ISSN 2579-7298

Detection of SQL Injection Attacks on MariaDB Using Hybrid Long Short-Term Memory

Hartono[a,1,*], Rama Apriando[b,2], Khusnul Khotimah[c,3]

[a] Universitas Muhammadiyah Kotabumi, North Lampung, 34511, Indonesia
[1] hartono@umko.ac.id*; [2] rama.apriando@umko.ac.id; [3] khusnul.khotimah@umko.ac.id;
* corresponding author

ARTICLE INFO

Article history
Received
Revised
Accepted

Keywords
SQL Injection
LSTM
Web Security

ABSTRACT

This study discusses the development of a SQL Injection attack detection system using the Long Short-Term Memory (LSTM) deep learning model. SQL Injection is a serious security threat to web applications that exploits vulnerabilities in user input to manipulate databases. The LSTM model was chosen due to its ability to process sequential data, which is relevant for analyzing the patterns and structure of SQL queries that are susceptible to attacks. The process begins by collecting and combining datasets from various sources, performing preprocessing to handle duplicate data, missing values, and gibberish queries, as well as analyzing the distribution of query lengths. The textual query data is then converted into a numerical representation through tokenization and padding. The processed dataset is divided into training and testing data. The Bi-directional LSTM model architecture is built with embedding, LSTM, dropout, and dense layers. The model is trained using the training data and its performance is evaluated using the test data, producing metrics such as accuracy, precision, recall, and F1-score. Evaluation results on the test data show a model accuracy of 99.99%, with precision of 99.99%, recall of 99.99%, and F1-score of 99.99% in distinguishing between normal queries and SQL Injection queries. The trained model and the tokenizer used are then saved for further testing purposes. This research demonstrates that the LSTM-based approach is highly effective in detecting SQL Injection attacks with high accuracy. Thus, the model can be deployed at the production level or production server.

1. Introduction

In today's highly interconnected digital era, cybersecurity is a critical component that must be addressed at the individual, organizational, and national levels. The increasing use of the internet and web-based applications for various activities—including financial transactions, communication, media, and the storage of sensitive data—has simultaneously expanded the attack surface that can be exploited by malicious actors. Cyberattacks can have far-reaching consequences, including financial losses, theft of personal and confidential corporate data, service disruptions, and reputational damage that is difficult to restore [1], [2], [3], [4], [5]. Websites, as the primary interface for user interaction with various online services, are often the main targets of attacks. Vulnerabilities in web applications can arise from multiple sources, such as coding errors, insecure configurations, architectural design weaknesses, or the use of vulnerable third-party components. In this context, one of the most common and dangerous types of attacks targeting web applications is the SQL Injection (SQLi) [6], [7], [8], [9], [10].

According to Aikido (2024) report, SQLi attacks continue to represent a significant proportion of the most frequently identified vulnerabilities. Out of a total of 2,219 recorded vulnerabilities in both open-source and closed-source projects, approximately 889 vulnerabilities were found in open-source projects (6.7%), while 1,330 vulnerabilities were identified in closed-source projects (10%). This fact also indicates that, despite SQLi having long been recognized as a fundamental weakness, modern software development practices still face substantial challenges in addressing it. The high number of vulnerabilities underscores the need for ongoing efforts to enhance security awareness and competence among application developers. Furthermore, the integration of advanced detection mechanisms can help reduce the risk of similar attacks. Collaboration among developers, researchers, and industry stakeholders can foster a more secure and resilient digital ecosystem [2], [3], [4], [5], [6].

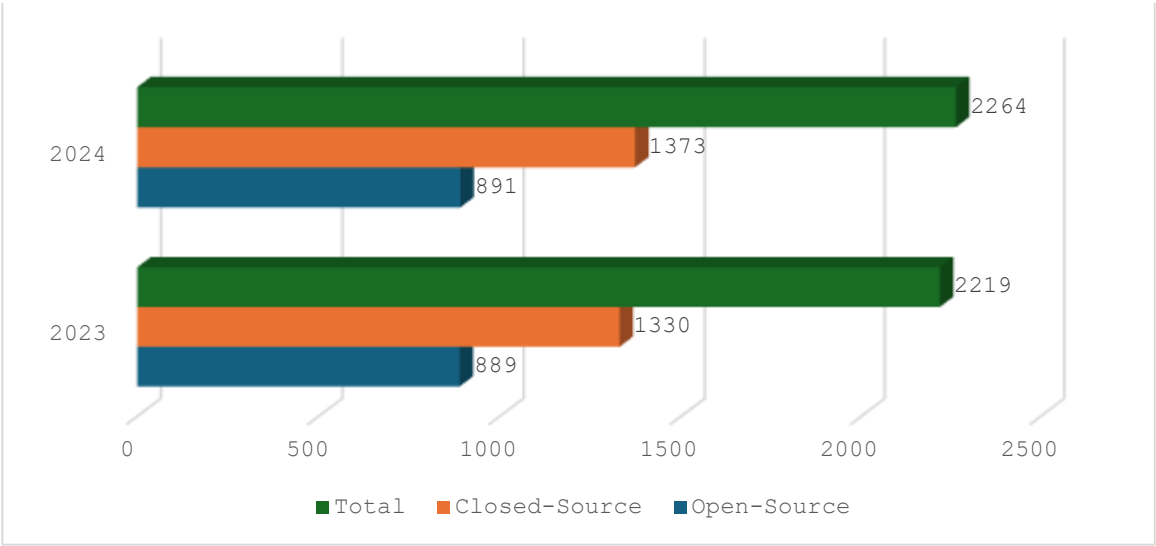


Figure 1. Proportion of SQL Injection Vulnerabilities in Open-Source and Closed-Source Projects (Source: Aikido Security, 2024)

Figure 1 illustrates the trend of SQLi vulnerabilities in both open-source and closed-source projects, indicating that this issue cannot yet be considered obsolete. In line with this, report [7] highlights that SQLi remains one of the most common application vulnerabilities, with severity levels ranging from high to critical across various industrial sectors. Based on Table 1, which presents the distribution of vulnerabilities by industry, several sectors, such as media and entertainment, medical, construction, software, and transportation, have experienced SQLi incidents characterized by CWE-89 and a CVSS score of 10, signifying extremely serious consequences if exploited. Meanwhile, other sectors face different types of vulnerabilities, such as Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CWE-79/CWE-352), with CVSS scores ranging from 6.1 to 9.8. These findings reinforce that SQLi continues to pose a tangible threat across domains, thus effective mitigation and prioritization of countermeasures based on CVSS are essential to safeguard systems and sensitive data from potential attacks.

Table 1. Most Common Application Vulnerabilities by Industry with Medium Severity Level (Source: Edgescan, 2024)

Industry	Vulnerability	CWE	CVSS
Media & Entertainment	SQL Injection	CWE-89	10
Legal	Information Disclosure - Sensitive Information in URL	CWE-200	7.1
Finance & Insurance	ICross-Site Request Forgery	CWE-352	6.8
Retail Hotels & Entertainment	Sensitive Data Enumeration	CWE-204	9
Venture Capital & Private Equity	Cross-Site Scripting (XSS) – Reflected	CWE-79	6.1

Manufacturing	Cross-Site Scripting (XSS) – Stored	CWE-79	9.3
Non Profit & Charitable	Sensitive File(s) Disclosure	CWE-200	8.6
Government	XML External Entity Injection	CWE-611	7.7
Medical	SQL Injection	CWE-89	10
Consulting	Malicious File Upload	CWE-434	9.8
Construction	SQL Injection	CWE-89	10
Banking	Cross-site Request Forgery	CWE-352	6.8
Energy & Utilities	Cross-Site Scripting (XSS) – Reflected	CWE-79	6.8
Software	SQL Injection	CWE-89	10
Transportation	SQL Injection	CWE-89	10

SQLi is one of the security vulnerabilities that arises in database-driven web applications [8]. This attack poses a serious threat as it can compromise the security of information and sensitive data stored within the application's database. Thus, SQLi can be defined as a type of security vulnerability that occurs in database-based web applications. This attack takes place when an attacker injects malicious code into the application to gain unauthorized access to sensitive information stored in the database. SQLi represents a significant threat to data security and has been a major concern for more than two decades, primarily due to insufficient input validation and inadequate security measures in web applications utilizing Structured Query Language (SQL) databases [9], [10], [11], [12], [13].

Websites that do not implement adequate user input sanitization and validation are highly susceptible to SQLi attacks. In such vulnerabilities, malicious input injected by an attacker can be executed as part of an SQL command on the connected database, thereby enabling unauthorized access to the contents and functions of the database [14], [15], [16], [17]. The fundamental mechanism of SQLi typically occurs when SQL queries are dynamically constructed using data directly sourced from users without proper cleansing or parameterization. For example, in a simple authentication form, a query structured as "*SELECT * FROM users WHERE username = 'input_username' AND password = 'input_password'*"' can be manipulated if an attacker enters a pattern such as ' OR '1'='1 in one of the input fields. The resulting query becomes "*SELECT * FROM users WHERE username* = " OR *'1'='1'* AND password = '..."', where the condition '1'='1' always evaluates to true, thus allowing authentication bypass without valid credentials.

The impact of SQLi exploitation is extensive and severe. These impacts may include: (1) data theft, whereby attackers read and export sensitive information such as user identities, credentials, or financial data; (2) data modification, referring to unauthorized changes to the contents of the database that can compromise the integrity of the application; (3) data deletion, which can result in data loss and service disruption; (4) command execution at the system level in certain vulnerable database configurations; and (5) denial of service caused by the execution of resource-intensive queries that exhaust server resources. Given the potential consequences of these impacts, the prevention and detection of SQLi through rigorous input validation techniques, the use of parameterized queries or prepared statements, and the implementation of intrusion detection mechanisms are crucial steps in maintaining the confidentiality, integrity, and availability of website database systems.

Various methods have been developed to detect and prevent SQLi attacks. The most effective prevention methods involve implementing secure coding practices, such as using parameterized queries or Object-Relational Mapping (ORM) frameworks that automatically handle input sanitization. According to [18] Object-Relational Mapping (ORM) is a technique that maps objects in conventional programming languages to table structures within relational databases. ORM enables users to manipulate data in the database using objects and conventional programming methods, without the need to write precise SQL statements. Some popular examples of ORM include Hibernate for Java, Django ORM for Python, and Entity Framework for .NET. In addition, server-side user input validation is a crucial step to ensure that every incoming value meets the expected format and constraints. Nevertheless, not all websites are developed in accordance with best security practices, so SQLi vulnerabilities are still frequently encountered in legacy applications as well as in newly developed applications where security aspects are insufficiently addressed [19], [20], [21], [22], [23]. MariaDB is a popular open-source relational database management system that originated as a fork of MySQL. It

is widely used in web applications for its robust performance, high scalability, and strong security features. MariaDB supports advanced SQL capabilities, making it compatible with a variety of programming languages and frameworks. One of its key strengths lies in the active development community, which consistently delivers security patches and new features to address emerging threats and vulnerabilities. Additionally, MariaDB offers built-in support for encryption, role-based access control, and audit plugins, making it a secure choice for organizations concerned about SQL injection and other database-related attacks.

Based on these considerations, the implementation of SQLi detection as an additional security layer within a defense-in-depth framework remains necessary. Traditional SQLi detection methods are generally classified into three categories: signature-based detection, anomaly-based detection, and rule-based detection [24], [25], [26]. ignature-based detection operates by matching network activity against a database containing known attack "signatures." This method is fast and efficient for known threats, but it is ineffective against zero-day attacks [27]. Anomaly-based detection is a methodology that compares normal activity with suspicious activity in network traffic to observe and identify potential attacks [28]. Rule-based detection works by identifying relevant facts or data. The knowledge base contains essential information that is used to make decisions [29]. The main limitation of traditional detection methods lies in their difficulty in addressing the ever-evolving variations of SQLi attacks. Attackers continuously seek new ways to obfuscate SQLi payloads so they are not detected by existing signatures or rules. The structure of SQLi queries can be highly diverse, utilizing combinations of logical operators, database functions, comments, encoding, and other techniques to evade detection. Furthermore, the large volume of query data in modern web applications renders manual analysis or rule-based approaches impractical. Recognizing these limitations, machine learning and deep learning-based approaches are increasingly being applied in the field of cybersecurity, including for the detection of SQLi attacks.

In cybersecurity, Machine Learning (ML) can be utilized to classify network traffic, files, or user inputs as either normal or malicious, detect behaviors that deviate from typical patterns, build behavioral profiles of entities such as users or applications to identify suspicious activities, as well as classify and analyze malware samples. Deep Learning (DL) models have demonstrated superior performance in handling tasks that involve processing complex data, including images, audio, and text. Considering that SQL queries are essentially a form of text with significant structural and sequential elements, the application of Natural Language Processing (NLP) techniques based on Deep Learning models becomes highly relevant for the detection of SQLi attacks. The application of Deep Learning (DL) and Natural Language Processing (NLP) enables systems to analyze patterns and relationships among elements within SQL queries, thereby allowing for accurate differentiation between legitimate queries and those manipulated for malicious purposes. One of the most effective artificial neural network architectures in deep learning for processing sequential data is the Long Short-Term Memory (LSTM) network. LSTM differs from Recurrent Neural Networks (RNN), which are DL methods capable of learning patterns present in sequential data [30]. In RNNs, when processing very large datasets, the gradients tend to either vanish (vanishing gradient) or explode (exploding gradient) during training. This phenomenon makes it difficult for RNNs to learn from long-term dependencies. LSTM is a deep neural network model and an extension of the RNN architecture with enhanced design [31]. STM possesses a complex internal structure in the form of gating mechanisms that regulate the flow of information within memory cells. The forget gate determines which information from previous memory should be discarded, the input gate regulates new information to be stored in the memory cell, and the output gate controls the output value to be forwarded based on the current state of the memory cell.

The gates in LSTM allow the model to retain important information from previous time steps and pass it on to subsequent steps, while selectively discarding irrelevant information. This capability is crucial in SQL query analysis, where the sequence of keywords, operators, and string values determines both the meaning and the potential risk of a query. For instance, sequences such as ' OR '1'='1' or UNION SELECT are typical sequential patterns frequently encountered in SQLi attacks. LSTM is able to learn and recognize such patterns, even when there are variations in the structure of the query, thereby improving the accuracy of attack detection. The use of a Bi-directional LSTM (BiLSTM) architecture, as implemented in this study, is even more effective because BiLSTM processes input sequences in both forward and backward direction [32]. This allows the model to capture the context of words on both sides of an element within a sequence, which can provide a richer understanding of the structure of SQL queries and facilitate the detection of attack patterns that may only become apparent when viewed in the context of the entire sequence. Although the use of parameterized queries and input validation is often considered an ideal preventive method, the reality in website development

demonstrates that SQLi vulnerabilities remain a serious threat. The limitations of traditional detection methods in addressing ever-evolving attacks highlight the need for more adaptive and accurate approaches [33], [34], [35]. This research is motivated by the need for a robust SQLi detection system capable of recognizing complex and diverse attack patterns. By leveraging the capabilities of LSTM in processing sequential data and learning long-term dependencies, it is expected that the model will be able to distinguish between normal queries and SQLi queries with a high degree of accuracy.

Based on a review of several previous studies, the detection of SQLi attacks has been conducted using various approaches, yielding diverse levels of success. SQLi attacks most frequently occurred on TCP port 80 (3,060 instances) and TCP port 3005 (2,020 instances), with the dominant variation being the use of the sleep function with the select statement, occurring 3,888 times. These findings underscore the importance of strengthening network and application configurations to enhance security [36]. A CNN-based IDS utilizing VGG-16, combined with normalization and data augmentation techniques on the CICDDoS2019 and CSE-CIC-IDS2018 datasets, produces models capable of achieving exceptionally high accuracy: VGG-16 attains 99.92% for Syn Flood and 99.99% for SQLi, ResNet-50 achieves 99.92% and 99.99%, while InceptionV3 reaches 99.77% and 99.98%. These findings demonstrate the effectiveness of Deep Learning in detecting complex attacks with near-perfect accuracy [37]. Meanwhile, custom Snort rules, when used to detect various SQLi variants, yield an accuracy of 92.65%, a recall of 92.31%, no false positives, and only 7.69% false negatives. This performance is significantly superior compared to the default Snort rules, which only achieve an accuracy of 8.8% and a recall of 4.6%, thus this research effectively enhances the detection capability for SQLi attacks [38]. Other studies indicate that the use of Support Vector Machine (SVM) results in a model with an accuracy of 96.84%, accompanied by good precision and recall [39]. Urthermore, the Random Forest and SVM algorithms provide satisfactory detection results, with accuracies of 99.78% and 94%, respectively. Notably, the combination of SVM with PALOSDM successfully increases accuracy to 99% [40].

Table 2. Comparison of Attack Detection Approaches Based on Datasets and Evaluation Metrics

Detection Approach	Dataset/Port	Attack Type/Variant	Metrics
SQLi Detection	TCP port 80	SQLi attacks	Instances: 3,060
SQLi Detection	TCP port 3005	SQLi attacks	Instances: 2,020
SQLi Detection	-	Sleep function with select	Instances: 3,888
CNN-based IDS (VGG-16)	CICDDoS2019, CSE-CIC-IDS2018	Syn Flood	Accuracy: 99.92
CNN-based IDS (VGG-16)	CICDDoS2019, CSE-CIC-IDS2018	SQL Injection	Accuracy: 99.99
CNN-based IDS (ResNet-50)	CICDDoS2019, CSE-CIC-IDS2018	Syn Flood	Accuracy: 99.92
CNN-based IDS (ResNet-50)	CICDDoS2019, CSE-CIC-IDS2018	SQL Injection	Accuracy: 99.99
CNN-based IDS (InceptionV3)	CICDDoS2019, CSE-CIC-IDS2018	Syn Flood	Accuracy: 99.77
CNN-based IDS (InceptionV3)	CICDDoS2019, CSE-CIC-IDS2018	SQL Injection	Accuracy: 99.98
Custom Snort rules	-	SQL Injection variants	Accuracy: 92.65, Recall: 92.31, FP: 0, FN: 7.69
Default Snort rules	-	SQL Injection variants	Accuracy: 8.8, Recall: 4.6
Support Vector Machine (SVM)	-	-	Accuracy: 96.84
Random Forest	-	-	Accuracy: 99.78
SVM	-	-	Accuracy: 94
SVM + PALOSDM	-	-	Accuracy: 99

In comparison to these studies, the use of the LSTM model offers distinct advantages, as it is specifically designed to process sequential data such as SQL queries, which possess unique patterns and structures. Unlike Snort, which is based on static rules, or CNNs that primarily focus on spatial pattern extraction, LSTM is more adaptive in capturing long-term dependencies within textual data. Consequently, the LSTM approach has the potential to provide more contextual, accurate, and relevant detection results in identifying SQLi attacks. Meanwhile, this study involves the development and integration of a comprehensive dataset containing both normal queries and SQLi queries from various sources, followed by effective data preprocessing, including handling duplicates, missing values,

cleaning irrelevant queries such as gibberish or excessively long normal queries, and performing tokenization and padding on query texts. Subsequently, the researchers designed and implemented the architecture of a BiLSTM model for SQL query classification, trained the model using the processed dataset, and evaluated its performance based on standard classification metrics such as accuracy, precision, recall, and F1-score on test data. The trained model, along with its tokenizer, is then saved for future use. Through this research, it is expected that the effectiveness of the LSTM-based approach in building an accurate, adaptive SQLi detection system capable of providing a robust security layer for websites can be demonstrated. The documentation of this study covers every stage of the process, from data preparation and model development to performance evaluation, as well as a demonstration of the application of the trained model for real-time detection of new queries.

2. Method

This study employs a deep learning-based BiLSTM model to detect SQLi attacks in MariaDB. In addition, this research applies natural language processing (NLP) during preprocessing to further enhance the level of accuracy. The research stages are carried out through several processes, namely data collection, data preprocessing, model architecture development, model training, and performance evaluation. All stages are systematically designed to ensure that the dataset used is of high quality, the constructed model achieves optimal performance, and the evaluation results are academically accountable.

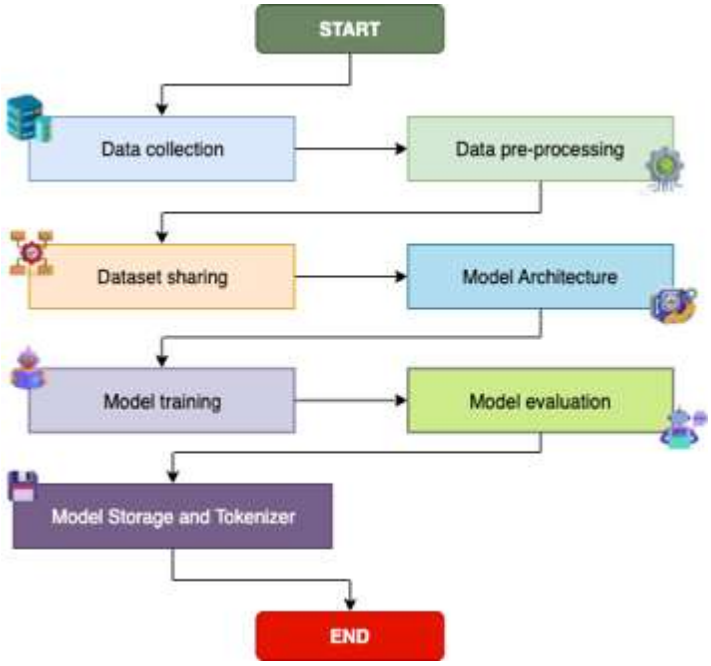


Figure 2. Research Steps for SQLi Detection Using BiLSTM

2.1 Dataset

The dataset is a crucial component in training deep learning models. To develop a robust SQLi detection model, this study collects datasets from various publicly available sources. The acquired datasets consist of collections of SQL queries, including both normal queries and those containing SQLi payloads, along with their respective labels, where a value of 0 typically denotes a normal query and a value of 1 indicates a SQLi query. The sources of the datasets in this research include several publications on the Kaggle platform, as presented in Table 3 below:

Table 3. Datasets Used

No	Dataset Name	Dataset Source Link
1.	SQLi Dataset Rayten	https://www.kaggle.com/datasets/rayten/sql-injection-dataset/
2.	SQLi Dataset Ayahkhaldi	https://www.kaggle.com/datasets/ayahkhaldi/sql-injection-dataset/
3.	SQLi Sajid576	https://www.kaggle.com/datasets/sajid576/sql-injection-dataset/
4.	SQLi XSS dataset AlexTrinity	https://www.kaggle.com/datasets/alextrinity/sqli-xss-dataset/

5.	Biggest SQLi Dataset Gambleryu	https://www.kaggle.com/datasets/gambleryu/biggest-sql-injection-dataset

Each dataset was accessed and loaded into a DataFrame using the pandas library. The integration of datasets from various sources aims to enhance the diversity of query patterns, including both normal and SQLi types, enabling the model to learn a broader spectrum of attack scenarios and thereby achieve better generalization.

2.2 Data Preprocessing

Data preprocessing is the process of cleaning and preparing data before it is further processed [41]. This study employs Natural Language Processing (NLP) methods to facilitate the preparation of data for subsequent processing. The NLP techniques applied include data cleaning, gibberish query detection, distribution analysis, and tokenization. This preprocessing stage is crucial for cleaning, standardizing, and preparing textual data to conform to the input format required by the LSTM model. Consequently, the accuracy of SQLi attack detection can be improved. Figure 3 show how data preprocessing done in this study. The preprocessing steps undertaken are as follows.

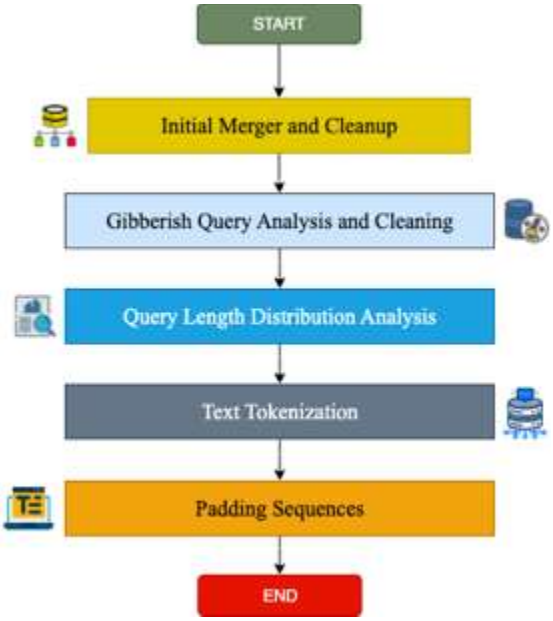


Figure 3. Data Preprocessing Stages

2.2.1 Merging and Initial Cleaning

All DataFrames obtained from various sources were first standardized in terms of column names and the data types of their label columns, and then merged into a single DataFrame. After merging, the DataFrame index was reset to ensure consistency. Initial cleaning was conducted to enhance data quality prior to model training. Duplicate data were removed to prevent bias caused by repeated queries. Additionally, rows containing missing values (NaN) in the 'Query' or 'Label' columns were deleted using the dropna() function to ensure data completeness in these key columns. Subsequently, the 'Label' column, which may have originally contained mixed data types, was converted to the integer data type using astype(int) to meet the requirements of binary classification tasks.

2.2.2 Gibberish Query Analysis and Cleaning

In some cases, SQLi queries appear as random strings of characters or lack clear meaning, particularly when attackers employ obfuscation techniques. However, normal queries may also exhibit unexpected formats, necessitating efforts to ensure dataset quality so that the model focuses solely on relevant attack patterns. To this end, detection and handling of queries categorized as gibberish—queries composed only of non-alphanumeric characters such as symbols or punctuation marks—were performed. This process involved defining a function named is_gibberish that utilizes regular expressions to identify queries that do not contain any letters (a–z, A–Z) or numbers (0–9). Queries

detected as gibberish and labeled as normal (0) were assumed to be noise data that are not relevant and were therefore removed from the dataset. This step aims to prevent the model from learning random patterns that do not represent the true structure of normal queries.

2.2.3 Query Length Distribution Analysis

Although the LSTM model has the capability to process long sequential data, there are limitations regarding input length due to computational constraints. Additionally, the presence of queries with abnormal lengths, particularly in data labeled as normal, often indicates noise such as fragments of HTML code, scripts, or other lengthy text that is irrelevant to the structure of SQL queries [42], [43], [44]. To address this issue, the researchers conducted an analysis of query length distribution by adding a length column representing the number of characters in each query. The distribution of query lengths was then visualized using histograms and boxplots to identify general characteristics and detect the presence of outliers. The analysis revealed that a significant number of normal queries (Label 0) exhibited extreme lengths, exceeding 3,000 characters, which implicitly indicates that these are not typical normal SQL queries. Therefore, excessively long queries were removed to prevent interference with the tokenization process and model learning.

2.2.4 Text Tokenization

Deep learning models can only process data in numerical form; therefore, query text must first be converted into a numerical representation. This is accomplished through the tokenization stage, utilizing the Tokenizer object from tensorflow.keras.preprocessing.text. During initialization, the tokenizer is configured with the parameter oov_token=" " to accommodate words that are not included in the vocabulary generated during training. Subsequently, the tokenizer is trained on all query texts in the dataset (fit_on_texts), resulting in a dictionary (word index) that maps each unique word to a specific numerical index. Once the dictionary is established, each query text is transformed into a sequence of numerical indices according to this mapping (texts_to_sequences). Thus, the query text is converted into a numerical format suitable for further processing by the model.

2.2.5 Sequence Padding

Neural network models require inputs with uniform dimensions. Since tokenization of queries produces sequences of varying lengths, it is necessary to normalize these lengths using a padding technique, implemented through the pad_sequences function from tensorflow keras preprocessing sequence. At this stage, the maximum sequence length (max_length) is set to 150, based on an analysis of query length distribution and considerations balancing relevant information coverage with computational efficiency. Each sequence shorter than max_length is supplemented with zeros (padding) at the end (padding='post'), while sequences exceeding this length are truncated at the end (truncating='post'). The result of this process is a NumPy array with dimensions (number_of_data, max_length), which is subsequently ready for use as input to the model.

2.3 Dataset Partitioning

The pre-processed dataset, consisting of x as input features and y as labels, was subsequently divided into two parts: training data and testing data. At this stage, the dataset was split with a ratio of 80% for training data and 20% for testing data, in accordance with the parameter test_size=0.2. Additionally, the researchers utilized the stratify=y parameter to maintain the proportion of label distribution between normal queries and queries containing SQLi attacks, ensuring balance in both data subsets. The implementation of stratification is crucial to minimize potential bias that may arise from class imbalance in the dataset. To ensure that the data splitting process is consistent and reproducible in every code execution, the random_state parameter was set to 42. Thus, each time the function is executed, the data split result will be identical, thereby enhancing the reproducibility of the experiment.

2.4 Model Architecture

The model employed in this study is an artificial neural network based BiLSTM. The selection of the BiLSTM architecture is motivated by its advantages in processing sequential data in both forward and backward directions[45], [46], [47]. This approach enables the model to capture richer contextual information from the structure of SQL queries, thereby enhancing its ability to detect complex patterns. Overall, the constructed model architecture consists of several layers. The first stage is the input layer, which receives input in the form of numerical sequences resulting from tokenization and padding processes with a specified maximum length. This input is then processed by the embedding layer, which functions to transform word indices into dense vector representations of fixed dimensions. This layer

not only maps words into vector space but also learns semantic representations that support model generalization. Subsequently, the embedding output is processed by the BiLSTM layer, which utilizes 64 LSTM units. With information flow in both directions, this layer is capable of capturing contextual relationships from the beginning to the end of the query and vice versa. To reduce the risk of overfitting, a dropout layer with a neuron dropout rate of 50 percent is added. This layer randomly deactivates a portion of neurons during training so that the model does not become overly dependent on specific patterns. The final stage of the architecture is the dense layer, which serves as the output layer. This layer consists of a single neuron with a sigmoid activation function, producing a probability value between 0 and 1. This probability represents the likelihood that a given query belongs to the SQLi attack class. The model is constructed using the Functional API approach from TensorFlow Keras, which allows each layer to be structurally connected from input to output. With this design, the model is expected to perform binary classification effectively, distinguishing between normal queries and those containing SQLi attacks.

2.6 Model Training

The designed BiLSTM model was subsequently trained using the training data (X_train, y_test) through parameter configurations tailored for binary classification tasks. The loss function employed was binary cross-entropy, which is widely recognized as the standard choice for two-class classification due to its ability to accurately measure the difference between the model's predicted probabilities and the actual labels. Network weight optimization was performed using the Adam optimizer, selected for its adaptive nature, efficiency, and proven stable performance across various DL models [48], [49], [50], [51]. During the training process, the monitored metric was accuracy, which measures the percentage of correct predictions out of the total data. The model was run for 5 epochs, with each epoch representing a complete cycle of the entire training data passing through the network. To improve computational efficiency, the training data was divided into batches of 256, allowing for incremental weight updates after each batch was processed. Additionally, to monitor the model's generalization, a validation split of 10% of the training data was used. Thus, a small portion of the data was separated as validation data, which was not used for weight updates but served to evaluate the model's performance on unseen data during training, while also helping to detect symptoms of overfitting. The entire training process was executed using the fit function, with the verbose parameter set to 1 so that the training progress for each epoch could be displayed in detail.

2.7 Model Evaluation

After the training process is completed, the BiLSTM model is evaluated using the test data (X_{test}, y_{test}), which has not been involved during training, so that the evaluation results can represent the generalization performance of the model. The evaluation stage is carried out through several steps. First, the model generates predictions in the form of class probabilities using the sigmoid activation function. These probabilities are then converted into binary labels by setting a threshold of 0.5. Thus, if $P(y = 1 \mid x) > 0{,}5$, the input is classified as an SQLi attack (label 1), whereas if $P(y = 1 \mid x) \leq 0{,}5$, the input is categorized as a normal query (label 0). The model's performance is measured using commonly used classification evaluation metrics: accuracy, precision, recall, and F1-score [52]. Accuracy is calculated as the ratio of correct predictions to the total number of test data, which is formulated as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Precision is used to measure the accuracy of the model in predicting the positive class, and is calculated using the following formula:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall (or sensitivity) is used to assess the model's ability to detect all positive data, using the following formula:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

Meanwhile, the F1-score provides a measure that balances both precision and recall, defined as the harmonic mean of the two:

$$F1\text{-}score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (4)$$

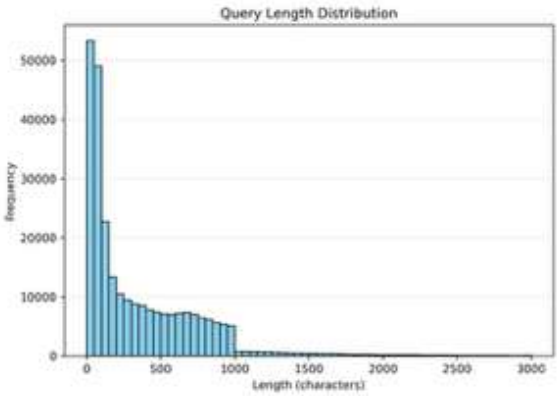
In addition to the aforementioned metrics, an analysis was also conducted using a classification report, which presents precision, recall, F1-score, and support for each class (0 and 1). Furthermore, a confusion matrix was constructed to illustrate the number of correct and incorrect predictions for each class, consisting of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). This confusion matrix was subsequently visualized in the form of a heatmap to facilitate interpretation. The evaluation results obtained from the metrics, classification report, and confusion matrix were further analyzed to assess the extent to which the model is capable of identifying SQLi and distinguishing from normal queries.

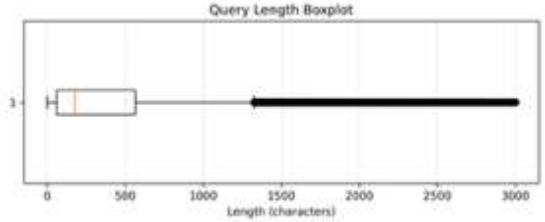
2.8 Model Deployment and Tokenizer

After completing the training and evaluation stages, the trained BiLSTM model was saved along with the tokenizer object used during the pre-processing phase. The model was stored in two formats, namely .keras and HDF5 (.h5). Storing the model in these formats aims to allow the model to be reloaded for inference or future testing purposes without the need for retraining, thereby maintaining computational efficiency. In addition to the model, the tokenizer object was also saved using the pickle library. The tokenizer plays a crucial role in converting input data in the form of text queries into numerical representations according to the scheme used during training. Thus, when the model receives new data, the transformation of text into numerical form remains consistent with the training data distribution. This saving step ensures that all key components, both the model and the pre-processing mechanism, are available and can be easily reintegrated during deployment or subsequent experiments. Overall, this research methodology follows the standard workflow in developing deep learning models for text classification, with specific adjustments to handle SQL query datasets. The selection of the BiLSTM architecture was based on its ability to capture bidirectional sequential patterns, thereby enabling a better representation of complex query structures.

3. Result And Discussion

The following presents the results and discussion for each stage of the research conducted, covering the initial characteristics of the dataset, the impact of pre-processing, the architecture and training process of the BiLSTM model, and a comprehensive evaluation of the model's performance in detecting SQLi attacks. The experimental results of the BiLSTM model trained to detect SQLi queries are presented through a series of tables and visualizations. These visualizations provide a comprehensive overview of the model's performance on the test data, which consists of 53,323 queries. The data pre-processing stage involved merging several datasets, tokenization, sequence padding, and the removal of irrelevant queries (such as gibberish in normal queries), ensuring that the model received clean and consistent data for both training and evaluation.





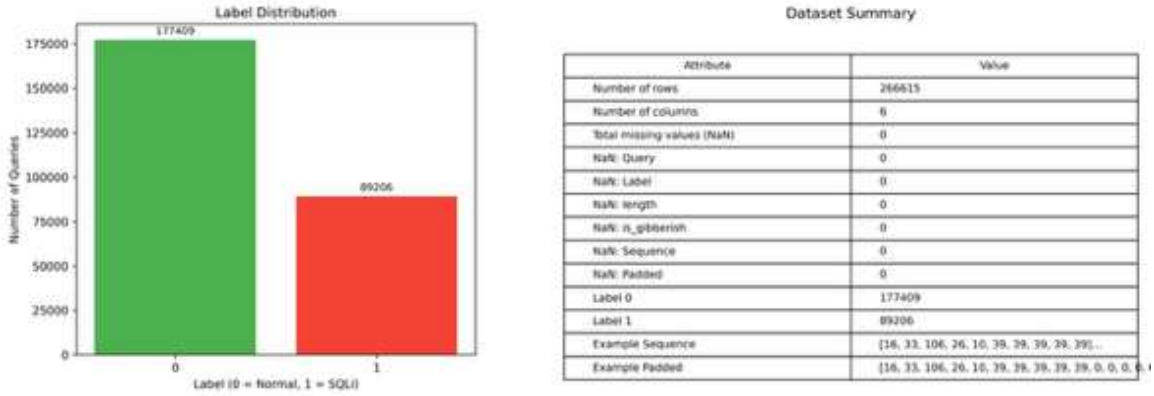


Figure 4. Visualization of the dataset after completing the data pre-processing stage

The BiLSTM model was constructed with the following architecture: an input layer with dimensions of (150), an embedding layer producing an output dimension of 128, a BiLSTM layer comprising 64 units, a dropout layer with a rate of 0.5, and a dense output layer utilizing a sigmoid activation function for binary classification. The model was compiled using the binary_crossentropy loss function, the Adam optimizer, and accuracy as the evaluation metric. The training process was conducted over 5 epochs with a batch size of 256 and a validation split of 0.1.

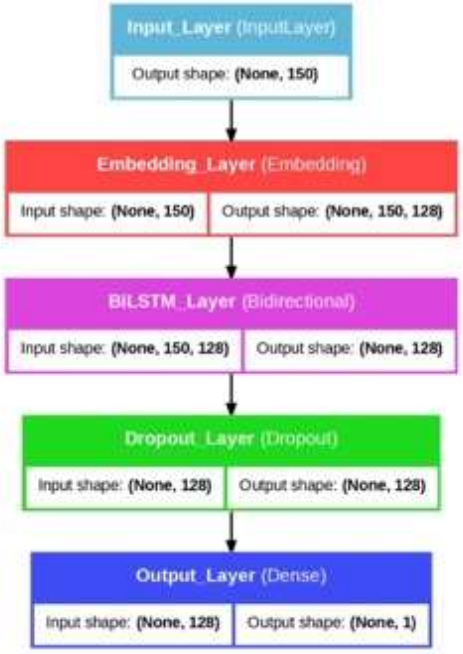


Figure 5. Implementation on BiLSTM Network Architecture for SQLi Attack Detection

3.1 Model Accuracy

After the BiLSTM model was trained, evaluation was conducted using previously unseen test data. The model evaluation was performed on test data that had not been encountered before. The evaluation results are presented in the form of a classification report and confusion matrix, which are summarized in Table 4.

Table 4. Evaluation Results of the Bidirectional LSTM Model on Test Data

Metric	Value	Interpretation
Overall Accuracy	99%	The percentage of total queries (Normal + SQLi) correctly classified by the model.
Precision (Normal, 0)	99%	Of all queries predicted as Normal, 99% are truly Normal queries. The False Positive rate is very low.

Metric	Value	Interpretation
Recall (Normal, 0)	100%	Of all actual Normal queries, 100% were correctly detected. The model did not miss any Normal queries (low False Negative).
F1-Score (Normal, 0)	99%	The harmonic mean between Precision and Recall for the Normal class, showing balanced performance.
Precision (SQLi, 1)	99%	Of all queries predicted as SQLi, 99% are truly SQLi attacks.
Recall (SQLi, 1)	98%	Of all actual SQLi queries, 98% were correctly detected. False Negative is very low, indicating maintained security.
F1-Score (SQLi, 1)	99%	The harmonic mean between Precision and Recall for the SQLi class, showing balanced performance.
Macro Avg	99%	The average of metrics (Precision, Recall, F1) across all classes, providing an overall performance overview.
Weighted Avg	99%	The weighted average of metrics based on the number of queries in each class, reflecting overall performance considering label distribution.

3.2 Confusion Matrix

The evaluation results are presented in the form of a classification report and confusion matrix, along with a visualization of classification percentages (Figure 6). The confusion matrix displays the number of True Positives (correctly detected SQLi), True Negatives, False Positives, and False Negatives. The visualization of classification percentages emphasizes that the proportion of correct predictions is very high, while incorrect predictions (off-diagonal) are very low, indicating that the model performs exceptionally well in distinguishing between Normal queries and SQLi attacks.

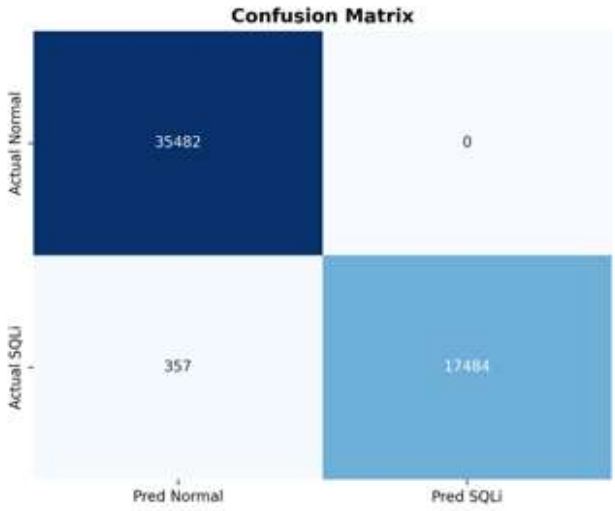


Figure 6. Confusion matrix

Based on the confusion matrix, this artificial intelligence model demonstrates exceptional performance in distinguishing between normal activities and SQLi-type cyber attacks. The model accurately identified 35,482 activities as "Normal" and correctly detected 17,484 attacks as "SQLi." Furthermore, the model missed only a small number of attacks, namely 357.

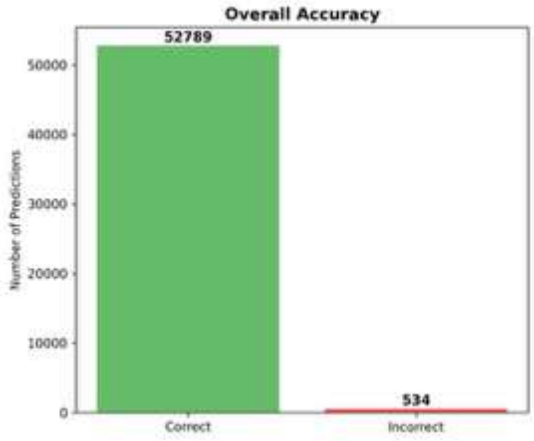


Figure 7. Overall Accuracy

The figure 7 provides an overview of the model's overall accuracy level. It is evident that the model's performance is exceptionally high. Out of all predictions made, a total of 52,789 were correct, as indicated by the prominent bar in the diagram in figure 8. Conversely, the number of incorrect predictions made by the model is very small, amounting to only 534. This stark visual contrast between the number of correct and incorrect predictions underscores the model's excellent overall accuracy in performing its task.

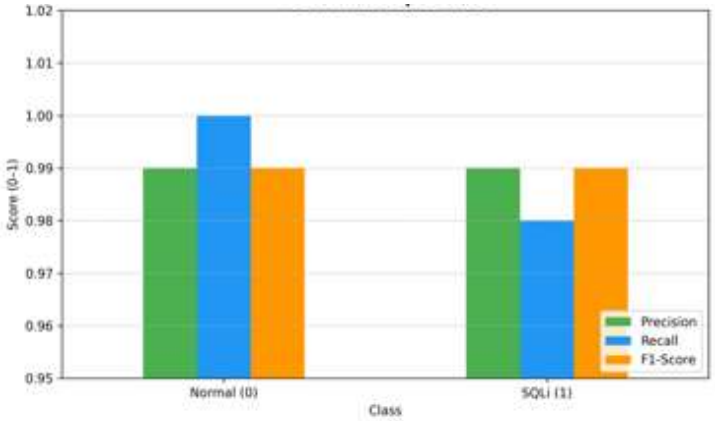


Figure 8. Performance by Class

This chart demonstrates that the model is highly effective at recognizing "Normal" data and never mistakenly classifies it as an attack (Recall 1.0). However, its performance in detecting "SQLi" attacks is slightly lower. This indicates that while the model successfully identifies the majority of attacks, there remains a small portion of attacks that are not detected.

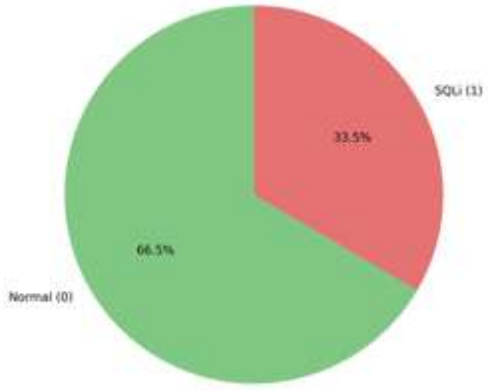


Figure 9. Distribution Testing

Figure 9. illustrates the composition of data used to test the model. It is evident that the dataset is imbalanced, with the majority—66.5%—consisting of "Normal" traffic, while the remaining 33.5% comprises "SQLi" attack data. This means the model was tested on a dataset where the number of normal data points is approximately twice that of attack data. Additionally, several new queries not included in the training or testing data were evaluated to assess the model's generalization capability. Normal queries, such as SELECT * FROM users WHERE username = 'admin' --, were correctly classified, whereas SQLi queries, like '1'='1', were predicted as attacks with high probability. Some ambiguous queries, such as DROP TABLE student;, were still classified as Normal, indicating that the model is able to distinguish attack patterns from legitimate SQL commands.

3.3 Model Resource in Production Environment

After the model was trained, the researchers saved and reloaded the model for use on the production server. This was done to observe how the trained model performs at the production level. In addition, this stage was carried out to determine the amount of resources consumed when running in a production environment. Based on testing on a production server with 8 cores, 12 GB of RAM, and 110 GB of disk storage, the model was able to operate effectively and efficiently with the following details.

Table 5. Using Models in Production Environments

	Resource Component	Usage
1	CPU (before model is loaded)	7.0%
2	CPU (after model is loaded)	Increase 4.0 % (total 11%)
3	RAM (before model is loaded)	10.7% (1.05GB)
4	RAM (after model is loaded)	Increase 1.69 GB (total 24%)

Based on resource usage monitoring during the model loading process, it was recorded that CPU utilization prior to loading the model was at 7.0%. After the model was loaded, CPU usage increased by 4.0%, bringing the total CPU utilization to 11%. Meanwhile, RAM consumption before loading the model was 10.7%, equivalent to 1.05 GB. Following the model loading process, there was an increase in RAM usage of 1.69 GB, resulting in a total RAM utilization of 24%. These data indicate that the loading process of the BiLSTM-based SQLi attack detection model has a considerable impact on resource consumption, particularly on CPU and RAM. However, this impact remains within reasonable limits for implementation in a production system. The following is a visualization:

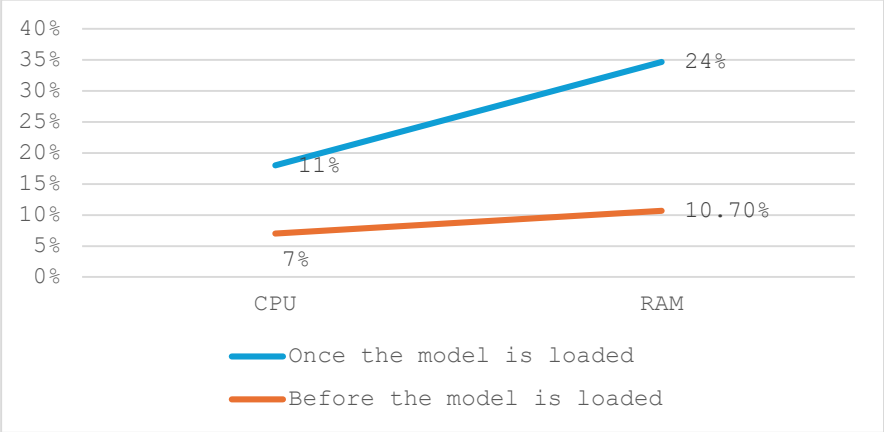


Figure 10. Use of models in a production environment

3.4 Model Performance in Production Environment

In addition to measuring resource usage in the production environment, the researchers also evaluated the average detection speed for SQLi attack queries. A total of 50,000 SQLi and Non-SQLi queries were utilized in this assessment. To ensure consistency in both value and processing speed, the detection of these 50,000 queries was performed 30 times. Based on the tests conducted on these queries, the BiLSTM model demonstrated an average detection speed of 0.000000206 seconds per query. The total time required by the model to process all queries was recorded at 54.1266 seconds.

3.5 Result Interpretation

The evaluation results of the BiLSTM model, as presented in Table 3 and visualized in Figure 6, provide a comprehensive overview of the model's performance on the test data consisting of 53,323 samples. Overall, the model demonstrates excellent capability in distinguishing between Normal and SQLi (SQLi) queries. The overall accuracy of the model is recorded at 0.99, indicating that 99% of the queries in the test data were correctly classified. The macro average and weighted average values for precision, recall, and f1-score, which consistently reach 0.99, reflect the model's stable performance across both classes, without bias toward either class. Specifically, for class 0 (Normal), a precision of 0.99 indicates that of all queries predicted as Normal, 99% are indeed truly Normal. A recall of 1.00 demonstrates that all Normal queries in the test data were correctly identified by the model, with no false negatives (FN), thus highlighting the model's reliability in recognizing benign queries. This finding aligns with the observations in Figure 6, where true negatives (TN) dominate the main diagonal of the confusion matrix, underscoring the model's effectiveness in identifying Normal query patterns. For class 1 (SQLi), a precision of 0.99 and a recall of 0.98 indicate that the majority of queries predicted as SQLi attacks are correct, although there is a small number of false negatives. The high proportion of true positives (TP) in the confusion matrix (Figure 6) confirms that the model is consistently able to detect attack patterns.

The low number of false positives (FP) in the Normal class and false negatives (FN) in the SQLi class suggest that the model maintains a good balance between sensitivity and specificity, thereby minimizing the risk of misclassification that could disrupt application operations or allow attacks to go undetected. In addition to quantitative evaluation, testing the model on new queries not included in the training or test datasets further demonstrates its generalization capability. Normal queries such as "SELECT * FROM users WHERE username = 'admin' --" and "I watched this movie yesterday and it was great!" were correctly classified as Normal, while SQLi queries such as "'1' = '1'" were predicted as attacks with high probability. Some ambiguous queries, like "DROP TABLE student;", were correctly identified as Normal with a low probability of being detected as SQLi, indicating that the model does not merely memorize simple attack patterns but is also able to distinguish legitimate SQL contexts from potential attacks. Thus, this interpretation confirms that the BiLSTM model, through its architecture capable of capturing bidirectional sequential context, has successfully learned the essential characteristics of both Normal and SQLi SQL queries. The dataset visualization after preprocessing (Figure 4) supports these findings, as cleaner data distribution, handling of duplicates, outliers, and gibberish queries enable the model to effectively learn relevant patterns. Overall, the combination of thoroughly processed data, tokenization, padding, and the BiLSTM architecture results in superior classification performance, making it practically applicable for SQLi attack detection systems.

3.6 Model Performance Factor

The high performance achieved by the BiLSTM model, as presented in Table 3 and visualized in Figure 6, can be explained by several interrelated factors, ranging from the characteristics of the dataset and data preprocessing to the architecture and training mechanisms of the model.

3.6.1 Dataset Characteristic

The dataset used in this study was compiled from five different sources with heterogeneous characteristics (see Section 1. Dataset Characteristics and Preprocessing). Careful preprocessing resulted in a final dataset consisting of 266,615 queries, with a class distribution of 177,409 Normal queries (66.54%) and 89,206 SQLi queries (33.46%). Although this distribution is not extremely imbalanced, it provides a sufficient proportion for the model to learn patterns from both classes. Furthermore, analysis of query length and outlier removal (Figure 4) ensured that excessively long or gibberish Normal queries were eliminated, thereby preventing the model from being disrupted by irrelevant noise. This is crucial, as long or random queries can hinder the model's ability to capture actual SQL syntax patterns.

3.6.2 Data Preprocessing and Representation

The tokenization and padding stages applied to the dataset ensure that each query is converted into a uniform numerical representation, with a maximum sequence length of 150 tokens. Padding and truncation maintain consistency of input into the LSTM network. This numerical representation, which is subsequently transformed into 128-dimensional embeddings, enables the model to learn the semantic relationships between tokens, rather than merely recognizing numbers. Such systematic preprocessing contributes to the model's ability to detect important patterns, in accordance with the standard evaluation formulas (1), (2), (3), and (4). These formulas are utilized in Table 3 to quantitatively assess

the model's performance. The high quality of the data ensures that the values of TN, TP, FP, and FN accurately reflect true characteristics of queries, thereby optimizing the model's accuracy and f1-score.

3.6.3 Bi-directional LSTM

The advantage of the BiLSTM architecture, as illustrated in Figure 5, lies in its ability to process SQL query sequences bidirectionally. This capability enables the model to comprehend the context of each token from both the beginning to the end and vice versa, which is essential for identifying attack patterns that may be concealed within SQL syntax. Furthermore, the inclusion of a Dropout layer set at 0.5 effectively prevents overfitting, thereby allowing the high performance achieved on the training data to be generalized well to the test data, as evidenced by validation accuracy consistently exceeding 99% during training.

3.6.4 Training Strategy Effect

The model was trained for 5 epochs with a batch size of 256 and a validation split of 0.1. This configuration is sufficient to learn the sequential patterns of queries without causing excessive overfitting. The rapid convergence of accuracy in the initial epochs indicates that SQLi patterns are clearly represented within the dataset, enabling the model to effectively learn the distinctive characteristics of attacks in a relatively small number of epochs. Overall, the high performance of the model is the result of synergy between a clean and representative dataset (Figure 4), thorough preprocessing, robust embedding representation, the BiLSTM architecture's ability to capture bidirectional context (Figure 5), and an efficient training strategy. Collectively, these factors explain why the model consistently achieves high precision, recall, f1-score, and accuracy values, as recorded in Table 3, and produces a prediction distribution that is predominantly concentrated along the main diagonal of the confusion matrix (Figure 6).

3.7 Theoretical Framework

The performance results of the BiLSTM model in detecting SQLi, as presented in Table 3 and visualized in Figure 6, can be associated with the theoretical principles of deep learning, machine learning, and the characteristics of SQLi.

3.7.1 Deep Learning and Sequential Representation

The BiLSTM model is a deep learning implementation capable of capturing sequential patterns in textual data. Theoretically, BiLSTM can process information in both forward and backward directions, enabling each token in an SQL query to be analyzed within the context of preceding and succeeding tokens. This capability aligns with the principles of sequential representation in deep learning, allowing the model to understand long-term dependencies and complex patterns that frequently occur in SQLi attacks. The high evaluation results on the test data demonstrate that this ability plays a significant role in distinguishing Normal queries from SQLi, particularly in cases involving ambiguous queries or those that resemble classic attack patterns such as ' OR '1'='1'.

3.7.2 Machine Learning and Generalization

Observations on new queries outside the training dataset (Figure 6) demonstrate the model's generalization capability. Machine learning theory emphasizes that a good model not only memorizes the training data but is also able to recognize patterns in previously unseen data. The balanced precision and recall values for both classes support this principle, indicating that the model has successfully internalized relevant features from SQL queries, thus enabling it to classify attacks with low false positive and false negative rates.

3.7.3 SQLi Characteristic

Conceptually, SQLi is an attack technique that exploits vulnerabilities in SQL query inputs to gain unauthorized access to or manipulate a database. The model's results, including its ability to distinguish between legitimate queries and malicious ones such as DROP TABLE student; (Figure 6), demonstrate alignment between empirical performance and the characteristics of attacks described in cybersecurity literature. This confirms that the BiLSTM model can recognize SQL syntax patterns susceptible to manipulation, consistent with the definitions and concepts of SQLi attacks. Overall, the model's high performance on evaluation metrics, the dominant prediction distribution along the main diagonal of the confusion matrix, and its ability to generalize to new queries all align with deep learning theory, machine learning principles, and security concepts related to SQLi. In other words, the findings of this study support the theory that the BiLSTM model is capable of learning and identifying complex patterns in sequential data, including those indicative of potential attacks on database systems.

3.7.4 Comparison with Previous Studies

The evaluation results of the Bi-directional LSTM model developed in this study (Table 3 and Figure 6) demonstrate exceptionally high performance, with an overall accuracy reaching 99%, precision and recall values of 0.99 for the Normal class, and 0.99 and 0.98 respectively for the SQLi class. The F1-score is recorded at 0.99 for both classes, indicating an optimal balance between the model's ability to detect attacks and its accuracy in classifying normal queries. This is the explanation when compared to previous studies:

a) This study emphasized the identification of SQLi attack patterns by analyzing Sophos XG330 firewall logs. The approach was primarily analytical, focusing on specific TCP ports to gain insights into attack patterns. However, it did not implement a predictive model based on deep learning and did not provide quantitative classification metrics such as accuracy, precision, or recall. In contrast, this study utilizes a Bi-directional LSTM model, which offers measurable performance metrics and a more adaptive detection capability for SQLi attacks [36].
b) Utilized CNN architectures (VGG16, ResNet50, InceptionV3) to detect SQLi and SYN Flood attacks. The best results were achieved by VGG16 with an accuracy approaching 100%, while ResNet50 and InceptionV3 achieved slightly lower accuracies. Although the accuracy is high, CNNs tend to focus more on spatial features and may be less optimal in capturing long-term sequential dependencies in SQL queries compared to the LSTM approach used in this study. BiLSTM approach in this tudy, on the other hand, is better suited for sequential data, making it more effective in detecting complex SQLi pattern [37].
c) Implemented Snort rules to detect variants of SQLi, resulting in good accuracy but still exhibiting false negatives for complex payloads. This rule-based approach demonstrates limitations in addressing newer or obfuscated attack variations, whereas the BiLSTM model is capable of adaptively recognizing sequential patterns, including queries that are ambiguous yet legitimate. BiLSTM model demonstrates superior adaptability by learning sequential patterns and effectively identifying both clear and ambiguous SQLi queries [38].
d) Applied the SVM algorithm, obtaining an accuracy of 96.10%. These results indicate that conventional machine learning is fairly effective, but still falls short compared to the performance of the BiLSTM model in this study (99% accuracy) [39].
e) Reported the use of Random Forest and SVM algorithms with accuracies of 99.78% and 94%, respectively, while the combination of SVM with PALOSDM achieved 99%. Although the Random Forest's accuracy is slightly higher than that of the BiLSTM, the LSTM approach offers advantages in understanding the sequential structure of SQL queries, thus minimizing false negatives in complex payloads that are difficult to detect using rule-based or non-sequential algorithms [40].

Overall, this quantitative comparison underscores that the Bi-directional LSTM approach achieves an optimal balance between the detection of SQLi attacks and the classification of normal queries. Its principal advantage lies in the ability to process sequential data, capture long-term dependencies, and generalize to new queries, making this model more adaptive than rule-based algorithms (such as Snort), conventional CNNs, or SVMs.

3.8 Finding Implications

The evaluation results of the BiLSTM model in this study (Table 3 and Figure 6) have significant implications from both academic and practical perspectives in the field of cybersecurity. First, the model's high performance, with an accuracy of 99% and balanced precision-recall metrics, demonstrates that a sequential deep learning approach can effectively distinguish between normal and SQLi queries. This reinforces the understanding that sequential models, such as BiLSTM, are more adaptive in detecting complex attack patterns compared to static rule-based approaches or conventional machine learning algorithms. Second, the model's ability to handle new queries not present in the training dataset confirms its strong generalization capabilities. This is particularly important in real-world practice, as SQLi attacks frequently employ varied payloads and obfuscation techniques that may not always be covered in historical data.

Therefore, this model can serve as a key component in intrusion detection or prevention systems that are responsive to new threats. Third, the application of comprehensive data preprocessing—including outlier removal, elimination of gibberish queries, tokenization, and padding—demonstrates that data quality directly influences model performance. The practical implication is that organizations seeking to implement SQLi detection systems must pay close attention to input data quality as a

preliminary step before model training. Fourth, the low distribution of FP and FN indicates a balance between security and usability. Low FN rates minimize the risk of undetected attacks, while low FP rates ensure smooth system operations by not blocking legitimate queries. This affirms that the model excels not only in technical detection but also takes operational aspects into account for real-world implementation. Finally, from an academic standpoint, these findings expand the understanding of applying BiLSTM for web application security. The model proves that sequential deep learning approaches can serve as effective alternatives to traditional methods, while also laying the groundwork for future research to combine sequential models with hybrid techniques or more complex datasets. Thus, this study's contributions encompass both theoretical and practical aspects in enhancing the reliability of SQLi detection systems.

The performance analysis of the BiLSTM model demonstrates clear advantages; however, it must still be critically reviewed based on the data and findings obtained (Table 3 and Figure 6). The model achieved an accuracy of 0.99, with precision and recall scores of 0.99/1.00 for the Normal class and 0.99/0.98 for the SQLi class, respectively. The high recall value for the Normal class indicates the absence of false negatives, whereas a recall of 0.98 for the SQLi class shows that only 2% of attacks evaded detection. This distribution is reflected in the percentage confusion matrix in Figure 6, where the main diagonal indicates a dominant proportion of correct predictions, while off-diagonal entries are minimal. The distribution of false positives (FP) and false negatives (FN) highlights the trade-off aspect. The model recorded a low FP rate, which is important to avoid mistakenly blocking normal queries, and a minimal FN rate, which is crucial for security. These results indicate that the constructed BiLSTM model is not only robust in technical detection but also optimizes the balance between security and operational smoothness.

The characteristics of the dataset are a determining factor in the model's performance. The training data consisted of 213,292 queries after preprocessing, which included outlier removal, elimination of gibberish queries, tokenization, and padding up to a maximum length of 150 tokens (Figure 4). The label distribution of 66.54% Normal and 33.46% SQLi ensures that the model learns from a relatively balanced data representation, although some class imbalance remains. This effect is reflected in the model's ability to distinguish the minority class (SQLi) with high, albeit not perfect, recall. Testing on new queries confirmed the model's generalization capability. For example, Normal queries such as "SELECT * FROM users WHERE username = 'admin' --" and SQLi queries such as "'1' = '1'" were correctly classified according to their labels, while ambiguous queries such as "DROP TABLE student;" were classified as Normal with a low probability of being SQLi. These results show that the model does not merely memorize simple patterns but is capable of recognizing relevant sequential contexts, although it still faces challenges with queries exhibiting high syntactic ambiguity.

Identified limitations include the fact that, despite being substantial (53,323 test data), the dataset is still limited to certain SQLi attack variations and may not cover the latest payloads or advanced obfuscation techniques. The max_length restriction of 150 tokens also potentially results in the loss of important information from very long queries, even though most outliers have been removed as they were considered noise. Overall, the data demonstrate that the BiLSTM model is effective, stable, and reliable for detecting SQLi. This critical analysis underscores the need for dataset enrichment and further exploration of advanced architectures to ensure the model remains resilient against more complex attacks in real-world scenarios.

4. Conclusion

This study successfully developed an SQLi detection system based on deep learning by utilizing a BiLSTM architecture. The research stages began with the collection of five relevant public datasets, followed by a preprocessing process that included data merging, handling of duplicates and missing values, outlier removal, and the elimination of non-representative gibberish queries. The preprocessing resulted in a final dataset comprising 266,615 queries with a proportional distribution between the Normal and SQLi classes. The constructed BiLSTM architecture consists of an embedding layer, a BiLSTM layer, a dropout layer, and an output layer with sigmoid activation. The model was trained for five epochs with a batch size of 256 and a validation split of 0.1, demonstrating rapid and stable improvements in accuracy and achieving validation accuracy above 99% from the initial epochs. Evaluation of the model on a test set of 53,323 samples showed excellent performance, with an overall accuracy reaching 99%.

The classification report indicated balanced precision, recall, and F1-score values for both classes. The perfect recall value for the Normal class (1.00) confirms the model's ability to correctly

detect all Normal queries, while the high recall for the SQLi class (0.98) demonstrates the model's effectiveness in detecting the majority of attacks. The confusion matrix supports these findings, with dominant numbers of true positives and true negatives and very low false positives and false negatives. These results indicate that the BiLSTM-based approach can effectively learn the sequential patterns and context of SQL queries, making it superior to signature-based or simple rule-based methods. The model's success was also influenced by the quality of the clean, representative, and consistent data produced during preprocessing. Nevertheless, this study acknowledges certain limitations, including the possibility that the dataset does not cover all variations of attacks encountered in real-world scenarios and the restriction on query length (max_length = 150 tokens), which may reduce information in very long queries. Overall, this study demonstrates that BiLSTM is a highly effective approach for text-based SQLi detection, offering high performance, low error rates, and potential for implementation in modern web application security systems.

5. Conclusion

Based on the findings of this research, several important directions for future development have been identified. First, it is crucial to utilize more diverse and up-to-date datasets. Expanding the dataset will enable the model to better detect the most recent variations of SQLi attacks, including new obfuscation techniques and payloads that are not represented in the current dataset. This will enhance the model's robustness and generalization when facing evolving threats. Second, there is significant potential in exploring alternative or hybrid deep learning architectures. For example, combining CNN with Long BiLSTM networks, or incorporating attention mechanisms, may improve the ability of the model to recognize and interpret complex patterns within SQL queries. Such approaches could further optimize detection performance and adaptability. From a data preprocessing standpoint, leveraging more advanced techniques could also yield substantial improvements. Methods such as employing pre-trained word embeddings or implementing specialized SQL query normalization have the capacity to enrich the semantic representation of queries, thereby increasing the overall accuracy and reliability of the model. Additionally, it is recommended that model testing be expanded beyond static datasets to real-world environments. Integrating the model into IDS or IPS would allow for performance evaluation using dynamic website traffic, providing insights into the model's effectiveness under practical conditions. Finally, optimizing the computational efficiency of the model is an important step toward practical implementation. Ensuring that the system can operate effectively in real-time scenarios, especially in environments with limited computational resources, will be essential for widespread adoption. Through these future development steps, deep learning-based SQLi detection systems are expected to evolve into even more reliable, adaptive, and relevant security solutions capable of addressing increasingly complex cyber threats.

References

[1] A. Aikido, "The State of SQL Injection." Diakses: 27 September 2025. [Daring]. Tersedia pada: https://www.aikido.dev/blog/the-state-of-sql-injections

[2] K. Chan, R. Gururajan, dan F. Carmignani, "A Human–AI Collaborative Framework for Cybersecurity Consulting in Capstone Projects for Small Businesses," *J. Cybersecurity Priv.*, Mei 2025, doi: 10.3390/jcp5020021.

[3] R. Kashef *dkk.*, "Bridging the Bubbles: Connecting Academia and Industry in Cybersecurity Research," *2023 IEEE Secure Dev. Conf. SecDev*, hlm. 207–213, Feb 2023, doi: 10.1109/SecDev56634.2023.00034.

[4] L. Miller dan M.-O. Pahl, "Collaborative Cybersecurity Using Blockchain: A Survey," *ArXiv*, vol. abs/2403.04410, Mar 2024, doi: 10.48550/arXiv.2403.04410.

[5] H. Zafar, C. Hollingsworth, T. Bandyopadhyay, dan A. Randolph, "Collaborative Pathways to Cybersecurity Excellence: Insights from Industry and Academia in the Southeastern US," *J. Cybersecurity Educ. Res. Pract.*, Jul 2024, doi: 10.62915/2472-2707.1183.

[6] N. Mmango dan T. Gundu, "Cultivating Collective Armor: Towards a Collaborative Cybersecurity Resilience Framework for SMEs," *Eur. Conf. Innov. Entrep.*, Sep 2024, doi: 10.34190/ecie.19.1.2799.

[7] E. Edgescan, “2024 Vulnerability Statistics Report 9th Edition,” 2025. [Daring]. Tersedia pada: https://www.edgescan.com/wp-content/uploads/2025/04/2024-Vulnerability-Statistics-Report.pdf?utm_source=chatgpt.com
[8] V. Abdullayev dan Dr. A. S. Chauhan, “SQL Injection Attack: Quick View,” *Mesopotamian J. CyberSecurity*, vol. 2023, hlm. 30–34, Feb 2023, doi: 10.58496/MJCS/2023/006.
[9] M. Souza, S. Ribeiro, V. Lima, F. Cardoso, dan R. Gomes, “Combining Regular Expressions and Machine Learning for SQL Injection Detection in Urban Computing,” *J Internet Serv Appl*, vol. 15, hlm. 103–111, Jul 2024, doi: 10.5753/jisa.2024.3799.
[10] A. Tagde, P. Jibhakate, Y. Pimpalikar, R. Agrawal, C. Dhule, dan N. Morris, “Comprehensive Data Access Protection Suite,” *2025 Int. Conf. Comput. Commun. Inf. Technol. ICCCIT*, hlm. 658–663, Feb 2025, doi: 10.1109/ICCCIT62592.2025.10928082.
[11] A. Hariyani dan P. Dolia, “Comprehensive Review of Advanced Techniques for Mitigating SQL Injection Vulnerabilities in Modern Applications,” *Int. J. Innov. Sci. Res. Technol.*, Apr 2025, doi: 10.38124/ijisrt/25mar1982.
[12] V. Babaey dan A. Ravindran, “GenSQLi: A Generative Artificial Intelligence Framework for Automatically Securing Web Application Firewalls Against Structured Query Language Injection Attacks,” *Future Internet*, vol. 17, hlm. 8, Des 2024, doi: 10.3390/fi17010008.
[13] D. Muduli *dkk.*, “SIDNet: A SQL Injection Detection Network for Enhancing Cybersecurity,” *IEEE Access*, vol. 12, hlm. 176511–176526, 2024, doi: 10.1109/ACCESS.2024.3502293.
[14] T. Ali, R. Dipke, V. Dhanshetti, dan N. Gadepally, “Data Leaks Using SQL Injection,” *Int. J. Adv. Res. Sci. Commun. Technol.*, Des 2023, doi: 10.48175/ijarsct-14255.
[15] A. Pawar, N. Kapadnis, P. Joshi, V. Kalyankar, R. Gharat, dan V. Khokle, “Detecting Data Leaks due to SQL Injection,” *INTERANTIONAL J. Sci. Res. Eng. Manag.*, Des 2024, doi: 10.55041/ijsrem39512.
[16] B. P. Singh dan Prof. M. K. Singhal, “Detection of SQL Injection Attack Using Machine Learning Techniques,” *Int. J. Sci. Res. Sci. Technol.*, Des 2024, doi: 10.32628/ijsrst24114323.
[17] Meenakshi dan Murugan, “Understanding the Threat: Exploring SQL Injection Attacks and Prevention Strategies,” *Int. Res. J. Mod. Eng. Technol. Sci.*, Mei 2024, doi: 10.56726/irjmets57129.
[18] A. A. Adriansyah dan M. I. P. Nasution, “KAJIAN TENTANG PERAN PENTING BASIS DATA BAGI PERPUSTAKAAN,” vol. 1, 2024, doi: https://doi.org/10.61722/jinu.v1i4.1819.
[19] A. Hariyani dan P. Dolia, “Comprehensive Review of Advanced Techniques for Mitigating SQL Injection Vulnerabilities in Modern Applications,” *Int. J. Innov. Sci. Res. Technol.*, Apr 2025, doi: 10.38124/ijisrt/25mar1982.
[20] S. S. Chinthalapudi, “Detecting and Mitigating SQL Injection in .NET Applications Using AI-Based Anomaly Detection,” *Int. J. Innov. Sci. Res. Technol.*, Apr 2025, doi: 10.38124/ijisrt/25mar1676.
[21] M. Begum, L. S. C, dan M. P, “Enhancement of Web Application Security using SQLMap and Machine Learning,” *Int. Res. J. Innov. Eng. Technol.*, Jan 2025, doi: 10.47001/irjiet/2025.inspire43.
[22] J. Tadhani, V. Vekariya, V. Sorathiya, S. Alshathri, dan W. El-Shafai, “Securing web applications against XSS and SQLi attacks using a novel deep learning approach,” *Sci. Rep.*, vol. 14, Jan 2024, doi: 10.1038/s41598-023-48845-4.
[23] A. Paul, V. Sharma, dan O. Olukoya, “SQL injection attack: Detection, prioritization & prevention,” *J Inf Secur Appl*, vol. 85, hlm. 103871, Sep 2024, doi: 10.1016/j.jisa.2024.103871.
[24] A. Bachir, A. Alali, A. Abed, H. Al-Jaberi, L. Dalloul, dan M. Uddin, “A Signature and NLP-based Network Traffic Detection Model For SQL Injections for Enhancing Web Security,” *2024 IEEEACM Int. Conf. Big Data Comput. Appl. Technol. BDCAT*, hlm. 91–96, Des 2024, doi: 10.1109/BDCAT63179.2024.00025.
[25] S. S. Chinthalapudi, “Detecting and Mitigating SQL Injection in .NET Applications Using AI-Based Anomaly Detection,” *Int. J. Innov. Sci. Res. Technol.*, Apr 2025, doi: 10.38124/ijisrt/25mar1676.
[26] A. Attri, P. Gundeboyena, V. Chigurla, S. Moluguri, dan N. Kasoju, “Network intrusion detection using hybrid approach,” *World J. Adv. Res. Rev.*, Feb 2025, doi: 10.30574/wjarr.2025.25.2.0367.
[27] A. M. R. Makkawaru dan H. Ashari, “Desain dan Analisis Kinerja Algoritma Pertahanan Aktif untuk Manajemen Aturan Firewall Melalui Simulasi Deteksi Intrusi Berbasis Python,” vol. 5, no. 1, 2025.

[28] O. D. Prasetyo, P. H. Trisnawan, dan A. Bhawiyuga, "Uji Kinerja Host-Based Intrution Detection System WAZUH terhadap Serangan Brute Force dan Dos," 2023.
[29] M. I. Ghozali, A. A. Riadi, D. A. Putra, dan W. H. Sugiharto, "Pengembangan Sistem Sortir Otomatis untuk Jeruk Citrus: Integrasi Teknologi Sensor dan Algoritma Rule-Based," vol. 4, no. 3, 2024.
[30] N. Lubis, Mhd. Z. Siambaton, dan R. Aulia, "Implementasi Algoritma Deep Learning pada Aplikasi Speech to Text Online dengan Metode Recurrent Neural Network (RNN)," *Sudo J. Tek. Inform.*, vol. 3, no. 3, hlm. 113–126, Sep 2024, doi: 10.56211/sudo.v3i3.583.
[31] Y. A. Susetyo, H. A. Parhusip, S. Trihandaru, dan B. Susanto, "LSTM-IOT (LSTM-based IoT) untuk Mengatasi Kehilangan Data Akibat Kegagalan Koneksi," *J. Teknol. Inf. Dan Ilmu Komput.*, vol. 12, no. 1, hlm. 175–186, Feb 2025, doi: 10.25126/jtiik.20251219157.
[32] R. R. Firdaus, "Rancang Bangun Sistem Monitoring Performa Ball Screw Berbasis LSTM-Autoencoder," 2024.
[33] N. S. Dasari, A. Badii, A. Moin, dan A. Ashlam, "Enhancing SQL Injection Detection and Prevention Using Generative Models," *ArXiv*, vol. abs/2502.04786, Feb 2025, doi: 10.48550/arXiv.2502.04786.
[34] A. Paul, V. Sharma, dan O. Olukoya, "SQL injection attack: Detection, prioritization & prevention," *J Inf Secur Appl*, vol. 85, hlm. 103871, Sep 2024, doi: 10.1016/j.jisa.2024.103871.
[35] Z. Gui *dkk.*, "SqliGPT: Evaluating and Utilizing Large Language Models for Automated SQL Injection Black-Box Detection," *Appl. Sci.*, Agu 2024, doi: 10.3390/app14166929.
[36] Celvine Adi Putra, Rianda Pratama, dan Tata Sutabri, "ANALISIS MANFAAT MACHINE LEARNING PADA NEXT-GENERATION FIREWALL SOPHOS XG 330 DALAM MENGATASI SERANGAN SQL INJECTION," *J. Manaj. Inform. Dan Sist. Inf.*, vol. 6, no. 2, hlm. 197–204, Jun 2023, doi: 10.36595/misi.v6i2.886.
[37] S. Sahren dan A. P. Lubis, "INTRUSION DETECTION SYSTEM BERBASIS DEEP LEARNING UNTUK PENINGKATAN MITIGASI SQL INJECTION DAN SYN FLOOD ATTACK," 2024.
[38] A. N. Maulana, M. Data, dan F. A. Bakhtiar, "Perancangan dan Implementasi Snort Rule Set untuk Deteksi Serangan SQL Injection," vol. 9, 2025.
[39] Pramono dan A. Arum Sari, "Prediksi Serangan Sql Injection Pada Jaringan Komputer Menggunakan Metode Support Vector Machine (SVM)," *J. TECNOSCIENZA*, vol. 8, no. 2, hlm. 317–326, Apr 2024, doi: 10.51158/tecnoscienza.v8i2.1184.
[40] A. Rahayu, E. Yulyanti, dan M. Ghalib, "Systematic Literature Review: SQL Injection Detection Vulnerability Using Machine Learning," vol. 21, 2025.
[41] C. S. Octiva, T. I. Fajri, E. B. Sulistiarini, S. Suharjo, dan U. W. Nuryanto, "Penggunaan Teknik Data Mining untuk Analisis Perilaku Pengguna pada Media Sosial," *J. Minfo Polgan*, vol. 13, no. 1, hlm. 1074–1078, Jul 2024, doi: 10.33395/jmp.v13i1.13936.
[42] R. K. R. Samantapudi, M. R. Dhanagari, dan S. Tarun, "Natural Language Query to SQL query generation using LSTMs, Transformers, LLMs and Gen AI," *Am. J. Technol.*, Mei 2025, doi: 10.58425/ajt.v4i1.352.
[43] D. Gandhi, A. Giri, dan S. Uparkar, "Overcoming Context Length Limitations in LLM's Integrating LSTM, Retrieval-Augmented Generation, and Agentic Frameworks for Enhanced Business Data Analysis," *Int. J. Multidiscip. Res.*, Jan 2025, doi: 10.36948/ijfmr.2025.v07i01.35767.
[44] H. Wei, "Research on the Application of Tree Model in Transforming Complex Natural Language Query Into Sql," *2025 Int. Conf. Digit. Anal. Process. Intell. Comput. DAPIC*, hlm. 379–384, Feb 2025, doi: 10.1109/DAPIC66097.2025.00076.
[45] R. Jin, Z. Chen, K. Wu, M. Wu, X. Li, dan R. Yan, "BiLSTM-Based Two-Stream Network for Machine Remaining Useful Life Prediction," *IEEE Trans. Instrum. Meas.*, vol. 71, hlm. 1–10, Jan 2022, doi: 10.1109/tim.2022.3167778.
[46] S. Singh dan S. Srivastava, "Enhancing the performance of deep learning models with fuzzy c-means clustering," *Knowl Inf Syst*, vol. 66, hlm. 7627–7665, Agu 2024, doi: 10.1007/s10115-024-02211-6.
[47] I. D. Mienye, T. Swart, dan G. Obaido, "Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications," *Inf.*, vol. 15, hlm. 517, Agu 2024, doi: 10.3390/info15090517.

[48] N. Das dan S. Begum, "An Empirical Study of Loss Functions for Aspect Category Detection in Imbalanced Data Scenario," *2025 10th Int. Conf. Signal Process. Commun. ICSC*, hlm. 247–252, Feb 2025, doi: 10.1109/ICSC64553.2025.10968230.

[49] U. Lilhore *dkk.*, "Hybrid convolutional neural network and bi-LSTM model with EfficientNet-B0 for high-accuracy breast cancer detection and classification," *Sci. Rep.*, vol. 15, Apr 2025, doi: 10.1038/s41598-025-95311-4.

[50] A. Rahman, S. Parvej, K. S. Alam, dan H. A. Fattah, "Optimizing SMS Spam Detection: Comparative Analysis of Hybrid Voting Ensembles and BiLSTM Networks with Stratified Cross-Validation," *2024 5th Int. Conf. Data Intell. Cogn. Inform. ICDICI*, hlm. 1030–1035, Nov 2024, doi: 10.1109/ICDICI62993.2024.10810777.

[51] J. Huang, G. Niu, H. Guan, dan S. Song, "Ultra-Short-Term Wind Power Prediction Based on LSTM with Loss Shrinkage Adam," *Energies*, Apr 2023, doi: 10.3390/en16093789.

[52] S. R. Darmawan, M. Fatchan, dan D. Maulana, "PREDICTION OF 2024 PRESIDENTIAL ELECTION USING K-NN WITH METRIC APPROACHES CHEBYSHEV AND EUCLIDEAN BASED ON TWITTER DATA INVESTIGATION," *J. Tek. Inform. Jutif*, vol. 5, no. 2, hlm. 475–485, Apr 2024, doi: 10.52436/1.jutif.2024.5.2.1720.