training and the testing set with a ratio of 80-20. With 80% of the data used for training and 20% for testing purposes.

3.7.1. CNN Model Training

The CNN model starts with an Embedding layer, which maps input sequences to dense vectors of embedding_dim size, appropriate for the subsequent layers. Following this layer is a Conv1D layer with 16 filters and a kernel size of 3 which uses the tanh activation function to extract some spatial features from the embedded input. Next, a MaxPooling1D layer is used to reduce dimensionality through pooling over a window size of 4 which helps keeping necessary features and reducing computational complexity. It's the Flatten layer that joins it together to make a long array of it, just like it needs to be for a fully connected layer. Following that, a Dense layer with 2 neurons and a tanh activation function adds non-linearity to the model, which is then followed by a Dropout layer with a 0.4 rate that randomly sets some outputs to zero during training to help prevent overfitting. A single neuron with a sigmoid activation function in the final Dense layer provides a probability score, indicating that the model is appropriate for binary classification. The Adam optimizer is used to compile the model with a learning rate = 0.001 to give a desired learning dynamic. During training the binary cross-entropy loss function is used, and accuracy is used as a performance metric to see how well the model is learning. The above architecture tries to have a good balance between the simplicity of the model and computational efficiency while catering to the binary classification task well.

3.7.1.1. CNN Model Architecture

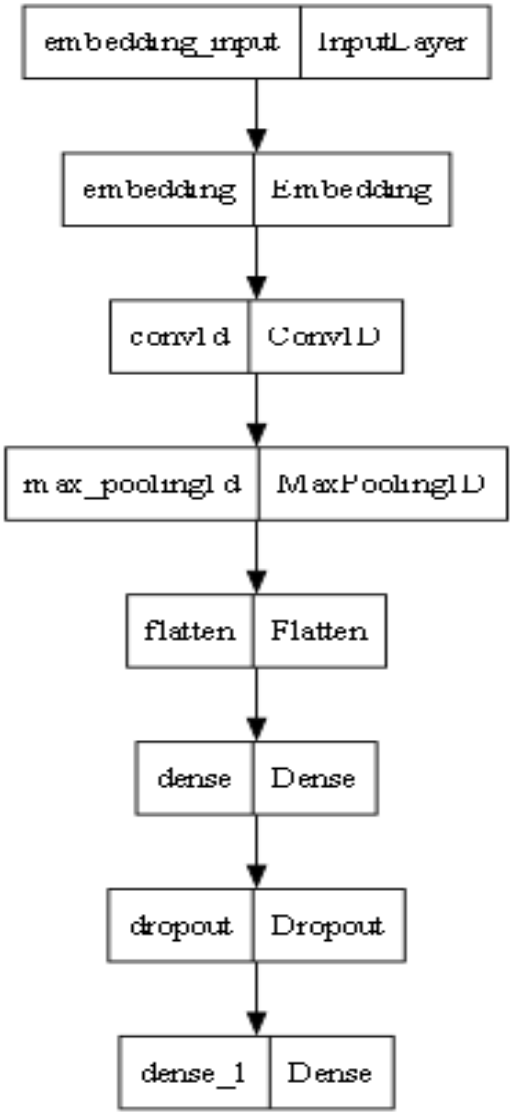


Figure 5 CNN Model Architecture created using Graphviz

3.7.2. LSTM Model Training

An LSTM based architecture for binary classification and analysis of temporal data. It starts with a trainable Embedding layer to encode the input sequences into dense vectors with a vocabulary size of 70 and the embedding dimension of embedding_dim. It thus gives the model an opportunity to learn task-related representations as part of its training. This will be followed by a predictive layer of type Bidirectional for LSTM 16 units having return_sequences=True. Next, we pass the output from the LSTM layer

to a Dense layer with a single neuron and a sigmoid activation function, which will give us the probability score for binary classification.

The model is compiled with the Adam optimizer (used in the previous model), we use the binary cross-entropy loss function, which is appropriate for binary-class problems. The performance of model learning is evaluated using accuracy as the performance metric. Bidirectional LSTMs combine the well-known advantages of LSTMs in the presence of temporal dependencies in sequence data capture with simplicity to ensure efficient training and generalization.

3.7.2.1. LSTM Model Architecture

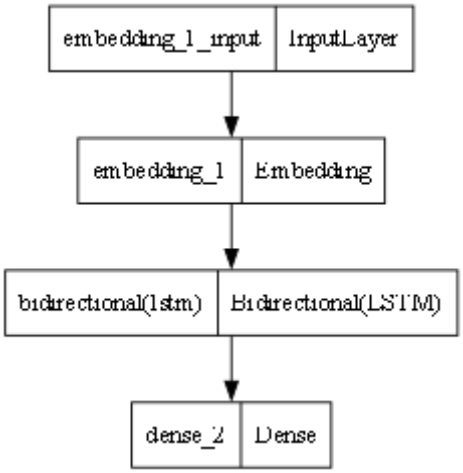


Figure 4 LSTM Model Architecture created using Graphviz

3.7.3. Hybrid Model Training

This Hybrid model employs the advantages of both CNN and LSTM to accomplish an effective SQL injection detection. The CNN branch begins with an input layer set to take sequences with a maximum length (max_len), followed by an embedding layer where there are 70 input dimensions and an embedding dimension defined by the user. In this branch, they use a 1D convolutional layer with 32 filters and a kernel size of 3 (as in previous papers), activated through ReLU to extract spatial features from input sequences. So, MaxPooling which is used for dimensionality reduction, followed by flattening the output and then dropout is used in order to avoid overfitting. The LSTM branch has the same embedding layer as the CNN branch

for processing the sequences. An LSTM layer with 16 units is used to extract temporal dependencies and contextual knowledge from the sequence data. Once more, dropout is used to improve generalization.

In this architecture, spatial and temporal features are concatenated from the CNN and LSTM branches. There is a dense layer with 32 units and ReLU that processes the merged representation followed by another dropout layer After obtaining the features from Time Distributed layer, we pass the output to an Average layer and finally to a Dense layer with sigmoid activation function that outputs a single value from the model, thus suitable for binary classification problems (i.e. classifying whether the SQL query is benign or malicious).

It compiles the model with the adam optimizer with 0.001 learning rate, sets binary cross entropy as the loss function and sets accuracy as evaluation metric.

3.7.3.1. Hybrid Model Architecture

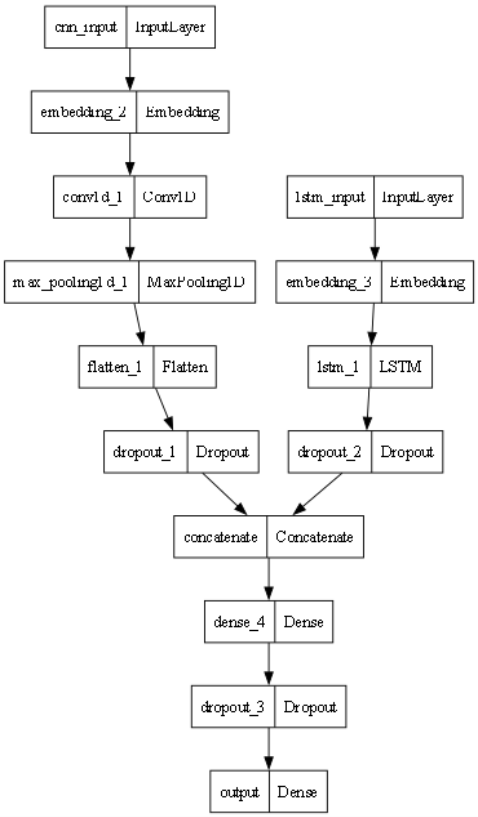


Figure 5 Hybrid Model Architecture created using Graphviz

3.8. Model Optimization and Parameter Tuning

Once the dataset is split into their respective training and testing subsets, the experimentation on the models begins. The time needed to train a particular model depends on its architecture and complexity so each model has its own training process. These variations highlight the distinguishing features of each form of model, in this case, CNN, LSTM, and the combined CNN-LSTM, such that they leverage and learn from diverse strengths and techniques. The models are trained in an iterative process, where the models adjust their internal parameters to discover and learn the complex patterns and relationships in the dataset. Batch Normalization and other methods are used throughout the training process to stabilize and speed up learning, allowing the system to converge faster and make more generalizable predictions. Not only does this phase focus on the optimization of the models, it also draws attention to the trade-off between computational efficiency and model complexity. By recognizing such trade-offs, the study guarantees the creation of agile, effective, high-performing models responsible for SQL injection detection and that manages varied real-world settings.

3.9. Model Testing

Model testing is an important step in machine learning to evaluate the performance of a trained model on data that it did not see. This step is crucial to evaluate whether the model can generalize to new, unseen examples and how well it will perform in practical applications. In this project, model testing aims to estimate whether the trained models—CNN, LSTM, and hybrid CNN-LSTM can successfully identify SQL injections or not. Testing models is primarily focused on generalization and robustness. In this stage, the testing dataset (20% of the total data) is applied to the trained models. A performance metric of accuracy, precision, recall, F1-score, and area under the ROC curve (AUC) is provided for each model output. These metrics give a detailed insight into the strengths and weaknesses of the model in prediction of SQL injection attacks. Batch Normalization, which can be included in the training process, is crucial in maintaining the stability of the models' performance in the testing phase by transforming input distributions of the layers to be normalized and consistent. This leads to better accuracy and reliability in the evaluation process. Additionally, when we test our models, we also examine false positives and false negatives to find areas of potential

improvement in the models. For practice, false positives which classify benign queries as malicious and false negatives which classify malicious queries as benign are closely analysed to help us understand the limitations of the models and improve them if required. A technical detail on the model testing process is provided in the following "Technical Evaluation" chapter. Here, we discuss our findings in-depth, covering the advantages, disadvantages, and best use cases for each model, including which models can be deployed in production and under what conditions they will (not) be appropriate. This thorough evaluation ensures the accuracy as well as the reliability and robustness of the models for practical SQL injection detection scenarios.

3.10. Real Time Testing.

we have setup a sever using flask to test my system for real time detection which is running on port 5000 locally. Below figures shows the result of my real time detection system.we used postman to make request to our server .

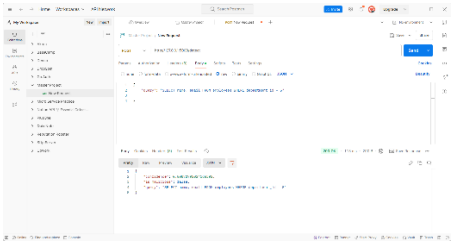


Figure 6 Melicious Query real time detection

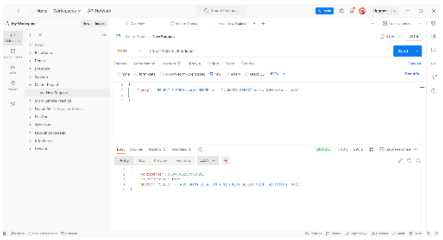


Figure 7 Benign Query real time detection

3.11. Graphviz

Graphviz is a powerful open-source software package for visualizing structured graphs of complex data. That is a common application for generating graphs and plots. Graphviz takes in graph descriptions in the DOT language, which is a simple plain-text format, and uses them to create beautiful output files, such as PNG, SVG, or PDF. For this project, we employed Graphviz to create visual models or diagrams of the model architectures, providing a clear and intuitive visualization of the structure and relationships between the layers within each neural network. These visualizations were useful for understanding the models' structure, and also helped us in analysing and presenting our models. Using Graphviz, you can see the complexity of the model in one look and communicate architectural decisions effectively. Its close integration with Python libraries like TensorFlow and Keras, made the workflow as smooth as possible, so that it became a requirement for the success for this project.

3.12. GitHub

GitHub was used in this project to manage the codebase as it is a version control and a collaboration platform. Scripts for data preprocessing, model training, and visualization were uploaded to a private repository on GitHub along with other files from the project. This allowed for secure storage, and easy access to the project's source code and related resources. Moreover, GitHub allowed for uninterrupted tracking of changes to code, facilitating the continuous enhancement of programs while preserving a clear history of changes. Additionally, and enabled collaboration and access to the progress from any device. This project was built using git and hosted on GitHub(code hosting service), ensuring project integrity and reproducibility.[28]

3.13. Technical Evaluation

However, in addition to accuracy, it is critical to assess the performance, functionality, and efficiency of the system to ensure practical applicability and reliability. This part cites the technical aspects of the work done throughout the project, analysing the models and their performance, aligned with the goals these models aimed for. This assessment includes various facets, for instance the computational resource usage of the models, response times, and

prediction stability across modified environments. Also, the system is tested for the robustness by examining the diversity of SQL structures and attack patterns, which makes it more probable to generalize for the real scenarios. In addition, the technical evaluation details performance metrics like precision, recall, and AUC to provide a deeper understanding of both the strengths and potential weaknesses of the system. It establishes an integrated evaluation environment for the designed system, so its accuracy is accompanied by its performance, allowing it to operate as an efficient tool to be deployed on real life SQL injection detection jobs.

3.13.1. Confusion Matrix

To analyse how well the developed models perform further confusion matrices were filled out based on the predictions. The confusion matrix has four important components:

- True Positives (TP): Number of attack queries detected correctly as attack.
- True Negatives (TN): The number of benign queries which have been correctly classified as benign.
- False Positives (FP): The benign queries incorrectly classified as malicious.
- False Negatives (FN): Benign queries classified in the model as malicious.

The confusion matrix is one such tool that allows the visualisation of the classifier performance along with the calculation of classification metrics like precision, recall and F1 score. The confusion matrix thus guides us to the specific areas of the model that may need improvement through analysing false positives and false negatives, thus balancing the performance across classes.

Table 1 Confusion matrix

	Predicted Negative	**Predicted Positive**
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

3.13.2. Accuracy

Accuracy measures the overall correctness of the model. It calculates the proportion of correctly classified queries (both benign and malicious) out of the total number of queries.

Accuracy = (TP + TN) / (TP + FP + FN + TN) [4]

3.13.3. Precision

Precision focuses on the model's ability to minimize false alarms. It evaluates the proportion of correctly predicted malicious queries among all queries predicted as malicious.

$$Precision = \frac{\text{TP}}{\text{TP+FP}} \quad [4]$$

High precision indicates that when the model predicts a query as malicious, it is likely to be correct. Precision is crucial in SQL Injection detection to avoid unnecessary disruptions caused by false positives.

3.13.4. Recall (Sensitivity or True Positive Rate)

Recall measures the model's ability to detect all actual malicious queries:

$$Recall = \frac{\text{TP}}{\text{TP+FN}} \quad [4]$$

High recall ensures that the model catches most malicious queries, minimizing the risk of undetected attacks. However, focusing solely on recall might increase false positives.

3.13.5. ROC-AUC (Receiver Operating Characteristic - Area Under Curve)

The ROC-AUC evaluates the model's ability to distinguish between classes (malicious vs. benign queries) across various classification thresholds:

1. **ROC Curve**: Plots the True Positive Rate (Recall) against the False Positive Rate.
2. **AUC**: The Area Under the ROC Curve measures the overall performance of the model. An AUC of 1 indicates perfect classification, while 0.5 indicates random guessing.

The ROC-AUC metric is particularly helpful in comparing models when threshold-independent performance needs to be assessed.

4. Result

In this section, we introduce the experimental results that evaluate how the proposed models perform for the problem of SQL injection detection. Evaluations were conducted to determine how well the models were able to classify benign and malicious SQL queries. The outputs focused particularly on the proposed CNN-LSTM based hybrid model trained and tested on a labelled dataset. These results were obtained from a binary classification problem focused on separating valid queries from potentially harmful SQL injection attacks, demonstrating the applicability of the model for security within web applications in real time.

4.1. Performance Evaluation

In this part we summarize the effectiveness and efficiency of the models based on the three datasets.

4.1.1. Results of CNN, LSTM, and CNN-LSTM Models during Training

Figure [5] compares the classification performance of CNN, LSTM and Hybrid models using Accuracy and Loss curves on training and validation datasets. The CNN model steadily increased the train accuracy in the early epochs and reached about 90% at epoch 5 and around pale at epoch 12. The loss curve for CNN showed consistent convergence of the model with a constant validation loss varying gently around the training loss indicating reasonable generalization. The CNN model did perform well, but it showed signs of having reached its peak quite quickly as it struggled to account for long-term dependencies between successive data points that you might often find in sequences. Due to its ability to capture sequential dependencies, the LSTM model performed much better. In the first 10 epochs, it had more than 95% accuracy on both training and validation datasets. A small scale up and down trend in training accuracy curve, according to orchestration of hyperparameter or randomness in sequential data. The curve for the loss showed fast decay and then convergence to stable low values with small oscillations probably revealing a small overfitting. Nevertheless, LSTM model had significant performance benefits over the CNN model because the nature of SQL query data is sequential in nature and the LSTM model can learn longer temporal relationships. The consistent Hybrid model incorporating CNN and LSTM architectures was the optimal-performing model. Training accuracy exceeded 95% by epoch 3 and validation accuracy reached around 98% by epoch 6. Training and validation losses for this model plotted

against epoch number show a strong drop off in loss at the beginning followed by a relatively plateauing validation and training loss (indicating that both losses are at similar low values). This behaviour showcases powerful generalization with scarce overfitting. Modelling with Hybrid approach the advantage of CNN layers extracting information from spatial features that can extract the common patterns of SQL injection in the data, combined with the sequential learning features in LSTM layers made the Hybrid model more comprehensive in learning complex patterns in the dataset. This helped it produce better accuracy and stability and faster convergence than the individual CNN and LSTM models

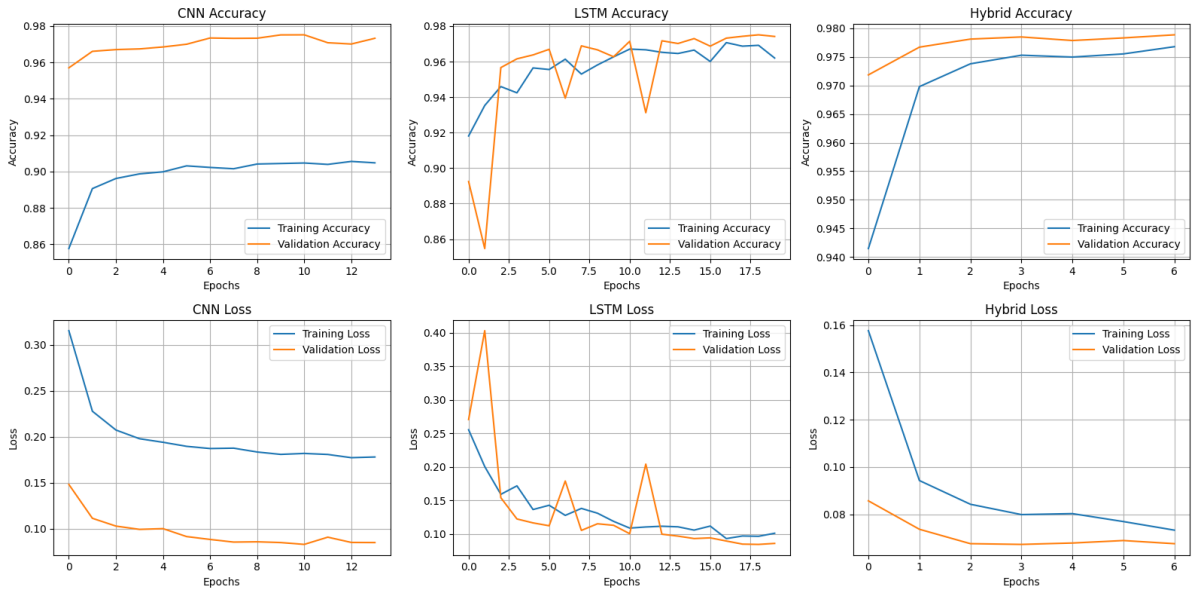


Figure 8 Training and loss Graph

4.1.2. Performance Analysis

The bar chart shown in the image depicts a detailed performance comparison for three models, i.e., CNN, LSTM, and the Hybrid on the SQL injection detection task. The analysis evaluates several performance metrics such as: accuracy, precision, recall or sensitivity, and ROC-AUC which are fundamental metrics of a model ability of classifying benign and malicious SQL queries.

4.1.2.1. Performance Metrics

Figure [6] shows the CNN model performs well overall but is, however, the worst on both accuracy and recall metrics of all the models. While CNNs are effective in evoking spatial features, the lack of temporal dependencies in CNNs makes it underutilized for SQL query

patterns. Even with this drawback, its accuracy and ROC-AUC scores show that it can reliably classify positive instances with high confidence. While, the performance of the CNN model indicates the efficiency of this approach, it is also indicative of its lack of understanding of temporal relationships, which is critical to realizing subtle patterns in SQL injection queries. The results demonstrated a significant improvement in the LSTM model performance as compared to the CNN model, especially in terms of accuracy and recall. As sequential data such as SQL queries have significant long-term dependencies and temporal relationships, the architecture of RNNs suits beautifully to this problem domain. The LSTM model significantly outperforms all others and demonstrates balanced and stable performance across all, suggesting good generalization to unseen data. The increased recall also indicates that it succeeds at correctly identifying a greater share of truly harmful queries, making it the front-runner for real-time SQL injection detection tasks. Among all these methods, the Hybrid model, which combines the unique advantages of the CNN and LSTM architectures, presents the best performance. The model has attained the highest scores in all metrics by merging the spatial feature extraction potentials of CNN layers with the sequential learning powers of LSTM layers. Its accuracy, precision, recall, and ROC-AUC are consistently better than the baselines, evidence of its ability to capture both local and temporal patterns of the data. Additionally, recognizing existing limitations in classifying inherently complex datasets, the model of choice for detecting SQL injection is the Hybrid model, given its superior performance in terms of both generalization and classification.

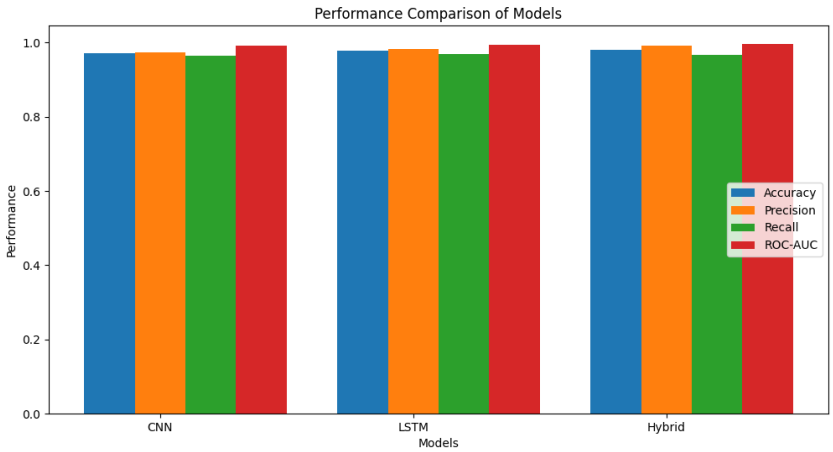


Figure 9 Performance Comparison

Table 2 Performance Comparison Table

Metric	CNN	LSTM	Hybrid(CNN-LSTM)
Accuracy	0.9708	0.9772	0.9808
Precision	0.9746	0.9825	0.9914
Recall	0.9636	0.9692	0.9679
ROC-AUC	0.9925	0.9940	0.9963

4.1.3. Confusion matrix

The confusing matrix result[7] shows the BiHave performance using CNN, LSTM and Hybrid models on detecting the benign and malicious SQL queries. The assessment focuses on how well each of the models can classify true positive (correctly predicted new malicious queries), true negative (correctly predicted new benign queries), false positive (benign queries misclassified as malicious) and false negative (malicious queries misclassified as benign). The matrices feature a comprehensive assessment of the advantage and disadvantage of each model, which is valuable in your task of identifying applicable models to detect SQL injection.

CNN Model Confusion Matrix, The CNN model performs well in correctly classifying 12,369 benign queries (true negatives) and 11,045 malicious queries (true positives). It does, however, show 288 false positives, benign queries that were improperly reported to as malicious and 417 false negatives, malicious queries that were classified as benign. Because the CNN model cannot capture the sequential dependencies between the input signals, the model is not as effective in making predictions as LSTM, CNN resulting in its false negative rate being relatively higher. This is anticipated because CNN filters a large region of data relatively than plumbing the temporal patterns essential to the classification of SQL queries.The LSTM model outperforms other models in classification performance, this is

mainly because it can capture sequential dependencies. It identifies 12,459 benign queries (true negative) and 11,109 malicious queries (true positive) and reduces that false negative down to 353. The number of false positives yielded by the model was also very low (198), indicating good generalizability of the model. This considerable enhancement over the CNN model highlights the importance of the sequential approach in SQL injection detection problems. The slight imbalance between these numbers suggests further tuning of hyperparameters would improve its preciseness of detection. Hybrid Model Confusion Matrix It can be seen that the Hybrid model which combines the benefits of CNN and LSTM, performed better than the other two models based on each evaluation metric. It can also be observed that it has the maximum & minimum True negative 12,561, True positive 11,094, False positive 96 and False negative count 368 respectively. The proposed method, Hybrid model, extracts spatial features using CNN, and sequential features using LSTM, making it possible to extract the complex association of SQL queries. In addition, it also has the strongest capability and effectiveness for real-time SQL injection detection, as it has the best performance for reducing the false positive and negative rate.

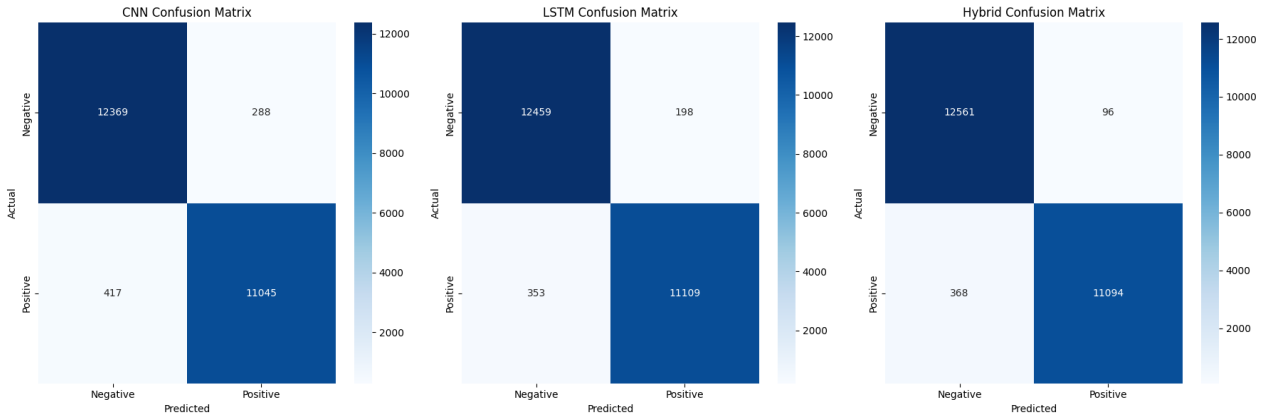


Figure 10 Confusion matrix result

Table 3 Confusion matrix result

Rate	CNN	LSTM	Hybrid
False Positive (FPR)	0.0228	0.0156	0.0076
False Negative (FNR)	0.0364	0.0308	0.0321

4.1.4. ROC Curve

The ROC (Receiver Operating Characteristic) curves[8] depicts the accuracy of the CNN, LSTM, and Hybrid in classifying SQL as benign or malicious. The high Area Under the Curve (AUC) scores of the three models suggest strong performance. Both CNN and LSTM models got AUC 0.99 which shows that for both cnn and lstm model is 99% chance that query will be detected as benign or malicious. Nevertheless, the Hybrid model, combining the feature extraction capabilities of CNN and the sequential learning strengths of LSTM, proved the best with an AUC of 1.00 which shows the near-perfect classification. The Hybrid Impressive Performance with Balanced Precision and Robustness the Hybrid model excels with its balanced precision and AUC score, indicating a robust ability to distinguish between good and bad SQL queries. As such, these results demonstrate that the Hybrid model outperforms the others, making it the best overall choice for accurately detecting real-time SQL injection, where it is crucial to minimize false alarms while maximizing detection accuracy.

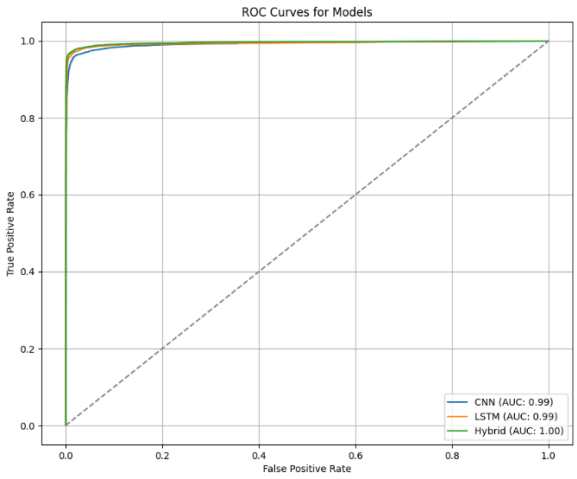


Figure 11 ROC Curves result

4.1.5. Computational Cost

The computational cost analysis of the models CNN, LSTM, and Hybrid gives insights on the resource-efficient nature, i.e. training time and memory usage, of these models, making it apt

for real-time deployment of SQL injection detection systems. Figure [9] shows the comparison between computational requirements and performance.

Among all models, the CNN model achieved the biggest computation performance, with only 79.83 seconds required for training. But it used a fair amount of memory at 794.38 MB. Thus, we have a faster training time with CNN as compared to RNN but CNN is not optimal because it is not suitable for capturing complex sequential dependencies, and SQL injection detection requires capturing significant sequent dependencies between tokens. Although the LSTM model has great potential and can effectively model temporal dependencies, it was the most expensive in terms of computing resources, taking 312.60 seconds to train and using 1075.98 MB of memory. Due to this high resource demand and its comparatively high training time, LSTM is also less favourable when it comes to real-time applications, where time and efficiency is the key requirements. The Hybrid approach, benefiting from the complementary nature of CNN and LSTM, achieved the highest performance-to-cost ratio. The Hybrid model achieved those high accuracy and stability mentioned above with a training time of 321.48 seconds and a memory usage of 327.12 MB which is much lower than LSTM stability at 747.06309 memory allocation. Hybrid model's ability to extract both spatial and sequential features from the sequences makes it highly competent for SQL injection detection tasks despite a marginal increase in training time.

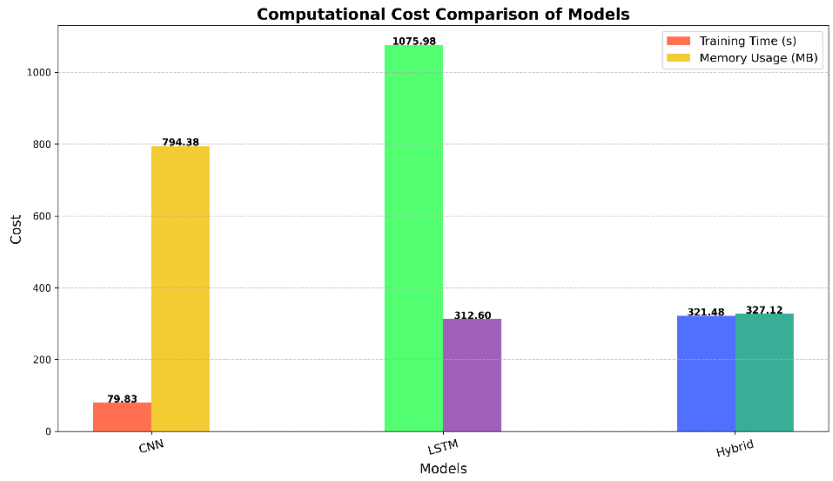


Figure 12 Computational Comparison

Conclusion

This research provides an in-depth assessment of the performance of CNN, LSTM, and Hybrid solutions for SQL injection detection, with particular emphasis on significant performance indices such as accuracy, precision, recall, and computational efficiency. Results reveal the advantages and pitfalls of each model, suggesting that the Hybrid model may be the best choice for developing this family of translational models peripherally due to its improved ability to strike a balance between total predictive accuracy and computational cost.

While the CNN model showed high efficiency in training time, it was ineffective in capturing sequential dependencies that are relevant for SQL injection detection. Conversely, the LSTM model performed exceptionally well with sequential data, yielding superior accuracy . But it is computationally more expensive in terms of memory and time to train and is therefore not suitable in resource-starved regimes. The Hybrid model, which utilizes the advantages of both CNN and LSTM architectures, performed the best. It performed the best across accuracy, precision, recall, and computational efficiency. It highlights the effectiveness of machine learning catering towards spatial and sequential features due to which this model can be considered as one of the strong candidates for Realtime SQL injection attack detection.The model had success due to the combination of the techniques applied, such as balanced datasets (SMOTE), optimized feature selection and designing hybrid architectures. By correcting data imbalance and building focus on high-impact features, the study achieved well-calibrated accuracy and generalizability.

As a final point, Hybrid yields the best detection for SQL injection given the resource and performance trade-off in comparison to other solutions. Overall, this study illustrates the value of hybrid strategies in cybersecurity, where both accuracy and efficiency matter most. Future studies can build on these models by testing robustness and scalability in dynamic and complex environments by adding, for example, some pre-trained embeddings and richer regularization mechanisms.

Future Directions

SQL injection detection by deep learning models has a great potential for future enhancement. Though present study has shown great results using CNN, LSTM and Hybrid

models and still there is a lot of potential for future work with advanced techniques for improved accuracy, robustness and computational efficiency. Hybrid architecture which combines the strongest points of different architectures such as CNN and LSTM has shown already benefits. By using more sophisticated techniques like transformers or attention-based mechanisms in this method might be able to further strengthen the model's ability to encode spatial as well as temporal features in complex SQL queries. As with a combination of models, improvements in accuracy and reductions in variance in predictions can also be gained from the use of ensemble learning techniques. To make the previous step easier, one must integrate pre-trained embeddings like Word2Vec, GloVe, or BERT, which may provide a deeper semantic understanding of SQL queries, helping the models to generalize better to unseen patterns. Additionally, techniques, such as transfer learning, can be utilized to fine-tune pre-trained models to a particular SQL injection dataset, decreasing the requirements for large amounts of training data and computation resources.To test latency, scalability, and resilience to adversarial for SQL-injection attacks, such an EMR would need to be deployed in production environments with real-time testing. Online learning techniques can also be incorporated to allow models to adapt to constantly changing attack patterns, enabling them to be reliable over the long term.

Finally, ethical and privacy considerations can be addressed through techniques such as federated learning, allowing organizations to train models collectively without exposing sensitive data. The course offers a multitude of opportunities and the potential for real-world impact in the realm of security, with this endeavor leading to the development of scalable, accurate and robust SQL injection detection systems.

References

1. E. Hosam, H. Hosny, W. Ashraf and A. S. Kaseb, "SQL Injection Detection Using Machine Learning Techniques," *2021 8th International Conference on Soft Computing & Machine Intelligence (ISCMI)*, Cario, Egypt, 2021, pp. 15-20, doi: 10.1109/ISCMI53840.2021.9654820.

2. Alghawazi, M.; Alghazzawi, D.; Alarifi, S. Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review. *J. Cybersecur. Priv.* **2022**, *2*, 764-777. https://doi.org/10.3390/jcp2040039

3. M. Hasan, Z. Balbahaith and M. Tarique, "Detection of SQL Injection Attacks: A Machine Learning Approach," *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, Ras Al Khaimah, United Arab Emirates, 2019, pp. 1-6, doi: 10.1109/ICECTA48151.2019.8959617.

4. Chen, Ding, et al. "Sql injection attack detection and prevention techniques using deep learning." *Journal of Physics: Conference Series*. Vol. 1757. No. 1. IOP Publishing, 2021.

5. Jemal, Ines, et al. "Sql injection attack detection and prevention techniques using machine learning." *International Journal of Applied Engineering Research* 15.6 (2020): 569-580.

6. Ross, Kevin, et al. "Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection." *Proceedings of the ACMSE 2018 Conference*. 2018.

7. Pattewar, Tareek, et al. "Detection of SQL injection using machine learning: a survey." *Int. Res. J. Eng. Technol.(IRJET)* 6.11 (2019): 239-246.

8. Muhammad, Taseer, and Hamayoon Ghafory. "Sql injection attack detection using machine learning algorithm." *Mesopotamian journal of cybersecurity* 2022 (2022): 5-17.

9. Deriba, Fitsum Gizachew, et al. "Development of a compressive framework using machine learning approaches for SQL injection attacks." *Przeglad Elektrotechniczny* 98.7 (2022): 181-187.

10. Abdulmalik, Yazeed. "An improved SQL injection attack detection model using machine learning techniques." *International Journal of Innovative Computing* 11.1 (2021): 53-57.

11. Sivasangari, J. Jyotsna and K. Pravalika, "SQL Injection Attack Detection using Machine Learning Algorithm," *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*, Tirunelveli, India, 2021, pp. 1166-1169, doi: 10.1109/ICOEI51242.2021.9452914.

12. K. Zhang, "A Machine Learning Based Approach to Identify SQL Injection Vulnerabilities," *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA, 2019, pp. 1286-1288, doi: 10.1109/ASE.2019.00164.

13. R. R. Choudhary, S. Verma and G. Meena, "Detection of SQL Injection attack Using Machine Learning," *2021 IEEE International Conference on Technology, Research, and*

Innovation for Betterment of Society (TRIBES), Raipur, India, 2021, pp. 1-6, doi: 10.1109/TRIBES52498.2021.9751616.

14. Kavitha, M. N., et al. "Prevention of SQL injection attack using unsupervised machine learning approach." *International Journal of Aquatic Science* 12.03 (2021).

15. Alam, Auninda, et al. *SCAMM: Detection and prevention of SQL injection attacks using a machine learning approach*. Diss. Brac University, 2021.
16. Ross, Kevin. "SQL injection detection using machine learning techniques and multiple data sources." (2018).

17. Demilie, Wubetu Barud, and Fitsum Gizachew Deriba. "Detection and prevention of SQL INJECTION attacks and developing compressive framework using machine learning and hybrid techniques." *Journal of Big Data* 9.1 (2022): 124.

18. Venkatramulu, S., et al. "Research on sql injection attacks using word embedding techniques and machine learning." *Journal of Sensors, IoT & Health Sciences* 2.01 (2024): 55-66.

19. Mustapha, Adeyinka Ayodeji, et al. "Comprehensive review of machine learning models for sql injection detection in e-commerce." *World Journal of Advanced Research and Reviews* 23.1 (2024): 451-465.

20. Moissinac, Béatrice, et al. "Detecting SQL Injection Attacks using Machine Learning." *CAMLIS*. 2023.

21. A. Luo, W. Huang and W. Fan, "A CNN-based Approach to the Detection of SQL Injection Attacks," *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*, Beijing, China, 2019, pp. 320-324, doi: 10.1109/ICIS46139.2019.8940196.

22. A. Setiyaji, K. Ramli, Z. Y. Hidayatulloh and G. S. Budhi Dharmawan, "A technique utilizing Machine Learning and Convolutional Neural Networks (CNN) for the identification of SQL Injection Attacks," *2024 4th International Conference of Science and Information Technology in Smart Administration (ICSINTESA)*, Balikpapan, Indonesia, 2024, pp. 1-6, doi: 10.1109/ICSINTESA62455.2024.10748116.

23. R. Lu, S. Wang and Y. Li, "Research on SQL Injection Detection Model Based on CNN," *2021 International Conference on Intelligent Computing, Automation and Applications (ICAA)*, Nanjing, China, 2021, pp. 111-114, doi: 10.1109/ICAA53760.2021.00028.

24. Hirani, Manav, et al. "A Deep Learning Approach for Detection of SQL Injection Attacks using Convolutional Neural Networks." *Department of Computer Engineering, MPSTME, NMIMS University, Mumbai, India* (2020).

25. 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.

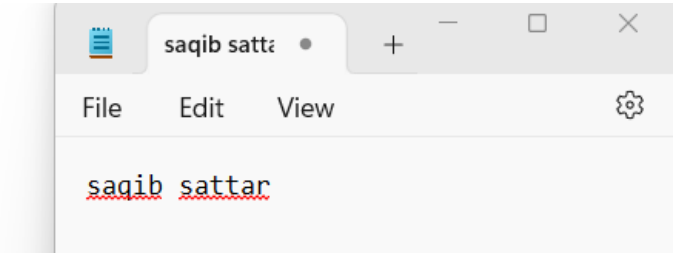
26. https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset/data

27. https://www.kaggle.com/code/alextrinity/sql-injection-detection-v0/input

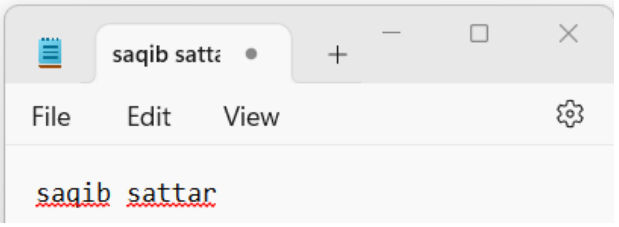
28. https://github.com/saqibsattar03/masterProject

Appendix-A

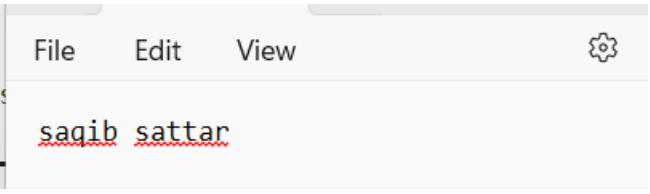
```
---- LSTM Metrics ----
Accuracy: 0.9772
Precision: 0.9825
Recall: 0.9692
ROC-AUC: 0.9940
False Positive Rate (FPR): 0.0156
False Negative Rate (FNR): 0.0308
```



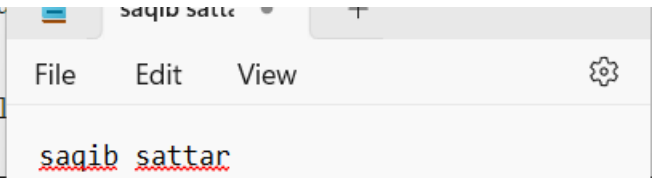

```
---- Hybrid Metrics ----
Accuracy: 0.9808
Precision: 0.9914
Recall: 0.9679
ROC-AUC: 0.9963
False Positive Rate (FPR): 0.0076
False Negative Rate (FNR): 0.0321
Figure(1600x800)
```



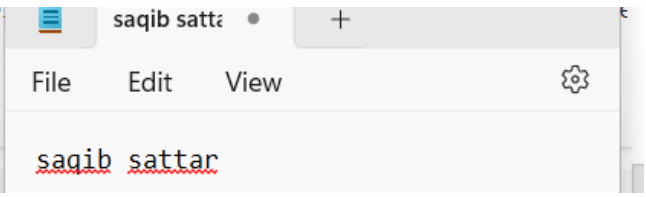

```
CNN Training Time: 50.57 seconds
CNN Memory Usage: 673.49 MB
An error occurred: This model has
Model: "sequential_1"
```




```
LSTM Training Time: 1075.98 seconds
LSTM Memory Usage: 312.60 MB
Model architecture saved successfull
Model: "functional_2"
```




```
Hybrid Training Time: 321.48 seconds
Hybrid Memory Usage: 327.12 MB
Evaluating CNN model...
```



```
754/754 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/st
---- CNN Metrics ----
Accuracy: 0.9708
Precision: 0.9746
Recall: 0.9636
ROC-AUC: 0.9925
False Positive Rate (FPR): 0.0228
False Negative Rate (FNR): 0.0364
```