SQL Injection Detection Using Machine Learning

By

Alvin Munene Muriungi

087496

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Information Systems Security at Strathmore University

School of Computing & Engineering Sciences

Strathmore University

Nairobi, Kenya

June, 2025

Declaration and approval page

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the dissertation contains no material previously published or written by another person except where due reference is made in the dissertation itself.

Student's Name: **Alvin Muriungi**

Sign: _____ Date: 19/05/2025

Approval

The dissertation of **Alvin Muriungi** was reviewed and approved for examination by the following:

Dr. Vitalis Ozianyi,

School of Computing & Engineering Sciences,

Strathmore University

Dr. Julius Butime,

Dean, School of Computing & Engineering Sciences,

Strathmore University

Prof. Bernard Shibwabo,

Director of Graduate Studies,

Strathmore University

Abstract

Web applications have become prime targets for cyberattacks due to their accessibility and the sensitive information they handled. SQL injection was a prevalent attack technique that exploited vulnerabilities in input validation to execute malicious SQL queries, potentially compromising application security and integrity. Traditional SQL injection detection methods, which relied on predefined patterns and rules, struggled to adapt to the evolving tactics of attackers. The purpose of this study was to design and implement a machine learning (ML)-based SQL injection detection system utilizing convolutional neural networks (CNNs). CNNs were selected due to their proven effectiveness in identifying complex patterns and anomalies, making them suitable for detecting union-based, error-based, and blind SQL injection attacks. An analytical prototyping methodology was employed to develop and evaluate the system. The model was trained on a diverse dataset of benign and malicious SQL queries and assessed using standard performance metrics. The CNN model achieved an accuracy of 98.21%, with macro-averaged precision, recall, and F1-score all at 0.98. These results demonstrated the model's robustness and effectiveness in distinguishing between legitimate and malicious input. The findings of this study indicated that CNN-based detection systems can significantly enhance the security of web applications by providing an adaptive and reliable defense against SQL injection attacks.

Keywords: SQL injection, Machine Learning, Convolutional Neural Networks, Anomaly Detection, Cybersecurity, Input Validation.

Table of content

List of Figures

Code Snippets

Abbreviations and Acronyms

CNN – Convolutional Neural Networks

NN – Neural Networks

RDMS – Relational Database Management System

SQL -Standard Query Language

SQLI – Standard Query Language Injection

SQLIA – Standard Query Language Injection Attacks

Definition of Terms

Anomaly Detection	A technique used to identify deviations from expected behaviour, commonly applied in cybersecurity to detect malicious activities (Chandola et al., 2009).
Blind SQL Injection	A type of SQL injection where no direct data is retrieved, but attackers infer information by analysing the application's response to queries (Rai et al., 2021).
Convolutional Neural Networks (CNNs)	A deep learning algorithm particularly effective in pattern recognition, used in this study to detect SQL injection attacks (Ghosh et al., 2020).
Dataset	A collection of data used to train and evaluate machine learning models, consisting of both benign and malicious SQL queries in this study (Azman et al., 2021).
Error-Based SQL Injection	A technique that forces database error messages to disclose information about the database structure (Khan et al., 2023).
Feature Extraction	The process of selecting key characteristics from raw data to improve machine learning model performance (Ghosh et al., 2020).
Machine Learning (ML)	a branch of Artificial Intelligence (AI) and computer science focuses on how data and complex algorithms can be used to emulate human thinking and ability to learn and improve its accuracy (Shaveta, 2023).
SQL Injection (SQLI)	A code injection technique that exploits security vulnerabilities in an application's database query execution by inserting malicious SQL statements (Atefeh Tajpour et al., 2010).
Supervised Learning	A type of machine learning where a model is trained on labelled data to learn patterns for classification or prediction tasks (Ali, 2013).
Union-Based SQL Injection	A type of SQL injection attack where attackers use the UNION SQL operator to retrieve data from additional database tables (Khan et al., 2023).
Web Application Firewall (WAF)	A security system designed to protect web applications by filtering and monitoring HTTP traffic to prevent cyberattacks (Kumar, 2023).

Acknowledgement

I wish to express my heartfelt gratitude to the Almighty for bestowing upon me the guidance, strength, and resilience to undertake this research journey. Without His divine assistance, I would not have been able to navigate the challenges and complexities inherent in the research process.

I would also like to express my sincere appreciation to my esteemed supervisor, Dr Vitalis Ozianyi. Your unwavering support, expert guidance, and selfless dedication have been instrumental in shaping the trajectory of my research and refining the quality of my work. Your willingness to engage in consultative meetings, provide constructive feedback, and offer invaluable insights have been invaluable assets throughout this research.

I would also like to give special thanks to my friend Miriam Njoroge for her assistance during the research.

I would also like to acknowledge the profound influence and support of my parents, whose boundless love, encouragement, sacrifices and belief in my abilities continue to resonate with my ambition, inspiring me to strive for greatness.

Chapter One: Introduction

1.1. Background of the study

Structured Query Language (SQL) is a programming language used to query and manage Relational Database Management Systems (RDBMS) such as MySQL, Oracle, and SQL Server. It plays a critical role in web applications that collect user inputs to construct SQL queries for database interactions. However, vulnerabilities in input handling can be exploited by attackers to inject malicious SQL code, resulting in what is known as a Structured Query Language Injection Attack (SQLIA). This type of attack undermines the functionality, confidentiality, and integrity of stored data by enabling unauthorized access or alterations, ultimately compromising database security and application reliability (Atefeh Tajpour et al., 2010).

SQLIA is recognized as one of the most prevalent threats to databases, affecting virtually any web application that uses a database as its backend. Hypertext Preprocessor (PHP) and Active Server Pages (ASP) applications are particularly susceptible due to their widespread use in dynamic web development. The primary objective of SQL injection attacks is often to compel databases to reveal private information, such as usernames, passwords, or secret keys. A report by Kigen et al. (2015) highlighted the rising frequency of cyberattacks targeting web applications, emphasizing the increasing exploitation of SQL injection vulnerabilities.

The ramifications of SQL injection attacks can be significant, as demonstrated in several real-world scenarios. For instance, in 2019, hackers exploited a SQL injection vulnerability in an Australian healthcare website, exposing sensitive patient data, including medical records and payment information. The breach was discovered when a user accidentally accessed another user's data and reported the issue (Taylor, 2019). Similarly, an attacker leveraged a SQL injection flaw in a content management system to gain administrative control, leading to the defacement of a news website and damage to its reputation (Johny et al., 2021). In another example, banking applications with SQL injection vulnerabilities allowed attackers to bypass login processes, gaining unauthorized access to customer accounts and facilitating illegal activities like unauthorized money transfers (Tiwari et al.,

2012). These cases illustrate how SQL injection can lead to data breaches, financial losses, and reputational damage.

To combat SQL injection, several security measures have been implemented. Web Application Firewalls (WAFs) are designed to detect and block malicious traffic but can inadvertently hinder legitimate requests and impact application performance (Kumar, 2023). Stored procedures, which encapsulate SQL statements, can improve security but pose challenges in terms of encryption and maintenance (Arasu et al., 2014). Input validation and whitelisting aim to enforce strict input standards but may be insufficient for complex or enterprise-level error handling (Baah & Kabari, 2012). Despite these defenses, the evolving nature of SQL injection techniques, such as obfuscation, encoding, and dynamic payloads, often outpaces traditional detection methods, rendering them less effective.

Machine learning (ML) has emerged as a promising approach to address these limitations by providing a dynamic and adaptive mechanism for detecting SQL injection attacks. Unlike static rule-based or signature-based systems, ML models can analyze patterns in raw query data, understand the context and purpose of queries, and detect anomalies indicative of potential SQL injection attempts. Convolutional Neural Networks (CNNs), for example, are particularly adept at identifying intricate patterns and adapting to new attack vectors (Ghosh et al., 2020). By continuously learning from historical data, these models improve detection accuracy and reduce false positives. Furthermore, ML algorithms can adapt to evolving attack methodologies, making them more effective in addressing the sophistication of modern cyber threats.

The sophistication of cyber attackers continues to evolve, making advanced detection methods essential. A 2015 report identified the top cybersecurity challenges in Kenya, which included data exfiltration, social engineering, insider threats, database breaches, and poor identity and access management (Kigen et al., 2015). These challenges highlight the urgency of implementing robust, scalable, and adaptive solutions to safeguard databases and web applications. By leveraging ML, particularly CNNs, it is possible to enhance SQL injection detection and prevention, contributing to a more secure and resilient digital environment. This dissertation explores the potential of ML-based models to address the

complexities of SQL injection threats, aiming to mitigate the risks posed by one of the most persistent and damaging cybersecurity challenges.

1.2 Problem statement

SQL Injection Attacks (SQLIAs) continue to be one of the most dangerous threats to web applications. By manipulating user inputs, attackers can inject malicious SQL commands that access, alter, or even delete sensitive data stored in Relational Database Management Systems (RDBMS). As more organizations move their operations online and store confidential data on web platforms, the risk posed by these attacks becomes even greater.

While traditional security measures like input validation, parameterized queries, stored procedures, and Web Application Firewalls (WAFs) have been helpful, they are no longer enough. Attackers are constantly evolving their methods, using techniques like obfuscation, encoding, and dynamic payloads to slip past these defenses. This leaves many systems vulnerable to breaches, financial losses, and reputational damage. The continued prominence of SQL injection on the OWASP Top 10 list as recently as 2021 is a clear sign that more effective and adaptive solutions are needed.

In recent years, researchers have turned to Machine Learning (ML) to tackle this challenge. ML models can learn from historical data to spot patterns and anomalies that might signal an attack. Among these, Convolutional Neural Networks (CNNs) have shown particular promise due to their strength in recognizing complex patterns even in noisy or unfamiliar input. This makes them well-suited to detecting SQLIAs, especially those crafted to evade traditional filters.

However, the use of CNNs for SQL injection detection is still relatively limited. Most existing research has focused on more conventional ML techniques, and only a few studies have explored how CNNs can be tailored specifically for this task. There's also a lack of work on how these models can be made robust and adaptable enough for real-world deployment, where attack methods are constantly evolving.

This research aims to fill that gap by exploring how CNNs can be used more effectively for detecting and preventing SQL injection attacks. By building on what has already been done and addressing current limitations, the goal is to develop a more intelligent and

reliable solution for protecting web applications. The next section—the literature review will take a closer look at previous studies in this area, identifying what has been achieved and where opportunities for improvement still exist.

1.3 Objectives

The purpose of this dissertation was to design, implement, and evaluate a Convolutional Neural Network (CNN) model for detecting SQL injection attacks in web applications.

1.3.1 Specific objectives

i. To analyse techniques for launching SQL injection attacks

ii. To assess the operation of existing SQL injection detection mechanisms

iii. To design and implement an algorithm for SQL injection detection using machine learning

iv. To test and validate the ML SQL injection detection algorithm

1.4 Research Questions

i. What techniques are used in SQL injection attacks?

ii. How do current tools used in detection of SQL injection attacks operate?

iii. How can an algorithm for SQL Injection detection be designed and implemented?

iv. How does the SQL injection detection algorithm perform?

1.5 Scope and Limitation

This dissertation focuses on developing and evaluating machine learning algorithms for SQL injection detection, including classic techniques like Union-based, Error-based, and Blind SQL injection. The goal is to provide robust defence against web application vulnerabilities. However, limitations include reliance on labelled datasets for training models, and the need for continuous monitoring and model retraining. The study focuses on supervised learning techniques, while unsupervised learning approaches may be limited by web application environments' complexity.

1.6 Justification of the study

This dissertation is worthwhile as the security and integrity of online/web applications are threatened by SQL injection attacks. For this reason, it is important to identify these attacks in order to protect sensitive data from unintended access. Through flaws in input fields, attackers are able to alter SQL queries and run harmful instructions to the database. These types of attacks can cause theft of private data, data breaches, financial loss, legal repercussions and damage to an organization's brand.

Because machine learning can evaluate large volumes of data and identify patterns that point to malicious conduct, it provides a viable method for SQL injection detection. Machine learning models may be trained to distinguish between suspicious and legitimate activity by using labelled datasets that contain instances of both benign and malicious SQL queries. Furthermore, machine learning algorithms are flexible and dynamic, which improves their capacity to recognize new and advanced attack methods. Organizations may strengthen their defences against SQL injection attacks and lower the risk of data breaches by incorporating machine learning-based detection algorithms into web application security frameworks. This will also strengthen the resilience of the systems against cyber threats.

The proposal doesn't explicitly address the fair distribution of and access to the benefits of the research. It is important to consider how the research outcomes (e.g., the developed SQL injection detection system) will be made available to the wider community, particularly to organizations that may not have the resources to develop their own solutions. This could involve open-sourcing the code, providing access to the system through a cloud-based service, or publishing detailed documentation and tutorials.

This research is designed to make SQL injection detection more effective and accessible to a wide range of users. Cybersecurity threats affect organizations of all sizes, and not all of them have the resources to develop advanced security solutions on their own. By using open-source tools like TensorFlow and Keras, this project ensures that the knowledge and technology can be shared with researchers, security professionals, and even smaller businesses that may not have the budget for expensive security systems.

The goal is to make the findings available through academic publications, open datasets, and possibly a cloud-based tool, so that anyone who needs it can benefit. This way, the research doesn't just serve large companies with deep pockets it helps strengthen cybersecurity for everyone.

Chapter Two: Literature Review

2.1. Introduction

This chapter presents review of literature on SQL injections. It expounds on characteristics of SQL injections, types of SQL injections and existing techniques for detecting them. The chapter also explores the use of ML techniques in cybersecurity, with focus on anomaly detection. Discussions on potential use of ML in detection of SQL injection are presented.

2.2. SQL Injection Attack Characteristics

In a report published by OWASP (2017) the top ten severe security risks targeted towards web applications are as shown in Figure 2.1:OWASP top ten sever security risks**Error! Reference source not found.** below, with SQL injection attacks leading. However, in a report published by OWASP in 2021, injection attacks have gone down to number 3 but should still be considered as a threat as it has the second greatest number of occurrences in applications (OWASP Top Ten | OWASP Foundation, n.d.). Figure 2-1 illustrates the top ten security risks and the changes in severity from 2017 to 2021.

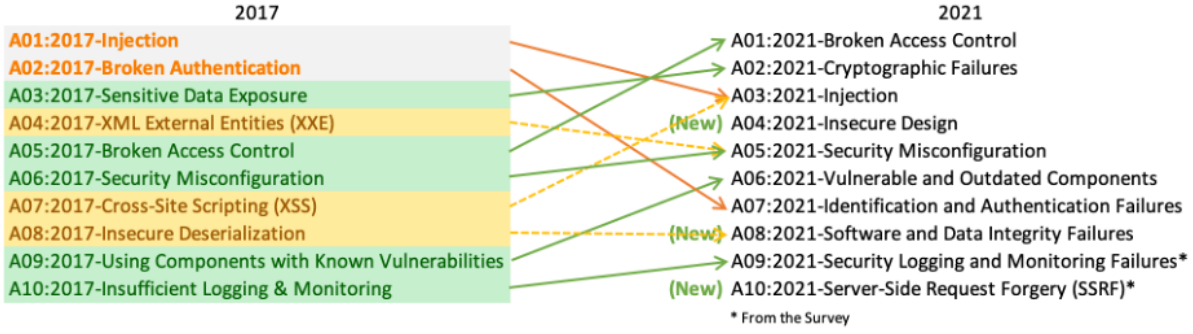


Figure 2.1:OWASP top ten sever security risks

Most organisations today rely on web applications to conduct business processes, and more and more organisations are slowly migrating from the traditional in-house systems to web applications allowing both clients and staff to access systems through smart devices such as mobile phones, laptops, tablets etc. However, because of the high demand for web applications, some security standards may be overlooked, which attackers may exploit. As a result, the vulnerabilities linked to these applications are rapidly increasing.

You Yu et al., (2011) states that weak web applications have enabled attackers to easily get access to systems and accounts of which are labelled as private and confidential. The typical working of web applications is as follows: (1) User submits login request via web interface including the username and password; (2) Before sending the request to the database, a dynamic SQL query is generated in the backend of the software and sent to the database; (3) The SQL query is received by the database and executed thus generating a response or output. It is during the second step that dangerous queries resulting from use of injected payloads would be created and subsequently sent to the database.

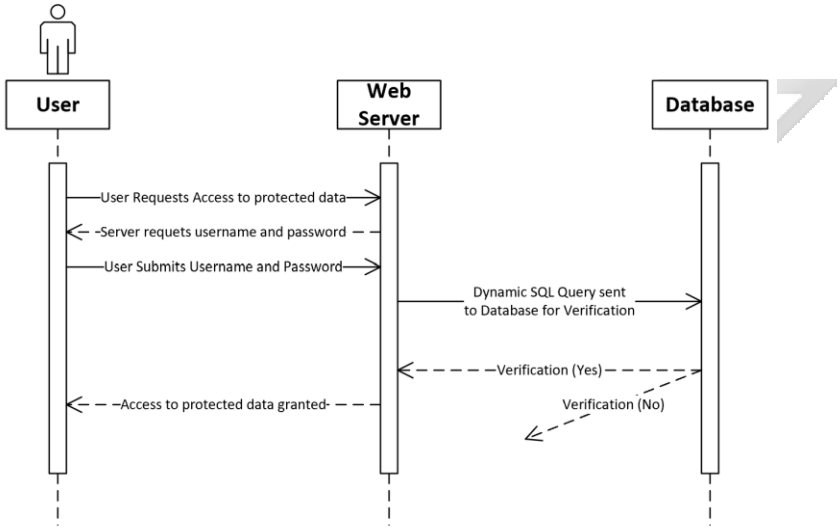


Figure 2.2: Sample Login Procedure

2.3. Types Of SQL Injection Attacks

SQL injection attacks may be classified mainly into three categories based on several factors. These are in-band (classic SQLi), Inferential (blind SQLi) and Out-of-Band SQLi (Khan et al., 2023).

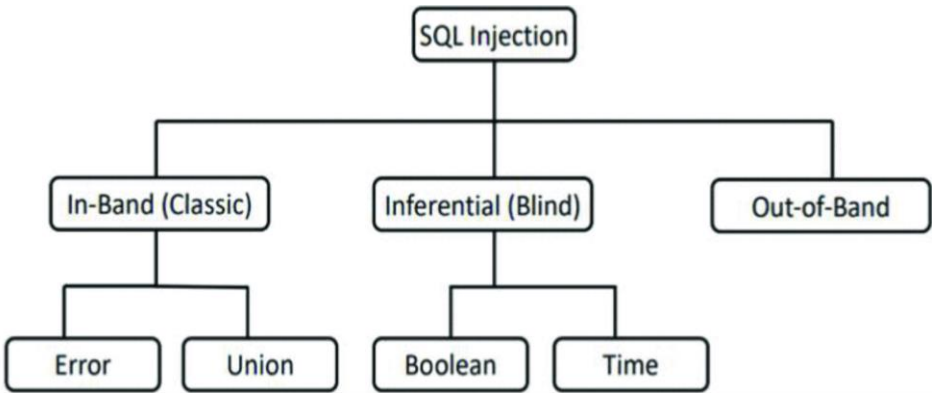


Figure 2.3: Type of SQLi Attacks

2.3.1. In-band (classic SQLi)

This is the easiest and most common type of SQL Injection attack in that it utilises the same channel for the attack and receiving the results after the attack. There are two types of in-band SQLi namely; error based SQLi and union based SQLi (Rai et al., 2021).

A. Error based SQLi

Error based SQLi relies on error messages received from database server that tend to disclose the entire database structure. From the error message an attacker can be able to recreate a payload that will eventually result in a successful attack (Khan et al., 2023).

B. Union based SQLi

Union based SQLi combines the results of more than two SQL SELECT statements to generate a unified (single) result. Commonly these commands are sent using an SQL UNION command (Khan et al., 2023).

2.3.2. Inferential (blind SQLi)

Inferential SQLi takes a while longer than in-band SQLi for an attacker to exploit but still poses a great threat. This type of attack does not involve the transfer of data i.e. no data is transferred from the web application to the database and no result is generated from the attack as well hence the name blind attacks. This type of attack uses Boolean type questions or time-based queries which are also the subtypes of the attacks (Rai et al., 2021).

A. Boolean Based SQLi

When the database generates or displays no output, we utilize the true false condition to provide a different response based on whether the supplied query returned true or false. If the system behaved different when subjected to either false or true payload, then it is vulnerable to Boolean based SQLi attacks.

Below are examples of Boolean based SQLIA

A.1. True Payload

```
SELECT id FROM users WHERE id =1 AND 1=2
```

Code Snippet 1: True Payload

A.2. False Payload

```
SELECT id FROM users WHERE id =1 AND id=2
```

Code Snippet 2: False Payload

If the result of the above is different, then the database is susceptible to Boolean based attacks.

B. Time based SQLi

This is an SQLi technique that sends an SQL query to the database that forces it to take some time before responding back. The time taken to respond determines if the result is true or false. This in turn make this type of attack really slow especially on larger databases as each character is enumerated one by one.

2.3.3. Out-of-band SQLi

This is a type of injection attack that the attacked application does not send a response to the attacker on the same channel the attack was sent, but instead has the application send the response to a remote site that the attacker has control over (Out-of-Band SQL Injection | Learn AppSec, 2022). This type of attack depends on features such as DNS or HTTP that allows the attacker to send the responses to a remote site.

2.4. Use of Legacy Solutions to Prevent SQL Injection Attacks

Below are some of the existing SQL injection techniques currently used to detect and prevent SQL injections.

2.4.1. Web Application Firewalls

A Web Application Firewall (WAF) is a type of application firewall designed to protect HTTP-based web applications. A WAF manages an HTTP interaction using a collection of rules. Generally, these rules protect against common vulnerabilities such as Cross-site scripting (XSS), SQL Injection, etc. A WAF is often used to protect a single web application (or a collection of web applications) and is considered a reverse proxy. A WAF can take the form of an appliance, a server plugin, or a filter and can be customized to a particular application (Omar et al., 2023).

WAFs have several limitations, including the potential to block legitimate traffic or requests, which can cause issues for users accessing the application, and the need for manual review and blacklisting by the administrator. Additionally, WAFs can slow down the application and add extra overhead, which can be problematic for critical applications with high traffic or high-performance requirements.

2.4.2. Stored Procedures

A set of SQL statements in SQL is called a stored procedure. These statements are stored together in the database. A stored procedure can perform one or several Data Manipulation Language (DML) operations in the database. The return value of the stored procedure depends on the statements contained in the procedure, and the arguments passed to it (Brimhall et al., 2015). Stored Procedures have two main drawbacks: they are unencrypted by default, allowing simultaneous access to the database, and they add complexity to application development and maintenance (Brimhall et al., 2015).

2.4.3. Input Validation and Whitelisting

Input validation is when an application checks the input it gets against a certain set of standards within the app. There are two types of input validation: whitelist (inclusion or positive) and blacklist (exclusion or negative). Input Validation and Whitelisting have limitations, including potential errors in enterprise-level input validation and insufficient coverage of all attack vectors, and may not be sufficient to adapt to evolving attack strategies (Baah & Kabari, 2012).

2.5. Use of Artificial Intelligence to Prevent SQL Injection Attacks

To improve web application security, it is necessary to strengthen the system itself through rigorous scanning and vulnerability detection. According to Betarte et al. (2018), using machine learning approaches can improve Web Application Firewalls' accuracy and efficiency in identifying SQL Injection attacks.

2.5.1. Machine Learning

Machine Learning as a branch of Artificial Intelligence (AI) and computer science focuses on how data and complex algorithms can be used to emulate human thinking and ability to learn and improve its accuracy (Shaveta, 2023). In the context of information security, machine learning can be used to efficiently determine and classify network traffic as either normal traffic or an attack in order to protect IT assets without interrupting normal day to day business transactions. There are multiple machine learning techniques/algorithms that can be used.

A. Supervised Learning

Supervised Learning (SL) also known as task-oriented learning, is the most widely used machine learning technique as it is the simplest method to employ. It utilises known input data and output data for training to predict future output data. It has multiple applications to face recognition and spam classification (Ali, 2013). Supervised Learning is divided into two categories: regression, which determines the relationship between variables for projections, and classification, which accurately groups data into specific categories based on labelling, such as true or false or spam or not spam.

Examples of supervised learning algorithms include Naïve Bayes, Decision Trees, KNN, Neural Networks, SVM, and Linear Regression.

I. **Neural Networks** – this involves the training of data by mimicking the human brain neural network through layers of nodes. They solve complex problems through the use of non-linearity and hidden relationships between both inputs and outputs (Hosam et al., 2021). Some examples of Neural Networks include:

a) Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of multilayer neural network generally used for special data. Typically, a CNN consists of one fully connected (FC) or more (FC) connected layers. The FC layer takes inputs from the final (pooling) or (convolution) layer and outputs the final (CNN) output. Both the convolution layer and the pooling layer perform feature extraction (Ghosh et al., 2020).

CNN offers advantages such as reducing trainable features to avoid overfitting and improve generalization, learning classification and feature extraction together to create more organized output, and being more efficient in large datasets compared to other neural networks.

b) Recurrent Neural Networks

A Recurrent Neural Network (RNN) is any network that has a loop in its network connections. In other words, the value of a unit depends directly or indirectly on its previous outputs as input. These networks are very powerful, but they are hard to model and train.

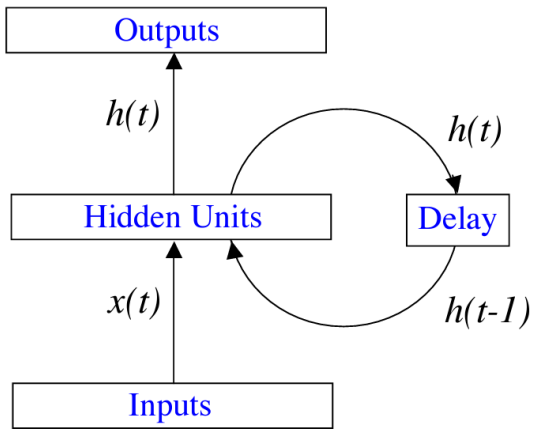


Figure 2.4: Recurrent Neural Network (RNN)

RNN offers several advantages, including the ability to model a set of records based on previous patterns, and the ability to combine RNN with convolution layers for a strong pixel neighbourhood, making each pattern dependent on the previous ones.

c) Feedforward Neural Networks

A feed forward neural network is an artificial neural network where the nodes' connections are not looped. Feed forward model is the most basic form of neural network because information only flows in one direction. The data may pass through several hidden nodes, but it always flows in one direction and not in the opposite direction.

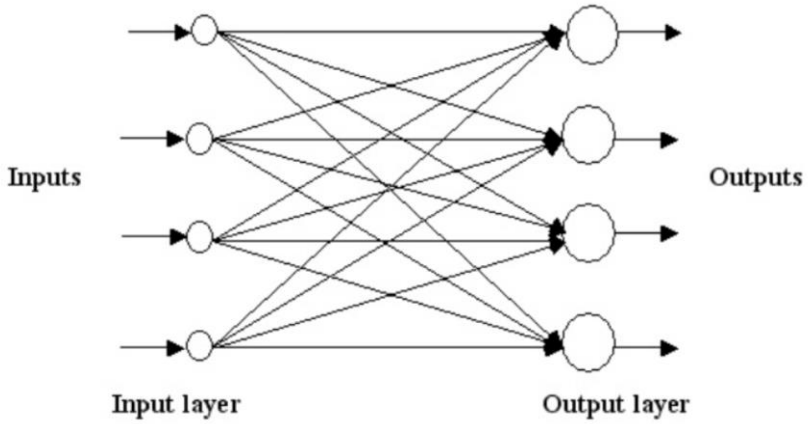


Figure 2.5: Feedforward Neural Networks

FNNs offer several advantages, including their ability to handle nonlinear data in a straightforward manner, addressing decision boundaries, and being suitable for regression and classification, unlike more complex perceptrons and sigmoid neurons. They also excel in regression and classification.

Neural networks identify SQL injection attacks by changing SQL queries into numerical forms and then learning the patterns through several models. If the data is of a fixed size, then a feedforward network is used. If the data is sequential to capture dependencies, then a recurrent network is used. A convolutional neural network is applied where local patterns are to be detected. The network is then trained using labelled queries so that it identifies other queries as either benign or malicious, making use of its capability to learn complex patterns to increase accuracy. The challenges in this method are large-labelled datasets and computational resources.

II. **Naïve Bayes** – This type of algorithm is used in the classification of tasks. Through the process of classification, one feature does not impact the probability of a given outcome as each identifier has its own and equal effect on the overall result.

Naive Bayes has limitations due to its conditional independence of features and its inability to handle complex numerical data with a Gaussian distribution. It may struggle with imbalanced datasets and skewed class distributions and may favour majority classes with few training samples. It may also struggle with complex correlations between characteristics, affecting its accuracy (Pajila et al., 2023).

For the detection of SQL injection attacks, Naïve Bayes looks for key features in SQL queries e.g. specific keywords, length of query, or unusual characters that might give way to an attack. The model then learns to associate these features with either normal or malicious behaviour. In this case, upon entry of a new query, the model will match its features against the likelihood that it is safe or that it is an injection attempt.

III. **Linear regression** – this algorithm uses the value of one variable to determine the next possible variable depending on the relationship between one dependent variable and one or more independent variables. It uses the method of least squares to plot a line of best fit on a graph.

Linear regression has a limitation as it requires interpretation to be limited to the range of predictor variables observed in the data, and it cannot be assumed to continue beyond the sample data range (Marill, 2004).

You can identify SQL injections using linear regression by generating features from SQL queries, for example by tokenization, special characters, query length, labelling each query as normal or malicious, and training a linear regression model on the data. After evaluating the model using accuracy and similar metrics, you can then deploy it to monitor and flag possibly harmful SQL queries in real-time.

IV. **Support Vector Machines** – developed by VladimirVapnik, Support Vector Machines (SVM) is used for both data classification and regression. It works by coming up with a hyperplane that separates the classes of data points on either side of plane (Yu & Kim, 2012). For this reason, SVM is known to find more optimal solutions over neural networks or rule-based techniques.

SVM has limitations such as not being suitable for large data sets, underperforming when data sets have more noise, and also underperforming when the number of features for

each data point exceeds the number of training data samples. SVM can be used in SQL injection detection by tokenizing and vectorizing the SQL queries into numerical feature vectors. After the creation of a dataset of benign and malicious queries, respectively, with appropriate labelling, one can train and evaluate the performance of an SVM model using appropriate metrics. The model can be deployed in real-time SQLi detection. Continuous learning and updating of the model help in coping with new attack patterns. Some key considerations include challenges such as feature selection and handing class-imbalanced data.

V. **K-Nearest Neighbor** – This algorithm classifies data based on how they associate with available data and their proximity to each other as it assumes similar data points are usually near each other. The processing time is faster for smaller data sets but as the dataset grows so does the processing time making it not desirable for data classification tasks.

   K-Nearest Neighbor (KNN), has limitations such as not scaling well, falling victim to the curse of dimensionality, and being prone to overfitting. It takes up more memory and data storage, which can be costly and time-consuming. Additionally, KNN is prone to the peaking phenomenon, where additional features increase classification errors, especially when sample sizes are smaller (Song et al., 2017).

   KNN includes extraction of features, such as keyword frequency and query length, or even pattern matching, followed by training using a labelled dataset, and then classifies new queries against the nearest neighbours in a multi-dimensional feature space. These challenges arise when dealing with high-dimensional data and class-imbalanced datasets. Optimization techniques like dimensionality reduction and weighted KNN can be applied to improve performance.

VI. **Decision Trees** – this is hierarchical algorithm that classifies data in the form of root node, branches, internal nodes and leaf nodes. It is commonly used for regression and classification tasks.

   Decision trees have limitations such as overfitting training data, capturing noise as a pattern, and producing unstable trees due to small variations in data. They may not always yield the most accurate predictions and may produce biased trees favouring the dominant

class in unbalanced datasets. Despite their ease of interpretation, decision trees may not always yield the most accurate models compared to other algorithms.

The use of decision trees involves collecting and labelling datasets containing benign and malicious SQL queries, extracting features such as keyword presence, structural analysis, and entropy, and training a decision tree model in classifying these queries as safe versus potentially harmful. To adapt to new methods of attack, the model will maintain its ability once integrated into the database layer of an application for the real-time detection of such queries, with regular updates and retraining.

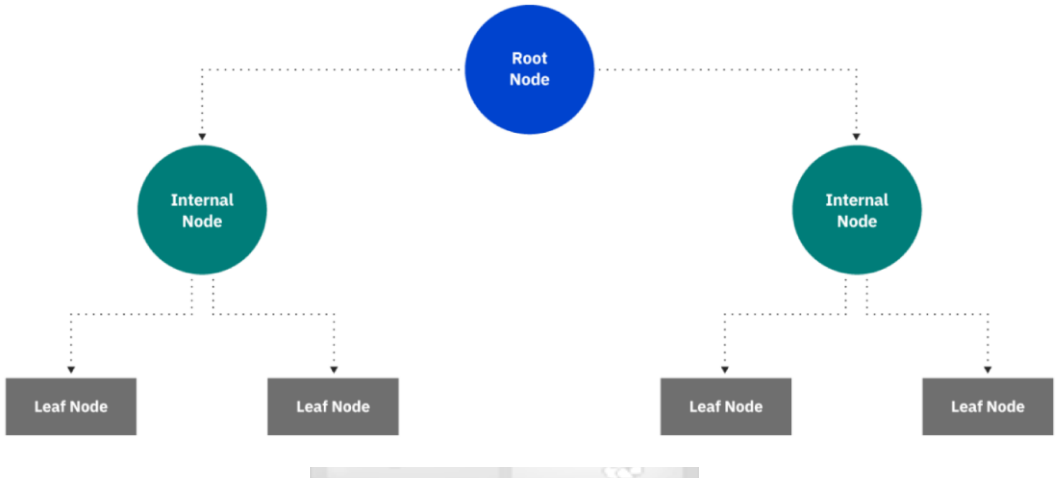


Figure 2.6: Decision Tree

B. Unsupervised Learning

Unsupervised learning is used in the analysis of unlabelled datasets in order to discover hidden data patterns without any human intervention. This ability to identify hidden patterns makes unsupervised learning ideal for exploratory data analysis, image recognition etc. Examples include Hidden Markov model, K-means clustering and Gaussian mixture models.

In the case of SQL injection detection, unsupervised learning can be applied by analyzing the SQL queries with clustering and anomaly detection techniques. Through tokenization of queries, extraction of structural features, and identification of patterns, unsupervised models will be able to detect unusual or anomalous queries that could indicate SQL injection. This approach requires continuous monitoring, manual

reviewing, and updating of models to adapt to new threats against which a balance needs to be struck between detecting true positives while minimising false positives.

Unsupervised learning has limitations such as learning one step at a time due to additional hierarchy needs, predicting clusters (K-value), affecting data direction, and being sensitive to outliers, making it linearly increase in learning time (Dike et al., 2018).

C. Reinforcement Learning

This is a technique that learns from past actions and experiences. It behaves similar to how human beings learn and grow based on their environment. It is commonly used in video games, industrial simulations etc. Examples include Temporal difference, Q-learning and Deep adversarial networks.

Reinforcement Learning (RL) can be used in SQL injection detection by training an agent to identify dangerous SQL queries using attributes such as keywords, patterns and query structures. The environment has been formed with states being SQL queries whereas actions are the classifications of the queries as safe or dangerous. For proper classification rewards have been offered to the agent, thus refining its policy so that it is accurate maximally. Training on known SQL injections datasets, some of these methods include Q-Learning and deep RL models.

Reinforcement learning has limitations such as high training steps, high sample size, and being environment-specific, making it less efficient and not transferable to other models (Lyu et al., 2022).

2.6. Existing Work on ML for SQLIA Detection

In recent years, researchers have increasingly turned to machine learning (ML) to tackle the challenge of detecting SQL injection attacks (SQLIA), largely because traditional rule-based and signature-based systems often fall short when dealing with new or cleverly disguised attacks.

Early approaches used algorithms like Support Vector Machines (SVM) and Decision Trees, which were trained to differentiate between normal and malicious SQL queries based on specific features extracted from the query strings. These models were fairly

effective, but they depended heavily on manual feature engineering and struggled with detecting unfamiliar attack patterns.

Later studies explored a wider range of ML techniques, including Naïve Bayes, Random Forest, and Logistic Regression. These models generally performed better and were easier to scale, especially when used alongside structured preprocessing methods like tokenization or parsing of SQL statements. For example, Sonali Mishra (2019) applied both Naïve Bayes and Gradient Boosting classifiers to detect SQL injection, highlighting that ensemble methods like boosting could improve accuracy compared to simpler models.

More recently, deep learning has started to gain traction in this area. One study proposed a deep neural network (DNN) that could outperform classical ML models in spotting SQL injection attempts, especially when the attack payloads were more complex or disguised using uncommon syntax (R et al., 2021). Some researchers have also experimented with Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) models, which can learn the sequence of characters in a query. These models are particularly useful for understanding context, such as how different parts of a query relate to each other over time.

While all of these models show promise, many rely on curated features or simplified data, and some still struggle to detect more subtle or obfuscated injection techniques.

2.7. Use of Convolutional Neural Networks (CNN) to Prevent SQL Injection Attacks

Before we dive into how Convolutional Neural Networks (CNNs) help detect SQL Injection Attacks (SQLIAs), it's important to first understand what CNNs are and how they work. CNNs are a type of deep learning model that excel at automatically learning patterns from data. They're most famously used in image recognition, but they've also proven to be quite effective for analyzing text (Abdulhamza & Al-Janabi, 2022).

At their core, CNNs are built from layers that each play a specific role. The first key layer is the convolutional layer, which uses small filters (or kernels) that scan across the input data. As they slide over the input, they perform calculations that help identify patterns—like edges in images or frequent word combinations in text (Falor et al., 2022).

Once these patterns are found, pooling layers help simplify the data by keeping the most important parts and discarding the rest, which makes the model more efficient and focused. Toward the end of the network, fully connected layers take all the learned patterns and use them to make a final decision or prediction (Falor et al., 2022).

The software tools, frameworks, or programming languages used to demonstrate the CNN are; TensorFlow, Keras, Scikit-learn and specific databases like MySQL. Additionally, Python is used as the primary programming language, with Jupyter Notebook as the development environment,

2.8. Related Work

In recent years, Convolutional Neural Networks (CNNs) have gained significant traction in the domain of SQL injection detection due to their ability to automatically extract features from raw data. Several notable studies have successfully applied CNNs to this problem. However, while CNN itself is not novel in this context, the unique positioning of this research lies in its scope, execution, and ethical framing.

For instance, Shahbaz et al. (2024) presented a robust CNN-based model trained on over 109,000 queries, achieving high precision and recall. Their work primarily focused on performance metrics and architectural design without an in-depth exploration of the model's ethical implications, accessibility, or deployment feasibility in resource-limited environments. Abdulhamza and Al-Janabi (2022) took a different route by leveraging a 2D-CNN architecture with word embeddings derived from the Skip-Gram model. Their approach treated SQL queries as image-like matrices, which enhanced contextual learning but introduced substantial computational overhead. While effective, such techniques may not be practical in settings where computational resources are constrained

More innovatively, Nguyen et al. (2025) proposed a graph-based approach using Graph Convolutional Networks (GCNs). Their work stands out by converting SQL queries into structured graph representations, enabling flexible input handling and improving the model's ability to understand relational data. Although powerful, this method introduces complexity and computational demands that may not be suitable for simpler deployment scenarios.

2.9. Research Gaps

Even with the progress made so far, there are still some noticeable gaps in current SQLIA detection research.

Firstly, many existing methods depend on manual feature extraction of things like keyword spotting or regular expressions to feed the machine learning models. This works to a point, but it doesn't always capture the deeper, structural patterns in SQL queries that could signal an attack, especially when attackers use tricks like adding harmless symbols or spacing to bypass detection (R et al., 2021).

Another challenge is generalization. Some models perform very well on the datasets they're trained on, but struggle with real-world data. That's often because training sets aren't diverse enough, and models end up overfitting; basically, learning the training data too well without being able to adapt to new, unseen inputs (Mishra, 2019).

There's also the issue of class imbalance. In most datasets, the number of benign queries far outweighs the number of SQL injection attempts. This can skew the model's learning, making it less sensitive to actual threats. While techniques like resampling or using ensemble models help, they don't fully solve the problem especially for attacks that don't follow known patterns (Krishnan et al., 2021).

One area that's still underexplored is the use of Convolutional Neural Networks (CNNs). While CNNs are more commonly known for image processing, they've shown strong potential for text classification tasks too, including detecting SQL injections. CNNs are good at spotting patterns in small windows of data, which makes them ideal for identifying fragments of SQL code that could be malicious. Unlike traditional ML methods, CNNs don't need you to define features manually, they learn them automatically during training.

Another key strength of CNNs is their resilience to obfuscation. Because they focus on local patterns, they can detect suspicious sequences even when attackers use unusual formats, symbols, or whitespace to try and hide their intentions.

Finally, early research shows that CNN-based models can match or even exceed the accuracy of models like logistic regression or SVM especially when combined with good preprocessing like tokenization and embedding layers (Krishnan et al., 2021). So, while

traditional ML models have laid a strong foundation, CNNs offer a powerful, scalable next step toward building smarter, more adaptable SQL injection detection systems.

In contrast, this research prioritizes a balance between performance, interpretability, and inclusivity. The model adopts a standard CNN architecture that is straightforward to implement and effective in capturing syntactic patterns within SQL queries. Moreover, this study incorporates stratified sampling, detailed data inclusion and exclusion criteria, and rigorous preprocessing to ensure data integrity and generalizability.

What sets this work apart is its human-centred approach. Ethical considerations are integrated throughout the methodology, from data privacy to open-source dissemination. The outcomes are not only intended for academic contribution but are also designed to support communities with limited access to advanced cybersecurity tools. This broader impact framework aims to ensure that the benefits of machine learning in security extend beyond academic and corporate boundaries.

2.10. Conceptual Framework

Figure 2-7 illustrates the conceptual framework, where detection of SQL injections is done through the use of a CNN. This involves several key stages that guide the process from data preparation to the final stage of model evaluation. The process begins with the organization and formatting of the dataset to ensure uniformity. The dataset is then loaded and split into training (70%), validation (15%), and testing (15%) sets in the working environment into training, validation and test sets for model training and evaluation allowing for robust evaluation and reduces the risk of overfitting. The sample size (consisting of an average of 1,000 SQL queries (both benign and malicious queries)) was chosen as the CNN model requires a large enough sample size to adequately train the model and provide significant results and ensure an equal representation of different types of SQL injection attacks, such as Union-based, Error-based, and Blind SQL injections. This approach helps to balance the dataset and improve model performance. At the text preprocessing phase, the data is cleaned and transformed to enhance its suitability for the CNN. Once finalized, model compilation and training is done to define the CNN architecture and tuning of the parameters. Finally, model evaluation is used to assess the performance of the trained CNN.

This research does not explicitly define the criteria for including or excluding SQL queries in the dataset, which is crucial to ensuring a balanced and representative dataset covering various SQL injection attack techniques. Inclusion criteria should involve real-world attack queries, diverse attack techniques, and different SQL dialects such as MySQL. Conversely, exclusion criteria should filter out duplicate queries, non-relevant database queries, and overly simplistic examples.

These criteria are crucial for building a reliable dataset. Including diverse attack techniques helps prevent the model from being biased toward just one type of SQL injection, while removing duplicates ensures the model doesn't just memorize patterns but actually learns to detect threats effectively. Supporting multiple SQL dialects also makes the model more adaptable to different database environments.

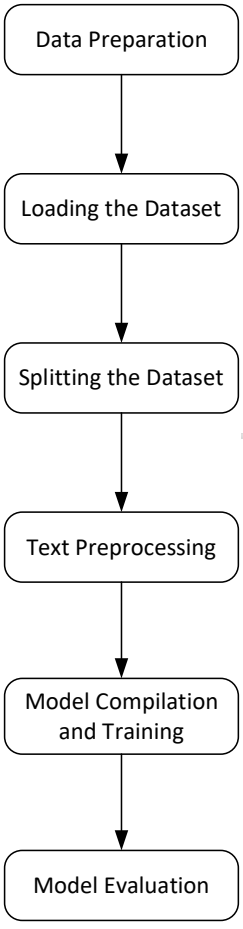


Figure 2.7: Conceptual Framework

Chapter Three: Research Methodology

3.1. Introduction

This chapter presents the methodology used in this research. It discusses how each research question was addressed, the type of data used, and the analysis approach. It then highlights the approach takes to design, implement and test the proof-of-concept SQL injection detection system using CNN. Finally, the ethical considerations are listed.

Below are the research questions addressed:

Research Question 1: *What techniques are used in SQL injection?*

This research question has been answered in the literature review presented in the chapter 2. The chapter depicts the many techniques that may be used to launch SQL Injection Attacks (SQLIA).

Research Question 2: *How do current tools used in detection of SQL injection attacks operate?*

This research question has been answered in the literature review presented in the chapter 2. These include Web Application Firewalls, stored procedures and input validation and whitelisting.

Research Question 3: *How can an algorithm for SQL Injection detection be designed and implemented?*

This research question has been addressed by the system methodology which is presented later in the chapter. The system methodology shows the methods used by the researcher to meet the research objectives.

Research Question 4: *How does the SQL injection detection algorithm perform?*

This research question is intended to demonstrate the validity of the researcher's dissertation. This was addressed by a set of system tests mentioned later in this chapter.

3.2. Research Design

The research involves the use of systematic and methodological approaches to developing an effective model capable of identifying SQL injection attacks. This study involves the collection of a comprehensive dataset comprising of both benign and malicious SQL queries to train and validate the machine learning model. The chosen machine learning model will be trained on the dataset to recognize patterns of SQL injection attacks. Feature extraction and selection will be applied to enhance the model's capability to adapt to various attack scenarios. Cross validation will also be employed to assess the model's performance and ensure its robustness.

A series of real-world situations and attack simulations will be used to assess the accuracy, precision, recall, and overall effectiveness of the proposed SQL injection detection system. The study will also investigate the model's scalability, taking into account variables like data volume and complexity, in order to evaluate its suitability for use in a variety of contexts.

3.3. Software Methodology

The chosen methodology for this dissertation is Kanban. Kanban is an agile methodology for managing agile and DevOps software development. It requires real-time capacity communication as well as complete work openness. Work items are visibly represented on a kanban board, allowing team members to monitor the status of each piece of work at all times (Atlassian, n.d.). Kanban was initially established in the late 1940s by a Toyota industrial engineer named Taiichi Ohno (SUGIMORI et al., 1977).

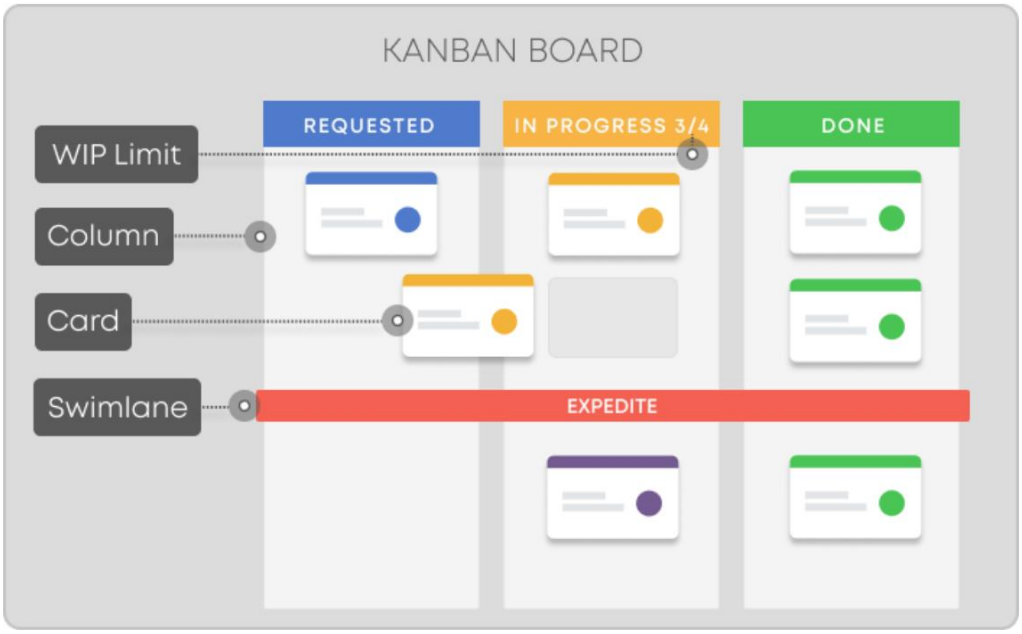


Figure 3.1: Kanban board

This methodology is split into 5 steps:

1. Visualization of workflow
2. Limit the amount of WIP
3. Manage and measure your workflow
4. Switch to explicit policies
5. Implement Feedback Loops
6. Use the scientific method for optimization

C.3.1. Visualization of workflow

At this stage, the researcher mapped out the software development workflow stages on a virtual Kanban Board using separate columns. The researcher tracked each stage of the development process, including research and requirement gathering, data collection and preprocessing, model design and implementation, training and testing, and model tuning, ensuring a comprehensive understanding of CNN and SQL injection patterns. A feasibility study was conducted to confirm that the requirements were clear and achievable. The gathered requirements were then assessed for their accuracy.

C.3.2. Limit the amount of WIP

To preserve efficiency, the researcher limited the number of tasks executed simultaneously. For example, while collecting SQL injection data, the researcher did not begin creating the CNN model until the dataset had been thoroughly pre-processed. Limiting work-in-progress (WIP) ensured that resources were not overstretched and that each phase received focused attention before moving forward (Ahmad et al., 2013).

This approach was especially useful during the model training and testing phases, when hyperparameter tuning and optimization required meticulous attention to detail. Limiting concurrent tasks during this stage helped minimize computing resource overload and kept the project on track.

C.3.3. Manage and measure your workflow

At this stage, the metrics tracked included cycle time (i.e., the length of time it took for a task to move from one stage to another), the number of tasks completed within a certain period, and work-in-progress limits. Through workflow measurement, the researcher was able to identify slowdowns; for example, when training time was too long or when debugging on CNN layers took an extended period. These metrics allowed the researcher to make informed decisions regarding resource investment for these stages, determining whether it was worthwhile to refine them further such as adjusting the model's architecture for improved detection accuracy (David J. Anderson, n.d.).

C.3.4. Switch to explicit policies

With the use of explicit policies, the researcher was able to formalize the rules guiding the workflow. For example, the researcher put criteria in place to ensure that a task could only move from "data preprocessing" to "model design" once specific conditions were met. The latter included tasks such as verifying that the dataset was balanced and that SQL injection patterns were well represented, among others. Similarly, explicit policies for transitioning from "training" to "evaluation" included meeting a minimum acceptable accuracy threshold on the validation dataset (Ahmad et al., 2013).

C.3.5. Implement Feedback Loops

This approach included the use of feedback loops to constantly monitor the status and performance of the research. Feedback was gathered after every stage of testing the CNN, particularly regarding its accuracy and its ability to detect SQL injections. For example, during the review of frequent false positives and negatives observed in testing, the researcher promptly readjusted the model and dataset to achieve better accuracy. These feedback loops ensured continuous improvement and provided valuable opportunities to adapt quickly to new insights (Ahmad et al., 2013).

C.3.6. Use the scientific method for optimization

Following the scientific method, each improvement iteration was treated as an experiment. By fine-tuning the CNN model, hypotheses were drawn regarding which changes in activation functions or learning rates would result in improvements or deterioration in detection performance. For example, the researcher hypothesized that increasing the depth of the CNN would enhance its feature extraction capabilities for SQL injection patterns. These hypotheses were then verified through controlled experiments, allowing the researcher to iteratively collect data to optimize the model. This structured approach aligned with the emphasis Kanban places on continuous improvement based on empirical evidence (Ahmad et al., 2013).

By applying these principles of Kanban, the researcher was able to efficiently manage the complexities involved in developing a CNN-based SQL injection detection algorithm, thereby ensuring that the project remained balanced, iterative, and data-driven.

3.4. System Analysis

The system analysis of this research involved an examination of the vulnerabilities and characteristics of SQLIAs outlined in the problem statement, while also exploring how CNNs could be used to handle these attacks. This analysis began by classifying SQLIAs, understanding their exploitation methods, and identifying deficiencies in conventional countermeasures such as input validation and Web Application Firewalls. The analysis of requirements therefore focused on identifying the capabilities that the model needed, such as handling obfuscation techniques, dynamic payloads, and encoding schemes employed

by attackers. It also considered the datasets required for training and testing, ensuring they included both known and novel SQLIA patterns, while performance metrics—accuracy, false positive rates, and detection speed—were clearly defined to evaluate the CNN-based solution.

3.5. System Implementation

The system implementation followed a phased approach to ensure a structured and effective development of the CNN-based SQL injection detection model. It began with the definition of the system specifications and the CNN architecture. This was then followed by the compilation and preparation of labelled datasets of SQL queries. The CNN was subsequently trained to identify and classify malicious and benign queries with performance optimization.

The model was developed using Python (version 3.10) in a Google Colab environment. Libraries included TensorFlow, Keras, Pandas, NumPy, and Scikit-learn. Google Colab provided a cloud-based platform with GPU acceleration, allowing for efficient training and evaluation of the CNN model.

The following steps were followed in the development of the CNN model. Data preprocessing involved tokenizing SQL queries and converting them into sequences. Vectorization was performed using one-hot encoding to convert tokens into numerical representations. The model architecture consisted of an embedding layer, convolutional layers, max-pooling layers, a flattening layer, and dense layers with ReLU and Softmax activations. During training, the model was compiled using categorical cross-entropy loss and the Adam optimizer and was trained for 10 epochs with a batch size of 32. Finally, the model's accuracy and other metrics were computed using test data.

3.6. System Testing and Validation

The model was tested on multiple samples of input datasets representing both legitimate and harmful SQL injection attacks, which were fed into the CNN model to determine its ability to detect SQL injection attacks and its capacity to learn after several attempts. This involved analysing the model’s precision, recall, and F1-score metrics using cross-validation techniques to confirm its robustness and efficacy across multiple attack vectors.

Furthermore, testing was conducted against various SQL injection attack techniques to evaluate the model's resilience and reliability in real-world deployment settings, ultimately aiming to protect database systems from potential security breaches.

3.7. Ethical Considerations

As the research focuses on cybersecurity and does not directly involve human subjects or sensitive data related to specific individuals or communities, the ethical considerations related to welfare, rights, beliefs, perceptions, customs, and cultural heritage are minimal. The research does address data privacy concerns and ensure that any data used for training and testing is anonymized and handled securely.

As the only researcher involved in the project, I took responsibility for tracking and documenting the progress of the research. While there were no external participants, I maintained a detailed record of key milestones, methods, and outcomes to ensure the research was on track and could be referenced in the future. Additionally, the research remained mindful of broader ethical principles, ensuring the research contributes positively to the field and respects privacy norms. By improving the detection of SQL injection attacks, this research could enhance the security infrastructure of organizations globally, indirectly supporting the protection of user data and privacy in a way that aligns with ethical values.

Upon completion of the project, I compiled a comprehensive final report that summarized the methodology, results, and conclusions. This report, along with any relevant code or documentation, was stored for future use, ensuring that the outcomes of the research can be shared and reviewed when needed, either for personal reference or in the context of broader dissemination.

When collecting and analysing sensitive information, the researcher will ensure that data is used responsibly and ethically, that privacy standards are followed, and consent is obtained. Furthermore, possible biases in the dataset used to train machine learning models must be mitigated in order to avoid unfair outcomes or discriminatory behaviours.

3.8. Dissemination and Utilization of Results

The findings from this research will be disseminated through various academic and professional channels, including academic publications, technical reports, open-source implementation, and industry collaboration. The results will be submitted to peer-reviewed journals and conference proceedings related to cybersecurity, artificial intelligence, and web security. A comprehensive technical report will be published, providing insights into the implementation and performance of the proposed ML-based SQL injection detection system. Additionally, the SQL injection detection algorithm will be made available as an open-source tool on platforms like GitHub to facilitate further research and real-world application. Lastly, the results will be shared with organizations and cybersecurity firms to enable the integration of ML-based SQL injection detection mechanisms into their web security infrastructure.

The results of this research will enhance cybersecurity practices across various domains by improving web application security, policy, and compliance. The proposed system can be integrated into web application firewalls (WAFs) to enhance real-time detection and prevention of SQL injection attacks. Additionally, insights from this research can inform best practices for securing web applications and influence security policies and compliance standards.

Chapter 4: System Design and Architecture

4.1. Introduction

This chapter focuses on the steps taken by the researcher in designing of the system. These steps involve requirement analysis, identification of the functional and non-functional requirements. The chapter also describes the architecture design for SQL injection detection using Machine Learning.

4.2. Requirement Analysis

At this stage the researcher is able to identify the resources and information that will aid in meeting the objectives set, which is; SQL Injection Detection using Machine Learning. In order to attain this, each requirement is broken down and flow of events/transaction performed assisting in the decision-making process is reviewed on their necessity (i.e. if the requirement is needed or not). This stage allows for the creating of the development framework required for this research paper. These system requirements can be categorized as functional and non-functional requirements.

4.2.1. Functional requirements

These describe the system's functionality. They provide the desired functionality that the user expects from the system. The functional requirements are as follows:

i. The system should be able to intercept incoming SQL queries submitted by users.
ii. The system should be able to apply Machine Learning Model for classification for submitted SQL queries.
iii. The system should be able to allow benign queries and reject malicious queries
iv. The system should have the capability to store intercepted queries along with their associated features and classification results in a database.

4.2.2. Non-Functional Requirements

The non-functional requirements are as follows:

i. The system should be able to process incoming queries with minimal latency to ensure real-time detection and response.

ii. The system should be easily scalable allowing for increasing loads and growing datasets.

iii. The system should be highly reliable, ensuring continuous operation.

iv. The system should be easy to maintain allowing for easy updates and enhancements.

4.3. The System Diagram

The design comprises the system flow chart, use case diagram and the system sequence Diagram. These are presented in the below subtopics.

4.3.1 Use Case Diagram

Figure 4.3.1 illustrates a use case diagram for a machine learning-based SQL injection detection system. It represents a user's engagement with the system, including the process flow and major components involved. The user initiates the process by running a script, which then presents the results. The key steps are to load the dataset, preprocess it, partition it, and convert queries to numerical representations. These procedures are followed by creating a Convolutional Neural Network (CNN) model, training it, assessing it, and lastly producing a classification report. The figure stresses the essential links between the processes and the sequential flow required to detect SQL injections successfully.

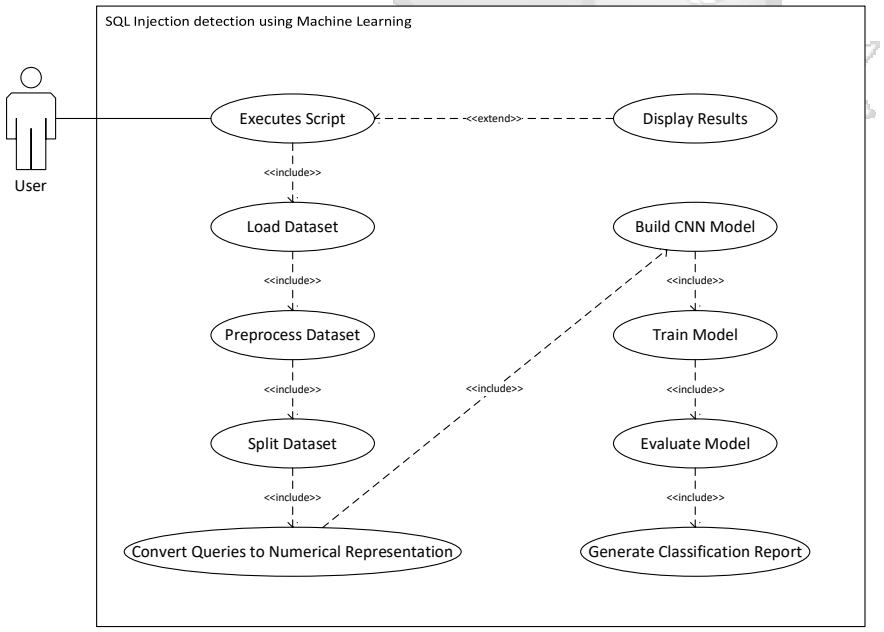


Figure 4.1: Use Case Diagram

4.3.2. Sequence Diagram

Figure 4.3.2 is a sequence diagram that depicts the process of detecting SQL injections using machine learning. The user initiates the process by executing a Python script, which triggers the DataLoader to load a dataset. The dataset is preprocessed, split into training and testing sets, and one-hot encoding is applied. A CNN model is built and trained using the training dataset, evaluated on the testing set, and a classification report is generated to summarize the model's effectiveness in detecting SQL injections. This systematic approach ensures a structured approach to machine learning-based SQL injection detection.

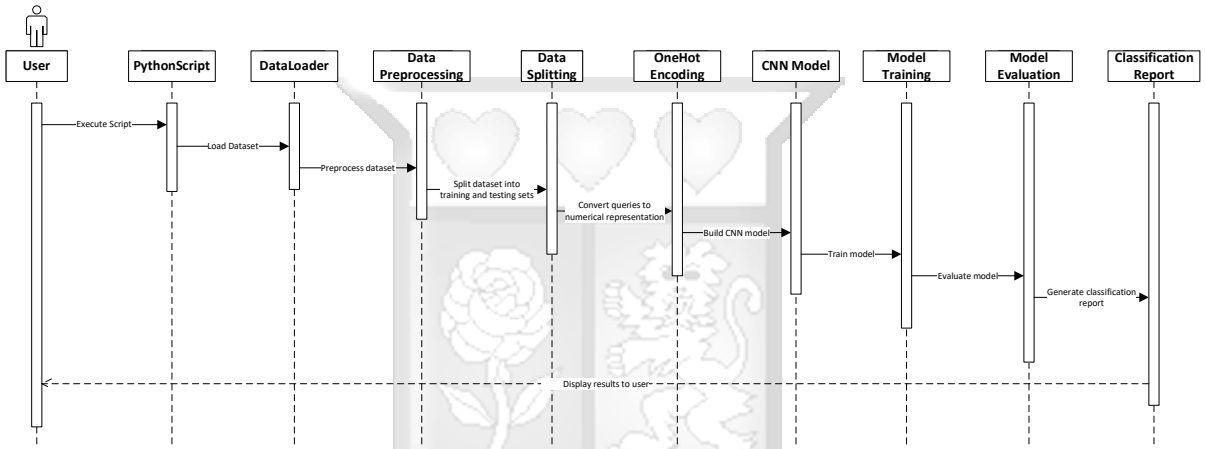


Figure 4.2: Sequence Diagram

4.3.3.Flow Chart Diagram

Figure 4.3.3 is a flowchart that outlines the process of detecting SQL injections using machine learning. It starts with a user executing a script, loading a dataset, preprocessing it, splitting it into training and testing sets, converting queries into numerical representations, building a Convolutional Neural Network (CNN) model, training it with the training data, and evaluating it with the testing data. A classification report summarizes the model's performance, and the results are displayed to the user.

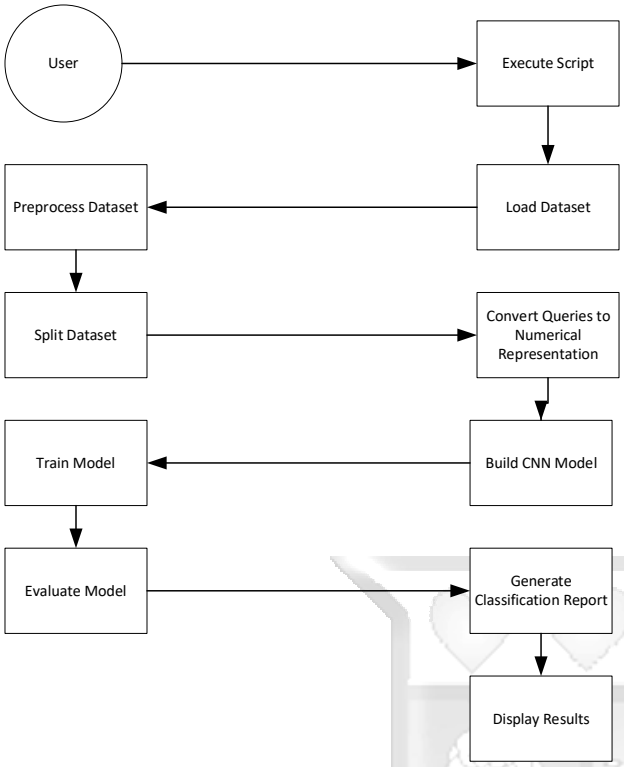


Figure 4.3: Flow Chart Diagram

Chapter 5: System Implementation and Testing

5.1 Introduction

Testing and system implementation are essential stages in the creation of every machine learning model. In this chapter, we will discuss the implementation and testing process of a Convolutional Neural Network (CNN) model for detecting malicious SQL injection (SQLi) queries. Python libraries like NumPy, Pandas, Scikit-learn, and TensorFlow are used in the provided code.

5.2 System Implementation

5.2.1 Dataset Loading and Preprocessing

The initial stage in system implementation is to load and preprocess the dataset. In the provided code, we load a dataset named'sqli.csv', which comprises SQL queries labeled as malicious or non-malicious. The dataset is loaded into a Pandas DataFrame, after which the input features (SQL queries) and output labels are extracted.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report

# Load the dataset
dataset = pd.read_csv('sqli.csv', encoding='utf-16')

# Preprocess the dataset
X = dataset['Sentence'].values
y = dataset['Label'].values
```

Figure 5.1: Dataset Loading and Preprocessing

5.2.2 Data Splitting

The dataset is loaded and then split into training and testing sets using Scikit-learn's '***train_test_split***' method. This ensures that the model's performance may be evaluated using previously unseen data.

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 5.2: Data Splitting

5.2.3 Text Preprocessing

Text processing is an important step before putting data into the model. This implementation converts textual inquiries into numerical representations using a basic one-hot encoding method. Each word in the query is hashed and assigned a unique index within a predefined vocabulary size. The resulting one-hot encoded vectors indicate the presence or absence of words in the requests.

```
vocab_size = 10000
X_train_encoded = np.zeros((len(X_train), vocab_size))
X_test_encoded = np.zeros((len(X_test), vocab_size))

for i, query in enumerate(X_train):
    if isinstance(query, float):  # Check if query is a float
        query = str(query)  # Convert float to string
    words = query.split()
    for word in words:
        index = hash(word) % vocab_size
        X_train_encoded[i, index] = 1

for i, query in enumerate(X_test):
    if isinstance(query, float):  # Check if query is a float
        query = str(query)  # Convert float to string
    words = query.split()
    for word in words:
        index = hash(word) % vocab_size
        X_test_encoded[i, index] = 1

X_train_encoded = np.expand_dims(X_train_encoded, axis=2)
X_test_encoded = np.expand_dims(X_test_encoded, axis=2)

y_train_categorical = to_categorical(y_train)
y_test_categorical = to_categorical(y_test)
```

Figure 5.3: Text Preprocessing

5.2.4 Model Building

The CNN model is built using TensorFlow's Keras API, which is fundamental to the system implementation. The model design starts with a convolutional layer, then moves on to max-pooling, flattening, and dense layers. The convolutional layer learns characteristics from incoming data, whilst the dense layers do classification.

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(vocab_size, 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(2, activation='softmax'))
```

Figure 5.4: Model Building

5.2.5 Model Compilation and Training

Once the model has been built, it is compiled with the proper loss function, optimizer, and evaluation metrics using the build method. Categorical cross-entropy loss and the Adam optimizer are used in this implementation. The model is then trained on the training data using the fit technique, which specifies the number of epochs and batch size.

```
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 5.5: Model Compilation

```
# Train the model
model.fit(X_train_encoded, y_train_categorical, epochs=10, batch_size=32)
```

Figure 5.6: Model Training

5.3 System Testing

5.3.1 Model Evaluation

The model's performance in classifying SQL queries as malicious or non-malicious is evaluated using testing data not exposed during training. The main evaluation metric is accuracy, which measures the proportion of correctly identified cases among all examples in the testing set. The **'evaluate'** method calculates the model's accuracy on the testing data, indicating its ability to apply its learnings to previously unknown data. Higher accuracy indicates better performance.

```
_, accuracy = model.evaluate(X_test_encoded, y_test_categorical)
print('Accuracy: %.2f%%' % (accuracy * 100))
```

Figure 5.7: Model Evaluation

However, accuracy alone may not provide a whole picture of the model's performance, particularly in imbalanced datasets with one class greatly outnumbering the other. As a result, it is critical to explore further into other evaluation measures such as precision, recall, and F1-score, especially when dealing with binary classification jobs like SQL injection detection.

Precision and recall are crucial metrics in binary classification tasks. Precision measures the proportion of true positive predictions, indicating a low false positive rate, and recall measures the proportion of true positive predictions out of all actual malicious instances. High precision indicates a low false negative rate, indicating the model effectively captures most malicious queries. The F1-score, the harmonic mean of precision and recall, provides a balanced measure of a model's performance, considering both false positives and false negatives. These metrics help stakeholders make informed decisions about the model's deployment and potential improvements. Evaluation was conducted on the test set to determine the model's effectiveness. The following metrics were recorded:

```
Epoch 1/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 81s 733ms/step - accuracy: 0.9048 - loss: 0.2206
Epoch 2/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 84s 760ms/step - accuracy: 0.9973 - loss: 0.0160
Epoch 3/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 80s 746ms/step - accuracy: 0.9974 - loss: 0.0083
Epoch 4/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 96s 881ms/step - accuracy: 0.9989 - loss: 0.0034
Epoch 5/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 81s 774ms/step - accuracy: 0.9998 - loss: 0.0017
Epoch 6/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 82s 772ms/step - accuracy: 0.9997 - loss: 0.0017
Epoch 7/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 82s 770ms/step - accuracy: 1.0000 - loss: 7.6053e-04
Epoch 8/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 81s 773ms/step - accuracy: 0.9993 - loss: 0.0028
Epoch 9/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 82s 770ms/step - accuracy: 0.9998 - loss: 0.0010
Epoch 10/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 82s 771ms/step - accuracy: 0.9994 - loss: 0.0023
27/27 ━━━━━━━━━━━━━━━━━━━━ 5s 193ms/step - accuracy: 0.9855 - loss: 0.0552
Accuracy: 98.21%
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 151ms/step
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       588
           1       0.97      0.97      0.97       252

    accuracy                           0.98       840
   macro avg       0.98      0.98      0.98       840
weighted avg       0.98      0.98      0.98       840
```

Figure 5.8: Findings and report

The results showed an accuracy of 98.21%. For precision, the model achieved 0.99 for benign queries and 0.97 for malicious ones. Recall values were also strong, with 0.99 for benign and 0.97 for malicious inputs. The F1-score was 0.99 for benign and 0.97 for malicious queries. The macro average for precision, recall, and F1-score was 0.98 across all classes. Similarly, the weighted average for precision, recall, and F1-score was also 0.98, indicating consistent and reliable performance across both classes.

Error! Reference source not found. confirms the model's high precision (low FP) and h igh recall (low FN). The balance between these indicates robust overall performance, capable of detecting most attacks while minimizing disruption to legitimate traffic. A total of 247 malicious queries were correctly identified as malicious (True Positives), while 583 benign queries were correctly identified as benign (True Negatives). Only 5 benign queries were incorrectly classified as malicious (False Positives), and 5 malicious queries were missed and classified as benign (False Negatives).

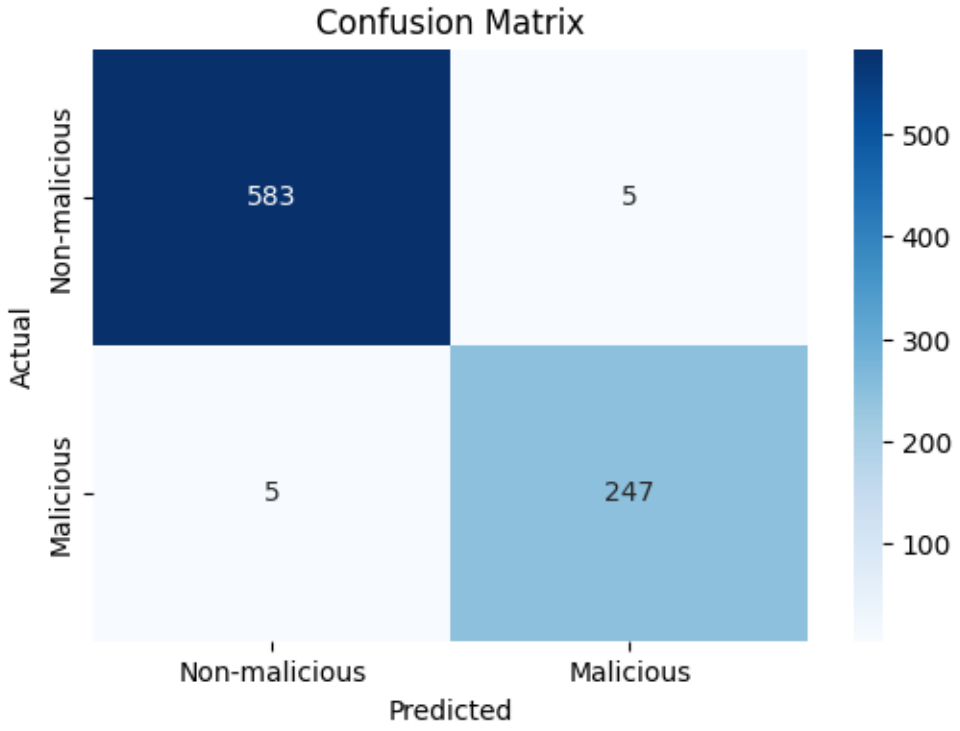


Figure 5.9: Confusion Matrix

5.3.2 Performance Analysis

The model was tested across multiple scenarios. Standard Queries: The model successfully detected all classical SQL injection attempts, including straightforward payloads using logical operators like 'OR 1=1'. Obfuscated Attacks: The model accurately detected obfuscated inputs involving encoded characters, concatenated commands, and payloads split across multiple parameters. This demonstrates the CNN's strength in learning non-linear and hidden structures within queries. Blind Attacks: Boolean-based and time-based SQL injections were accurately classified, showcasing the model's ability to recognize subtle behavioural patterns that do not involve direct data leakage.

After evaluating the model's accuracy, the provided code snippet generates a classification report using Scikit-learn's **'classification_report'** function, which includes precision, recall, and F1-score for both classes (malicious (1) and non-malicious(0)), as well as other metrics like support (the number of occurrences of each class in the testing set) and the weighted average of precision, recall, and F1-score. This comprehensive study provides a full examination of the model's performance across multiple dimensions, allowing stakeholders to accurately analyze its strengths and limitations.

```
y_pred = np.argmax(model.predict(X_test_encoded), axis=1)
report = classification_report(y_test, y_pred)
print(report)
```

Figure 5.10: Performance Analysis

CHAPTER 6: Conclusions and Recommendations

6.1. Introduction

This dissertation analyses SQL injection detection techniques, existing detection mechanisms and the development of a machine-learning based algorithm in an attempt of enhancing cybersecurity measures against SQLi attacks that pose a threat to web applications, compromising data integrity and security.

6.2. Summary of Findings

The main objective of this research was to develop a Convolutional Neural Network (CNN) model capable of detecting SQL injection attacks in web applications. The model was trained and evaluated using a balanced dataset containing both benign and malicious SQL queries.

Figure 5.8: Findings and report demonstrated that the Convolutional Neural Network (CNN) model effectively detected SQL injection patterns across a wide range of attack types, including standard queries, obfuscated inputs, and blind injection techniques, confirming its robustness and adaptability to real-world scenarios. The model achieved a high test accuracy of 98.21%, supported by a low training loss of 0.0023, indicating strong convergence during the training process. Precision and recall for malicious queries were both recorded at 0.97, while for benign queries, both metrics reached 0.99, reflecting the model's strong ability to differentiate between legitimate and malicious inputs with minimal false positives or false negatives. The confusion matrix further highlighted this reliability, showing 247 true positives, 583 true negatives, and only 5 false positives and 5 false negatives. Performance remained consistent across various testing scenarios, including encoded payloads, split-payload attacks, and semantic query variations, suggesting that the model captured deeper patterns rather than relying solely on fixed signatures. Moreover, the close alignment between validation and test accuracy throughout training confirmed that the model generalized well and was not overfitted to the training data.

Overall, these findings validate the effectiveness of CNN-based models in detecting SQL injection attacks and support their application in intelligent, automated security systems for web environments.

6.3. Conclusion

This research aimed first to analyze the techniques used to launch SQL injection (SQLi) attacks. The study identified a variety of methods attackers employ to exploit vulnerabilities in SQL queries, such as altering input parameters to manipulate database operations. Understanding these common patterns and tactics was essential in revealing how attackers gain unauthorized access, extract sensitive data, or disrupt system functionality. These insights provided a critical foundation for designing more effective and adaptive detection mechanisms. The second objective involved evaluating existing SQLi detection systems. By testing these systems under diverse attack scenarios and environmental conditions, the research uncovered key limitations, including their susceptibility to sophisticated attacks and the occurrence of false positives and negatives. These findings informed the need for a more robust and intelligent detection approach.

To address this, the third objective focused on designing and implementing a machine learning-based detection algorithm using Convolutional Neural Networks (CNNs). CNNs were chosen due to their proven effectiveness in recognizing spatial patterns, making them particularly suitable for analyzing structured query inputs. The model was carefully architected to balance accuracy with computational efficiency. The final objective was to test and validate the algorithm using a comprehensive dataset comprising both benign and malicious SQL queries. The evaluation, based on metrics such as accuracy, precision, recall, and F1-score, demonstrated high performance, confirming the model's practical applicability. The results highlighted the algorithm's strength in accurately detecting SQLi attacks while also identifying areas for further refinement, such as improving the detection of edge cases.

In conclusion, this dissertation represents a significant step forward in improving cybersecurity defences against SQL injection attacks. It has offered significant insights into the numerous strategies and methods used by hostile actors to exploit SQL vulnerabilities through a thorough investigation of attack approaches. The researcher was able to evaluate

current detection mechanisms critically and pinpoint both their advantages and disadvantages by grasping this technique. The creation and application of sophisticated machine learning methods intended especially for SQL injection detection were made possible by this examination. The resultant machine learning-based detection solution exhibits a high level of accuracy and efficiency in recognizing and mitigating SQL injection threats by utilizing the capabilities of Convolutional Neural Networks (CNNs). A strong basis for building a more secure and resilient digital environment has been established by this convergence of cutting-edge technology and analytical insights.

The importance of ongoing research, innovation, and collaboration in cybersecurity cannot be overstated. As cyber threats become more sophisticated, it's crucial to continuously refine detection mechanisms. Future research should focus on machine learning techniques, algorithm robustness, and real-time detection systems. Fostering a collaborative environment where organizations share threat intelligence and best practices is essential for building a united front against cybersecurity risks. This commitment can better safeguard digital infrastructures and protect against evolving cyber threats.

6.4. Recommendations

This dissertation suggests several recommendations to improve SQL injection detection and cybersecurity measures. These include integrating advanced machine learning algorithms like the CNN-based model into cybersecurity frameworks, implementing continuous monitoring and updates, conducting comprehensive security audits, enhancing text preprocessing to improve input data quality, and encouraging collaborative threat intelligence among organizations. These measures aim to enhance detection accuracy, maintain high detection accuracy, and mitigate potential vulnerabilities in web applications. Additionally, incorporating sophisticated text preprocessing techniques can improve detection accuracy.

6.5. Future Research Work

This dissertation has made significant contributions to SQL injection detection, but there are areas for future research and development. These include exploring different types of database systems, including NoSQL and alternative machine learning techniques such as recurrent neural networks (RNNs), or reinforcement learning, improving adversarial

robustness, developing real-time detection systems capable of processing SQL queries at scale, integrating the algorithm with existing security frameworks, conducting large-scale validation studies using diverse datasets, and developing user-friendly tools and interfaces.

These areas aim to enhance the security and resilience of web applications against evolving threats. This dissertation also focuses on integrating the machine learning-based detection algorithm with existing security information and event management systems and intrusion detection systems to provide comprehensive threat coverage. By addressing these future research directions, the cybersecurity community can continue to advance the state-of-the-art in SQL injection detection and enhance the security and resilience of web applications.

References

Abdulhamza, F. R., & Al-Janabi, R. J. S. (2022). SQL Injection Detection Using 2D-Convolutional Neural Networks (2D-CNN). *2022 International Conference on Data Science and Intelligent Computing (ICDSIC)*, 212–217. https://doi.org/10.1109/ICDSIC56987.2022.10075777

Ahmad, M. O., Markkula, J., & Oivo, M. (2013). *Kanban in Software Development: A Systematic Literature Review* (p. 16). https://doi.org/10.1109/SEAA.2013.28

Ali, S. (2013). Intrusion Detection Using the WEKA Machine Learning Tool.

Arasu, A., Eguro, K., Kaushik, R., & Ramamurthy, R. (2013). Querying encrypted data. *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 1262-1263. https://doi.org/10.1145/2588555.2588893.

Arif, A. A. S., Purwoko, R., Qomariasih, N., & Setiawan, H. (2022). Analysis of SQL Injection Attack Detection and Prevention on MySQL Database Using Input Categorization and Input Verifier. 2022 IEEE 8th Information Technology International Seminar (ITIS), 190–194. https://doi.org/10.1109/ITIS57155.2022.10010201

Atefeh Tajpour, Mohammad Zaman Heydari, Maslin Masrom, & Suhaimi Ibrahim. (2010). SQL injection detection and prevention tools assessment. *2010 3rd International Conference on Computer Science and Information Technology*, 518–522. https://doi.org/10.1109/ICCSIT.2010.5563777

Azman, M. A., Marhusin, M. F., & Sulaiman, R. (2021). Machine Learning-Based Technique to Detect SQL Injection Attack. *Journal of Computer Science*, *17*(3), 296–303. https://doi.org/10.3844/jcssp.2021.296.303

Betarte, G., Pardo, Á., & Martínez, R. (2018). Web Application Attacks Detection Using Machine Learning Techniques. 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 1065–1072. https://doi.org/10.1109/ICMLA.2018.00174

Bieniusa, A., Zeller, P., & Barke, S. (2018). Collaborative Work Management with a Highly-Available Kanban Board. , 59-72. https://doi.org/10.1007/978-3-319-98047-8_4.

Booch, G. (1994). *Object-oriented analysis and design with applications* (2nd ed). Benjamin/Cummings Pub. Co.

Brimhall, J., Gennick, J., & Sheffield, W. (2015). Chapter 17: Stored Procedures. , 417-436. https://doi.org/10.1007/978-1-4842-0061-2_17.

Brindavathi, B., Karrothu, A., & Anilkumar, C. (2023). An Analysis of AI-based SQL Injection (SQLi) Attack Detection. 2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), 31–35. https://doi.org/10.1109/ICAISS58487.2023.10250505

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

David J. Anderson. (n.d.). *Kanban—David J. Anderson, PDF*. PocketBook4you. Retrieved May 16, 2025, from https://pocketbook4you.com/en/read/kanban-successful-evolutionary-change#google_vignette

Digital 2020: Kenya. (n.d.). DataReportal – Global Digital Insights. Retrieved April 11, 2021, from https://datareportal.com/reports/digital-2020-kenya

Dike, H. U., Zhou, Y., Deveerasetty, K. K., & Wu, Q. (2018). Unsupervised Learning Based On Artificial Neural Network: A Review. 2018 IEEE International Conference on Cyborg and Bionic Systems (CBS), 322–327. https://doi.org/10.1109/CBS.2018.8612259

Falor, A., Hirani, M., Vedant, H., Mehta, P., & Krishnan, D. (2022). *A Deep Learning Approach for Detection of SQL Injection Attacks Using Convolutional Neural Networks* (pp. 293–304). https://doi.org/10.1007/978-981-16-6285-0_24

Ghosh, A., Sufian, A., Sultana, F., Chakrabarti, A., & De, D. (2020). Fundamental Concepts of Convolutional Neural Network (pp. 519–567). https://doi.org/10.1007/978-3-030-32644-9_36

Hosam, E., Hosny, H., Ashraf, W., & Kaseb, A. S. (2021). SQL Injection Detection Using Machine Learning Techniques. 2021 8th International Conference on Soft Computing & Machine Intelligence (ISCMI), 15–20. https://doi.org/10.1109/ISCMI53840.2021.9654820

Jang, Y.-S. (2020). Detection of SQL Injection Vulnerability in Embedded SQL. *IEICE Transactions on Information and Systems*, E103.D(5), 1173–1176.

Johny, J., Nordin, W., Lahapi, N. M., & Yu Beng, L. (2021). *SQL Injection Prevention in Web Application: A Review* (pp. 568–585). https://doi.org/10.1007/978-981-16-8059-5_35

Kabari, L., & Baah, B. (2012). Data Validation System For A Relational Database. *International Journal of Advanced Research in Computer Science*, 3, 56-61. https://doi.org/10.26483/IJARCS.V3I5.1322.

Khan, J., Farooqui, S., & Siddiqui, A. (2023). A Survey on SQL Injection Attacks Types & their Prevention Techniques. *Journal of Independent Studies and Research Computing*. https://doi.org/10.31645/jisrc.23.21.2.1. What is Machine Learning? | IBM. (n.d.). Retrieved March 24, 2023, from https://www.ibm.com/topics/machine-learning

Kigen, P. M., Muchai, C., Kimani, K., Mwangi, M., Shiyayo, B., Ndegwa, D., Kaimba, B., Mueni, F., & Shitanda, S. (2015). *Kenya Cyber Security Report 2015*. Serianu Ltd.

Kigen, P. M., Muchai, C., Kimani, K., Mwangi, M., Siyayo, B., Ndegwa, D., . . . SHitanda, S. (2015). *Achieving Enterprise Cyber Resilience Through Situational Awareness.* Nairobi: Serianu. Retrieved from http://serianu.com/downloads/KenyaCyberSecurityReport2015.pdf

Krishnan, S. S. A., Sabu, A. N., Sajan, P. P., & Sreedeep, A. L. (2021). *SQL Injection Detection Using Machine Learning*. *11*(3).

Kumar, V. (2023, August 4). Web Application Firewalls Pros and Cons. *Haltdos - Enterprise Application Security & Delivery Platform | Haltdos*. https://www.haltdos.com/waf/web-application-firewalls-pros-and-cons/

Lyu, L., Shen, Y., & Zhang, S. (2022). The Advance of Reinforcement Learning and Deep Reinforcement Learning. 2022 IEEE International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA), 644–648. https://doi.org/10.1109/EEBDA53927.2022.9744760

Marill, K. (2004). Advanced statistics: linear regression, part II: multiple linear regression.. *Academic emergency medicine : official journal of the Society for Academic Emergency Medicine*, 11 1, 94-102 . https://doi.org/10.1197/J.AEM.2003.09.006.

Martin, J. (n.d.). What is Rapid Application Development (RAD). 34.

Mishra, S. (2019). *SQL Injection Detection Using Machine Learning* [Master of Science, San Jose State University]. https://doi.org/10.31979/etd.j5dj-ngvb

Nguyen, D.-C., Ha, M.-H., Do, M.-T., & Chen, O. T.-C. (2025). Towards lightweight model using non-local-based graph convolution neural network for SQL injection detection. *Egyptian Informatics Journal*, *30*, 100684. https://doi.org/10.1016/j.eij.2025.100684

Omar, F., Ahmed, D., Elnakib, O., Ahmed, M., Farhan, N., Hindy, H., Abdel-Hamid, M., & Badawi, Y. (2023). Towards a User-Friendly Web Application Firewall. *2023 Eleventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, 483-488. https://doi.org/10.1109/ICICIS58388.2023.10391117.

Out-of-Band SQL Injection | Learn AppSec. (2022, March 28). Invicti. https://www.invicti.com/learn/out-of-band-sql-injection-oob-sqli/

OWASP Top Ten | OWASP Foundation. (n.d.). Retrieved March 3, 2023, from https://owasp.org/www-dissertation-top-ten/

OWASP. (2017). *OWASP Top 10 2017—The Ten Most Critical Web Application Security Risks* (OWASP Top 10 2017, p. 25). The OWASP Foundation. https://owasp.org/www-dissertation-top-ten/2017/

Pajila, P. J. B., Sheena, B. G., Gayathri, A., Aswini, J., Nalini, M., & R, S. S. (2023). A Comprehensive Survey on Naive Bayes Algorithm: Advantages, Limitations and Applications. 2023 4th International Conference on Smart Electronics and Communication (ICOSEC), 1228–1234. https://doi.org/10.1109/ICOSEC58147.2023.10276274

R, J. K., Balaji B, S., Pandey, N., Beriwal, P., & Amarajan, A. (2021). An Efficient SQL Injection Detection System Using Deep Learning. *2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, 442–445. https://doi.org/10.1109/ICCIKE51210.2021.9410674

Rai, A., Miraz, MD. M. I., Das, D., Kaur, H., & Swati. (2021). SQL Injection: Classification and Prevention. 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), 367–372. https://doi.org/10.1109/ICIEM51511.2021.9445347

Shahbaz, M., Mumtaz, G., Zubair, S., & Rehman, M. (2024). Evaluating CNN Effectiveness in SQL Injection Attack Detection. *Journal of Computing & Biomedical Informatics, 7*(02), Article 02. https://doi.org/10.56979/702/2024

Shaveta. (2023). A review on machine learning. *International Journal of Science and Research Archive*, *9*(1), 281–285. https://doi.org/10.30574/ijsra.2023.9.1.0410

Song, Y., Liang, J., Lu, J., & Zhao, X. (2017). An efficient instance selection algorithm for k nearest neighbor regression. *Neurocomputing*, 251, 26-34. https://doi.org/10.1016/j.neucom.2017.04.018.

Stored Procedure in SQL: Benefits And How to Create It. (2021, May 10). Simplilearn.Com. https://www.simplilearn.com/tutorials/sql-tutorial/stored-procedure-in-sql

Sugimori, Y., Kusunoki, K., Cho, F., & Uchikawa, S. (1977). Toyota production system and Kanban system Materialization of just-in-time and respect-for-human system. *International Journal of Production Research*, 15, 553-564. https://doi.org/10.1080/00207547708943149.

Tian, H., Xu, J., Lian, K., & Zhang, Y. (2009). Research on strong-association rule based web application vulnerability detection. *2009 2nd IEEE International Conference on Computer Science and Information Technology*, 237–241. https://doi.org/10.1109/ICCSIT.2009.5234394

Tiwari, V., Makhija, K., & Ratra, J. (2012). *A STUDY OF SQL INJECTION IN BANKING TRANSACTION*. *1*, 70–75.

Wilson, V. (2014). Research Methods: Triangulation. *Evidence Based Library and Information Practice*, 2.

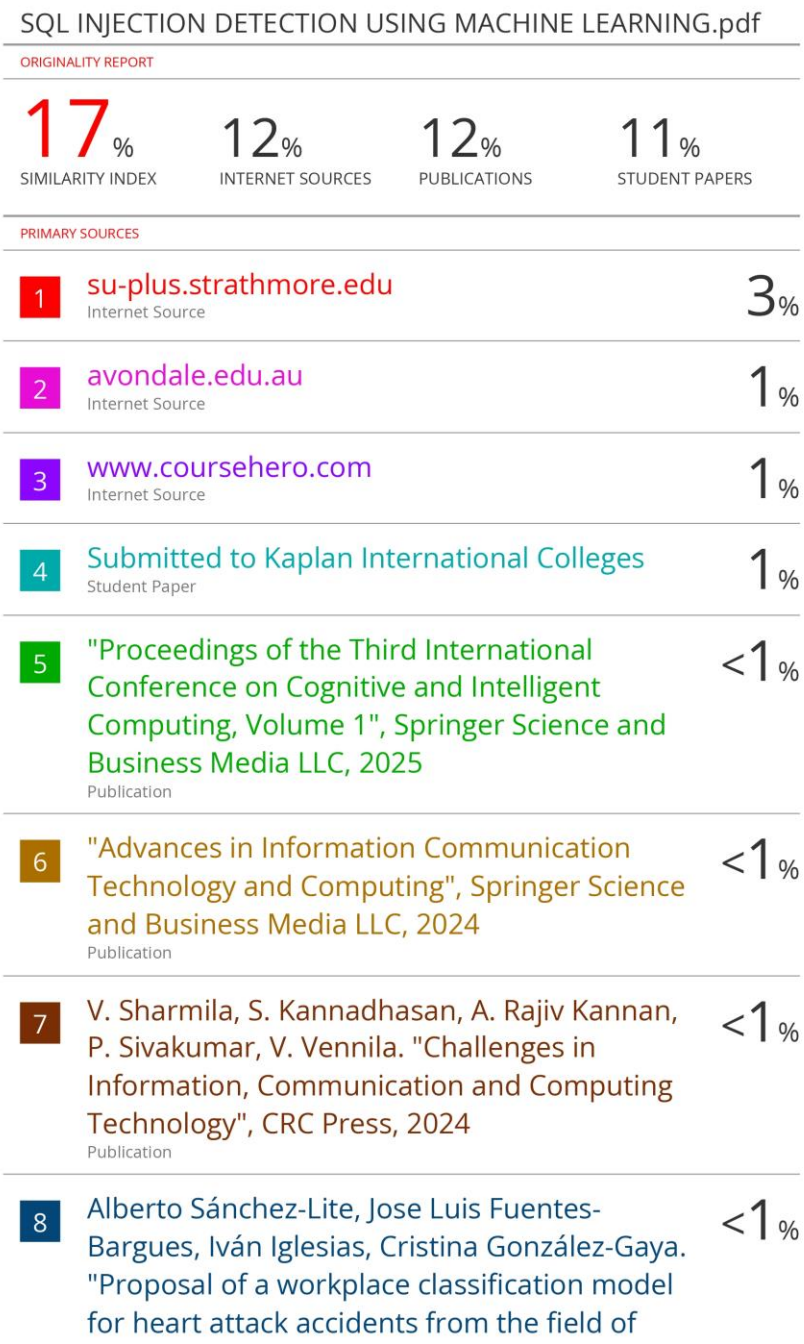
You Yu, Yuanyuan Yang, Jian Gu, & Liang Shen. (2011). Analysis and suggestions for the security of web applications. *Proceedings of 2011 International Conference on Computer*

Science and Network Technology, *1*, 236–240. https://doi.org/10.1109/ICCSNT.2011.6181948

Yu, H., & Kim, S. (2012). SVM Tutorial: Classification, Regression, and Ranking. *Handbook of Natural Computing*. https://doi.org/10.1007/978-3-540-92910-9_15

Appendices

Appendix A: Similarity Report



SQL INJECTION DETECTION USING MACHINE LEARNING.pdf

ORIGINALITY REPORT

17%	12%	12%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	su-plus.strathmore.edu Internet Source	3%
2	avondale.edu.au Internet Source	1%
3	www.coursehero.com Internet Source	1%
4	Submitted to Kaplan International Colleges Student Paper	1%
5	"Proceedings of the Third International Conference on Cognitive and Intelligent Computing, Volume 1", Springer Science and Business Media LLC, 2025 Publication	<1%
6	"Advances in Information Communication Technology and Computing", Springer Science and Business Media LLC, 2024 Publication	<1%
7	V. Sharmila, S. Kannadhasan, A. Rajiv Kannan, P. Sivakumar, V. Vennila. "Challenges in Information, Communication and Computing Technology", CRC Press, 2024 Publication	<1%
8	Alberto Sánchez-Lite, Jose Luis Fuentes-Bargues, Iván Iglesias, Cristina González-Gaya. "Proposal of a workplace classification model for heart attack accidents from the field of	<1%

	occupational safety and health engineering", Heliyon, 2024 Publication	
9	dspace.library.uvic.ca Internet Source	<1 %
10	Submitted to Ahlia University Student Paper	<1 %
11	R. N. V. Jagan Mohan, Vasamsetty Chandra Sekhar, V. M. N. S. S. V. K. R. Gupta. "Algorithms in Advanced Artificial Intelligence", CRC Press, 2024 Publication	<1 %
12	Submitted to Staffordshire University Student Paper	<1 %
13	Submitted to Bay of Plenty Polytechnic Student Paper	<1 %
14	Submitted to Kennesaw State University Student Paper	<1 %
15	Submitted to Napier University Student Paper	<1 %
16	R. N. V. Jagan Mohan, B. H. V. S. Rama Krishnam Raju, V. Chandra Sekhar, T. V. K. P. Prasad. "Algorithms in Advanced Artificial Intelligence - Proceedings of International Conference on Algorithms in Advanced Artificial Intelligence (ICAAAI-2024)", CRC Press, 2025 Publication	<1 %
17	ijrpr.com Internet Source	<1 %
18	home.simula.no Internet Source	<1 %

Appendix B: Ethical Clearance Confirmation

Strathmore UNIVERSITY

21st March 2025

Mr Muriungi Alvin,
alvin.muriungi@strathmore.edu

Dear Mr Muriungi,

RE: SQL Injection Detection Using Machine Learning

This is to inform you that SU-ISERC has reviewed and **approved** your above **SU-masters** proposal. Your application reference number is **SU-ISERC2622/25.** The approval period is from **21st March 2025 to 20th March 2026.**

This approval is subject to compliance with the following requirements:

i. Only approved documents including (informed consents, study instruments, MTA) will be used.
ii. All changes including (amendments, deviations, and violations) are submitted for review and approval by SU-ISERC.
iii. Death and life-threatening problems and serious adverse events or unexpected adverse events whether related or unrelated to the study must be reported to SU-ISERC within 72 hours of notification.
iv. Any changes anticipated or otherwise that may increase the risks or affected safety or welfare of study participants and others or affect the integrity of the research must be reported to SU-ISERC within 72 hours.
v. Clearance for the export of biological specimens must be obtained from relevant institutions.
vi. Submission of a request for renewal of approval at least 60 days prior to the expiry of the approval period. Attach a comprehensive progress report to support the renewal.
vii. Submission of an executive summary report within 90 days of completion of the study to SU-ISERC.

Before commencing your study, you will be expected to obtain a research license from National Commission for Science, Technology, and Innovation (NACOSTI) https://research-portal.nacosti.go.ke/ and obtain other clearances needed.

Yours sincerely,

Mr Ambrose Rachier,
Chairperson; SU-ISERC

Ole Sangale Rd, Madaraka Estate. PO Box 59857-00200, Nairobi, Kenya. Tel +254 (0)703 034000
Email admissions@strathmore.edu www.strathmore.edu

Appendix C: Reviewer Comments

Comment on the appropriateness of Selected Methods and Tools: The tools should be explicitly listed either in the documentation or as an appendix.

The research employs Convolutional Neural Networks (CNNs) for SQL injection detection, which is appropriate given CNNs' capability to identify complex patterns. The software tools used include TensorFlow, Keras, Scikit-learn, and MySQL as the database management system. Additionally, Python is used as the primary programming language, with Jupyter Notebook as the development environment, and Git for version control. These tools are explicitly listed here to ensure clarity in the research methodology. Additionally, the Kanban methodology was selected for its suitability in managing the iterative development and evaluation of the proposed solution.

Reference: Section 2.6, Page 18.

Comment on the appropriateness of Sampling Methods and Sample Size: There is no information to demonstrate the population and the resultant sample appropriate for the development of the solution. You need to a scientific justification for the sample required for the development.

The dataset comprises approximately 1,000 SQL queries (both benign and malicious) and is split into training (70%), validation (15%), and testing (15%) sets. The sampling method used is stratified sampling to ensure an equal representation of different types of SQL injection attacks, such as Union-based, Error-based, and Blind SQL injections. This approach helps to balance the dataset and improve model performance. This sample size was chosen to adequately train the CNN model while ensuring significant results. The data split allows for robust evaluation and reduces the risk of overfitting.

Reference: Section 2.7, Conceptual Framework, Page 19.

Comment on the appropriateness and completeness of inclusion and exclusion criteria: We need clear and explicit information on how or criteria to include or exclude the data used for the development of the solution.

The research defines clear inclusion and exclusion criteria to ensure a balanced dataset. Inclusion criteria encompass SQL queries sourced from real-world attack datasets, representing diverse techniques such as Union-based, Error-based, and Blind SQL injection attacks. It also includes queries from various SQL dialects like MySQL, PostgreSQL, and SQL Server to enhance model adaptability, along with queries that exhibit meaningful variations in structure and complexity to improve generalization.

Conversely, exclusion criteria eliminate duplicate queries that may cause overfitting and bias, non-relevant database queries unrelated to SQL injection, overly simplistic examples that fail to reflect real-world attack patterns, and queries with missing or incomplete data that could impact model training and evaluation. These measures ensure the model learns effectively and generalizes well across different database environments.

Reference: Section 2.7, Conceptual Framework, Page 19.

Comment on the consistency of the cited references: There is no list of definitions of the key terms with authentic citations in the documentation and MUST be added.
The proposal references relevant sources, including OWASP reports and recent machine learning studies. However, ensuring that key terms such as "SQL Injection" and "Machine Learning" are explicitly defined with citations from authoritative sources will further improve clarity and consistency.

Reference: References, Page 26.

Comment on the fair distribution of and access to benefits of the research: There is no information of the fair distribution of and access to benefits of the research and MUST be included in the documentation.
The proposal doesn't explicitly address the fair distribution of and access to the benefits of the research. It is important to consider how the research outcomes (e.g., the developed SQL injection detection system) will be made available to the wider community, particularly to organizations that may not have the resources to develop their own solutions. This could involve open-sourcing the code, providing access to the system through a cloud-based service, or publishing detailed documentation and tutorials.

This research is designed to make SQL injection detection more effective and accessible to a wide range of users. Cybersecurity threats affect organizations of all sizes, and not all of them have the resources to develop advanced security solutions on their own. By using open-source tools like TensorFlow and Keras, this project ensures that the knowledge and technology can be shared with researchers, security professionals, and even smaller businesses that may not have the budget for expensive security systems.

The goal is to make the findings available through academic publications, open datasets, and possibly a cloud-based tool, so that anyone who needs it can benefit. This way, the

research doesn't just serve large companies with deep pockets it helps strengthen cybersecurity for everyone.

Reference: Protocol Section 1.6, Justification of the Study, Page 5.

Comment on the outlines of the procedures that will be followed to keep participants informed of the progress and outcome of the research: There is no information on the fair distribution of and access to benefits of the research and MUST be included in the documentation.

This research doesn't involve human participants in the traditional sense, this point is less directly applicable.

As the only researcher involved in the project, I took responsibility for tracking and documenting the progress of the research. While there were no external participants, I maintained a detailed record of key milestones, methods, and outcomes to ensure the research was on track and could be referenced in the future.

Upon completion of the project, I compiled a comprehensive final report that summarized the methodology, results, and conclusions. This report, along with any relevant code or documentation, was stored for future use, ensuring that the outcomes of the research can be shared and reviewed when needed, either for personal reference or in the context of broader dissemination.

Reference: Section 3.7, Ethical Considerations, Page 26.

Comment on whether the proposal has due regard for the welfare, rights, beliefs, perceptions, customs and cultural heritage of those to be involved: There is no information on the adequate explanation been given on how the results will be utilized and MUST be included in the documentation.

As the research focuses on cybersecurity and does not directly involve human subjects or sensitive data related to specific individuals or communities, the ethical considerations related to welfare, rights, beliefs, perceptions, customs, and cultural heritage are minimal. The research does address data privacy concerns and ensure that any data used for training and testing is anonymized and handled securely.

Additionally, although the research does not directly involve human subjects, the research remained mindful of broader ethical principles, ensuring the research contributes positively to the field and respects privacy norms. By improving the detection of SQL injection attacks, this research could enhance the security infrastructure of organizations globally,

indirectly supporting the protection of user data and privacy in a way that aligns with ethical values.

Reference: Section 3.7, Ethical Considerations, Page 26.

The changes have not been made in the document directly: it MUST be included there and not hidden in the appendices. In addition: 1. Remove the listing of items using bullet points, some dotted within the documentation (acceptable for presentation and not documentation). 2. List of References are supposed to be in alphabetical order: check out the listing. 3. There is no Section to indicate the dissemination of results and utilization of the results: just next to the Ethical considerations.

All additional content that was previously placed in the appendices, including details on dissemination of results and ethical considerations, has been integrated into the main sections of the document.

Inappropriate use of bullet points has been replaced with structured paragraph formatting to maintain formal documentation style.

The references have been alphabetized according to standard citation guidelines.

This section has been added after the ethical considerations section to outline how the research findings will be distributed and utilized effectively.

Reference: References, Page: 29 & Section 3.8 Dissemination and Utilization of Results, Page: 27