Scientific Research Publishing

Journal of Computer and Communications, 2025, 13(8), 40-79
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Designing a Detection Model for SQL Injection Attack

Daniel Schilling Weiss Nguyen*, Dawood F. Alrubie

School of Business and Technology (SBT), Aspen University, Phoenix, USA
Email: dans515e@gmail.com, alrubiedawood@gmail.com

How to cite this paper: Nguyen, D.S.W. and Alrubie, D.F. (2025) Designing a Detection Model for SQL Injection Attack. *Journal of Computer and Communications*, **13**, 40-79.
https://doi.org/10.4236/jcc.2025.138003

Received: June 17, 2025
Accepted: August 4, 2025
Published: August 7, 2025

Abstract

SQL injection attacks pose a critical threat to web application security, exploiting vulnerabilities to gain access, or modify sensitive data. Traditional rule-based and machine learning approaches have limitations when applied to real-time web application environments, making detection of injection attacks difficult. Therefore, this study seeks to design and evaluate a Deep Learning-Based SQL Injection Detection System Specifically Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). These models can be utilized to detect SQL injection attacks in real-time web applications. The empirical results show that the proposed model is very accurate, with a mean accuracy of 94.5%, a low standard deviation, and a precision of 96%, meaning that the system correctly identified all the SQL injections as malicious without any false positives. Furthermore, the distributions of the amounts, frequencies, time, and duration of the transactions, all of which are derived from behavioral analyses of the transaction data, were also useful in identifying the existence of unusual patterns of activities that could be associated with fraudulent transactions.

Keywords

SQL Injection Detection (SQLi), Authorized Push Payment (APP) Fraud: Feature Engineering, Transaction Data, User Behavior Patterns, Convolutional Neural, Networks (CNNs), Recurrent Neural Networks (RNNs)

1. Introduction

The Internet has created complex network infrastructures with multiple interlinked devices, servers, and services, which must remain protected from attack. Complexity poses an increasing security challenge, exposing an expanded attack

surface to malicious actors. Web applications' rapid proliferation has resulted in complex ecosystems where data flows freely among various platforms, devices, and cloud services. Web applications can be attacked for various reasons, including system vulnerabilities caused by improper coding, compromised web servers, poor application design, and failure to validate forms [1]-[3]. At least a single flaw that attackers may exploit at greater complexity exists in every web application. These web application vulnerabilities enable criminals to obtain immediate and public access to databases containing sensitive data (such as financial or confidential information), making them a common target of attacks. Although these ecosystems provide businesses and individuals with advanced functionality and seamless connectivity, their complexity also opens them up to an increased attack surface from malicious actors. Network complexity gives malicious actors more chances to penetrate and compromise systems, increasing endpoints and connections over time [4]. As networks become intricate, so do their attack surface: computers, servers, mobile phones, Internet of Things (IoT) devices, and cloud services can all be targeted by cybercriminals. Structured Query Language (SQL) Injection stands out among the myriad online attacks plaguing the digital world as one of the most notorious and menacing threats.

SQL injection attacks exploit vulnerabilities in web application inputs to inject malicious SQL queries directly into their databases and cause data breaches, unauthorized access, data manipulation, and even compromise of entire systems. According to [5], many web applications collect user input through forms, URL parameters, or cookies and use this input in SQL queries that interact with databases. Unscrupulous web applications may fail to validate or sanitize user data before including it in these SQL queries, leaving their databases vulnerable. Attackers can leverage this vulnerability to inject malicious SQL code [6]. Attackers can input specially crafted SQL statements as part of user input; for instance, instead of entering their valid username and password, they might type something like: OR 1=1 --. When web applications construct SQL queries based on user input, they often do not differentiate between user-provided text and SQL code injection. As a result, some of this code ends up as part of the query and is then executed as expected by database servers. As demonstrated above in the example 'OR 1=1--,', the 1 = 1 condition constantly evaluates to true, bypassing any login checks [7]. An attacker could use this vulnerability to gain unauthorized access to an application or database. Conventional rule-based methods and basic input validation techniques have been employed to mitigate these threats; however, their effectiveness remains limited due to attackers' ever-evolving strategies and web application's increasing complexity. SQL Injection stands out among the myriad online attacks plaguing our digital world as one of the most notorious and menacing threats. SQL injection attacks exploit vulnerabilities in web application inputs to inject malicious SQL queries directly into their databases and cause data breaches, unauthorized access, data manipulation, and even compromise of entire systems. Conventional rule-based methods and basic input validation techniques

have been employed to mitigate these threats; however, their effectiveness remains limited due to attackers' ever-evolving strategies and web application's increasing complexity.

1.1. Statement of the Problem

SQL injection-based APP fraud is an unexplored threat in online banking. Such an attack involves hacking of web applications with the aim of manipulating banking systems in order to approve fraudulent transactions using SQL injection techniques. As mobile banking applications interact with both mobile devices and the bank's backend database, a successful SQL injection attack could have ripple effects across multiple platforms [8]. An attacker who compromises one part of the system could gain access to others, potentially affecting the bank's entire IT infrastructure. The lack of coherent and timely detection systems weakens financial institutions, exposes them to massive losses, and diminishes customers' confidence in online banking. SQL injections occur when attackers exploit weaknesses in how a bank's application processes user input, allowing them to inject malicious SQL commands into the database. This can lead to unauthorized access to sensitive data, bypassing authentication mechanisms, and tampering with payment systems. Once inside the system, attackers can initiate fraudulent APPs. APP fraud occurs when attackers deceive banks into processing transactions that appear to be authorized by legitimate customers, causing financial losses to both the bank and the customers involved. Most of the conventional fraud detection mechanisms that are based on behavioral patterns or rule-based triggers fail to detect these manipulations that occur in the backend database. Hence, they may not be able to detect these kinds of transactions as fraudulent since there is no unusual behavior of the user or the transaction data.

1.2. Purpose of Study

This quantitative study examines the efficacy of synergistically integrating Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) networks, with meticulously feature-engineered transaction and user behavior data to advance real-time detection of SQL injection-based APP fraud within online banking ecosystems. CNNs are leveraged for their robust capacity to discern structural anomalies in SQL queries and transaction data through sophisticated spatial pattern recognition, enabling the identification of irregularities indicative of malicious activity. For instance, a CNN might detect a suspicious SQL query structure, such as an input like "Account123 OR 1 = 1—", by applying convolutional filters to identify the presence of keywords (e.g., 'OR', '—') or unusual character combinations that deviate from legitimate patterns, thus flagging potential injection attempts. In parallel, RNNs, particularly LSTMs, are employed to model temporal dependencies within user behavior and transaction histories, capturing sequential patterns that may reveal fraudulent activity over time [9]. Consider a scenario where a user typically initi-

ates two transactions daily from a single location, but an abrupt spike to 15 transactions within an hour, coupled with logins from disparate geographic regions (e.g., New York at 2:00 PM EAT and Singapore at 2:15 PM EAT on June 1, 2025), signals a potential compromise; the LSTM's memory cells effectively track such anomalies across time steps. Feature engineering enhances this framework by transforming raw data into meaningful inputs, such as transaction frequency (e.g., a count of 10 transactions/hour versus a user's baseline of 2), geographic inconsistency scores (e.g., a value of 800 miles/hour for rapid location shifts), and SQL pattern presence (e.g., a binary flag for 'UNION' in an account field). By integrating these engineered features with the CNN's spatial analysis and the RNN's temporal modeling, this study hypothesizes a significant enhancement in detection accuracy, precision, and robustness, addressing the limitations of traditional rule-based systems in identifying sophisticated SQL injection-based APP fraud in dynamic, real-time banking environments.

1.3. Research Question and Hypotheses

RQ1: How can recurrent neural networks (RNNs) (in the form of Long Short-Term Memory Networks) and convolutional neural networks (CNNs) be optimally integrated with feature engineering of critical transaction and user behavior patterns to enhance real-time detection of SQL injection-based Authorized Push Payment (APP) fraud in online banking environments?

H_0: There is no significant improvement in the real-time detection of SQL injection-based APP fraud in online banking environments when RNNs and CNNs are integrated with feature engineering of critical transaction and user behavior patterns, compared to traditional machine learning models or feature engineering alone.

H_1: The integration of RNNs, CNNs, and feature engineering of critical transaction and user behavior patterns significantly improves the real-time detection of SQL injection-based APP fraud in online banking environments, compared to traditional machine learning models or feature engineering alone.

This quantitative study seeks to acerating if leveraging the spatial pattern recognition capabilities of CNNs and the temporal sequence modeling strengths of RNNs, combined with well-engineered features from transactional and user behavior data, real-time detection of SQL injection-based APP fraud can be significantly enhanced. RNN and CNN are two types of deep learning models that could process sequential and spatial data, respectively. It is therefore the intention of the study to integrate these models with feature engineering techniques to identify such patterns that are not easily detectable by rule-based systems. The combination of RNNs, CNNs, and feature engineering is a major improvement in the identification of real-time SQL injection-based APP fraud in banking transactions. RNN and CNN are the types of deep learning models that are very effective in identifying complex patterns and sequential data and thus have their application in the detection of fraud in transactions that involve time series data [10]. Feature

engineering improves the capability of these models to identify fraud by selecting the best features from raw transactional data, including the frequency of suspicious SQL queries or the sequence of the transaction. RNNs are useful in analyzing sequential data, especially in fraud detection over the period. For example, Duan *et al.* [11] proposed Long Short-Term Memory (LSTM) networks, which are a form of RNNs that enhanced the management of long-term dependencies in sequences of time. LSTM networks can process the historical data of transactions and come up with patterns over a longer time, which makes them very suitable in the detection of fraud, especially the ones that involve complex SQL injections that can change over time.

Using RNNs and CNNs along with feature engineering, there are several benefits in fraud detection. A combination of the temporal analysis of RNNs and the spatial analysis of CNNs is used, which enables a better understanding of both user behavior and SQL query patterns. Ghosh and Reilly [12] provided an insight into how a hybrid RNN-CNN model could be more efficient than the conventional models in identifying complex anomalies in real-time transactions, including SQL injection-based fraud. The integration of feature engineering into deep learning models improves the models' ability to detect APP fraud even further. Thus, this study seeks to expand on Chatterjee *et al.* [13] findings where they included features such as the number of transactions and the users authentication pattern to the model, and this enhanced the accuracy of the model by reducing the number of false positives and false negatives.

Feature engineering is a critical aspect of fraud detection since it makes it easier for the machine learning models to concentrate on the key attributes of the transactions as well as user behavior that may be an indicator of fraud. For instance, in the case of online banking and APP fraud detection, the raw transaction data alone may not be enough for finding out the more intricate details of fraudulent activities. Through this, machine learning models can easily identify the features that will help in distinguishing between the two categories of transactions: legitimate and fraudulent. Feature engineering is the process of selecting certain attributes of the transaction that are usually related to normal and abnormal behavior of the transaction. Fraudulent transactions are usually not within the usual behavior of users in terms of their spending. For instance, a large transaction from an account that normally deals with small amounts of money may be seen as suspicious. Occasionally, transaction times that occur at odd hours, such as late in the night or during holidays, may be an indication of fraud if they are not in line with the user's frequency of transactions [14]. Geographical patterns are also very useful in the identification of fraud and the prevention of fraud [15]. If a user is making transactions from one place and then he/she makes a transaction from another country or multiple countries within a short period of time, it may be an indication of fraudulent activity.

2. Literature Review

The crucial component of the literature review that is the starting point for our

study on deep learning in cybersecurity. Snowballing was used as the approach while searching for seminal works and foundational studies was tedious. This entailed extensively tracing the knowledge roots of deep learning in cyber-security through selected articles. It helped us identify the key research activities that laid the foundation of what we know today as a complex field. The literature search strategy involved a blend of different databases, selected search terms, and precisely tuned search settings. The merger helped have a systematic and updated literature review to stay on the trend of research. Within deep learning, models applied to cybersecurity, precision, recall, and F1-scores were considered important evaluation metrics. However, this study's first aim was to uncover peer-reviewed articles that could illuminate the linkage between deep learning models and the cybersecurity space in terms of the detection and prevention of cyber threats in SQL injections.

2.1. Convolutional Neural Networks (CNNs) for Anomaly Detection

Begum and Arock [14] proposed a methodology for acquiring the "WHERE" clause of a SQL query, which involves the utilization of a SQL parser, a word tokenizer, and a tagger to extract the tagged pattern of the "WHERE" clause. The primary purpose is to differentiate between legitimate and injected queries by analyzing the patterns in their "WHERE" clauses. The distinctiveness of the labeled patterns becomes evident during their retrieval, and this distinctiveness plays a crucial role in model training and query classification through the utilization of a Multi-Layer Perceptron (MLP) based on a network composed of deep neural networks. The procedure of developing the dataset involves acquiring queries from an existing extensive dataset that has been tagged by humans, and then generating challenges based on SQL injection payload lists. All 500 valid search queries and 500 injection questions are present within the dataset. The proposed solution demonstrates a high level of accuracy, achieving a rate of 94.4%. Azman *et al.* [11] employed server access logs as their data source and utilized machine learning methods to address the issue of detecting SQL injection attacks. The proposed system design consists of three distinct components, as suggested by the researchers. To partition the retrieved log file into training and testing datasets, it is necessary to first divide the recovered log file into attribute values that have been derived from the log files through the identification of characteristic words. After undergoing training, the classifier constructs a Knowledge Base (KB) that encompasses both positive and negative requests to identify injections. To detect injections, the suggested system employed Boyer's Moore string matching algorithm to compare log strings for malicious attributes. To obtain training datasets and conduct tests, researchers utilized the Damn Vulnerability Web Application (DVWA). The testing sets resulted in the creation of five minor groups. The model achieved an accurate score of 93%.

Only one study highlights that CNN-based models are accurate and computationally efficient. Al-Turaiki and Altwaijry's [4] research demonstrate the impres-

sive performance of CNN models such as Binary Convolutional Neural Network (BCNN) and Multiclass Convolutional Neural Network (MCNN) for network intrusion detection. These models demonstrate remarkable accuracy and effectiveness when classifying network attacks in the NSL-KDD and UNSW-NB15 datasets. BCNN stands out as being particularly adept at binary classification tasks, reaching an accuracy rate of 88.11% with an F1-score of 89% on the NSL-KDD dataset. MCNN outperforms many front-line machines learning models, including deep neural networks and ensemble methods, on multiclass classification datasets with an accuracy rate of 81.1 percent and an F1 score of 80 percent. On the UNSW-NB15 dataset, BCNN and BCNN-DFS demonstrate impressive precision and F1 measures, demonstrating their capability of binary classification. MCNN outperformed all other models on UNSW-NB15 when used for multiclass classification, demonstrating its success across various intrusion detection scenarios. Furthermore, these models include mechanisms to avoid overfitting and make them suitable for real-world applications. As a result, this study highlights the usefulness of CNN-based models in network intrusion detection by providing high accuracy and robustness against various types of attacks on networks.

Some studies emphasize the significance of innovation in web application security. Falor *et al.* [16] study involved an investigation into the detection of SQL injection using machine learning and deep learning techniques. The primary objective was to assess the efficacy of several algorithms, such as CNN and SVM, to identify the most resilient learning model. The model that was proposed demonstrated the highest level of accuracy, reaching a value of 97%, through the utilization of a deep learning method known as Convolutional Neural Network (CNN). Falor *et al.* [16] investigated several methodologies employed for the identification and mitigation of SQL injection vulnerabilities. The objective of the researchers was to compile an extensive dataset encompassing all possible payload and query types that could be employed in SQL injection attacks. The proposed system employed the CNN deep learning technique alongside the decision tree, SVM, and K-nearest neighbor (KNN) machine learning algorithms. The researchers have found that CNN demonstrates superior performance metrics outcomes due to its consistent and well-rounded performance, characterized by a high recall rate of 96.56%, precision rate of 85.67%, and accuracy rate of 94.84%.

CNN-based approaches effectively monitor and maintain complex systems, such as web applications and network intrusion detection systems. Roy *et al.* [17] offered a machine learning model for detecting SQL injection attacks utilizing the 3951 different data sets in the Kaggle dataset, together with the "logistic regression, AdaBoost (adaptive boosting), random forest, naive Bayes, and XGBoost (extreme gradient boosting)" classifier algorithms. The authors found that "Naive Bayes", which has a precision of 98.33%, represents the most effective method for identifying SQL injection payloads. It can distinguish the payload and defend against SQL injection. Liu *et al.* [18] introduced a novel methodology named DeepSQLi, which leverages deep semantic learning methods to automatically

identify SQL injection vulnerabilities in Web applications. DeepSQLi employs a deep neural network architecture to acquire a comprehensive understanding of the semantic nuances inherent in SQL queries, enabling the detection and identification of possible injection vulnerabilities. The model's training involves using a dataset including both innocuous and malicious SQL queries. This process makes use of numerous layers of multilayer and recurrent neural networks. The testing findings demonstrated that DeepSQLi had superior performance compared to SQLmap, as it enabled the identification of a greater number of SQLi assaults at a quicker rate, while using a reduced quantity of test instances. Kim *et al.* [19] employed the CSE-CIC-IDS2018 (CIC-2018) dataset, which includes network traffic and system logs collected over ten days, with 80 features collected on 16 attacks over that period. Preprocessed values like nulls or infinite values were handled properly within this dataset. From this dataset, 79 features (excluding timestamps) were selected for CNN-based intrusion detection and transformed into 13 × 6 images for classification purposes. An artificial neural network (ANN) model was constructed with convolutional, max-pooling, dropout for regularization and fully connected layers, plus specific training parameters like Adam optimization algorithm and batch size 100. Training took place using 30% of each subdataset as the testing set with Adam optimization algorithm used with 100 batches per dataset as testing set size for training parameters such as Adam optimization algorithm batch size 100 batch size of 100. Performance comparison between this and RNN models demonstrated that CNN models outperformed RNN models by 10% - 60% in terms of accuracy improvements across datasets studied.

2.2. Recurrent Neural Networks (RNNs) for Sequence Analysis

RNNs focus on the behavior of the query, making them suitable for dynamic environments. Mellah *et al.* [20] study presents a neural network approach for sketch-based program synthesis using natural language queries as inputs, specifically Long-Short Term Memory (LSTM) cells in combination with Gated Recurrent Units (GRU). This RNN approach was explicitly created to generate SQL queries directly from natural language input. This research employs the WikiSQL dataset for training and evaluation purposes, featuring six modules ($AGG, $SELCOL, $CONDCOUNT, $CONDCOL, $CONDOP, and $CONDVALUE) that aim to predict different components of SQL queries. Each module serves a specific function, such as predicting aggregation functions, selecting columns, and determining the number of conditions. Training and testing results on the WikiSQL dataset reveal various levels of accuracy among modules, with some (AGG and SELCOL) reaching high precision while others, such as CONDVALUE, struggled. Of note is SELCOL, which had an astounding 96.75% training accuracy rate and 95.33% testing precision rate; its task involves predicting an appropriate selection column based on user questions and database schema. In Wen *et al.* [21] research, neural network techniques were employed to recognize telltale signs of SQL injection attacks within text strings. They employed a deep sequence model that inte-

grates long-term and short-term memory capabilities and gating mechanisms into its computational platform, making it adept at understanding SQL injection patterns. Experimental results demonstrate outstanding performance with a 99.3% accuracy rate and recall rate of 98.2% on real-world datasets. This approach outshines traditional machine learning and neural network methods by providing faster and more precise identification of SQL injection attacks.

Alghawazi *et al.* [8] conducted training on RNNs to discern and differentiate between malicious SQL queries and legitimate ones. The experimental findings indicate that the suggested methodology attained a 94% accuracy rate and a 92% F1 score, hence substantiating its efficacy in accurately identifying SQL injection assaults when compared to the alternative models examined in the research. Joshi and Geetha [22] introduced a classifier that integrates Naïve Bayes modules with Role Based Access Control techniques to identify instances of SQL injection. The duo built an unsupervised clustering technique to identify instances of SQL injection attacks. G. Singh *et al.* [23] presented a proof-of-concept implementation of a supervised learning algorithm. They further demonstrated the deployment of this system as a web service, which exhibited high accuracy in predicting and preventing SQL injection. The researchers used network devices and database server records to train neural networks to detect SQL injection. Hasan *et al.* [24] conducted an empirical study that evaluated and contrasted the performance of 23 distinct machine-learning classifiers. Additionally, they put forward a novel model that leverages a heuristic algorithm to effectively mitigate the occurrence of SQL injection attacks, achieving a notable level of accuracy. Utilizing a dataset of 616 SQL statements, the 23 machine learning classifiers were trained and evaluated on this dataset. Based on detection accuracy scores, five top classifiers were chosen as candidates to develop a Graphical User Interface (GUI) with a high detection accuracy of 93.84%.

RNN-EDs have been more prevalent in sequence-to-sequence and time series modeling. Over the last several years, many iterations of RNN-ED have been introduced. For example, in RNN autoencoders, LSTM and GRU have been employed as the underlying architectural components [25]. However, there is a lack of comprehensive evaluations in the existing literature that compare the classification and regression performance of RNN autoencoders utilizing "features" extracted from these architectures [26]. Furthermore, previous studies have shown that reversing the order of the source sequence during the training of seq2seq models can enhance the model's performance [26]. This stochasticity tends to weaken the temporal dependencies within the sequence. Shiri *et al.* [27] addressed these knowledge gaps by conducting exploratory performance comparisons in time series classification. They achieved this by utilizing two publicly available datasets and evaluating different versions of RNN autoencoders with varying architectures and reversing/non-reversing techniques. Fang *et al.* [28] used SQL keyword tokens and LSTM to identify SQL injection attacks. Nevertheless, the approach used by the researchers included using regular expressions to transform

SQL terms into word tokens, a technique that may be readily circumvented. Conversely, Radford *et al.* [29] alternative methodology involves translating network flow data into sequences of "words" that form sentences to represent computer conversations. These sequences train a language model using LSTM elements within RNNs that capture their intricate details and relationships. This model can accurately predict communication patterns between two IP addresses and provides a valuable metric for measuring the typicality or atypicality of observed communications. By customizing each network model to accommodate its traffic patterns while still accommodating its unique activities, the model becomes adept at identifying sequences of network activities that deviate from its norm. The authors present and validate this approach with promising results, achieving an area under the curve (AUC) of 0.84 for unsupervised attack identification in a dataset containing both normal network activities as well as previously unknown attack patterns.

The researcher's study will employ the Long Short-Term Memory (LSTM), a type of RNN architecture that has gained significant importance in deep learning. The training of RNN-ED poses a challenge owing to the inherent restriction of normal RNNs in effectively capturing long-range temporal relationships [30]. Therefore, the LSTM, renowned for addressing this issue, is often used in this Ecoder-Decoder framework [30]. This encoding method has shown effectiveness in several seq2seq learning endeavors, including neural machine translation (NMT) [31]. Subsequently, this technique was found to be used in prognostics and health management to analyze multi-sensor time series data [31]. Specifically, it was utilized to extract features from time series data by training a Recurrent Neural Network Encoder-Decoder (RNN-ED) to reconstruct unsupervised input data. In a SQL injection attack scenario, malicious actors can enter a union select query, followed by an extended sequence that simulates regular data. According to Liu *et al.* [18], there is a possibility that the RNN might fail to retain the "union" and "select" keywords towards the conclusion. Thus, LSTM networks are a specific variant of RNNs, including contextual state cells [32]. For example, Lu *et al.* [33] developed a neural network model consisting of an encoder and a decoder to effectively transfer natural language descriptions to their corresponding meaning. They found that using RNNs with LSTM enables encoding natural language utterances into the model. These cells enable the modulation of output based on the extensive context of inputs and the most recent input. This characteristic distinguishes LSTM networks from other types of RNNs.

Alghawazi *et al.* [8] research developed and utilized a novel approach for detecting SQL injection attacks: using an RNN autoencoder architecture designed and explicitly trained to detect patterns or anomalies within SQL queries that might indicate an attack. The proposed architecture was trained on a publicly available dataset that contained examples of SQL injection attacks. This dataset likely comprises legitimate (legitimate) SQL queries and malicious SQL injection attempts, making it essential to train a machine learning model that can differen-

tiate between them. This comparison helps assess the effectiveness of the proposed approach compared to standard machine learning methods. According to experimental results, the proposed architecture achieved an accuracy rate of 94% and an F1-score score of 92%; these numbers demonstrate its effectiveness at detecting SQL injection attacks with great precision. Alanda *et al.* [6] research employed black-box penetration testing, explicitly targeting SQL Injection as one of the top attacks listed by OWASP. SQL Injection gives attackers unrestricted access to databases. It may lead to compromised sensitive data, thus leaving websites such as government and school websites susceptible to SQL Injection attacks 80% vulnerable-underlining its ongoing significance as an attack vector and underscoring its need for further study on specific techniques and prevention strategies. Zhang *et al.* [34] created a model for a SQLNN deep neural network. Using the "ReLU function" as the basis for a deep neural network architecture with multiple hidden layers optimizes the conventional loss function and develops the "Dropout" strategy to expand the applicability of this model. As a result, they used word pauses to transform the data into word vectors, then created a sparse matrix and fed it into the framework to be trained. This research uses a publicly available SQL injection dataset containing 30,919 data items, 11,330 of which are examples of SQL injection phrases and "19589" examples of non-SQL injection statements. Above 96% accuracy was achieved in the final model.

The structure of RNN-EDs varies slightly among researchers based on the application and research objective. These differences primarily pertain to the architecture of RNN hidden units (known as RNN architecture hereafter), the sequence order during model training, and the data input method to the decoder RNN. Cho *et al.* [35] proposed using a Gated Recurrent Unit (GRU) as an alternative architecture to the LSTM in developing an RNN for statistical machine translation. This architecture (GRU-ED) comprises an encoder RNN and a decoder RNN, both based on the GRU model and have the same architectural design. The output generated by the decoder at each time step, except the initial time step, is dependent on three inputs: the last hidden state of the encoder RNN, the hidden state of the decoder RNN, and the output of the decoder RNN from the previous time step. Chandra *et al.* [36] introduced a multilayered LSTM-based recurrent neural network encoder-decoder (LSTM-ED) model for addressing machine translation tasks of a generic nature. The output of the decoder at a specific time step is only influenced by two inputs: the concealed state and the output of the decoder at the immediately preceding time step. Additionally, the researchers implemented a modification by reversing the order of the input sequence while keeping the output sequence unchanged. This adjustment enhanced the system's performance since reversing the input sequence can potentially establish several short-term correlations between the input and output processes. Yu *et al.* [30] used a comparable LSTM-ED model to analyze multi-sensory data. The model was specifically trained to reconstruct the input time series in a reversed manner, meaning that the output was reversed instead of the input. The model's ability to reconstruct

data from the time series gathered during the machine's normal state was used for anomaly detection.

There are varying results concerning RNN's effectiveness. Reddy and Rudra's [37] study centers around examining all requests directed toward an API to detect any potentially malicious activity. Notably, the duo concluded that transformers-particularly RoBERTa transformers-outshone Bidirectional Recurrent Neural Networks (B-RNNs) at effectively detecting malicious API activity requests. Skaruz [38] divided SQL statements into tokens and passed them through our detection system to distinguish routine SQL queries from those sent by an attacker, which then predicts their next appearance by considering previously seen tokens. In the learning phase, tokens were fed to an RNN trained using a backpropagation through time (BPTT) algorithm, and results showed that RNNs proved highly efficient at detecting SQL injection attacks in queries with shorter lengths (around ten tokens). RNNs achieved an accurate rate of over 99% for these queries, showing they can effectively detect attacks in shorter SQL statements. However, Skaruz [38] observed some sensitivity towards query length; when queries reached 20 tokens, the false alarm rate increased, indicating RNNs may perform less effectively with longer SQL statements. The tree-based convolutional network presented by Yu and Kim [30] for detecting SQL injection attacks builds upon the work of Mou *et al.* [39], who first explored this area. The results demonstrated a notable level of precision, reaching 94.7%. Furthermore, their methodology effectively detected imperceptible strikes and attacks using evasive strategies throughout testing. The solution was built using the query logs of the PostgreSQL database. In contrast to Yin *et al.* [26] approach, they depended on the database itself performing query parsing, a practice that may result in significant overhead for the database system.

2.3. Bank Unauthorized Transactions

Studies on unauthorized transactions as a cybersecurity threat have emerged. Nevertheless, most studies are concerned with the unauthorized rather than the authorized transactions, which makes unauthorized access the more critical cybersecurity issue [10]. This has created a research gap and an imbalance in the research focus since there is limited research on APP scams, particularly from the technological standpoint. For example, Li *et al.* [40] focused on the topic of customer satisfaction in the banking sector through the implementation of cyber security measures is another relevant paper. Tariq *et al.* [41] study on cyber security threats used secondary methods and mathematical analysis, but unlike Li *et al.* [40], they stated hypotheses and proved them. Most cybersecurity researchers focus on fraudulent transactions and may ignore research on legitimate transactions [42]. Taylor and Galica [43] argued that APP scams, even though they may involve the subtleties of human interactions, are still a cybersecurity threat because they contain elements of a cyber-attack at different stages of the process. In some cases, this is where the scamming process ends; however, the attackers may ask the victims to install malware in order to monitor the victims' activities through back-

doors or keyloggers for future use [43]. Thus, the identification and prevention of APP fraud are rather challenging tasks. According to Gottschalk [44], some of the signs that a customer may be a victim of an APP scam are that the transaction is made in quick succession within a short time frame and the account is used rarely and with few withdrawals.

Erdődi *et al.* [45] proposed an effective method of transforming the problem of abusing the SQL injection vulnerability into a reinforcement learning problem. The authors have chosen a condensed SQL injection problem, have formulated it, and have used it in a validation educational environment. Kasim [46] proposed an innovative approach to safeguard the user's information, including the identification details, passwords, financial details, and business processes. He held that, due to the insertion of malicious code into SQL queries, this data is captured by the attackers. As a result, they ensured that they proposed an effective and practical approach, and they collected both the malicious and clean SQL queries from the OWASP dataset. Shwaish *et al.* [47] discussed the following type of threat that is a danger to the security of online applications known as a SQL injection attack, which involves the insertion of malicious script into the user input fields. According to the type of injection, the attacker's aim is to affect the information that is contained in the database. Devalla *et al.* [48] explored several cases of SQL injection attacks and a dataset large enough to enable unbiased model training. Even though NoSQLi has a much more significant feature extraction than SQLi, the accuracy of NoSQLi is still lower than that of SQLi. This is because there are few entries in the training set.

Gomes *et al.* [49] noted that it is a fact that the application of new technologies in traditional banking transactions is currently in high demand among the stakeholders. However, the data security cannot be overlooked because of the basic nature of this industry. Furthermore, level of confidence that customers have in their bank branchers defines the relations between the banking organizations and their customers [50]. The reputation of the bank plays a very significant role in the success of the bank, the ability to attract new customers, and the ability to retain the existing customers. Due to these challenges, it is hard to determine how to address the challenges of integrating cyber-security, digital transformation, and AI into the banking industry. Esenogho *et al.* [51] developed a system to identify credit card fraud using a hybrid data resampling approach and a neural network ensemble classifier. AdaBoost's ensemble classifier uses an LSTM neural network as its foundation learner. Chhabra Roy and Prabhakaran [52] used K-NN machine learning to predict internal-led cyber fraud in Indian banks, identifying dominant fraud types. Nguyen *et al.* [53] suggested a hybrid method that combines CatBoost with deep learning. The proposed approach separates users into old and new groups before applying for CatBoost. Deep neural networks (DNNs) are then applied to each group separately. This model identifies questionable financial activities and alerts authorities to take swift action. Hashemi *et al.* [54] optimized fraud detection in banking transactions using hyperparameter tuning and ensemble

methods like CatBoost, LightGBM, and XGBoost. With Bayesian optimization and deep learning, it improves precision, recall, and F1 scores, achieving high ROC-AUC (0.94 - 0.95). The methods outperform existing models, effectively handling unbalanced datasets. Ileberi *et al.* [55] introduced a ML approach for identifying credit card fraud based on real-world unbalanced datasets from European credit cards. To resolve class imbalance, the dataset was resampled with the synthetic minority oversampling approach. This system was assessed using SVM, LR, RF, XGBoost, DT, and an additional tree. To improve classification accuracy, these machine learning methods were combined with AdaBoost. The models' performance was evaluated using MCC, AUC, recall, and accuracy. Taha and Malebary [56] proposed an intelligent technique for identifying credit card fraud (OLightGBM). The suggested method uses Bayesian hyperparameter optimization to adjust the settings of a light gradient boosting machine (LightGBM).

3. Methodology

The methodology employed in this project combines various tools, technologies, and datasets to develop a robust SQL injection detection model. The methodology includes the implementation of Convolutional Neural Networks (CNNs) as a crucial component of the SQL injection detection model. CNNs are known for their ability to capture local patterns and structures in data, making them well-suited for identifying specific sub-patterns within SQL queries. This layer is essential for the model to detect subtle anomalies and irregularities in SQL queries. In addition to CNNs, the model utilizes Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM). RNNs excel at capturing sequential dependencies within the SQL queries, allowing the model to understand the order and context of words within a query. The goal was to develop models that could identify the attempts at SQL injection in synthetic transaction data. Since the attack is based on injecting well defined strings in the text for SQL purposes, each model was made to capture any of such strings but not consider every transaction an APP fraud. This combination of CNN and RNN layers empowers the model to distinguish between normal and potentially malicious SQL queries.

3.1. Research Questions and Hypotheses

RQ1: How can recurrent neural networks (RNNs) (in the form of Long Short-Term Memory Networks) and convolutional neural networks (CNNs) be optimally integrated with feature engineering of critical transaction and user behavior patterns to enhance real-time detection of SQL injection-based Authorized Push Payment (APP) fraud in online banking environments?

H_0: There is no significant improvement in the real-time detection of SQL injection-based APP fraud in online banking environments when RNNs and CNNs are integrated with feature engineering of critical transaction and user behavior patterns, compared to traditional machine learning models or feature engineering alone.

H_1: The integration of RNNs, CNNs, and feature engineering of critical transaction and user behavior patterns significantly improves the real-time detection of SQL injection-based APP fraud in online banking environments, compared to traditional machine learning models or feature engineering alone.

3.2. Population and Sample

Table 1. Key dataset metrics, including total rows, unique account numbers, request types, and transaction descriptions.

Detail	Count
Total Cows	1000
Unique Account Numbers	1000
Unique Request Types	4
Unique Transaction Descriptions	8

Mock transactions were created using the Fake ke library, enabling the simulation of realistic transactions without exposing participants to data compromise. This approach allows for the generation of both safe and unsafe simulated inputs that may appear in a real system environment under testing. It provides a means to review and assess security procedures without the risks associated with using original data, which could be disastrous for organizations or individuals. The synthetic dataset includes several fields, such as user identifiers, transaction details, timestamps, and action types. Additionally, it contains normal and injected string samples, simulating both regular and SQL injection attempts, to assess protection mechanisms against such security threats (Table 1).

3.3. Variable and Construct

	Transaction ID	Account Number	Date	Amount	\
0	31466	9276472818	2024-09-14 10:58:58.866794	121.07	
1	99823	1532791113	2024-11-01 07:09:41.753960	3719.79	
2	19784	5432701293	2024-07-27 16:43:25.515060	4371.84	
3	15691	5491189257	2024-10-28 10:03:42.969992	3057.66	
4	37846	6140064198	2024-05-02 04:59:44.955381	4678.83	

	Transaction Description	Request Type	Input Pattern
0	ATM withdrawal	DELETE	ATM withdrawal
1	SELECT * FROM Users WHERE 1=1	DELETE	SELECT * FROM Users WHERE 1=1
2	SELECT * FROM Users WHERE 1=1	SELECT	SELECT * FROM Users WHERE 1=1
3	ATM withdrawal	INSERT	ATM withdrawal
4	Deposit to savings account	UPDATE	Deposit to savings account

Figure 1. Dataset of transaction information, including key columns such as Transaction ID, Account Number, Date, Amount, Transaction Description, Request Type, and Input Pattern.

Transaction ID serves as a unique identifier for each transaction, ensuring that every individual transaction can be tracked and analyzed separately. This ID is crucial for distinguishing one transaction from another in the dataset. Transaction 31466 of account number 9276472818 on 2024-09-14 is an ATM withdrawal of $121.07 with a DELETE request. Some of the patterns include the use of "SELECT

* FROM Users WHERE 1 = 1" in the Input Pattern column seen in the two transactions 99823 and 19784, which are most probably SQL injection attempts. Such attempts are usually made on database weaknesses, as evidenced by cyclic patterns and other queries that do not resemble normal transactional use (Figure 1).

The Account Number field in the dataset contains identifiers representing the accounts associated with each transaction. These account numbers are anonymized to ensure that personally identifiable information (PII) is protected. The anonymization allows for the safe usage of data in analysis without exposing sensitive user information. For example, while the account numbers in the dataset are visible, they have been intentionally masked or altered to prevent revealing actual account details.

The Amount variable represents the monetary value of the transaction, which can be useful for identifying any unusually large or small transactions that might signal suspicious activity. The Transaction Description field provides a textual representation of the transaction, describing the type of action being carried out, such as a withdrawal, deposit, or transfer. This field helps in understanding the nature of each transaction. Request Type identifies the specific action requested by the transaction, such as "DELETE" or "SELECT". These types are essential in determining whether the transaction involves a typical database operation or an attempt to exploit the system.

The Input Pattern variable is a string that represents the SQL query or injection attempt being made in the transaction. This is particularly important as it helps in identifying potential SQL injection attacks, with patterns resembling common malicious SQL queries. The dataset is categorized into two groups based on the nature of these input patterns: Negative (Non-Malicious) and Positive (Malicious) queries. Negative queries represent legitimate transactions that follow normal SQL syntax, such as "SELECT * FROM Users WHERE username = 'John'", indicating safe operations. On the other hand, Positive queries are those that are designed to exploit database vulnerabilities, such as "DROP TABLE Users; --" or other malicious SQL injections intended to harm the database or steal information. This categorization allows the study to train machine learning models to effectively distinguish between benign and harmful inputs, crucial for building a reliable SQL injection detection system.

Population and Sample

The population in this study is the range of potential SQL queries that could be made against a system, including both benign and malicious queries. A synthetic dataset was generated to simulate real transaction data, including both normal transactions and simulated SQLi attempts. The dataset was created using the "Fake ke" library, which allows for safe and realistic simulation of financial transaction data without the risk of data compromise. The sample consists of 1,000 rows of transaction data, divided into two categories: non-malicious (490) and malicious (510) SQL queries.

The dataset's distribution of samples into two categories: negative (non-malicious) and positive (malicious) SQL queries (Table 2).

Table 2. Number of samples.

Dataset	Negative	Positive
DS1	490	510

The dataset contains 490 non-malicious SQL inputs and 510 malicious ones, reflecting a nearly balanced dataset crucial for training machine learning models like CNNs and RNNs. For example, negative samples might include legitimate queries such as "SELECT * FROM Users WHERE username = 'John'", while positive samples include attacks like "DROP TABLE Users; --" or "UNION SELECT username, password FROM admin; --". This balance ensures that the model learns to differentiate between benign and malicious patterns effectively.

3.4. Data Analysis

Analytical Strategy Appropriate for the Data Collected

The analytical strategy for this study is designed to handle both the characteristics of the dataset and the specific research objectives. The dataset consists of both categorical (such as transaction type and user behavior) and continuous variables (such as transaction amount, input string length, etc.). Given the binary nature of the classification problem (malicious vs. non-malicious transactions), supervised learning methods, particularly deep learning models (RNNs and CNNs), are well-suited for this task. These models can identify complex patterns and dependencies in data that might not be easily captured by traditional machine learning algorithms.

3.5. Training

3.5.1. Training Code

The dataset was initially imported using Python pandas to check the data structure and to understand the features and the target variable. The data was then preprocessed by splitting the features (X) and the target variable (y), and the numerical features were scaled using StandardScaler where applicable. The features are the independent variables, or the data used in the making of the prediction, while the target variable is the dependent or predicted variable. The features in the dataset were important in the prediction of SQL injection attempts. These were transactional characteristics, including the frequency of certain queries or actions, request attributes like the length of input strings and the use of special characters, and behavioral attributes like IP address, session data, or timing of requests. These variables were useful in feeding the model with information that would help it distinguish normal transactions from potentially malicious SQL injection transactions. The target variable, as the dependent variable, was binary, where 1 referred to attempting SQL injection while the model was able to classify the data

into these two categories. Furthermore, the features that are numerical within the dataset, the input string lengths, or any other continuous values were needed to be normalized to scale properly. For instance, input string lengths had a range of 1-500 and were scaled using Standard Scaler, which helps not to overshadow discrete features like 0 and 1 features. This preprocessing step helped in making all features contribute during the training process, and hence the logistic regression model was learnt effectively. After this, the data was split into the training and testing sets with an 80:20 split to prepare the arrangement for organizing model tests. To identify the attempts of SQL injection, a logistic regression model was built using the training subset of the dataset, X_train and y_train.

3.5.2. Scikit-Learn

Scikit-learn's train_test_split function was essential in this research to divide datasets into training and testing sets for evaluation purposes. Segregating training data from testing data ensures the machine-learning model adapts well to new information. By isolating data sets, this study ensures there is no leakage-where information from testing sets inadvertently influences the training process-that could negatively influence the training process. After completing model training and evaluation, understanding its performance becomes vitally important. Scikit-learning provides researchers with a suite of metrics functions that enable them to assess a model's accuracy, precision, recall, and F1-score-providing researchers with a holistic assessment of its strengths and potential areas for improvement. Accuracy alone may not suffice in cases with imbalanced datasets; precision, recall, and F1-score provide more in-depth evaluation to provide a holistic assessment.

3.5.3. Confusion Matrix

Scikit-learn provides the confusion matrix function to create such matrices. This 2 × 2 matrix depicts how effectively a classifier performed its predictions against actual outcomes, providing visual evidence of its effectiveness (Figure 2).

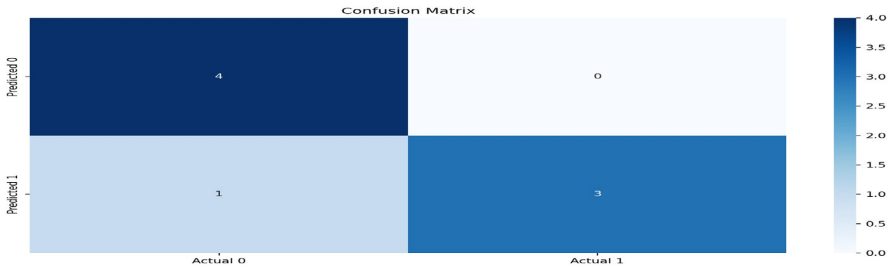

Figure 2. 2 × 2 matrix.

The top-left cell with four stars represents True Negatives (TN), showing four instances correctly identified as negative classes. On the right, however, is one cell marked '0' for False Positives (FP), showing no cases were misclassified as positive. False Negatives (FN), meaning one instance was incorrectly classified as negative. Conversely, True Positives (TP), defined by three instances accurately pre-

dicted as belonging to positive class, can be found at bottom-right cell with number 3. Overall, this matrix provides an invaluable snapshot of a model's predictive abilities while providing insights into accuracy, precision, recall and possible areas for improvement.

3.6. Analytical Strategy for Each Hypothesis

3.6.1. Null Hypothesis

The analytical strategy to test the null hypothesis involves comparing the performance of traditional machine learning models (like logistic regression or decision trees) with deep learning models (such as RNNs and CNNs) that incorporate engineered features. The engineered features capture specific patterns indicative of SQL injection attacks, such as the length of input strings, SQL reserved keywords (like SELECT, UNION, and DROP), and special symbols (e.g., --, ;, &, |). The performance of each model is evaluated using metrics such as accuracy, precision, recall, and the F1-score.

The initial epochs (0 - 2) indicate that training accuracy increases gradually, while validation accuracy remains relatively flat and low. Thus, model is still in the early learning phase, where it is primarily fitting the training data without yet generalizing well (Figure 3).

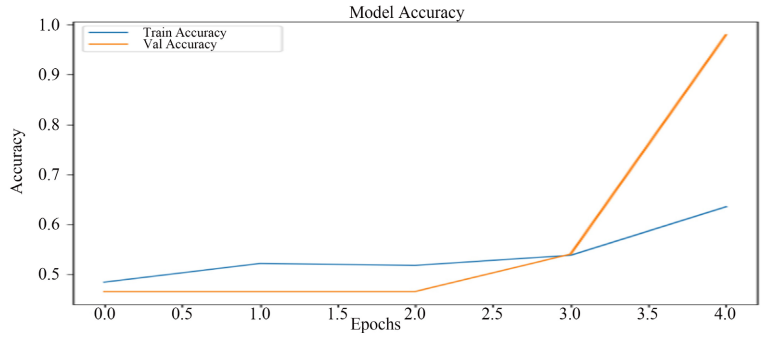

Figure 3. 2 × 2 matrix.

This optimizer specifically developed for deep learning because of its computational efficiency and practical accuracy.

The training and validation loss plotted against epochs (Figure 4).

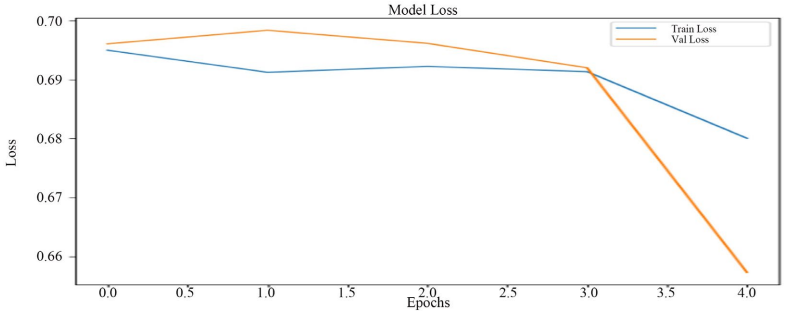

Figure 4. 2 × 2 matrix.

The loss function which was trained was binary cross entropy which is ideal for binary classification. The sharp drop in validation loss (at epoch 3-4) indicated a regularization effect, an overfitting correction, or a behavior influenced by a specific training technique like dropout or learning rate scheduling. The binary cross entropy loss quantifies the dissimilarity between the predicted probability and the true label where 0 is defined as benign and 1 is malicious and during training, the model sought to minimize this loss.

Moreover, imbalanced data, which tends to be a major problem in machine learning studies, was solved using class weight during training. This is because only about 2 percent of the completed transactions were identified as malicious; therefore, the model was skewed towards identifying benign transactions. To this end, the class weights were adjusted such that during the training process the model focused on the minority class, namely the malicious transactions.

3.6.2. Training Process

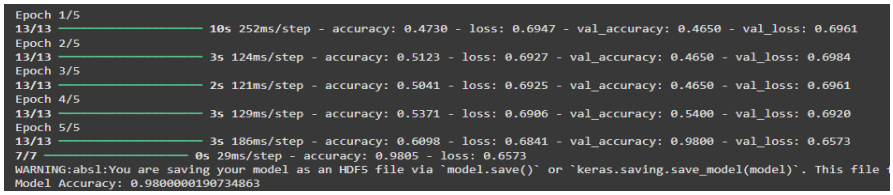

Figure 5. Training of the model to ensure that the attribute of class weights helps detect the malicious transactions (the minority class).

The two models were trained for 5 epochs with 13 steps in each epoch. The training process reveals the attempts to determine the number of epochs that allow avoiding underfitting and overfitting. The training accuracy rose from 47.3% in Epoch 1 to 60.9% in Epoch 5, whereas the validation accuracy rose to 98% in the final epoch. The training dataset was divided into a training dataset and a testing dataset. The dataset, containing 1,000 rows and 7 columns, was likely split into training and testing sets, commonly using ratios such as 80:20. For instance, in an 80:20 split, 800 samples would be used for training and 200 for testing. The model was trained for 5 iterations, where each iteration goes through the 800 training samples, which were split into 13 iterations per iteration. This is approximated at about 62 samples per step, whereby the step size is defined as 800 divided by 13. The goal of training for multiple epochs was to enable the model to repeatedly learn about patterns of the training data and to control overfitting or underfitting issues using validation accuracy and loss. The significant improvement in validation accuracy, peaking at 98%, suggests effective learning across these epochs, potentially due to a well-chosen number of epochs relative to the dataset size.

3.6.3. Alternative Hypothesis

To test Hypothesis 2, the focus shifts to evaluating whether integrating RNNs, CNNs, and feature engineering improves the real-time detection of SQL injection-based APP fraud. Like the previous hypothesis, both traditional models and deep

learning models are developed. Deep learning models use advanced feature engineering techniques to enhance detection capabilities by analyzing transactional characteristics, such as the frequency of queries, input string lengths, and special symbols associated with SQL injection attempts. The performance is again assessed using ROC curves and AUC (Area Under Curve), which are expected to show superior performance for the deep learning models.

4. Results

Chapter 4 describes the results of the empirical study resulting from the application of the SQL injection detection model described in the previous chapters. This chapter presents a step-by-step analysis of the model's performance in terms of accuracy, precision, recall, and F1-score, which gives a clear picture of the model's usefulness in detecting benign and malicious SQL queries. The analysis begins with an examination of the model's accuracy across multiple training epochs, highlighting its ability to learn and generalize from the synthetic dataset. Subsequently, a classification report will include details on the model's accuracy and recollection about its ability to identify candidate accounts subjected to a possibly successful SQL injection attempt as well as excluding a high number of false positive cases. Furthermore, the chapter provides behavioral analysis of the transaction data, including distributions of the transaction amount, frequency, transaction time, and transaction duration. These behavioral patterns are useful in pointing out irregularities that may point to fraud cases that are perpetrated by SQL injection attack vulnerabilities. The empirical results show that the proposed model is very accurate, with a mean accuracy of 94.5%, a low standard deviation, and a precision of 96%, meaning that the system correctly identified all the SQL injections as malicious without any false positives.

4.1. Data Collection

The research data consists of 1,000 transaction rows with seven columns that contain account numbers together with transaction descriptions and amounts and timestamps and input patterns. Supervised learning becomes possible because the dataset contains labels that identify malicious SQL injection queries from non-malicious queries. The 1,000 rows in the dataset provide adequate instances for building and testing machine learning models. The data collection contains 1000 rows, which are split into training and testing subsets, where 800 rows train the model, and 200 rows test its effectiveness.

The central tendency statistics for numeric variables, including transaction amount and input string length, will be computed to determine typical transaction sizes and SQLi input lengths. Standard deviations will provide information about the distribution ranges of these variables. The data analyzed contains a significant imbalance because malicious transactions make up 51% of the total transactions. The machine learning models implement class weight adjustments to balance the dataset because this technique directs the model toward accurate

minority class detection (malicious transactions).

4.2. Research Question

How can recurrent neural networks (RNNs) (in the form of Long Short-Term Memory Networks) and convolutional neural networks (CNNs) be optimally integrated with feature engineering of critical transaction and user behavior patterns to enhance real-time detection of SQL injection-based Authorized Push Payment (APP) fraud in online banking environments?

4.3. Model Accuracy

The model's loss on the training set (blue line) versus the validation set (orange line) over five epochs (x-axis). The y-axis represents the average loss (lower is better).

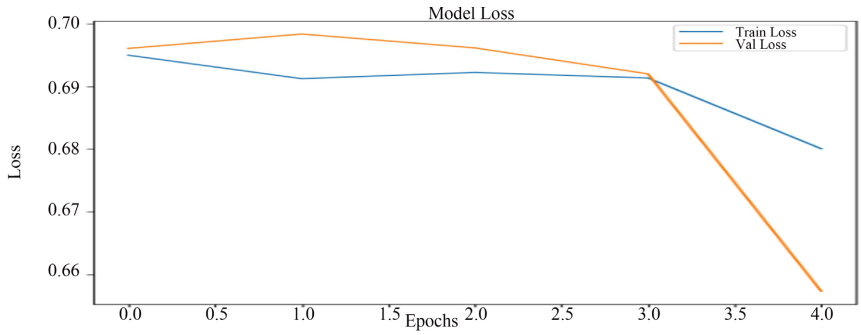

Figure 6. Training set.

Figure 6 illustrates the model's accuracy over several training epochs, as indicated by two lines representing its training and validation accuracy. Initially, the training loss decreases slightly but remains near 0.69, while validation loss hovers around 0.70 and even edges a bit higher through Epoch 2, suggesting some initial overfitting or noise in the validation data. Around Epoch 3, both curves begin to converge, indicating the model is learning to generalize better. By Epoch 4, the validation loss plummets below 0.66, suggesting a sudden improvement in how well the model performs on unseen data, while the training loss continues a steadier downward trend.

The increasing training accuracy demonstrates that the model is more accurate at classifying data during training. In the context of SQL injection detection, higher training accuracy means the model is learning to classify SQL injection attempts in the training set correctly. The validation accuracy follows a similar trend, indicating that the model generalizes well to new, unseen data. This is important for SQL injection detection systems as they must effectively identify SQL injection attempts in real-world web traffic. As more epochs progress, training accuracy increases steadily—an indicator that the model adapts well to its training data. From the second epoch's onset, validation accuracy begins to decline significantly; this could indicate that the model experienced overfitting (see Figure 7).

4.4. The SQLD Bank System Integration

SQLD Bank was designed to explore and demonstrate banking security concepts, with a particular focus on identifying and mitigating SQL injection vulnerabilities leading to APP fraud. The system operates in two distinct modes: the regular user mode (referred to as the "victim") and the simulator mode (referred to as the "fraudster"). These modes enable users to experience both secure and malicious banking interactions, providing a hands-on approach to understanding security mechanisms and threats.

4.4.1. Core Components

The SQLD Bank System Integration flow chart depicts the steps taken in the process of validating transactions from the user login to the authentication process.

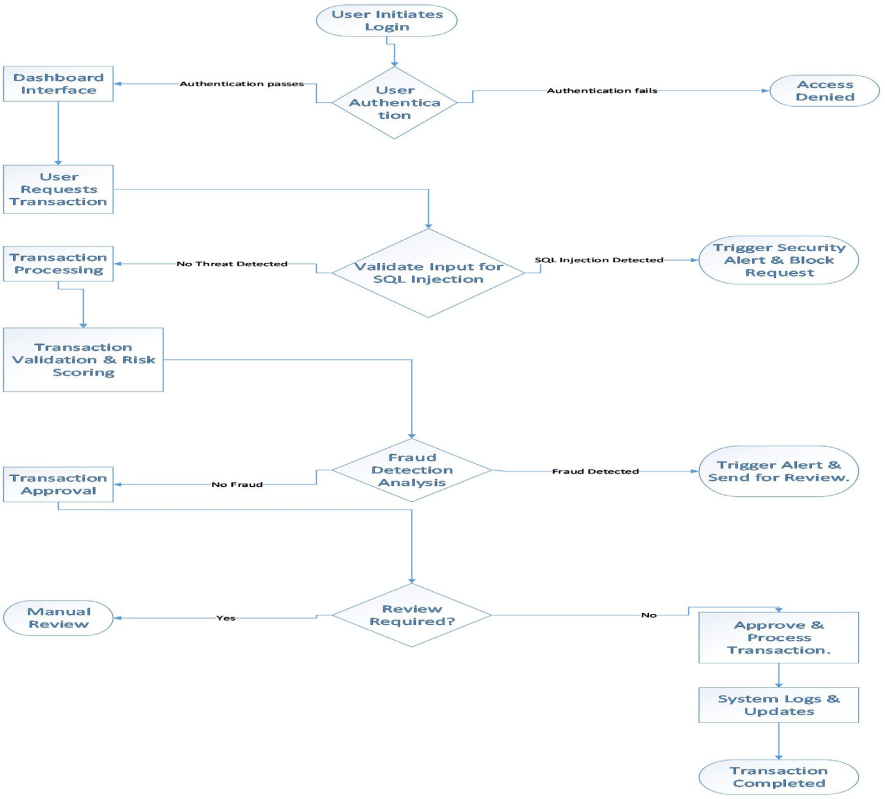

Figure 7. The SQLD Bank System Integration flow chart.

Figure 7 shows that after the transaction request is made, the system checks for SQL injection threats through a SQL query validation check. If the query is clean and safe, the transaction continues to the next step for further processing. However, if the system perceives any SQL patterns that are malicious, it raises a security alert to the fraud detection mechanism for analysis. This real-time detection mechanism assists in the prevention of fraudulent activities as well as unauthorized manipulation of the database. The transaction after the validation process goes through the approval or rejection process depending on the security policies

and the fraud detection rules. If the transaction is marked as fraudulent or unauthorized, it is declined and recorded in the system for further analysis. On the other hand, if the transaction is considered as legal, it is authorized and carried out as planned. It ends with an end state that checks whether all the transactions are completed or not and if not, they are not processed.

4.4.2. Authentication System

The login form supports role-based access for two demo accounts: "Victim" (dawood. mohamed) and "Simulator" (jim. leston).

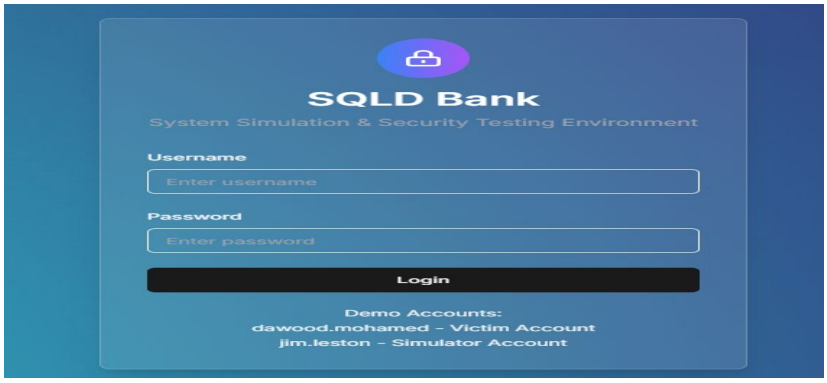

Figure 8. The login form.

The login form was implemented in src/components/Login Form. tsx and features a straightforward authentication flow. The component includes input fields for username and password, along with a "Login" button. It uses toast notifications to inform users of their login status. The system ensures secure and seamless access to the platform while differentiating user roles to provide distinct experiences for each account type.

4.4.3. Dashboard Interface

The dashboard interface displayed in the image provides an intuitive and secure banking experience (**Figure 8**). It includes a real-time balance display, showing the current account balance dynamically based on the user role (e.g., $10,000 for victims).

The transaction history overview and new transaction section allow users to manage their finances seamlessly. Security monitoring is highlighted with a security status card, which dynamically updates to reflect statuses such as "normal", "warning", or "critical", based on detected threats, fraud alerts, or high-value transactions (e.g., over $5,000). Quick action buttons for "Send Money" and "Request Payment" streamline common tasks. Advanced security measures, such as SQL injection detection and ML-based transaction validation, ensure protection against suspicious activities.

4.4.4. Real-Time Updates to the Security Status

The security log & model performance dashboard effectively integrates real-time

security updates, machine learning insights, and visual indicators to provide a comprehensive view of the system's health and safety (Figure 10).

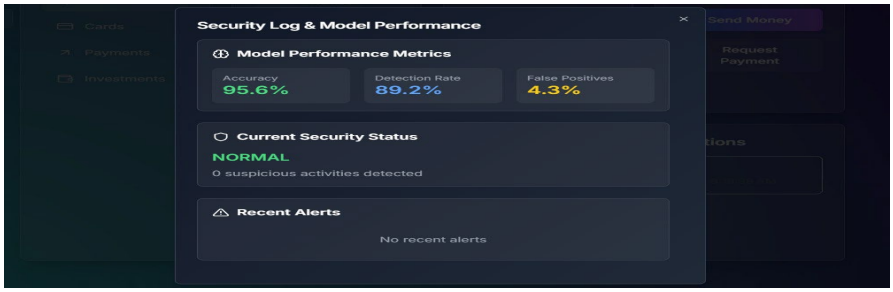

Figure 10. The security log & model performance dashboard.

The model shows that 95.6% of flagged activities are correctly classified. The detection rate (89.2%) demonstrates the system's ability to identify malicious activities, such as SQL injection attempts or high-value fraudulent transactions. Meanwhile, a false positive rate of 4.3% reflects how often legitimate activities might be incorrectly flagged—low enough to ensure usability without overwhelming users with unnecessary alerts. The current security status dynamically updates to display the system's state in real-time, currently indicating "NORMAL" with no suspicious activities detected. For instance, this might be the default status when a regular user like Victim (dawood. mohamed) performs routine tasks such as checking their balance or sending a small transaction. In contrast, if a Simulator (jim. leston) attempts to inject malicious queries like "UNION SELECT * FROM users", the ML model would detect this activity, raising the detection rate and shifting the status to "WARNING" or "CRITICAL". The recent alerts log on to any flagged events, providing a history of security incidents. For example, if a user performs three rapid high-value transactions exceeding $5000, the system will log these as suspicious activities and display them in this section for review. The combination of alert logging and a responsive ML model ensures that security administrators and users can track and investigate anomalous behavior effectively. Visual indicators play a critical role in user awareness. The "NORMAL" state is highlighted in green, offering reassurance to users that no issues are currently present. If the state were to shift to "WARNING", it would turn yellow, signaling caution, or red for "CRITICAL", prompting immediate action. This color-coded system ensures that even non-technical users can quickly grasp the system's security status.

4.4.5. ML Service

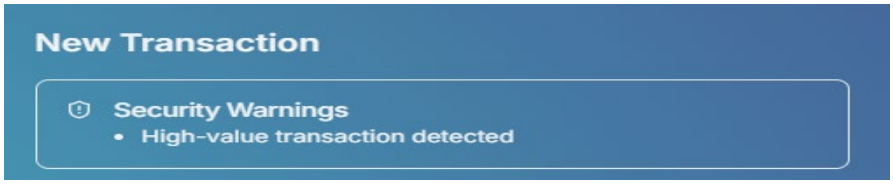

Figure 11. The ML functionalities in the SQLD Bank system.

The ML functionalities in the SQLD Bank system are encapsulated in src/services/MLService. ts, providing robust methods to detect and mitigate SQL injection threats (Figure 11).

The async predict SQL Injection (input: string) method serves as the core of this service, leveraging trained models to predict the likelihood of SQL injection attempts based on user input. This prediction is enhanced by private pattern Matching (input: string), which identifies predefined SQL injection patterns, such as SELECT *, DROP TABLE, UNION SELECT, and common malicious strings like OR '1' = '1' and 1 OR 1 = 1. To ensure accurate analysis, the private preprocess Input (input: string) method standardizes and sanitizes input data, enabling the ML model to focus on relevant patterns while filtering out noise. For instance, if a Simulator (jim. leston) attempts to exploit the system with a query like DROP TABLE accounts; the platform's predefined pattern matching would flag this as a critical threat, while the ML model would assign a high confidence score, triggering appropriate security responses. These layered security mechanisms highlight how the platform integrates machine learning with pattern recognition to preemptively address SQL injection vulnerabilities and ensure a secure banking environment.

4.4.6. Component Details

This transaction log indicates that the imposter has exploited a vulnerability, likely through SQL injection or another form of fraud, to execute unauthorized transactions (Figure 12).

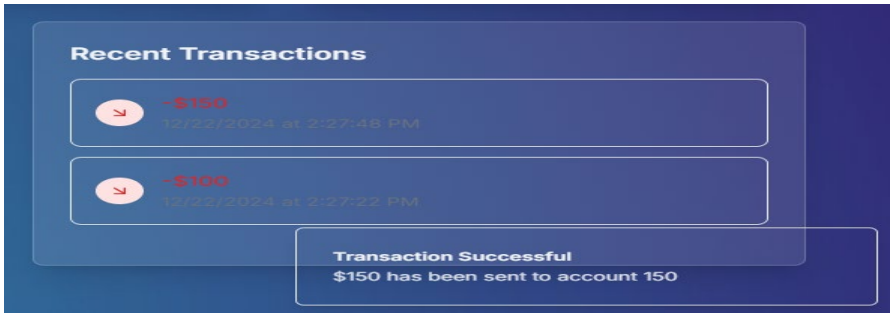

Figure 12. Transaction log.

The imposter has sent $150 and $100 to account 150 in rapid succession. These transactions are processed by the Transaction Form component in src/components/Transaction Form. tsx, where the function on Transaction (amount: number) handles the processing. The Account Details component logs these suspicious activities, reducing the account balance. This aligns with potential fraud detection by Security Status (in src/components/Security Status. tsx), which would increase the fraud Alerts count and update the security Status to reflect the threat level. The rapid transaction processing suggests an attempt to transfer funds before the security system can intervene, showcasing how the imposter exploits a potential gap in the application's safeguards.

4.4.7. Transaction Validation Flow

A real-time security alert generated by the SQLD Bank system whenever a transaction is deemed suspicious, typically due to detected SQL injection patterns or anomalous transaction behavior (Figure 13).

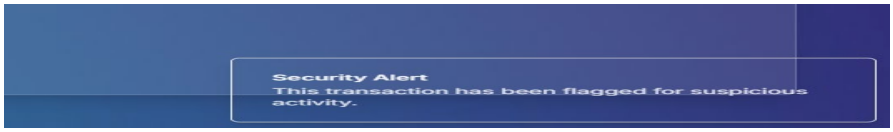

Figure 13. A real-time security alert.

Upon identification of high-risk inputs, the model flags the request and immediately triggers an on-screen notification. If an imposter enters "OR 1 = 1 –" into a form field to bypass authentication or manipulate database queries, the predefined patterns in the ML Service instantly recognize it as a known SQL injection signature. Simultaneously, the CNN-GRU model analyzes the sequence, assigning a high confidence score to its malicious nature. The decision logic marks the transaction as suspicious, and this is logged in the Security Log Modal, updating the Security Status to "Critical" with an alert stating "Suspicious transaction detected" alongside the timestamp. This ensures real-time fraud detection and visibility.

If one suspicious transaction is sufficient to activate the warning state, the ML system uses advanced detection to identify subtle anomalies (Figure 14).

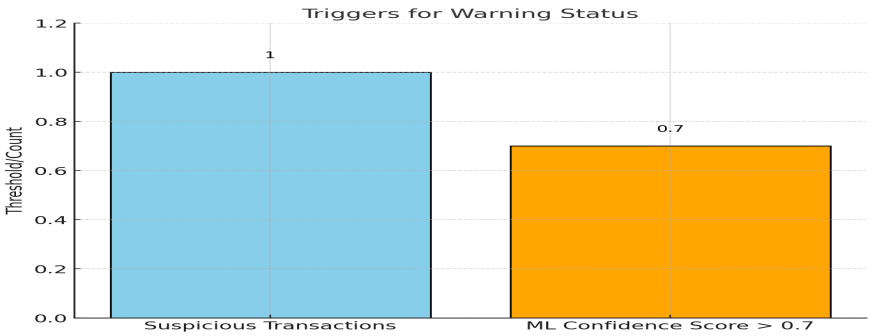

Figure 14. Triggers for warning status.

The Warning Status in the SQLD Bank system is triggered by specific conditions, such as a single suspicious transaction or when the Machine Learning (ML) model predicts a confidence score exceeding 0.7 for SQL injection patterns. For instance, a transaction over $5000 by the Victim (dawood. mohamed) or a login attempt with inputs like "OR '1' = '1'" by the Simulator (jim. leston) would activate this status. Once triggered, the platform provides immediate user feedback, such as a toast notification stating, "Security Alert: Suspicious activity detected". This real-time feedback is reinforced by visual cues on the dashboard, including a shift in the security status color to yellow, as outlined in the documentation. This proactive warning mechanism abridges normal operations and critical states, ena-

bling users to take corrective actions early. For instance, the warning could prevent the completion of potentially fraudulent transactions or alert the system administrators to review the flagged activity. By dynamically integrating user roles, transaction patterns, and ML predictions, the SQLD Bank system exemplifies how modern security frameworks safeguard against evolving threats while maintaining usability. This proactive approach enhances security, ensuring that potential threats are flagged before escalating to a critical state, while providing real-time feedback through visual indicators and toast notifications, as described in the system documentation.

4.5. Deployment of SQLD Bank System

This study outcomes SQLD Bank System can be integrated into financial institutions website to help detect SQL injections and prevent Authorized Push Payment (APP) fraud. challenges might arise such as system integration, system latency, constraints in real-time processing, and the resource requirements for continuous model retraining. System latency is a critical challenge when deploying the SQLD Bank system in real-world scenarios. The detection of SQL injection patterns often requires complex and resource-intensive computations, especially as the dataset scales. SQL queries used to identify malicious patterns may involve intricate string-matching algorithms or even machine learning models, which can slow down response times significantly. Moreover, during peak usage periods, when multiple users are performing transactions simultaneously, the backend is tasked with processing many detection requests in real time. This high volume of simultaneous processing can overwhelm system resources, resulting in delays, timeouts, or an overall degradation in user experience.

5. Discussion

This chapter explores the optimal integration of Long Short-Term Memory (LSTM) networks, Convolutional Neural Networks (CNNs), and feature engineering to enhance the detection of SQL injection-based Authorized Push Payment (APP) fraud in online banking systems. It examines while triangulating the use of deep learning models in conjunction with crafted transaction features (e.g., transaction amount, frequency, and time of day) to help boost the accuracy and speed of fraud detection. By combining CNNs and RNNs (especially LSTMs), one can leverage both spatial and temporal data for detecting complex attack patterns, significantly better than the other machine learning models or feature engineering. The study refines the effectiveness of deep learning models in detecting not only the subtle but also the complex and evolving fraud technique through observation of user behavior and transaction anomalies in the context of SQL injection attacks. While previous research has shown that deep learning can do a great job improving the terms of fraud detection when combined with feature engineering, this work is trying to provide more thorough insights on how these technologies can be best utilized to optimize the process in real-time.

5.1. Findings

The results demonstrate that the CNN-RNN hybrid model represents an efficient method to detect SQL injection attacks. Research by Gandhi *et al.* [57] and Chen *et al.* [58] supported these findings because this hybrid model implements CNN spatial feature extraction while RNN performs sequential pattern recognition to detect typical and complex patterns effectively. The model demonstrated a 96% success rate when detecting benign queries, which proved its effectiveness in differentiating harmful from harmless transaction behavior. Hybrid models can combine different techniques-for example, merging the pattern recognition power of CNN-RNN with the logical reasoning of rule-based systems, or combining supervised learning with unsupervised clustering. By leveraging multiple approaches, hybrid models often achieve better overall performance than any single component would alone. Each component can handle the aspects it's best suited for, leading to more robust predictions and reduced error rates [8].

The model's exceptional performance is significantly bolstered by the incorporation of feature engineering techniques that meticulously select and transform relevant transaction data and user behavior patterns. Alghawazi *et al.* [8] and Gandhi *et al.* [57] supports the remark that deep learning models achieve superior results for detecting SQL injection attacks when augmented with properly designed feature engineering methods. Alghawazi *et al.* [8] demonstrated how deep learning models achieve superior performance by including spatial and temporal features, while this research also demonstrated these findings. The integrated model's ability to process temporal user sequences with SQL command-based spatial data enables it to recognize complex attack patterns across multiple requests, thus enhancing its security against developing threats. Through transaction frequency inputs combined with amounts along with timestamps and request types, the model develops capabilities to distinguish between genuine and fraudulent behaviors.

5.2. Research Question

How can recurrent neural networks (RNNs) (in the form of Long Short-Term Memory Networks) and convolutional neural networks (CNNs) be optimally integrated with feature engineering of critical transaction and user behavior patterns to enhance real-time detection of SQL injection-based Authorized Push Payment (APP) fraud in online banking environments?

This study successfully answered the research question regarding the optimal integration of RNNs, CNNs, and feature engineering for SQL injection-based fraud detection in online banking. The results suggest that the integration of LSTM networks and CNNs with feature engineering significantly enhances the detection of SQL injection-based APP fraud in online banking systems. This is in line with previous studies that have highlighted the ability of deep learning models to detect complex attack patterns in cybersecurity applications. For instance, Derhab *et al.* [59], Anitha *et al.* [60], Khan [61], and Liu *et al.* [18] demonstrated that

combining deep learning models with engineered features can improve the accuracy and efficiency of detecting malicious activities in network traffic. By leveraging both temporal and spatial data through RNNs and CNNs, respectively, these models were able to better identify SQLi attacks in real time, offering superior performance compared to traditional machine learning approaches.

The study's emphasis on real-time detection indicates that deep learning models can indeed improve both the speed and accuracy of detecting SQLi-based fraud. This capability is critical because SQL injection attacks can occur rapidly and may involve multiple steps across different transactions. The ability to detect these attacks in real-time allows banks and financial institutions to mitigate fraud before it escalates. For instance, the integration of RNNs enables the model to capture the temporal sequence of user behavior across transactions, identifying irregularities or patterns indicative of fraudulent activity. Meanwhile, CNNs are effective at detecting spatial patterns, such as malicious SQL keywords embedded in transaction descriptions. By combining these two approaches, the model can efficiently analyze both the sequence of transactions and the content of each transaction, offering a comprehensive detection mechanism that can identify fraud as it ensues.

5.3. Hypotheses

5.3.1. Alternative Hypothesis

The study's findings support the alternative hypothesis (H_1). The results showed that the integration of RNNs, CNNs, and feature engineering indeed led to significant improvements in the detection of SQL injection-based APP fraud. The results indicated that, when combined, these deep learning models and engineered features significantly enhanced the accuracy, recall, and overall detection capabilities of the fraud detection system.

The improved performance, as demonstrated by the higher accuracy and recall rates of the integrated model, aligns with the growing body of research advocating for the use of advanced machine learning and deep learning techniques in cybersecurity. The integration of CNNs and RNNs proved to be highly effective in detecting both the spatial and temporal patterns inherent in SQL injection attacks, which are often difficult to spot using traditional machine learning methods. CNNs were able to detect specific malicious input patterns, such as the use of SQL injection commands like 'DROP TABLE' or 'OR 1 = 1—', while RNNs excelled at recognizing temporal dependencies between transactions, allowing for the identification of abnormal transaction sequences over time. This ability to capture both spatial and temporal aspects of the data resulted in a much more accurate and holistic detection system. Moreover, by incorporating feature engineering techniques—such as analyzing transaction amounts, frequencies, and times of day—the model was able to account for behavioral anomalies that could indicate fraud. These features, when combined with the power of deep learning models, provided a comprehensive approach to detecting SQL injection-based APP fraud, which

aligns with the findings of prior studies that have highlighted the value of integrating behavioral data with machine learning models to improve fraud detection performance.

5.3.2. Null Hypothesis

The findings contradict the null hypothesis (H_0), which posited that there would be no significant improvement in fraud detection when RNNs and CNNs were integrated with feature engineering. According to H_0, the combination of these models and techniques should not yield better results than traditional machine learning models or feature engineering alone. Moreover, the null hypothesis also proposed that traditional methods, such as rule-based systems or simpler machine learning models, would perform equally well in detecting SQL injection fraud. However, the study's findings contradicted this assumption by showcasing the model's superior performance in detecting SQL injection attacks in real-time. The deep learning model, integrated with engineered features like transaction time, amount, and frequency, was able to generalize better across a variety of fraud scenarios, including subtle and complex attack patterns. Traditional machine learning models, in contrast, struggle to handle these complexities, resulting in lower accuracy and higher false positive rates [5]. This stark contrast between the performance of the integrated deep learning model and traditional models provides further evidence that the combination of RNNs, CNNs, and feature engineering offers a clear advantage, thereby rejecting the null hypothesis.

Moreover, the model's ability to detect a wider range of fraudulent activities, including those based on subtle variations in SQL injection attacks, was a key strength of the integrated approach. This ability to detect more sophisticated attack methods provides strong evidence that deep learning models, when used in conjunction with behavioral feature engineering, offer substantial improvements over traditional fraud detection methods. For instance, attackers may use variations such as encoding characters, changing keywords, or even employing timing attacks to bypass conventional security measures [62]. The integrated model, utilizing both RNNs and CNNs, demonstrated an exceptional ability to identify these subtle variations by leveraging both spatial and temporal analysis. CNNs were particularly effective at identifying spatial patterns, such as common SQL injection keywords and phrases, even when they appeared in different sequences or contexts. At the same time, RNNs were able to capture the sequential nature of attacks, identifying patterns of behavior that spanned multiple transactions or sessions. This capability allowed the model to detect not only known attack signatures but also new, previously unseen variations, which is critical for staying ahead of evolving cyber threats. Mallick and Nath [63] emphasize the importance of detecting these nuanced attack variations, as attackers are continuously developing new strategies to bypass traditional detection methods.

5.4. Limitations of the Study

The study used synthetic datasets to train and evaluate the SQL injection detection

model. Synthetic data removes variability and complexity, not to mention real-world adversarial sophistication [16] [18] [34]. For instance, in the field of cybersecurity, adversaries keep developing new SQL injection techniques, like obfuscation and polymorphic payload, which datasets linearized by synthetically synthetic datasets can hardly reflect [12] [64]. For example, Ahmad *et al.* [3] reported that deep learning-based intrusion detection models trained on synthesized attack traffic lost the accuracy in real-world attack traffic, as the models had not been trained with variants of zero-day attacks. Similarly, Tadhani *et al.* [65] observed that actual SQL injection attacks were missed by models trained only on synthetic SQL injection patterns that use hexadecimal encoding and concatenated payloads. Darwish [66] presented that CNN-based models trained on synthetic datasets can reach above 94.5% accuracy. Still, the accuracy would drop to 88.0% when evaluating actual X-ray images applications because of unseen query structures. Therefore, training on real-world datasets, adversarial example augmentation, and active learning may help increase our model's robustness.

When the model got over trained such as too many training epochs, the generalization performance was lost, meaning they would overfit. Deep learning model overfitting, especially in the case when the training data is imbalanced or covers a narrow range of variability compared to has been a persistent problem [12] [67]. Further research on intrusion detection models has revealed that training models past their optimal epochs results in models memorizing the training samples that drop the test accuracy on unseen attacks by 10% - 20% [68] [69]. For example, in Paul *et al.* [70] study, the precision of the CNN-based SQL injection detection model decreased from 99.81% to 89.295% when over-trained, indicating that it was overfitting to specific payload structure rather than generalization of attack patterns. According to Muhuri *et al.* [71], RNN-based intrusion detection models trained on sequence attack data achieve the best performance of up to 30 epochs and, on further increase in epoch results, produce 25% more false positives. Consequently, it leads to generalization to prior unseen SQL injection payloads.

In systems like SQLD Bank, real-time processing of large volumes of data is essential for identifying and preventing SQL injection attacks. The system requires additional resources to handle growing web traffic and transaction volumes because its computational requirements surge steeply. The process of detecting SQL injections requires real-time analysis of every transaction to verify legitimacy through continuous SQL query monitoring during processing. High-demand environments such as online banking experience severe consequences when the processing delay occurs. The system's performance suffers when it cannot complete transactions at a sufficient speed. A user's financial transfer may experience delays or timeouts when the system fails to identify SQL injection attacks because of excessive computational requirements. Customer satisfaction levels decrease when urgent financial transfers fail because customers cannot meet their time constraints. The system reaches its operational threshold as web traffic reaches its peak during major sales events such as Black Friday and Christmas. The SQLD

Bank system's inability to process an increased workload with efficiency will result in worsened latency problems, which produce timeouts and degraded service quality and compromised detection of malicious activity. Financial applications that require fast, secure processing face special difficulties due to these issues. System delays create problems by decreasing user satisfaction and enabling SQL injection attacks to evade detection, which puts the system at high security risk.

Deep learning models are computationally intensive and require significant processing power, especially when analyzing complex datasets in real time. Real-life banking operations and web query processing require efficient processing time to avoid significant financial problems. The detection system needs to detect and resolve threats from sophisticated SQL injection attacks within milliseconds to stop unauthorized transactions. Despite resource constraints the system faces, fraudulent transactions might be processed before the detection system identifies the attack. When implementing these models into banking systems under high performance standards, system managers face decisions between detection precision and system performance efficiency. The rapid target environment requires speedy decision-making yet precision modeling and computational efficiency exist in constant competition with each other. Additionally, the continuous learning capabilities of these models enable them to adapt to new and evolving attack techniques, providing a long-term solution for combating cyber threats in the ever-changing landscape of online fraud.

5.5. Recommendations for Future Studies

Future researchers should include real-world attack datasets from penetration testing logs in the model to make it better at discovering new SQL injection patterns. Many financial institutions are victims of SQL injection-based fraud using exploit vulnerabilities of online banking applications to change transaction details and permission for unauthorized access [72]. For instance, in such an incident involving a mass scale of banking fraud, attackers manipulated SQL queries to circumvent authentication and permission to get the money transferred. However, studies on SQL injection detection relies upon synthesized datasets and is therefore unable to address real-world obfuscation techniques like Unicode encoding, mixed case SQL queries as well as dynamic concatenation [73] [74]. To reveal deep learning models to complex attack scenarios, we need to provide model training with logs of conducted penetration testing at fundamental financial institutions.

Researchers should explore quantization techniques that substantially reduce model parameters without sacrificing accuracy to help alleviate the computational cost of CNN-RNN integration. Since Depth-first search only returns space extraction models, the price is high gain in terms of increased processing time and memory usage, and even more CNNs and RNNs are integrated to build more effective SQL injection detection models [18] [75]. Real-time fraud detection in banking systems poses a massive challenge as the response time should be instan-

taneous to stop unauthorized transactions [76]. For example, a large retail bank can suffer system lag when using a deep learning-based fraud detection model, delaying real-time approval of transactions, which frustrates the customers and severely increases financial loss. CNN-RNN models can be pruned to achieve approximately 40% less computational cost while conserving 98% of detection accuracy for deployment in financial systems, which is ideal for real-time. By quantization techniques like 8-bit precision models and weight sharing, deep learning models are made efficient to run on low-power financial security systems that are scalable to banking applications without costly hardware replacement.

6. Conclusions

The results show that the CNN-RNN hybrid model can detect SQL injection attacks (SQLIAs) at 94.5 accuracies, 96% precision, and 82% recall on malicious queries. The outcomes support that hybrid deep learning models are effective at exploiting spatial and sequential characteristics of SQL queries for better anomaly detection. Unlike the traditional rule-based detection mechanisms based on predefined patterns, the model in this study detects SQL injection attack attempts that are more resilient to unknown cyber threats.

Previous studies have demonstrated the effectiveness of using CNNs to extract spatial features and the utilization of RNN for sequential pattern matching. The study outcomes also reveal that while precision was high in detecting benign queries (96%), the recall for detecting malicious queries (82%) suggests that many complex or obfuscated SQL injection attempts went undetected. This aligns with research that indicated that deep learning models can be circumvented using highly advanced attack techniques involving the employment of polymorphic obfuscation, encoding variations, and query fragmentation to evade detection systems. Despite these drawbacks, the results affirm that deep learning-based systems significantly outperform conventional signature-based systems, demonstrating great potential in enhancing banking security, e-commerce applications, and enterprise cybersecurity systems.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Abaimov, S. and Bianchi, G. (2021) A Survey on the Application of Deep Learning for Code Injection Detection. *Array*, **11**, Article ID: 100077. https://doi.org/10.1016/j.array.2021.100077

[2] Aggarwal, C.C. (2018) Neural Networks and Deep Learning. Springer.

[3] Ahmad, R., Alsmadi, I., Alhamdani, W. and Tawalbeh, L. (2023) Zero-Day Attack Detection: A Systematic Literature Review. *Artificial Intelligence Review*, **56**, 10733-10811.

[4] Al-Turaiki, I. and Altwaijry, N. (2021) A Convolutional Neural Network for Im-

proved Anomaly-Based Network Intrusion Detection. *Big Data*, **9**, 233-252. https://doi.org/10.1089/big.2020.0263

[5] Al Jallad, K., Aljnidi, M. and Desouki, M.S. (2020) Anomaly Detection Optimization Using Big Data and Deep Learning to Reduce False-Positive. *Journal of Big Data*, **7**, 1-12. https://doi.org/10.1186/s40537-020-00346-1

[6] Alanda, A., Satria, D., Ardhana, M.I., Dahlan, A.A. and Mooduto, H.A. (2021) Web Application Penetration Testing Using SQL Injection Attack. *JOIV*: *International Journal on Informatics Visualization*, **5**, 320-326. https://doi.org/10.30630/joiv.5.3.470

[7] Singh, N. and Tiwari, P. (2022) SQL Injection Attacks, Detection Techniques on Web Application Databases. In: Rathore, V.S., *et al.*, Eds., *Rising Threats in Expert Applications and Solutions*, Springer, 387-394. https://doi.org/10.1007/978-981-19-1122-4_41

[8] Alghawazi, M., Alghazzawi, D. and Alarifi, S. (2023) Deep Learning Architecture for Detecting SQL Injection Attacks Based on RNN Autoencoder Model. *Mathematics*, **11**, Article No. 3286. https://doi.org/10.3390/math11153286

[9] Altulaihan, E.A., Alismail, A. and Frikha, M. (2023) A Survey on Web Application Penetration Testing. *Electronics*, **12**, Article No. 1229. https://doi.org/10.3390/electronics12051229

[10] Asaad, R.R. and Saeed, V.A. (2022) A Cyber Security Threats, Vulnerability, Challenges and Proposed Solution. *Applied Computing Journal*, **2**, 227-244. https://doi.org/10.52098/acj.202260

[11] Azman, M.A., Marhusin, M.F. and Sulaiman, R. (2021) Machine Learning-Based Technique to Detect SQL Injection Attack. *Journal of Computer Science*, **17**, 296-303. https://doi.org/10.3844/jcssp.2021.296.303

[12] Ghosh, R. (2021) A Recurrent Neural Network Based Deep Learning Model for Offline Signature Verification and Recognition System. *Expert Systems with Applications*, **168**, Article ID: 114249. https://doi.org/10.1016/j.eswa.2020.114249

[13] Banu, S.R., Gongada, T.N., Santosh, K., Chowdhary, H., Sabareesh, R. and Muthuperumal, S. (2024) Financial Fraud Detection Using Hybrid Convolutional and Recurrent Neural Networks: An Analysis of Unstructured Data in Banking. 2024 10*th International Conference on Communication and Signal Processing* (*ICCSP*), Melmaruvathur, 12-14 April 2024, 1027-1031. https://doi.org/10.1109/iccsp60870.2024.10543545

[14] Begum, M. and Arock, M. (2021) Efficient Detection of SQL Injection Attack (SQLIA) Using Pattern-Based Neural Network Model. 2021 *International Conference on Computing*, *Communication*, *and Intelligent Systems* (*ICCCIS*), Greater Noida, 19-20 February 2021, 343-347. https://doi.org/10.1109/icccis51004.2021.9397066

[15] Bello, H.O., Ige, A.B. and Ameyaw, M.N. (2024) Adaptive Machine Learning Models: Concepts for Real-Time Financial Fraud Prevention in Dynamic Environments. *World Journal of Advanced Engineering Technology and Sciences*, **12**, 21-34. https://doi.org/10.30574/wjaets.2024.12.2.0266

[16] Falor, A., Hirani, M., Vedant, H., Mehta, P. and Krishnan, D. (2021) A Deep Learning Approach for Detection of SQL Injection Attacks Using Convolutional Neural Networks. *Proceedings of Data Analytics and Management*, Volume 2, 293-304. https://doi.org/10.1007/978-981-16-6285-0_24

[17] Roy, P., Kumar, R. and Rani, P. (2022) SQL Injection Attack Detection by Machine

Learning Classifier. 2022 *International Conference on Applied Artificial Intelligence and Computing* (*ICAAIC*), Salem, 9-11 May 2022, 394-400. https://doi.org/10.1109/icaaic53929.2022.9792964

[18] Liu, L., Wang, P., Lin, J. and Liu, L. (2021) Intrusion Detection of Imbalanced Network Traffic Based on Machine Learning and Deep Learning. *IEEE Access*, **9**, 7550-7563. https://doi.org/10.1109/access.2020.3048198

[19] Kim, J., Shin, Y. and Choi, E. (2019) An Intrusion Detection Model Based on a Convolutional Neural Network. *Journal of Multimedia Information System*, **6**, 165-172. https://doi.org/10.33851/jmis.2019.6.4.165

[20] Mellah, Y., Kaddari, Z., Bouchentouf, T., Berrich, J. and Belkasmi, M.G. (2021) Intelligent Sketch-Based Recurrent Neural Networks Models to Handle Text-to-SQL Task. *Proceedings of the* 2*nd International Conference on Big Data*, *Modelling and Machine Learning*, Vol. 1, 201-205. https://doi.org/10.5220/0010731000003101

[21] Wen, P., He, C., Xiong, W. and Liu, J. (2021) SQL Injection Detection Technology Based on BiLSTM-Attention. 2021 4*th International Conference on Robotics*, *Control and Automation Engineering* (*RCAE*), Wuhan, 4-6 November 2021, 165-170. https://doi.org/10.1109/rcae53607.2021.9638837

[22] Joshi, A. and Geetha, V. (2014) SQL Injection Detection Using Machine Learning. 2014 *International Conference on Control*, *Instrumentation*, *Communication and Computational Technologies* (*ICCICCT*), Kanyakumari, 10-11 July 2014, 1111-1115. https://doi.org/10.1109/iccicct.2014.6993127

[23] Singh, G., Kant, D., Gangwar, U. and Singh, A.P. (2015) SQL Injection Detection and Correction Using Machine Learning Techniques. In: Satapathy, S.C., *et al.*, Eds., *Emerging ICT for Bridging the Future*, Springer International Publishing, 435-442. https://doi.org/10.1007/978-3-319-13728-5_49

[24] Hasan, M., Balbahaith, Z. and Tarique, M. (2019) Detection of SQL Injection Attacks: A Machine Learning Approach. 2019 *International Conference on Electrical and Computing Technologies and Applications* (*ICECTA*), Ras Al Khaimah, 19-21 November 2019, 1-6. https://doi.org/10.1109/icecta48151.2019.8959617

[25] Mobtahej, P., Zhang, X., Hamidi, M. and Zhang, J. (2022) An LSTM-Autoencoder Architecture for Anomaly Detection Applied on Compressors Audio Data. *Computational and Mathematical Methods*, **2022**, Article ID: 3622426. https://doi.org/10.1155/2022/3622426

[26] Yin, C., Zhang, S., Wang, J. and Xiong, N.N. (2022) Anomaly Detection Based on Convolutional Recurrent Autoencoder for IoT Time Series. *IEEE Transactions on Systems*, *Man*, *and Cybernetics*: *Systems*, **52**, 112-122. https://doi.org/10.1109/tsmc.2020.2968516

[27] Shiri, F.M., Perumal, T., Mustapha, N. and Mohamed, R. (2023) A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU.

[28] Fang, Y., Peng, J., Liu, L. and Huang, C. (2018) WOVSQLI: Detection of SQL Injection Behaviors Using Word Vector and LSTM. *Proceedings of the* 2*nd International Conference on Cryptography*, *Security and Privacy*, Guiyang, 16-19 March 2018, 170-174. https://doi.org/10.1145/3199478.3199503

[29] Radford, B.J., Richardson, B.D. and Davis, S.E. (2018) Sequence Aggregation Rules for Anomaly Detection in Computer Network Traffic.

[30] Yu, W., Kim, I.Y. and Mechefske, C. (2021) Analysis of Different RNN Autoencoder Variants for Time Series Classification and Machine Prognostics. *Mechanical Sys-*

tems and Signal Processing, **149**, Article ID: 107322.
https://doi.org/10.1016/j.ymssp.2020.107322

[31] Thaler, S., Menkovski, V. and Petkovic, M. (2018) Deep Learning in Information Security.

[32] Staudemeyer, R.C. and Morris, E.R. (2019) Understanding LSTM—A Tutorial into Long Short-Term Memory Recurrent Neural Networks.

[33] Lu, D., Fei, J. and Liu, L. (2023) A Semantic Learning-Based SQL Injection Attack Detection Technology. *Electronics*, **12**, Article No. 1344.
https://doi.org/10.3390/electronics12061344

[34] Zhang, B., Li, J., Ren, J. and Huang, G. (2021) Efficiency and Effectiveness of Web Application Vulnerability Detection Approaches: A Review. *ACM Computing Surveys*, **54**, 1-35. https://doi.org/10.1145/3474553

[35] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., *et al.* (2014) Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the* 2014 *Conference on Empirical Methods in Natural Language Processing* (*EMNLP*), Doha, October 2014, 1724-1734.
https://doi.org/10.3115/v1/d14-1179

[36] Chandra, R., Goyal, S. and Gupta, R. (2021) Evaluation of Deep Learning Models for Multi-Step Ahead Time Series Prediction. *IEEE Access*, **9**, 83105-83123.
https://doi.org/10.1109/access.2021.3085085

[37] Reddy, A.S. and Rudra, B. (2022) Detection of Injections in API Requests Using Recurrent Neural Networks and Transformers. *International Journal of Electronic Security and Digital Forensics*, **14**, 638-658. https://doi.org/10.1504/ijesdf.2022.126451

[38] Skaruz, J. (2019) Database Security: Combining Neural Networks and Classification Approach. *Studia Informatica*, No. 1-2, 95-115.

[39] Mou, L., Li, G., Zhang, L., Wang, T. and Jin, Z. (2016) Convolutional Neural Networks over Tree Structures for Programming Language Processing. *Proceedings of the AAAI Conference on Artificial Intelligence*, **30**, 1287-1293.
https://doi.org/10.1609/aaai.v30i1.10139

[40] Li, F., Lu, H., Hou, M., Cui, K. and Darbandi, M. (2021) Customer Satisfaction with Bank Services: The Role of Cloud Services, Security, e-Learning and Service Quality. *Technology in Society*, **64**, Article ID: 101487.
https://doi.org/10.1016/j.techsoc.2020.101487

[41] Tariq, U., Ahmed, I., Bashir, A.K. and Shaukat, K. (2023) A Critical Cybersecurity Analysis and Future Research Directions for the Internet of Things: A Comprehensive Review. *Sensors*, **23**, Article No. 4117. https://doi.org/10.3390/s23084117

[42] Shabbir, A., Shabir, M., Javed, A.R., Chakraborty, C. and Rizwan, M. (2022) Suspicious Transaction Detection in Banking Cyber-Physical Systems. *Computers & Electrical Engineering*, **97**, Article ID: 107596.
https://doi.org/10.1016/j.compeleceng.2021.107596

[43] Taylor, J.L. and Galica, T. (2020) A New Code to Protect Victims in the UK from Authorised Push Payments Fraud. *Banking & Finance Law Review*, **35**, 327-332.

[44] Gottschalk, P. (2022) Financial Crime Issues: Fraud Investigations and Social Control. Springer Nature.

[45] Erdődi, L., Sommervoll, Å.Å. and Zennaro, F.M. (2021) Simulating SQL Injection Vulnerability Exploitation Using Q-Learning Reinforcement Learning Agents. *Journal of Information Security and Applications*, **61**, Article ID: 102903.
https://doi.org/10.1016/j.jisa.2021.102903

[46] Kasim, Ö. (2021) An Ensemble Classification-Based Approach to Detect Attack Level of SQL Injections. *Journal of Information Security and Applications*, **59**, Article ID: 102852. https://doi.org/10.1016/j.jisa.2021.102852

[47] Shwaish, A.K., Hussain, M.A. and Al-Kashoash, H.A. (2020) Encoding Query Based Lightweight Algorithm for Preventing SQL Injection Attack. *Journal of Basrah Researches* (*Sciences*), **46**, 1-11.

[48] Devalla, V., Srinivasa Raghavan, S., Maste, S., Kotian, J.D. and Annapurna, D.D. (2022) Murli: A Tool for Detection of Malicious Urls and Injection Attacks. *Procedia Computer Science*, **215**, 662-676. https://doi.org/10.1016/j.procs.2022.12.068

[49] Gomes, L., Deshmukh, A. and Anute, N. (2022) Cyber Security and Internet Banking: Issues and Preventive Measures. *Journal of Information Technology and Sciences*, **8**, 31-42. https://doi.org/10.46610/joits.2022.v08i02.005

[50] Hidayat, K. and Idrus, M.I. (2023) The Effect of Relationship Marketing towards Switching Barrier, Customer Satisfaction, and Customer Trust on Bank Customers. *Journal of Innovation and Entrepreneurship*, **12**, Article No. 29. https://doi.org/10.1186/s13731-023-00270-7

[51] Esenogho, E., Mienye, I.D., Swart, T.G., Aruleba, K. and Obaido, G. (2022) A Neural Network Ensemble with Feature Engineering for Improved Credit Card Fraud Detection. *IEEE Access*, **10**, 16400-16407. https://doi.org/10.1109/access.2022.3148298

[52] Chhabra Roy, N. and Prabhakaran, S. (2022) Internal-Led Cyber Frauds in Indian Banks: An Effective Machine Learning-Based Defense System to Fraud Detection, Prioritization and Prevention. *Aslib Journal of Information Management*, **75**, 246-296. https://doi.org/10.1108/ajim-11-2021-0339

[53] Nguyen, N., Duong, T., Chau, T., Nguyen, V., Trinh, T., Tran, D., *et al.* (2022) A Proposed Model for Card Fraud Detection Based on Catboost and Deep Neural Network. *IEEE Access*, **10**, 96852-96861. https://doi.org/10.1109/access.2022.3205416

[54] Hashemi, S.K., Mirtaheri, S.L. and Greco, S. (2023) Fraud Detection in Banking Data by Machine Learning Techniques. *IEEE Access*, **11**, 3034-3043. https://doi.org/10.1109/access.2022.3232287

[55] Ileberi, E., Sun, Y. and Wang, Z. (2021) Performance Evaluation of Machine Learning Methods for Credit Card Fraud Detection Using SMOTE and Adaboost. *IEEE Access*, **9**, 165286-165294. https://doi.org/10.1109/access.2021.3134330

[56] Taha, A.A. and Malebary, S.J. (2020) An Intelligent Approach to Credit Card Fraud Detection Using an Optimized Light Gradient Boosting Machine. *IEEE Access*, **8**, 25579-25587. https://doi.org/10.1109/access.2020.2971354

[57] Gandhi, N., Patel, J., Sisodiya, R., Doshi, N. and Mishra, S. (2021) A CNN-BiLSTM Based Approach for Detection of SQL Injection Attacks. 2021 *International Conference on Computational Intelligence and Knowledge Economy* (*ICCIKE*), Dubai, 17-18 March 2021, 378-383. https://doi.org/10.1109/iccike51210.2021.9410675

[58] Chen, D., Yan, Q., Wu, C. and Zhao, J. (2021) SQL Injection Attack Detection and Prevention Techniques Using Deep Learning. *Journal of Physics*: *Conference Series*, **1757**, Article ID: 012055. https://doi.org/10.1088/1742-6596/1757/1/012055

[59] Derhab, A., Aldweesh, A., Emam, A.Z. and Khan, F.A. (2020) Intrusion Detection System for Internet of Things Based on Temporal Convolution Neural Network and Efficient Feature Engineering. *Wireless Communications and Mobile Computing*, **2020**, Article ID: 6689134. https://doi.org/10.1155/2020/6689134

[60] Anitha, T., Aanjankumar, S., Poonkuntran, S. and Nayyar, A. (2023) A Novel Methodology for Malicious Traffic Detection in Smart Devices Using BI-LSTM-CNN-De-

pendent Deep Learning Methodology. *Neural Computing and Applications*, **35**, 20319-20338. https://doi.org/10.1007/s00521-023-08818-0

[61] Khan, M.A. (2021) HCRNNIDS: Hybrid Convolutional Recurrent Neural Network-Based Network Intrusion Detection System. *Processes*, **9**, Article No. 834. https://doi.org/10.3390/pr9050834

[62] Mahboubi, A., Luong, K., Aboutorab, H., Bui, H.T., Jarrad, G., Bahutair, M., *et al.* (2024) Evolving Techniques in Cyber Threat Hunting: A Systematic Review. *Journal of Network and Computer Applications*, **232**, Article ID: 104004. https://doi.org/10.1016/j.jnca.2024.104004

[63] Mallick, M.A.I. and Nath, R. (2024) Navigating the Cyber Security Landscape: A Comprehensive Review of Cyber-Attacks, Emerging Trends, and Recent Developments. *World Scientific News*, **190**, 1-69.

[64] Rahul, S., Vajrala, C. and Thangaraju, B. (2021) A Novel Method of Honeypot Inclusive WAF to Protect from SQL Injection and Xss. 2021 *International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications* (*CENTCON*), Vol. 1, 135-140. https://doi.org/10.1109/centcon52345.2021.9688059

[65] Tadhani, J.R., Vekariya, V., Sorathiya, V., Alshathri, S. and El-Shafai, W. (2024) Securing Web Applications against XSS and SQLi Attacks Using a Novel Deep Learning Approach. *Scientific Reports*, **14**, Article No. 1803. https://doi.org/10.1038/s41598-023-48845-4

[66] Darwish, D. (2024) Improving Techniques for Convolutional Neural Networks Performance. *European Journal of Electrical Engineering and Computer Science*, **8**, 1-16. https://doi.org/10.24018/ejece.2024.8.1.596

[67] Yao, P., Shen, S., Xu, M., Liu, P., Zhang, F., Xing, J., *et al.* (2022) Single Model Deep Learning on Imbalanced Small Datasets for Skin Lesion Classification. *IEEE Transactions on Medical Imaging*, **41**, 1242-1254. https://doi.org/10.1109/tmi.2021.3136682

[68] Jagielski, M., Thakkar, O., Tramer, F., Ippolito, D., Lee, K., Carlini, N., Wallace, E., Song, S., Thakurta, A. and Papernot, N. (2022) Measuring Forgetting of Memorized Training Examples.

[69] Leino, K. and Fredrikson, M. (2020) Stolen Memories: Leveraging Model Memorization for Calibrated {White-Box} Membership Inference. 29*th USENIX Security Symposium* (*USENIX Security* 20), Boston, 12-14 August 2020, 1605-1622.

[70] Paul, A., Sharma, V. and Olukoya, O. (2024) SQL Injection Attack: Detection, Prioritization & Prevention. *Journal of Information Security and Applications*, **85**, Article ID: 103871. https://doi.org/10.1016/j.jisa.2024.103871

[71] Muhuri, P.S., Chatterjee, P., Yuan, X., Roy, K. and Esterline, A. (2020) Using a Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) to Classify Network Attacks. *Information*, **11**, Article No. 243. https://doi.org/10.3390/info11050243

[72] Chattopadhyay, A. and Sripada, D. (2023) Security Analysis and Threat Modelling of Mobile Banking Applications. 2023 14*th International Conference on Computing Communication and Networking Technologies* (*ICCCNT*), Delhi, 6-8 July 2023, 1-6. https://doi.org/10.1109/icccnt56998.2023.10307577

[73] Bakır, R. and Bakır, H. (2024) Swift Detection of XSS Attacks: Enhancing XSS Attack Detection by Leveraging Hybrid Semantic Embeddings and AI Techniques. *Arabian Journal for Science and Engineering*, **50**, 1191-1207. https://doi.org/10.1007/s13369-024-09140-0

[74] Grange, A. (2024) Learning SQL from within: Integrating Database Exercises into the

Database Itself.

[75] Jemal, I., Haddar, M.A., Cheikhrouhou, O. and Mahfoudhi, A. (2020) M-CNN: A New Hybrid Deep Learning Model for Web Security. 2020 *IEEE/ACS 17th International Conference on Computer Systems and Applications* (*AICCSA*), Antalya, 2-5 November 2020, 1-7. https://doi.org/10.1109/aiccsa50499.2020.9316508

[76] Sekar, J. (2023) Real-Time Fraud Prevention in Digital Banking a Cloud and AI Perspective. *Journal of Emerging Technologies and Innovative Research*, **10**, P562-P570.