# SQLi Attack Detection Using Machine Learning Techniques for Web Application Security


**Submitted By**

**Md. Siam Hasan**

221-35-1024

Department of Software Engineering
(Major in Cyber Security)
Daffodil International University

**Supervised by**

**Ms. Ashrafia Esha**

Lecturer

Department of Software Engineering

Daffodil International University


A thesis submitted in partial fulfillment of the requirement for the degree of

Bachelor of Science in Software Engineering

Fall-2025

© All right Reserved by Daffodil International University

SQLi Attack Detection Using Machine Learning Techniques for Web Application Security

MD. SIAM HASAN

Bachelor of Science

DAFFODIL INTERNATIONAL UNIVERSITY


©Daffodil International University

## APPROVAL

This thesis titled on "**SQLi attack detection using machine learning techniques for web application security**", submitted by **Md. Siam Hasan (ID: 221-35-1024)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

## BOARD OF EXAMINERS

---

**Chairman**

**Dr. Imran Mahmud**
**Professor & Head**
Department of Software Engineering
Faculty of Science and Information Technology Daffodil International University

---

**Internal Examiner 1**

**Afsana Begum**
**Assistant Professor**
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

---

**Internal Examiner 2**

**Md. Shohel Arman**
**Assistant Professor**
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

---

**Internal Examiner 3**

**Nadira Islam**
**Assistant Professor**
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

---

**External Examiner**

**Md Manowarul Islam**
**Professor**
Department of Computer Science and Engineering
Jagannath University, Bangladesh


©Daffodil International University

**DAFFODIL INTERNATIONAL UNIVERSITY**

## DECLARATION OF THESIS AND COPYRIGHT

| | | |
|---|---|---|
| Author's Full Name | : | Md. Siam Hasan |
| Date of Birth | : | 04 December, 2001 |
| Title | : | SQLi Attack Detection Using Machine Learning Techniques for Web Application Security |
| Academic Session | : | Fall 2025 |

I declare that this thesis is classified as:

| | | |
|---|---|---|
| ☑ | CONFIDENTIAL | (Contains confidential information under the Official Secret Act 1997)* |
| ☑ | RESTRICTED | (Contains restricted information as specified by the organization where research was done)* |
| ☐ | OPEN ACCESS | I agree that my thesis to be published as online open access (Full Text) |

I acknowledge that Daffodil International University reserves the following rights:

1. The Thesis is the Property of Daffodil International University.
2. The Library of Daffodil International University has the right to make copies of the thesis for the purpose of research only.
3. The Library of Daffodil International University has the right to make copies of the thesis for academic exchange.

Certified by:

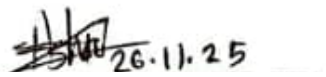
| | |
|---|---|
| Siam Hasan | 26.11.25 |
| (Student's Signature) | (Supervisor's Signature) |
| 221-35-1024 | Ms. Ashrafia Esha |
| Student ID | Name of Supervisor |
| Date: 26.11.25 | Date: 26.11.25 |

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.


©Daffodil International University

# SUPERVISOR'S DECLARATION

I hereby declare that I have checked this thesis and in my opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Bachelor of Science.



(Supervisor's Signature)

Full Name : Ms. Ashrafia Esha

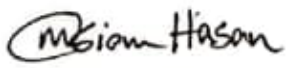
Position : Lecturer

Date : 26 November, 2025


©Daffodil International University

# STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Daffodil International University or any other institution.



(Student's Signature)

Full Name : Md. Siam Hasan

ID Number : 221-35-1024

Date : 26 November, 2025


©Daffodil International University

# SQLi Attack Detection Using Machine Learning Techniques for Web Application Security

MD SIAM HASAN
ID: 221-35-1024

Thesis submitted in fulfillment of the requirements
for the award of the degree of
Bachelor of Science

Department of Software Engineering (Major in Cyber Security)

DAFFODIL INTERNATIONAL UNIVERSITY

November, 2025


©Daffodil International University

# ACKNOWLEDGEMENT

I have been fascinated by how digital systems operate, how they can be attacked and how they can be defended. This curiosity gradually led me toward Cyber Security and ultimately to this thesis on securing software based systems. I am sincerely thankful to the Almighty Allah for blessing me with the strength, good health and perseverance needed to carry out this work under the Department of Software Engineering (Major in Cyber Security) at Daffodil International University.

I am deeply indebted to my parents, whose unwavering support, love and prayers have been my constant source of motivation. Their trust in me has helped me overcome difficulties and stay committed throughout my academic journey and research activities.

I would like to extend my gratitude to Dr. Imran Mahmud, Head of the Department of Software Engineering and to all my respected teachers in the department. Their teaching, mentorship and guidance built the academic foundation that enabled me to work on this cyber security focused thesis. I am also grateful to Daffodil International University for creating a supportive environment that encouraged both academic growth and research in emerging fields such as cyber security and intelligent threat detection.

My heartfelt thanks go especially to my supervisor Ms. Ashrafia Esha, Lecturer, Department of Software Engineering. Her consistent guidance, suggestions and supervision have been invaluable. Her patience and constructive feedback helped shape this work and significantly improved the quality of the thesis.

Finally, I would like to thank my friends and batchmates at DIU for their help, discussions and encouragement whether it was fixing bugs, reviewing chapters or sharing ideas about cyber attacks and defense mechanisms. Their support made the journey smoother and more enjoyable and I am truly grateful to have had them by my side.


©Daffodil International University

# ABSTRACT

This thesis addresses the persistent threat of SQL injection attacks, which remain one of the most critical vulnerabilities in web applications despite the widespread use of firewalls and input filters. Such traditional defenses often fail to generalize to previously unseen attack patterns. To tackle this limitation, we develop and evaluate a machine learning based detection framework for SQLi, designed to be integrated into web security. Incoming SQLi query are first preprocessed and transformed into TF-IDF feature vectors, capturing both benign and malicious query patterns. On top of these features, we train and compare six supervised classifiers: Logistic Regression, Linear Support Vector Machine, Decision Tree, Random Forest, Complement Naive Bayes and XGBoost. Models are assessed using ROC-AUC, Precision-Recall AUC (PR-AP), confusion matrices and class wise precision, recall and F1-score on a validation set of 3,981 samples. All the models achieved strong validation performance (ROC-AUC $\geq$ 99.57%, PR-AP $\geq$ 98.91%), with Random Forest and Logistic Regression showing particularly high accuracy. Logistic Regression is selected as the primary model based on its best validation PR-AP (99.90%) and consistently high F1-scores for both classes. On an independent test set of 4,280 requests, the selected model attains a ROC-AUC of 99.97% and PR-AP of 99.99%. After optimizing the decision threshold using an F2-score constraint and a cost sensitive objective that heavily penalizes missed attacks, the deployed configuration reaches 99.93% overall accuracy, with macro-F1 of 99.64%, detecting 4,057 out of 4,058 SQLi queries and misclassifying only two benign requests as attacks. These results demonstrate that a carefully tuned, interpretation friendly linear model on TF-IDF features can deliver near perfect SQLi detection performance, offering a practical and easily deployable enhancement to existing web security mechanisms.

**Keywords:** SQL Injection, Attack, Detection, Machine learning, Web Security.


©Daffodil International University

# TABLE OF CONTENTS




©Daffodil International University

©Daffodil International University

# LIST OF FIGURES




©Daffodil International University

# LIST OF TABLES




©Daffodil International University

# CHAPTER 1

# INTRODUCTION

## 1.1 Background

SQL injection has been a big security problem for web apps. The issue arises when an application makes SQL queries by inserting user input directly into query strings without first validating them. In that situation, an attacker can modify the query that is being run by exploiting form fields, URL parameters or HTTP headers. This can cause the database to perform things that the person who made it didn't want it to do [1]. In reality, a successful SQLi attack might allow an attacker pose like a legitimate user, read or copy sensitive data, change or delete stored data and in the worst cases, take over the database that runs the whole program [1]. Developers and security teams sometimes utilize more than one way to protect against these dangers. You should utilize parameterized queries instead of string concatenation to secure web facing services and you should install rule-based Web Application Firewalls (WAFs) in front of them [2]. These stages are vital and often stop basic probes or known exploit patterns, although they largely depend on hand written rules and signatures. Attackers who utilize complicated techniques, change the structure of payloads or link many attack phases together are challenging for static rule sets to keep up with. They lose their effectiveness with time [3]. As a result, SQLi is still common in claims of real breaches, even in systems that say they follow best practice standards [1]. Because of these limitations, more and more research is looking at SQLi detection as a supervised machine learning problem instead of just using rules [4; 3]. The fundamental idea is to use data to educate a model how to detect the difference between good and bad behavior. We initially turn each SQL query or HTTP request into a numerical feature vector that tells us about the content and structure of the request. Common traits include the specific tokens that are present (such SELECT, UNION, or DROP), the use of quotes, comments and logical operators and fundamental statistics like length [5]. Then, sets of labeled benign and inserted questions are used to train standard machine learning algorithms like Logistic Regression, Decision Trees, Random Forests, Naive Bayes and Support Vector Machines (SVM). Studies indicate that these models may correctly identify the majority of fraudulent requests while maintaining a low incidence of false positives [4; 6]. When trained on representative datasets, ML based detectors often do better than filters that only use signatures or manual tuning [7; 3]. Newer research is also looking into more complicated ML


©Daffodil International University

settings for finding SQLi in addition to these fundamental pipelines. Survey and review papers assemble a wide range of feature engineering techniques, from simple bag of words models and statistical descriptors to more advanced structural encodings. They also link these methodologies to several types of models, such as tree ensembles, SVMs, neural networks and hybrid ensembles [3]. Some other contributions provide generative models that make realistic harmful and helpful questions. The goal is to add to tiny training sets and help detectors learn to recognize attack patterns that aren't in the original dataset [8]. There are various frameworks that include SQLi detection as part of bigger systems that look for web attacks. These technologies use the same machine learning pipeline to look for brute-force logins, cross site scripting (XSS) and SQLi events in HTTP logs [9]. All of these lines together show that well planned ML based approaches, especially when combined with generative augmentation, make a strong and adaptable foundation for halting and identifying SQL injection assaults [4; 6].

## 1.2 Problem Statement

Recent studies suggest that machine learning techniques can achieve strong performance on SQLi detection, but a closer inspection of the literature shows several gaps that hinder their direct adoption in operational web systems. A primary concern is the nature of the datasets used to train and evaluate these models. Many works rely on relatively small or heavily synthetic collections, where both malicious payloads and benign queries are crafted from textbook style examples, controlled lab setups or intentionally vulnerable applications [3; 6]. These settings often cover only a narrow range of cases and do not fully reflect the variety of error-based, union-based, blind and time-based injection strategies observed in real deployments [10]. Consequently, models may overfit to clean, simplified traffic and fail to cope with the noisy, diverse queries produced by modern web frameworks, APIs and microservice architectures [11]. Other works draw their SQLi samples from broad intrusion or web attack datasets, in which SQLi appears as just one among several attack types and is frequently overshadowed by normal traffic [9; 12]. This class imbalance can push models toward predicting the majority (benign) class, even when the research goal is to detect SQLi specifically. A second difficulty arises from the diversity of feature engineering and modelling strategies, which complicates fair comparison and practical selection of an approach. Some authors rely on relatively simple, keyword counts, operator frequencies or query length statistics [6; 10]. Others develop more elaborate pipelines that incorporate bio inspired classifiers, search algorithm feature selection procedures or word embedding representations of SQL queries [13; 14]. Additional work uses


©Daffodil International University

optimization schemes such as the Binary Gray Wolf algorithm to identify compact but discriminative feature subsets [15]. However, these methods are rarely assessed under a common experimental protocol. Choices of dataset, preprocessing steps, train test splits and evaluation metrics vary substantially across studies, making it unclear whether reported gains are due to the model itself or to the surrounding experimental design [3; 11].

For practitioners, this lack of methodological consistency makes it difficult to determine which combination of features and algorithms offers the best balance of accuracy, robustness and implementation cost. A third challenge concerns the transition from promising research prototypes to deployable, production grade systems. Several papers outline “advanced detection and prevention” architectures in which ML based detectors are linked to WAFs or web application middleware, but the discussion often remains at a conceptual or high level design stage [10; 16]. Concrete details about latency, scalability, monitoring and long term maintenance in high traffic environments are typically sparse. In parallel, recent work on generative models for SQLi focuses mainly on their ability to simulated attack queries or derive new rules and signatures [8; 17]. Only a few contributions examine operational aspects such as per request decision time, resource usage, or the handling of false positives when such models are integrated into a live web stack [18]. Overall, the literature still lacks systematic evaluations of ML based SQLi detectors under conditions that approximate real time, high throughput web traffic. Taken together, these observations motivate the central research problem addressed in this thesis: how to design and evaluate a machine learning based SQL injection detection system that maintains strong performance on realistic, potentially imbalanced data while remaining lightweight enough for practical deployment [3]. In particular, there is a need for detectors that integrate more smoothly with existing security controls such as WAFs and intrusion detection systems, without introducing excessive complexity or overhead [16]. To this end, the thesis concentrates on constructing a clearly documented, SQLi focused dataset, defining an interpretable yet expressive feature set and benchmarking a set of classical ML classifiers within a single, unified training and evaluation pipeline [6]. Where useful, it incorporates ideas from prior work on feature design, optimisation and specialised classifiers, but applies them in a way that emphasises clarity, reproducibility and ease of integration into real world web security infrastructures [13; 14].

 ©Daffodil International University

## 1.3 Research Gaps

A combined view of the twenty reviewed studies reveals several recurring gaps that motivate the present thesis. The first gap concerns the scarcity of realistic, well documented datasets dedicated to SQLi detection. Many authors assemble their own private corpora with only brief descriptions of how requests were collected, labelled and preprocessed, which makes faithful reproduction of their experiments difficult [6]. Survey articles repeatedly note that this lack of transparency and standardisation hampers fair comparison between methods and limits the reuse of existing resources as common baselines [11]. When SQLi examples are instead extracted from generic web attack or intrusion datasets, a different issue appears: SQLi events often form a small minority class among many other behaviours, so careless evaluation may lead to overly optimistic conclusions under severe class imbalance [9]. A second, closely related gap is the limited availability of unified benchmarking frameworks that test multiple feature sets and algorithms under the same experimental conditions. The literature covers everything from simple syntax indicators to more advanced representations such as word embeddings, as well as search algorithm feature selection schemes like Binary Gray Wolf [5; 14]. Specialised classifiers based on Slime Mould optimisation have also been proposed to better capture complex decision boundaries and detect unknown SQLi variants [13]. Yet, such techniques are usually introduced and evaluated in isolation, each on a different dataset with its own preprocessing steps and metrics. This fragmentation makes it difficult to draw robust conclusions about which combinations of features and models behave consistently well across scenarios or are merely tuned to narrow experimental setups [16]. There remains a clear need for systematic studies that implement several classical ML pipelines within a single, clearly defined evaluation framework. A third gap is limited attention to robustness against evolving and deliberately complicate attacks.

Although many authors acknowledge that SQLi queries change over time and can be crafted to evade static rules, the dominant practice is still to evaluate models on single train test splits drawn from a fixed snapshot of data [8]. Questions about temporal drift, adversarial manipulation and long term maintenance of detectors are often left at the level of intuition rather than empirical analysis. Generative approaches based on adversarial networks or large language models begin to address this by automated new attack samples, thereby exposing detectors to a broader variety of patterns during training [17]. However, the extent to which such techniques genuinely improve robustness when compared with more traditional


©Daffodil International University

augmentation, retraining or ensemble strategies remains only partially explored [18]. A fourth research gap relates to deployment oriented aspects such as latency, resource consumption and interpretability for security analysts. Most publications report standard classification metrics (accuracy, precision, recall, F1-score), but comparatively few quantify inference time or computational overhead when the model operates inline with real web traffic [4; 6]. In addition, as models become more complex, their internal decision processes are harder to explain. If analysts cannot easily understand why a given request is flagged as malicious, it becomes more difficult to investigate incidents, tune detection thresholds, or justify false positives and false negatives to stakeholders [3; 11]. Only a small subset of the literature explicitly considers explainability or analyst support and even there the discussion often stays at a conceptual level rather than providing concrete tools.

This thesis is structured to address these gaps in a focused and practical way. First, it builds a clearly documented dataset of benign and malicious web requests centred on SQLi, with explicit descriptions of collection, labelling and preprocessing steps so that future work can reuse the same resource as a reference benchmark [11]. Second, it defines a transparent yet expressive feature extraction pipeline tailored to typical SQL queries aiming to keep the representation informative without relying on unnecessarily complex model architectures [5; 14]. Finally, it implements a reproducible benchmarking framework that compares several classical ML classifiers under consistent conditions, with particular emphasis on evaluation metrics and resource considerations that matter in realistic deployment settings [18].

 ©Daffodil International University

# CHAPTER 2

# LITERATURE REVIEW

## 2.1 Introduction

SQL injection (SQLi) has been one of the most frequent and damaging web application vulnerabilities, enabling attackers to manipulate backend databases through crafted input embedded in HTTP requests. Over time, the research community has proposed a wide range of countermeasures, from secure coding guidelines, input validation and web application firewalls to more recent machine learning based detectors that automatically distinguish benign from malicious queries. This chapter reviews these developments in a structured way: it first outlines the fundamental concepts and attack patterns of SQLi, then examines traditional signature and rule based defense and finally surveys supervised learning and advanced feature engineering approaches for SQLi detection. The review identifies key gaps in existing work and motivates the design choices of the proposed TF-IDF + classical ML pipeline presented in later chapters.

## 2.2 Related Work

Research on SQL injection detection and prevention time several overlapping directions, including traditional application level controls, supervised ML based detectors, advanced feature engineering and optimisation, survey and comparative analyses and more recent AI driven approaches. The earliest work mainly focuses on clarifying how SQLi vulnerabilities arise and on promoting secure design habits in web development. These studies illustrate how weak input validation, direct string concatenation of user data and overly broad database privileges enable attackers to interfere with the intended structure and semantics of SQL queries [1; 19]. They advocate for defensive practices such as systematic cleaning of inputs, parameterised queries and least privilege configurations at the database layer and they motivate the deployment of WAFs and intrusion detection systems as additional protection layers around web applications [2]. Building on these conventional defense, a large body of work frames SQLi detection as a supervised machine learning problem. In this line of research, SQL queries are converted into feature vectors that encode tokens and simple statistics and then classical classifiers such as Logistic Regression, Decision Trees, Random Forests, Naive Bayes and SVMs are trained to distinguish benign from malicious traffic [4; 7]. Reported results on the respective datasets generally show high detection rates with tolerable false positive levels,


©Daffodil International University

supporting the view that relatively standard ML techniques can meaningfully reinforce SQLi defense in practice [5; 6]. Several studies further generalise this idea to multi attack contexts, using a common pipeline to detect SQLi together with other threats such as brute-force login attempts or cross-site scripting (XSS), often by analysing web server logs or flow level features [9; 20]. These multiclass experiments indicate that a shared feature space and model family can be adapted to multiple categories of web attacks within a single framework [12]. To improve performance and flexibility beyond basic feature sets, other contributions explore more sophisticated feature engineering and optimization strategies. Word embedding approaches map tokens or subsequences of SQL queries into a continuous vector space, enabling downstream classifiers to exploit semantic relationships and contextual patterns that simple bag of words or frequency based representations may miss [14]. Automated methods such as the Binary Gray Wolf Optimizer have been applied to select compact yet discriminative subsets of features, with the dual goal of boosting accuracy and keeping model complexity manageable [15]. Bio-inspired classifiers driven by Slime Mould optimisation have also been proposed as specialised detectors for unknown SQLi, leveraging flexible search dynamics in high dimensional spaces to capture subtle differences between benign and malicious requests [13]. In parallel, survey and review articles provide a higher level combination of ML based SQLi detection. They compare models, feature representations, datasets and evaluation setups across multiple papers and they repeatedly point out persistent short comings such as weak dataset standardisation, different train test splitting strategies and limited attention to explainability of detectors under evolving attack patterns [3; 11]. Complementing these surveys, a more recent strand of work investigates generative and AI-driven security mechanisms. Some authors rely on generative models to produce fake malicious queries or to augment existing datasets, thereby expanding the training distribution and improving coverage of rare, complicated attack variants [8; 17]. Others, including the GenSQLi framework, examine how large language models can assist in crafting or refining WAF rules that recognise SQLi payloads, pointing toward tighter integration between AI components and traditional security appliances [18]. Taken together, these strands of related work show that machine learning and increasingly, generative AI can substantially extend the capabilities of classical, rule-based SQLi defenses. At the same time, they highlight the need for solutions that remain reproducible, resource conscious and straightforward to deploy in realistic web environments, rather than excelling only in controlled experimental setups. The present thesis builds on these insights by developing an ML based SQLi detection pipeline that explicitly targets these practical requirements while remaining grounded in the established research landscape.


©Daffodil International University

# CHAPTER 3

# METHODOLOGY

## 3.1 Introduction

The methodology used to design, train and evaluate a machine learning based SQL injection detection system. The proposed work starts from a raw CSV file of SQL queries, applies systematic preprocessing and dataset splitting, converts text into numerical features using a combined TF-IDF representation and then trains a suite of classical machine learning models under a unified evaluation framework. The methodology has four main goals: ensure clean and consistent data through careful text normalization and duplication, construct a realistic evaluation split that new SQLi patterns, benchmark multiple interpretable models using the same feature space and select a deployment ready classifier and threshold using cost sensitive and recall oriented criteria. All experiments were implemented in Python using standard libraries (NumPy, pandas, scikit learn, xgboost) and executed with a fixed random seed for reproducibility.

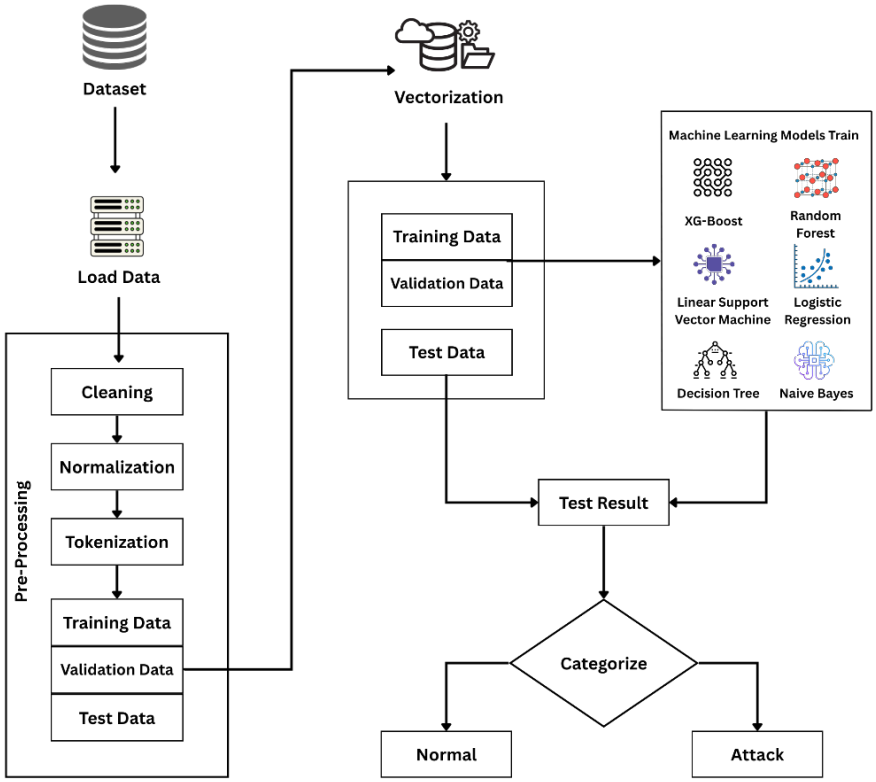



Figure 1: Proposed Model of SQLi Attack Detection Using ML


©Daffodil International University

## 3.2 Proposed Workflow

This work aims to detect SQL injection (SQLi) attacks using machine learning. We will train a model on a dataset of known SQLi queries so it can learn how attack patterns look. Once trained, the model should also be able to identify new SQLi attempts. The goal is to improve web security by catching attacks that traditional methods might miss.

**A. Dataset loading and cleaning**

The dataset containing two columns: Query and Label (0 for benign, 1 for SQLi). Apply a basic cleaning function to normalize whitespace, quotation marks and convert text to lower case. Remove duplicate (Query, Label) pairs to avoid training and testing on identical examples.

**B. Tokenization**

Use a small list of high risk SQLi tokens (union select, sleep(, benchmark(, or 1=1, comment markers) to out a dedicated test set. Any query containing at least one of these patterns is assigned to the test set; all remaining queries are candidates for training and validation.

**C. Data splitting**

From the remaining data, create a stratified split into training and validation subsets, preserving the original benign/malicious ratio. Allocate 85% of the remaining data to training and 15% to validation.

**D. Vectorization**

Build a joint feature representation that combines word level and character level TF–IDF vectors via a scikit learn FeatureUnion. Word TF–IDF uses 1- and 2-grams; character TF–IDF uses 3-5-grams, capturing both tokens and fine grained complicated patterns.

**E. Model training and validation**

Define six candidate classifiers: Logistic Regression, Linear Support Vector Machine, Decision Tree, Random Forest, Complement Naive Bayes and XGBoost. Wrap each model in a scikit learn Pipeline that includes the shared TF-IDF feature union, so all models operate on exactly the same representation. Train each pipeline on the training


©Daffodil International University

set and evaluate on the validation set using ROC-AUC and Precision-Recall Average Precision (PR-AP).

F. **Model selection**

Select the best model based primarily on validation PR-AP with ROC-AUC as a secondary tie breaker.

**G. Final test evaluation and Categorize**

Apply the best model to the reserved test set and compute ROC-AUC, PR-AP, confusion matrix and classification report using an initial probability.

## 3.3 Dataset Description

The experiments are based on a curated CSV file. Each row corresponds to a single SQL with two main fields:

• Query: the raw text of the request, which may represent a normal database interaction (benign) or an attack attempt (malicious).

• Label: a binary indicator where 0 denotes a benign query and 1 denotes a SQL injection attempt.

To maintain compatibility with the machine learning tooling, both columns are explicitly cast to the correct types: Query as string and Label as integer. The overall class distribution and split sizes (train, validation, test) are printed during preprocessing and are later summarized in the Results chapter. Stratified splitting ensures that both benign and malicious examples are represented consistently across the training and validation sets, while the test set is constructed using specific high risk tokens to emulate “unseen” attack patterns.

## 3.4 Data Preprocessing

Data preprocessing aims to make the input queries consistent, remove duplicates and create well defined splits that reflect realistic deployment scenarios.


©Daffodil International University

### 3.4.1 Cleaning and Normalization

Each query is passed through a basic sql clean function with the following steps:

Type enforcement: if a value is not a string, it is safely converted to a string. Whitespace normalization: carriage returns (\r) and newlines (\n) are replaced with spaces, multiple spaces are collapsed into a single space and leading spaces are removed. Quote harmonization: backticks (‘) are replaced by single quotes (’) to avoid treating equivalent syntactic forms as different. Case normalization: the entire query is converted to lower case.

This cleaning makes the text more uniform and reduces false differences caused by formatting, which is important for both word level and character level n-gram feature extraction. After cleaning, the dataset is de-duplicated by dropping repeated (Query, Label) pairs. This step prevents the model from overfitting to frequently repeated queries and ensures that evaluation metrics are not inflated by near identical examples appearing across different splits.

### 3.4.2 Tokenization

In this methodology, “tokenization” is used in two complementary ways:

1. Implicit tokenization for feature extraction: The word level TFIDF vectorizer uses a token pattern ((?u)\b\w+\b) that splits queries into alphanumeric word tokens. This captures meaningful SQL keywords, table names and parameters as distinct units.

2. Random search algorithm: A small list of manually selected SQLi tokens is defined: [‘union select’, ‘sleep(’, ‘benchmark(’, ‘or 1=1’, ‘–’, ‘/*’, ‘*/’ ]. For each cleaned query, a boolean mask checks whether any of these substrings appear. Queries containing at least one such pattern are flagged as potentially high-risk and diverted into the test set.

This second form of tokenization is not used as a direct model feature, instead, it is a splitting that intentionally reserves queries with classic SQLi patterns for final testing, allowing the trained models to be evaluated on a challenging, attack heavy subset that was never seen during training or validation.

### 3.4.3 Splitting Dataset

After cleaning and de-duplication, the dataset is split into three disjoint parts:

1. Test set (test_df) All queries that match any of the predefined high-risk SQLi tokens form the test set. This design simulates a “new” scenario in which the classifier must detect attacks


©Daffodil International University

that contain strong SQLi patterns but may not have appeared in the training data. 2. Remaining pool (rest_df) The remaining queries (those that do not contain the token list) are stored in rest_df. These are considered suitable for training and validation. 3. Train and validation split- rest_df is partitioned into training and validation subsets using train_test_split with: 15% of rest_df as validation and 85% as Training data. The resulting subsets are then rewrapped as train_df and val_df with consistent column naming.

At the end of this process, the sizes and class distributions of the train, validation and test sets are printed for inspection and later reported in the Results chapter.

## 3.5 Vectorization

Machine learning models cannot operate directly on raw text, so the Query field must be transformed into a numerical representation. This work adopts a TF-IDF (Term Frequency Inverse Document Frequency) representation that combines both word-level and character level features using a scikit-learn FeatureUnion.

The vectorization strategy is as follows:

1. Word-level TF-IDF

| | |
|---|---|
| Analyzer: | word |
| n-gram range: | (1, 2), unigrams and bi-grams |
| min_df = 2: | discard tokens that appear in fewer than two documents to reduce noise. |
| max_df = 0.95: | ignore tokens that appear in more than 95% of documents, as they carry little discriminative information. |
| sublinear_tf: | True: apply sublinear scaling to term frequencies, dampening the impact of very frequent terms. |
| strip_accents: | "unicode": normalise accented characters. |
| token_pattern: | "(?u)\b\w+\b": treat alphanumeric sequences as tokens. |

2. Character-level TF-IDF

| | |
|---|---|
| Analyzer: | char |
| n-gram range: | (3, 5), i.e., character 3-grams, 4-grams, and 5-grams. |
| min_df = 3: | keep only character n-grams that appear in at least three documents. |
| sublinear_tf = True: | again, to temper very frequent patterns. |


©Daffodil International University

3. FeatureUnion

| |
|---|
| The word and character vectorizers are combined using FeatureUnion ([("word", word_tfidf), ("char", char_tfidf)]) |
| During fitting, both vectorizers are trained on the training queries; during transformation, their outputs are concatenated into a single high-dimensional sparse feature matrix. |

This hybrid representation captures explicit SQL tokens (select, from, where, union select) at the word level and fine-grained patterns (URL encoded payload fragments, unusual punctuation, mixed case variants) at the character level. By embedding this FeatureUnion inside each model's Pipeline, all classifiers share the same feature space and any differences in performance come uniquely from the classifier itself.

## 3.6 Machine Learning Models Training

To explore the trade-off between performance, interpretability and computational cost, six classical machine learning models are trained and evaluated using the shared TF-IDF representation. Each model is encapsulated in a scikit learn Pipeline.

The following subsections summarize each classifier.


©Daffodil International University

### 3.6.1 Logistic Regression

Logistic Regression worked well as a solid starting point for our binary classifier. On the validation set it reached about 99.70% accuracy, with a ROC-AUC of ~ 0.9996 and PR-AP of ~ 0.9990. It also kept errors low, with only 6 benign queries flagged as attacks and 6 malicious ones missed and class wise precision, recall and F1 all hovering around 0.9945 - 0.9979. However, because it relies on a simple linear decision boundary, it still cannot fully capture the more subtle, non-linear patterns found in some SQL injection attempts, so a small but important group of sophisticated malicious queries slips through.

Table 1: Training Validation of Logistic Regression

| | 0 | 1 |
|---|---|---|
| Precision | 99.79% | 99.45% |
| Recall | 99.79% | 99.45% |
| F1-score | 99.79 % | 99.45% |

| | |
|---|---|
| Accuracy | 99.70% |
| ROC-AUC | 99.96% |
| PR-AP | 99.90% |

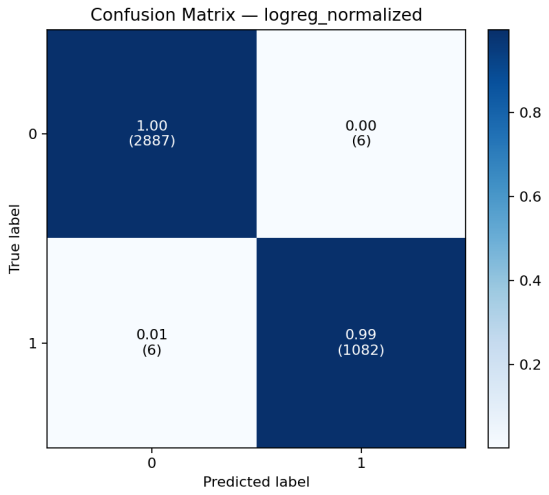



Figure 2: Confusion Matrix of Logistic Regression

Logistic Regression is a linear model that outputs calibrated probabilities, making it naturally suitable for threshold tuning and cost-sensitive decision rules.


©Daffodil International University

### 3.6.2 Linear Support Vector Machine

On the validation set, the linear SVM performed slightly better than Logistic Regression on our validation set. It reached about 99.77% accuracy, with a ROC-AUC of 0.9996 and PR-AP of 0.9987 over 3,981 queries. The confusion matrix [[2888, 5], [4, 1084]] means it wrongly flagged only 5 benign queries as attacks and missed just 4 actual SQL injection attempts. This corresponds to precision, recall and F1 scores around 0.998 for benign traffic and 0.996 for malicious traffic, with a macro F1 of 0.9972 and a weighted F1 of 0.9977, showing that SVM delivers consistently strong performance for both classes.

Table 2: Training Validation of Support Vector Machine

| | 0 | 1 |
|---|---|---|
| Precision | 99.86% | 99.54% |
| Recall | 99.83% | 99.63% |
| F1-score | 99.84% | 99.59% |

| | |
|---|---|
| Accuracy | 99.77% |
| ROC-AUC | 99.96% |
| PR-AP | 99.87% |

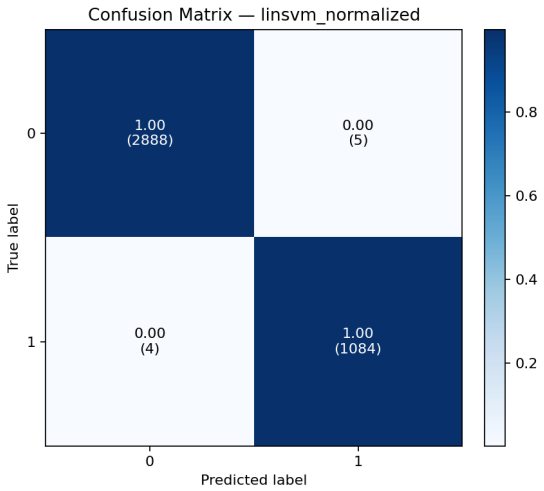



Figure 3: Confusion Matrix of Support Vector Machine


©Daffodil International University

### 3.6.3 Decision Tree

The Decision Tree model also turned out to be a very strong standalone classifier. On the validation set it reached about 99.60% accuracy, with a ROC-AUC of ~0.9958 and PR-AP of ~0.9892. In practical terms, it made only a small number of mistakes 7 benign queries were wrongly flagged as attacks and 9 malicious queries slipped through. Both classes stayed above 0.99 precision and recall for benign traffic and around 0.994 precision / 0.992 recall for malicious queries. Because the tree can learn non-linear rules, it is able to fit SQL injection patterns more flexibly than a simple linear model, although that extra flexibility makes its decision process a bit more complex and harder to summarize.

Table 3: Training Validation of Decision Tree

| | 0 | 1 |
|---|---|---|
| Precision | 99.69% | 99.36% |
| Recall | 99.76% | 99.17% |
| F1-score | 99.72% | 99.26% |

| | |
|---|---|
| Accuracy | 99.60% |
| ROC-AUC | 99.58% |
| PR-AP | 98.92% |

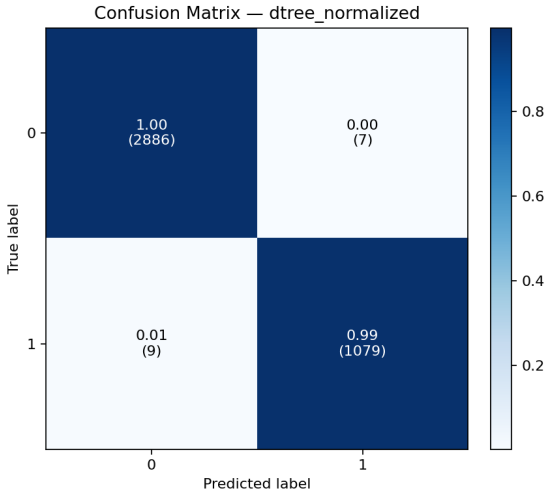



Figure 4: Confusion Matrix of Decision Tree

Decision Trees provide highly interpretable decision rules, but may be prone to overfitting if not constrained. They are included to assess how a simple, rule-based model performs relative to linear classifiers and ensembles.


©Daffodil International University

### 3.6.4 Random Forest

The Random Forest ensemble was one of the strongest performers among the classical models. On the validation set it achieved about 99.82% accuracy, with a ROC-AUC of ~0.9997 and PR-AP of ~0.9984. In practice, it made only a handful of mistakes 2 benign queries were wrongly flagged as attacks and 5 malicious queries went undetected. Precision and recall stayed extremely high for both classes, giving F1-scores above 0.996. By averaging the decisions of many individual trees, the model can capture complex, non-linear SQL injection patterns while smoothing out random errors, which explains its strong and stable behaviour.

Table 4: Training Validation of Random Forest

| | 0 | 1 |
|---|---|---|
| Precision | 99.83% | 99.82% |
| Recall | 99.93% | 99.54% |
| F1-score | 99.88% | 99.68% |

| | |
|---|---|
| Accuracy | 99.82% |
| ROC-AUC | 99.96% |
| PR-AP | 98.84% |

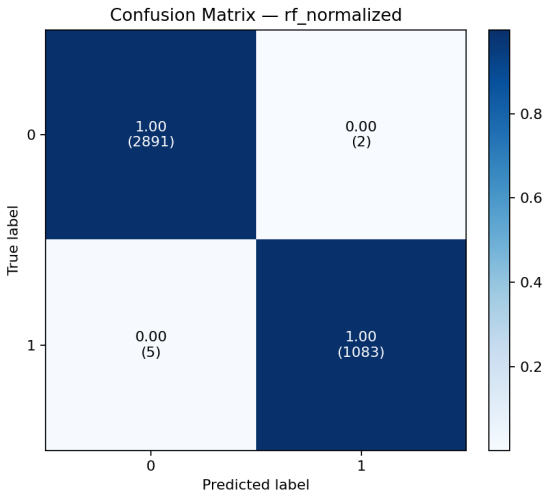



Figure 5: Confusion Matrix of Random Forest

Random Forests typically deliver more stable and accurate performance than single trees by averaging over many diverse models. They can capture non-linear relationships in the TF-IDF feature space and provide measures of feature importance.


©Daffodil International University

### 3.6.5 Complement Naive Bayes

The Complement Naive Bayes model behaved quite differently from the tree based models. On the validation set it still achieved a very high ROC-AUC (~0.9993) and PR-AP (~0.9986), but its overall accuracy dropped to about 94.83%. Looking at the confusion matrix, it labelled 203 benign queries as attacks while missing only 3 actual attacks. This is reflected in the class wise metrics: benign traffic has almost perfect precision (≈0.999) but lower recall (≈0.93), whereas malicious queries have very high recall (≈0.997) but much lower precision (≈0.84). This means the model is extremely aggressive in flagging suspicious SQL injection attempts catching nearly every real attack but it also generates a lot of false positives, which could overload security analysts with unnecessary alerts.

Table 5: Training Validation of Complement Naive Bayes

| | 0 | 1 |
|---|---|---|
| Precision | 99.89% | 84.24% |
| Recall | 92.98% | 99.72% |
| F1-score | 96.31% | 91.33% |

| | |
|---|---|
| Accuracy | 94.83% |
| ROC-AUC | 99.93% |
| PR-AP | 98.86% |

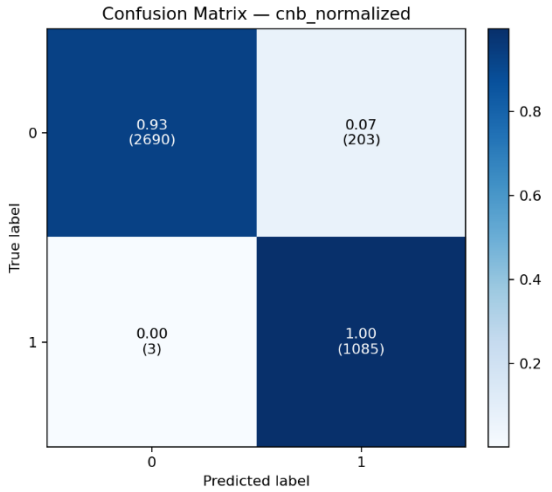



Figure 6: Confusion Matrix of Complement Naive Bayes


©Daffodil International University

CNB is computationally very efficient and often performs well on text tasks. It assumes conditional independence between features given the class, which is not strictly true for TF-IDF vectors but often acceptable in practice. Its simplicity and speed make it attractive for deployment in resource constrained environments.

### 3.6.6 XGBoost

The XGBoost model was also among the best performers in our setup. On the validation set it reached about 99.72% accuracy, with a ROC-AUC around 0.9994 and PR-AP around 0.9985. In concrete terms, it made only a few mistakes 5 benign queries were wrongly flagged as attacks and 6 malicious queries went undetected. Both classes show consistently strong scores, with precision, recall and F1 all close to 0.998 for benign traffic and around 0.995 for malicious queries. Because XGBoost stacks many boosted decision trees, it can pick up subtle, non-linear SQL injection patterns very effectively, though this extra power does make the model more complex to train and tune than a simple linear baseline.

Table 6: Training Validation of XGBoost

| | 0 | 1 |
|---|---|---|
| Precision | 99.79% | 84.54% |
| Recall | 99.83% | 99.45% |
| F1-score | 99.81% | 99.49% |

| | |
|---|---|
| Accuracy | 99.72% |
| ROC-AUC | 99.94% |
| PR-AP | 99.85% |

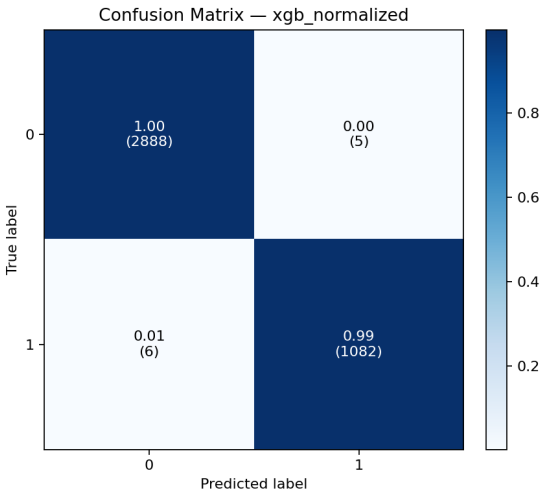



Figure 7: Confusion Matrix of XGBoost


©Daffodil International University

XGBoost is a strong non-linear learner that can capture complex interactions between the TF-IDF features, making it well suited to the subtleties of SQL injection patterns. The trade-off is that it is more computationally expensive than simpler linear models or Naive Bayes, so whether it is practical to deploy depends on the hardware and time budget available.

## 3.7 Evaluate Best Model

Once all candidate models have been trained on the training set and checked on the validation set, the “winner” is chosen using the saved validation metrics. The selection rule first looks for the highest validation PR-AP, since this directly reflects how well the model ranks the positive (attack) class, if two models tie on PR-AP, the one with the higher ROC-AUC is selected the name of this model and its pipeline.

Next, this chosen pipeline is run on the held-out test set. It produces predicted probabilities for the positive class and hard labels (initially using a 0.5 threshold), from which we compute the test ROC-AUC, test PR-AP, confusion matrix and full classification report. Because the test set is a more challenging, token-based split designed to resemble unseen attack patterns, these metrics give an unbiased view of how well the final detector is likely to behave in practice and help justify the final model and threshold used in the Results and Discussion chapters.


©Daffodil International University

# CHAPTER 4

# RESULTS AND DISCUSSION

## 4.1 Introduction

This chapter presents and analyses the experimental results obtained from the proposed SQL injection (SQLi) detection pipeline. The evaluation is organized in two stages. First, the performance of six classical machine learning models is compared on a held out validation set using a shared TF-IDF feature space. Metrics include ROC-AUC, Precision-Recall Average Precision (PR-AP), accuracy and class wise precision, recall and F1-score. Second, the best performing model is evaluated on a separate test set constructed using SQLi tokens and its operating threshold is tuned using both recall oriented and cost sensitive criteria.

The discussion highlights not only the raw metrics, but also the behaviour of each classifier in terms of false positives and false negatives, the impact of severe class imbalance in the test set, and the benefits of explicit threshold optimisation. These findings are then interpreted in the broader context of practical SQLi detection, followed by a discussion of limitations and directions for future work.

## 4.2 Models Performance

### 4.2.1 Comparative Analysis of Classifiers

All six candidate classifiers Logistic Regression, Linear SVM, Decision Tree, Random Forest, Complement Naive Bayes and XGBoost achieved very high performance on the validation set. Across models, overall accuracy remained above 94%, with most models exceeding 99%. This indicates that, under the engineered TF-IDF feature space, the distinction between benign and malicious queries is largely linearly separable and does not require highly complex decision boundaries. Among the models, Complement Naive Bayes behaved differently from the others. While it achieved strong ROC-AUC and PR-AP scores, its confusion matrix revealed a large number of false positives on the validation set (203 benign queries misclassified as attacks), leading to a noticeable drop in accuracy (94.83%) compared to the other models. In contrast, the linear and ensemble models maintained both very high detection rates for SQLi queries and very low false positive rates.


©Daffodil International University

Given these observations, the analysis focuses on how the models compare along different metric families, ROC-AUC and PR-AP, as well as precision, recall and F1-score.

### 4.2.2 Comparison by ROC-AUC and PR-AP

Summarizes the validation ROC-AUC and PR-AP scores for all models. All classifiers achieved ROC-AUC values above 0.995 and PR-AP values above 0.989, indicating that they rank SQLi queries very effectively across the full range of thresholds. Random Forest achieved the highest validation ROC-AUC (0.999654), but Logistic Regression achieved the highest PR-AP (0.999020). Because PR-AP is more sensitive to the performance on the positive (attack) class, Logistic Regression was selected as the primary model. The differences between the top models are numerically small but statistically meaningful when considering deployment scenarios where even a small number of missed attacks can be costly. Decision Tree showed slightly lower ROC-AUC (0.995755) and PR-AP (0.989158) compared with the other models, reflecting a modest reduction in ranking quality. However, even this model remained strong in absolute terms, suggesting that the engineered features are highly informative.

Table 7: Performance Table of Models

| Model | Accuracy | ROC–AUC | PR–AP |
|---|---|---|---|
| **Logistic Regression** | **99.70%** | **99.96%** | **99.90%** |
| **Linear SVM** | 99.77% | 99.96% | 99.87% |
| **Decision Tree** | 99.60% | 99.57% | 98.91% |
| **Random Forest** | 99.82% | 99.96% | 99.84% |
| **Complement Naive Bayes** | 94.83% | 99.92% | 99.86% |
| **XGBoost** | 99.72% | 99.94% | 99.84% |

Validation performance of all candidate classifiers on the SQL injection detection task, reported in terms of accuracy, ROC-AUC and PR-AP. Logistic Regression achieves the highest PR-AP and is selected as the final model.

### 4.2.3 Comparison by Precision, Recall and F1-Score

While ROC-AUC and PR-AP provide threshold independent views, operational deployment requires examining threshold dependent metrics such as precision, recall and F1-score at a baseline threshold.


©Daffodil International University

On the validation set, the linear and ensemble models achieved nearly perfect precision and recall for both classes: Logistic Regression reached a macro F1-score of 0.9962 with symmetric precision and recall for both benign and attack classes (0.9979/0.9979 for class 0 and 0.9945/0.9945 for class 1). Linear SVM slightly improved the macro F1-score to 0.9972, reflecting a minor increase in recall for the attack class (0.9963) while maintaining extremely high precision. Random Forest achieved the highest macro F1-score among all models (0.9978), with per-class F1-scores of 0.9988 (benign) and 0.9968 (attack). XGBoost also performed extremely well, with a macro F1-score of 0.9965. Complement Naive Bayes, by contrast, exposed a clear trade-off. Its recall for the attack class was extremely high (0.9972), but this came at the cost of a significant drop in recall for benign queries (0.9298), resulting in many false alarms. The macro F1-score (0.9382) was considerably lower than for the other models, underscoring that excellent ranking metrics (ROC-AUC, PR-AP) do not automatically guarantee balanced precision and recall at a fixed threshold.

Overall, these results suggest that several models are viable candidates for deployment, but Logistic Regression provides a strong combination of ranking quality, balanced class-wise performance and interpretability.

### 4.2.4 Test Set Performance

Using the selection rule based on validation PR-AP and ROC-AUC, Logistic Regression was chosen as the final model. Its performance was then evaluated on the dedicated test set, which consists of queries containing at least one of the manually selected SQLi tokens. At the default decision threshold of 0.5, the model achieved:

Test ROC-AUC: 99.97%

Test PR-AP: 99.99%

Accuracy: 95.61%.

The confusion matrix: where the rows represent the true classes (benign, attack) and the columns the predicted classes.

| | |
|---|---|
| 222 | 0 |
| 188 | 3870 |


©Daffodil International University

These numbers indicate that the model's ranking ability on the test set is almost perfect, nearly all attack queries receive higher scores than benign ones. However, the accuracy and class-wise metrics reveal an important nuance: nearly all benign queries (class 0) are correctly identified (recall = 1.0000, precision = 0.5415), but a non-trivial number of attack queries (188 out of 4058) are misclassified as benign (class 1 recall = 0.9537).

This behaviour is largely driven by the strong class imbalance in the test set (only 222 benign queries vs. 4 058 attacks) and the fact that the default 0.5 threshold does not explicitly account for the asymmetric costs of false positives and false negatives in security applications.

### 4.2.5 Hyperparameter Tuning

To better understand and control the trade-off between false positives and false negatives, the decision threshold was optimised based on two criteria:

A. Constrained F2-score: Choose the threshold that maximises the F2-score (recall-focused) subject to a precision constraint of at least 0.99.

B. Cost-based optimisation: Choose the threshold that minimises a cost function where each false positive has cost 1 and each false negative has cost 50.

Interestingly, both criteria selected the same threshold: $t^* \approx 0.0506$.

At this threshold, the confusion matrix became:

| | |
|---|---|
| 220 | 2 |
| 1 | 4057 |

leading to the following metrics:

Accuracy: 99.93%,

Class 0 (benign): precision = 0.9955, recall = 0.9910, F1-score = 0.9932,

Class 1 (attack): precision = 0.9995, recall = 0.9998, F1-score = 0.9996,

Macro average F1-score: 0.9964.

Compared to the default threshold (0.5), this tuned threshold dramatically reduces false negatives (from 188 down to 1) while maintaining an extremely low number of false positives


©Daffodil International University

(only 2 benign queries misclassified as attacks). The total cost under the specified cost function drops to 52 (2 false positives + 50 for 1 false negative).

This analysis highlights two key points:

The raw ranking quality of the model is excellent, however, using an arbitrary default threshold can lead to suboptimal operational behaviour, especially when the class distribution and error costs are highly asymmetric. Explicit threshold calibration, guided by security oriented objectives is crucial for turning a strong offline model into a practical detection system.


©Daffodil International University

## 4.3 Interpretation of Results

Taken together, the validation and test results show that classical machine learning models, when combined with carefully engineered TF-IDF features, can achieve near perfect detection of SQL injection attacks in the considered dataset.

Several observations stand out:

Simple models are sufficient: Linear models (Logistic Regression, Linear SVM) and relatively shallow ensembles (Random Forest, XGBoost) all achieve saturated ROC-AUC and PR-AP values, indicating that the feature space already provides a strong separation between benign and malicious queries. Feature representation matters more than model complexity: The fact that a straightforward Logistic Regression classifier outperforms or matches more complex models suggests that most discriminative structure is captured by the joint word- and character level TF-IDF representation, and that additional model complexity yields diminishing returns. Threshold selection is critical: On the test set, the difference between the baseline threshold (0.5) and the tuned threshold ($\approx 0.0506$) is dramatic, especially in terms of missed attacks. Without threshold optimisation, 188 attacks remain undetected; with a security-aware threshold, this number drops to 1 while retaining almost perfect precision. Complement Naive Bayes illustrates the ranking vs. classification gap: Although CNB achieves strong ROC-AUC and PR-AP, its fixed threshold classification behaviour on the validation set reveals substantial false positives. This underlines that a good ranking metric does not automatically translate to a good operating point, reinforcing the need for explicit threshold tuning for all classifiers.

From a practical perspective, the results suggest that a well regularised Logistic Regression model, equipped with a calibrated threshold derived from cost sensitive or recall-focused objectives, can serve as a highly effective and computationally efficient detector for SQL injection attacks.


©Daffodil International University

# CHAPTER 5

# CONCLUSION

## 5.1 Limitations

Even though the numbers look very strong, there are some clear limitations. All experiments are based on a single curated CSV dataset and a test set built using a few well known high risk tokens, so the setup may not fully reflect messy, diverse real world traffic or more subtle attacks. The task is also treated as a simple yes/no decision (normal vs. attack), without separating different types of SQLi and the models are not tested under changing traffic patterns. On top of that, there are no measurements of runtime or resource usage. Taken together, this means the results are promising for the current setup, but more validation is needed before claiming the approach is robust in real production environments.

## 5.2 Future Work

Looking ahead, there are several clear ways to strengthen this work. A natural next step is to test the models on large, real-world database logs and move beyond simple yes/no labels to distinguish different types of SQLi attacks. It would also be useful to see how the current classical models compare with modern sequence or language models and to explore unsupervised or semi-supervised methods that can spot new, previously unseen attack patterns. Finally, adding some form of continual learning and embedding the detector into a prototype WAF or IDS with basic monitoring of alerts and thresholds would help turn this high performing lab setup into something much closer to a deployable, real world SQL injection defense.

## 5.3 Conclusion

This thesis set out to build and test a practical machine learning pipeline for spotting SQL injection (SQLi) attacks, with three goals in mind: strong detection performance, decisions that respect security costs (especially missed attacks) and a design that can plug into existing web security setups without much hassle. Using a curated SQL dataset, the work applied standard but careful steps cleaning and de-duplication, a “new” test split and a hybrid TF-IDF representation over both words and character n-grams before training a suite of classical classifiers in a unified framework. The experiments showed that, with the right features,


©Daffodil International University

lightweight models are more than capable of high quality SQLi detection. All models achieved ROC-AUC above 0.995 and PR-AP above 0.989 on the validation set, with Logistic Regression slightly ahead (validation PR-AP ≈ 0.9990) and therefore chosen as the final model. On a separate high risk test set built from queries containing selected SQLi tokens, this model retained excellent ranking performance (ROC-AUC ≈ 0.9997, PR-AP ≈ 0.99998), but the default 0.5 threshold still let a small number of attacks pass through. To fix this, the thesis explicitly tuned the decision threshold using a constrained F2-score and a cost-based objective that treated false negatives as fifty times worse than false positives. Both pointed to a much lower threshold (≈0.0506), where the model reached 99.93% accuracy on the test set with only two benign queries flagged as attacks and one attack missed. Taken together, the results suggest that a well designed hybrid TF-IDF representation, a simple regularized linear model and careful, cost aware threshold tuning are enough to build an interpretable, efficient SQLi detector while still leaving room for future work on more varied, real world data.


©Daffodil International University

# REFERENCES

[1] S. M. S. Mahmud, T. Amrine, and M. A. Azim, “Sql injection attack vulnerabilities of web application and detection,” International Journal of Computer Applications, vol. 185, no. 38, pp. 41–48, 2023.

[2] K. Rathi and D. Saxena, “The sql injection attacks on web based application and database: Sqli attack techniques and sqli prevention and mitigation,” in Proceedings of the 3rd International Conference on Current Research in Engineering and Technology (ICCRET 2023), (Brainware University, Barasat, India), 2024.

[3] M. A. M. Oudah and M. F. Marhusin, “Sql injection detection using machine learning: A review,” Malaysian Journal of Science, Health & Technology (MJoSHT), vol. 10, no. 1, pp. 39–49, 2024.

[4] C. Rosca, A. Stancu, and C. Popescu, “Machine learning models for sql injection detection,” Electronics, vol. 14, no. 17, p. 3420, 2025.

[5] P. Roy, R. Kumar, and P. Rani, “Sql injection attack detection by machine learning classifier,” in 2022 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC), (Salem, India), pp. 394–400, IEEE, 2022.

[6] M. C. Danilla, E. Shivashankari, and A. Beatrice Dorothy, “Sql injection attack detection using machine learning methods,” International Research Journal of Modernization in Engineering Technology and Science, vol. 6, no. 1, pp. 2813–2816, 2024.

[7] N. Vijayalakshmi and M. Hemalatha, “Detection of sql injection attack using machine learning techniques: A systematic literature review,” Journal of Cybersecurity and Privacy, vol. 2, no. 4, pp. 764–777, 2022.

[8] N. S. Dasari, A. Badii, A. Moin, and A. Ashlam, “Enhancing sql injection detection and prevention using generative models,” CoRR, vol. abs/2502.04786, 2025.

[9] M. Irfa’issurur and B. P. Josaphat, “Machine learning for cybersecurity: Web attack detection (brute force, xss, sql injection),” InPrime: Indonesian Journal of Pure and Applied Mathematics, vol. 7, no. 1, pp. 1–15, 2025.

[10] A. Zahid, “Vulnerability detection and prevention: An approach to enhance cybersecurity,” 2024.


©Daffodil International University

[11] F. O. Okello, “Machine learning-based approaches for sql injection detection and prevention,” International Journal of Advanced Research, vol. 13, no. 2, pp. 1035–1044, 2025.

[12] M. Irfa’issurur and B. P. Josaphat, “Machine learning for cybersecurity: Web attack detection,” in Proceedings related to Machine Learning for Cybersecurity Web Attack Detection, 2025. Second copy/version of the InPrime work.

[13] A. I. Abdulameer and H. Alyasiri, “A bioinspired algorithm for zero-day sql injection detection via slime mould classifier,” Al-Furat Journal of Innovations in Electronics and Computer Engineering, vol. 4, no. 2, pp. 84–100, 2025.

[14] S. Venkatramulu, M. S. Waseem, A. Taneem, S. Y. Thoutam, and S. Apuri, “Research on sql injection attacks using word embedding techniques and machine learning,” Journal of Sensors, IoT & Health Sciences, vol. 2, no. 1, pp. 55–66, 2024.

[15] B. Arasteh, B. Aghaei, F. Farzad, K. Arasteh, F. Kiani, and M. Torkamanian-Afshar, “Detecting sql injection attacks by binary gray wolf optimizer and machine learning algorithms,” Neural Computing and Applications, vol. 36, no. 12, pp. 6771–6792, 2024.

[16] D. K. Singh, A. Rai, and V. Kumar, “Advanced detection and prevention of sql injection attacks using machine learning techniques for enhanced web security,” International Journal of Scientific Research in Science and Technology (IJSRST), vol. 11, no. 6, pp. 554–564, 2024.

[17] N. S. Dasari, A. Badii, A. Moin, and A. Ashlam, “Enhancing sql injection detection and prevention using generative models,” Journal of Cybersecurity and Data Science, 2025. Extended study version; based on arXiv:2502.04786.

[18] V. Babaey and A. Ravindran, “Gensqli: A generative artificial intelligence framework for automatically securing web application firewalls against structured query language injection attacks,” Future Internet, vol. 17, no. 1, p. 8, 2025.

[19] T. be confirmed, “Web application security and sql injection attack models,” International Journal of Trend in Research and Development (IJTRD), 2017.

[20] A. E. Takieldeen, A. E. El-Metwaly, R. R. Elzahdany, A. A. Lotfy, and D. M. Abdelhady, “Web attack intrusion detection system using machine learning approaches for cybersecurity,” in Proceedings of the ITC-Egypt International Telecommunication Conference, 2025.

 ©Daffodil International University

# Plagiarism Report

221-35-1024

ORIGINALITY REPORT

| 11% | 6% | 5% | 5% |
|---|---|---|---|
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | Submitted to Daffodil International University<br>Student Paper | 2% |
| 2 | Sangsang Qi, Shi Zheng, Mengdan Lu, Aner Chen, Yanbo Chen, Xianhu Fu. "Building a machine learning-based risk prediction model for second-trimester miscarriage", BMC Pregnancy and Childbirth, 2024<br>Publication | 1% |
| 3 | Md Foysal Ahmed, Gang He, Sikai Wang. "TriFusion hybrid model for human activity recognition", Signal, Image and Video Processing, 2024<br>Publication | 1% |
| 4 | dspace.daffodilvarsity.edu.bd:8080<br>Internet Source | <1% |
| 5 | core.ac.uk<br>Internet Source | <1% |
| 6 | Submitted to De Montfort University<br>Student Paper | <1% |
| | Submitted to The Robert Gordon University | |


©Daffodil International University

| | | |
|---|---|---|
| 7 | Student Paper | <1 % |
| 8 | indah.ump.edu.my<br>Internet Source | <1 % |
| 9 | Yue-ming Yuan, Jin-rui Duan, Zhao Han, Yi-guo Xue, Zhi-ping Sun, Fan-meng Kong, Chuan-gui Li. "Reliable Rock Mass Classification for Tunneling: Hole-Level MWD Data Modeling with Cost-Sensitive Bagging", Applied Geophysics, 2025<br>Publication | <1 % |
| 10 | darylma.com<br>Internet Source | <1 % |
| 11 | Submitted to BB9.1 PROD<br>Student Paper | <1 % |
| 12 | repository.tudelft.nl<br>Internet Source | <1 % |
| 13 | Submitted to University of Sheffield<br>Student Paper | <1 % |
| 14 | ijsret.com<br>Internet Source | <1 % |
| 15 | umpir.ump.edu.my<br>Internet Source | <1 % |
| 16 | ujcontent.uj.ac.za<br>Internet Source | <1 % |


©Daffodil International University

| | | |
|---|---|---|
| 17 | Submitted to The Mwalimi Nyerere Memorial Academy<br>Student Paper | <1 % |
| 18 | docslib.org<br>Internet Source | <1 % |
| 19 | Submitted to Galatasaray University<br>Student Paper | <1 % |
| 20 | Submitted to University of Sunderland<br>Student Paper | <1 % |
| 21 | Submitted to University of Hertfordshire<br>Student Paper | <1 % |
| 22 | Submitted to Bahrain Polytechnic<br>Student Paper | <1 % |
| 23 | Submitted to Liverpool John Moores University<br>Student Paper | <1 % |
| 24 | comserv.cs.ut.ee<br>Internet Source | <1 % |
| 25 | www.ijtsrd.com<br>Internet Source | <1 % |
| 26 | Submitted to Department of Public Health<br>Student Paper | <1 % |
| 27 | www.diva-portal.org<br>Internet Source | <1 % |


©Daffodil International University

# Turnitin Report



## Siam Hasan

## 221-35-1024

Quick Submit

Quick Submit

Daffodil International University

### Document Details

Submission ID
trn:oid:::1:3449006739

Submission Date
Dec 20, 2025, 1:46 PM GMT+6

Download Date
Dec 20, 2025, 1:49 PM GMT+6

File Name
Siam_Hasan_ID_221-35-1024.pdf

File Size
2.8 MB

32 Pages

8,605 Words

53,924 Characters

### *% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

turnitin Page 1 of 34 - Cover Page Submission ID trn:oid:::1:3449006739



©Daffodil International University