Iraqi Journal of Science, 2026, Vol. 67, No. 10, pp: 5428-5445
DOI: 10.24996/ijs.2026.67.10.25

ISSN: 0067-2904

Lightweight DL for Early SQLi Detection in Web Apps Security

Baydaa Flayyih Hasan*[1], Hadeel Sadeq Ahmed[2], Wafaa Ayoub Kassara[3]
[1]*Total Quality Management Techniques Department, Technical College of Management /Baghdad, Middle Technical University, Baghdad, Iraq*
[2]*Medical Devices Technology Department, Electrical Engineering Technical College, Middle Technical University, Baghdad, Iraq*
[3]*Accounting Techniques department, Technical College of Management /Baghdad, Middle Technical University, Baghdad, Iraq*

Received: 10/5/2025 Accepted: 8/1/2025 Published: 30/10/2026

Abstract

SQL injections are still one of the most significant threats to online applications. It is hard to discover SQL injection attacks since SQL queries are always changing. As these attacks become more complicated and happen more often, current methods of finding them that rely on keyword denial lists are becoming less successful. To address this problem, we suggest a new model that turns text-based features into images. This study introduces a Lightweight Deep Learning system for SQL Injection Attack Detection, designated as LWD-SQLIAS. The system has three main steps: first, it uses NLP approaches to preprocess the dataset. Then, it uses a hybrid feature extractor that combines TF-IDF with the neural layers of VGG16 filters to extract features and create image features that are 244×224 pixels in size. In the end, the proposed Lightweight MobileNetV2 (LW-MNV2) model, which uses the MobileNetV2 architecture and fine-tuning method, is used to optimize these feature images. The proposed LWD-SQLIAS was implemented on dataset 30919 of SQL injection samples obtained from Kaggle. The results show the LW-MNV2 performs better than the original MobileNetV2, achieving 99.85% accuracy, 99.74% precision, 99.86% recall, and 99.80% score.

Keywords: SQLi attack, VGG16, MobileNetV2, TF-IDF.

تأمين تطبيقات الويب بناءً على الكشف المبكر عن طريق حقن SQL وباستخدام طريقة خفيفة الوزن DL

بيداء فليح حسن*[1] , هديل صادق احمد[2], وفاء ايوب كسارة[3]
[1] قسم تقنيات ادارة الجودة الشاملة، الكلية التقنية الادارية , بغداد، الجامعة التقنية الوسطى، بغداد، العراق
[2] قسم تقنيات الاجهزة الطبية، الكلية التقنية الهندسية الكهربائية , بغداد، الجامعة التقنية الوسطى، بغداد، العراق
[3] قسم تقنيات المحاسبة، الكلية التقنية الادارية, بغداد ، الجامعة التقنية الوسطى، بغداد، العراق

الخلاصة

حقن SQL يظل أحد أكثر التهديدات انتشاراً لتطبيقات الويب. التقييم المستمر لبنية الاستعلام من قبل

*Email: Baydaa@mtu.edu.iq

Open Access Articles. This articles is licensed under

الهجمات يعقد اكتشاف هذه الاختراقات. مع زيادة تكرار وتعقيد مثل هذه الهجمات، أصبحت طرق الكشف التقليدية المعتمدة على قوائم منع الكلمات الرئيسية غير فعالة بشكل متزايد. لمعالجة هذا التحدي، اقترح الباحثون نموذجاً مبتكراً يحول الميزات النصية إلى صور، تقترح هذه الورقة نظام التعلم العميق الخفيف للكشف عن هجمات حقن SQL (LWD-SQLIAS) يتضمن هذا النظام ثلاث خطوات رئيسية: معالجة مسبقة لمجموعة البيانات باستخدام تقنية معالجة اللغة الطبيعية، يليها نموذج Deep TF-IDF المقترح المبني على TF-IDF ومرشحات VGG16 لاستخراج الميزات وإنتاج صور الميزات بحجم 244×224، وأخيراً، يتم التدريب المسبق لصور الميزات هذه باستخدام نموذج LW-MNV2 المقترح، الذي يستخدم بنية MobileNetV2 وتقنية الضبط الدقيق. تم تنفيذ LWD-SQLIAS المقترح على مجموعة بيانات تضم 30919 عينة من حقن SQL تم الحصول عليها من Kaggle تظهر النتائج أن LW-MNV2 يؤدي أفضل من MobileNetV2 الأصلي، محققاً دقة 99.85%، ودقة 99.74%، واسترجاع 99.86%، ودرجة 99.80%.

1. Introduction

A severe online vulnerability known as SQL injection lets attackers transmit specially designed input data to a web server to influence the communication between the server and the SQL database. This enables attackers to control SQL queries executed by server-side scripts, potentially extracting unauthorized data. Attackers may modify databases or run remote operations via the web server on the server side. Detecting SQL injection vulnerabilities is essential for ethical hackers and penetration testers to secure systems. Based on the Open Web Application Security Project (OWASP) 2021, which summarized the results of the Top 10 "Web Application Security threats", SQL Injection is ranked third among the most hazardous threats across 94% of tested apps, as well as 33 Common Weakness Enumeration (CWE) maps into the Injection category [1][2][3]. Organizations and the software development industry now prioritize preventing these kinds of attacks. To avoid negative consequences, early and accurate identification is essential [4]. Traditionally, preventing these kinds of attacks included checking data input for particular characters linked to frequent assaults and verifying it. Though still used today, this technique is no longer helpful in fending off other assaults [5]. Owing to the constant advancement of technology, companies and developers are always looking for new approaches to deal with this issue as it keeps coming up. Researchers have also recently proposed several detection methods, which may be generally divided into three categories: deep neural network-based, machine learning, and traditional [6]. The conventional method works well for typical string matching. Still, attackers are using more advanced tools these days to carry out automated injection attacks, and the classic method is often unable to keep up.

The machine learning-based method solves the problems with the old technique but has limitations, namely overfitting and the need to manually extract features through filtering [7]. Approaches based on deep neural networks do not suffer from overfitting or need manual feature extraction [8]. The industry has widely employed deep learning to increase SQL injection detection accuracy and decrease human labor, using methods like text representation and tainted analysis [9]. Text representation techniques may be further subdivided into token-based [10], N-gram-based [11], payload-based [12], and information-carrying [13]. These techniques could have better accuracy and higher complexity. This is primarily due to the machine learning method's poorly characterized input vector, which makes it challenging to identify the correct model. In addition, when processing the SQL statement dataset, text vectorization algorithms like the TFIDF algorithm [14] can produce datasets with improved representation; however, this algorithm's slightly imprecise representation of attributes will result in an inaccurate description of keyword weight.

In this paper, the researchers propose a novel feature extraction method by integrating TF-IDF and VGG16 models to transform input text data into feature images suitable for the proposed lightweight deep learning classification model. Additionally, this work proposes a lightweight deep learning model for SQL injection detection by integrating MobileNetV2 and a fine-tuning technique, aiming to reduce computation complexity while enhancing detection accuracy.

The rest of the paper is broken into the following sections: Section 2 discusses the relevant work from the literature. Section 3 provides an explanation of the SQL injection attack and its classes. In Section 4, the recommended Deep TF-IFD and Lightweight MobileNetV2 (LW-MNV2) methods for feature extraction and SQL injection attack detection are described. Section 5 provides a description and discussion of the results. Finally, Section 6 concludes this study.

2. Related Works

This section will focus on previous works that relied on the TF-IDF feature extraction and the deep learning classification approaches in SQL injection attack detection.

An SQLNN deep neural network model was recently suggested by Zhang et al. [6] for the detection of SQL injection threats. They used NLP to preprocess the 30,919 data items in the Kaggle dataset and retrieved TF-IDF characteristics. With an accuracy of 96.16%, precision of 97.28%, recall of 92.23%, and F1-score of 94.68%, the SQLNN model performed well. However, this study fails to elucidate the model's complexity and computing efficiency, concentrating solely on accuracy metrics.

[7] Suggested an approach combining Text CNN, LSTM, and the attention mechanism for SQL injection detection. It uses Text CNN for local features, Bi-LSTM for sequence information, TF-IDF and Word2Vec for feature vectorization, and an attention technique to enhance the process of the lengthy sequence. The accuracy of the suggested approach was higher than 99.57%. Although the model attains impressive accuracy, the combination of CNN, Bi-LSTM, and the Attention mechanism leads to considerable computational complexity owing to its several components . Additionally, the use of TF-IDF and Word2Vec increases the cost of feature extraction and reduces the model's efficiency in terms of time and memory usage.

[14] Used a Kaggle dataset to examine the efficacy of Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs) for SQL injection detection. With 99.70% accuracy, 99.91% precision, 99.25% recall, and a 99.58% F1-score, CNN performs the best. However, the study lacks computational analysis because it does not take into account aspects that affect computing efficiency, such as the length of training and inference.

[15] utilized a CNN to identify SQL injection attempts within a dataset of 3951 records from Kaggle. It compared how well CNN worked with other machine learning methods. The CNN did better than the other models, with an accuracy of 94.84%, a precision of 85.67%, and a recall of 96.56%. The study's limitation is the use of a small database, which makes it more likely to overfit. This makes the system less useful in general. Also, the Low Precision Value (85.67%) compared to Overall Accuracy means that the model makes a certain number of false positives. This makes it not good for sensitive uses.

Using the 30,919-item Kaggle dataset, [16] introduced a new recurrent neural network methodology for SQL injection detection. The approach got a score of 92% and an accuracy of 94%, which is lower than the best results because it focused on sophisticated attack patterns.

[17] Used transformers (BERT) to create a deep learning system that uses long short-term memory (LSTM) networks and bidirectional encoder representations. The system achieved results with an accuracy of 97.3%, a precision of 96.3%, a recall of 96.2%, and a F1-score of 95.8% utilizing 10,852 sentences from the HTTPParams dataset. A limitation of this work is the intricate model structure, a hybrid LSTM-BERT, which renders it computationally expensive.

[18] created a CNN-based classification system for SQLI detection using Word2Vec and non-background rule-dependent NLP approaches. For their tests, they employed 4,000 normal and 4,000 SQLI samples. According to the reported results, the system's accuracy and F1 score are nearly 98%. The CNN model required around 26 seconds to process 4000 data, nearly twice the duration of the MLP, demonstrating low time efficiency in actual application.
A hybrid CNN-BiLSTM model was developed by [19] to identify SQL injection threats. With a dataset of 4,200 questions, the CNN-BiLSTM outperformed other machine learning algorithms examined in this study, with an accuracy of almost 98%. The limitation of this study is that the limited amount of data utilized is inadequate for assessing the model in real-world scenarios or different environments, and although a hybrid model was employed, the computing performance of the model was not disclosed.

[20] proposed employing an autoencoder network (AE-Net) for SQL injection detection to improve data representation with deep characteristics. Their approach achieved a k-fold highest accuracy of 99% when combined with an extreme gradient boosting classifier. The integration of an autoencoder with XGBoost amplifies the model's complexity, hence complicating its implementation and real-time functionality.

[21] combined SQL injection detection systems, a multi-layer perceptron (MLP), and LSTM networks. Eighty-seven thousand ninety-five queries were included in the dataset derived from DPI data that the ISP sent via the FTP protocol. Over 99% accuracy was attained by the MLP. Nonetheless, the research exhibited Limited Data Diversity, as it relied exclusively on data from a single ISP. This limits the model's generalizability when applied to different network configurations or various SQL attack methodologies.

[22] proposed the application of Bi-LSTM and TF-IDF for the detection of SQL injection. Three Kaggle datasets were utilized, with total samples of 4,200, 33,861, and 30,648, respectively. The accuracy, recall, precision, and F1 scores for TF-IDF and Bi-LSTM reached 99%. However, the study lacks computational analysis, including training duration and inference.
Table 1: summarizes related work, detailing the utilized dataset, the employed methodology, and the results of the studies.

Table 1: Summary of Related Work on SQL Injection Detection.

Ref.	Dataset	Method	Results
[6]	Kaggle dataset with 30919 samples	SQLNN deep neural network	accuracy =96.16%, precision =97.28% recall = 92.23%, F1-score = 94.68%.
[7]	Collected a new dataset with 10,052 samples	CNN, LSTM and attention	accuracy=99.57%
[14]	Kaggle dataset	ANNs, CNNs, and RNNs	Accuracy =99.70% Precision =99.91% Recall =99.25% F1-Score =99.58%
[15]	3951-record Kaggle dataset	CNN	Accuracy =94.84% Precision =85.67% Recall =96.56%
[16]	Kaggle dataset with 30919 samples	RNNs	Accuracy =94% F1-Score =92%
[17]	HTTPParams dataset with 10852 samples	Hybrid LSTM and BERT	Accuracy =97.3% Precision =96.3% Recall =96.2% F1-Score =95.8%
[18]	4,000 normal and 4,000 SQLI samples	CNNs	Accuracy = 98.2% Precision =97.4% Recall =99% F1-Score =98.2%
[19]	SQL dataset with 4200 queries	hybrid CNN-BiLSTM	Accuracy = 98%
[20]	Custom SQLI dataset	AE-Net	Accuracy =99%
[21]	DPI dataset with 87095 queries	MLP and LSTM	Accuracy > 99%
[22]	Three Kaggle datasets with total samples, (4200), (33861), (30648)	Bi-LSTM	Accuracy, Precision, Recall, and F1-Score =99%

Table 1 demonstrates that several recent studies have attained very accurate results by employing advanced deep learning models like CNNs and RNNs, as well as hybrid models based on BiLSTM with attention processes. Even though these models work well, they often need a lot of computing resources and a large number of parameters. This makes it hard to employ them on systems with limited resources or in applications that need quick processing speeds. It is also important to note that a lot of the earlier studies only looked at accuracy percentage and didn't adequately assess model efficiency or inference time. These problems made it necessary to create a lightweight and useful model, which is why the LW-MNV2 model was suggested in this study.

3. SQLi Attacks

SQL injection vulnerabilities emerge when attackers modify application database queries. It usually allows an attacker access to restricted data. It may contain user data, table structures, login information, and other program-accessible data. An attacker may edit or erase such data, causing long-term program functionality or content changes. Sometimes, an attacker may escalate a SQL injection attack to perform denial-of-service attacks or compromise the server's foundation or back-end architecture [23]. The SQL injection consists of several types as follows [24]:

- Impair Input: Cross-site scripting may bypass security if a client-side script solely controls input at the application's front end. This allows attackers to transmit malicious data to the server without verification.

- Error message displayed: the web page displays an error message related to SQL code. The most frequent reasons for this error are setup problems or mistakes made by website programmers. Table and database names are examples of exposed information that attackers might use to perform SQL Injection.
- Uncontrolled variable: When a script doesn't restrict a database variable, an attacker may use an SQL command to alter the data.
- Tautologies: to ensure that a query constantly evaluates to TRUE, the code is added to the WHERE clause using the conditional OR function. Since the query's condition is a tautology, it is evaluated as true for each row in the table.

SQL injection puts a server's or network's database system and the security of its data at risk. This might cause a network to go down, alter data, reveal private information, propagate malware, and crash a website [25]. Figure 1 illustrates the secure web-based applications from SQLi attacks and consists of three steps as shown in the figure below:

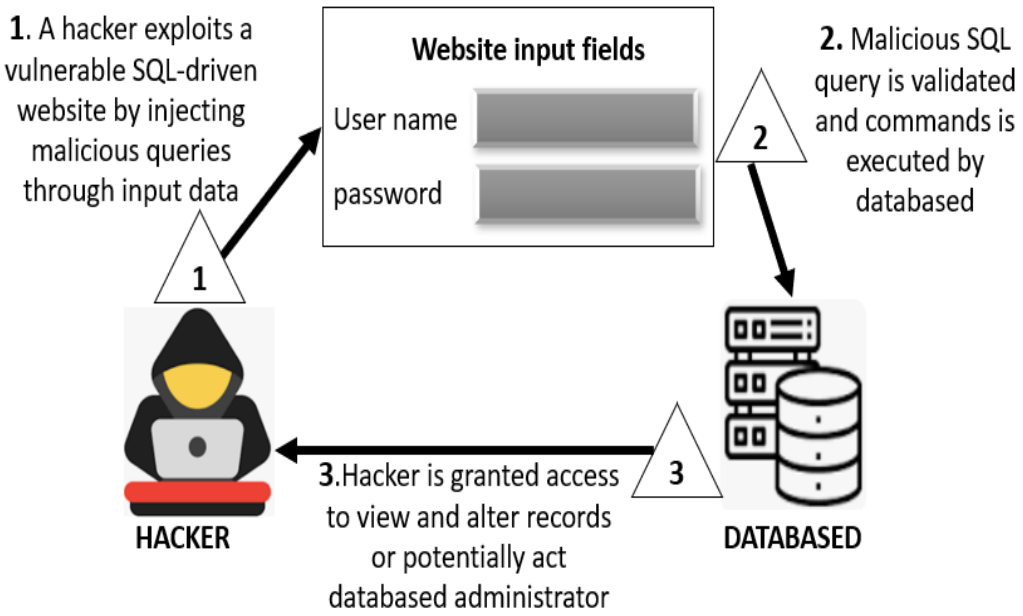


Figure 1: Secure web-based applications from SQLi attacks.

4. Lightweight DL Model to SQLi Detection

Sensitive data protection and online application security depend on the early and highly accurate detection of SQL injection threats. To address this, this paper proposes the lightweight SQL injection attack detection system (LWD-SQLIS), designed to efficiently detect such attacks with minimal computation. The core idea of the LWD-SQLIS framework is to extract essential features from the dataset and transform them from text to image format, ensuring compatibility with the deep learning classification structure. This model is referred to as Deep TF-IDF with VGG16 architecture. Additionally, this work proposes a lightweight classification model, known as LW-MNV2, which is designed to minimize training time while achieving higher accuracy in attack detection by integrating MobileNetV2 and fine-tuning techniques. The proposed LWD-SQLIS framework, illustrated in Figure 2, includes four main steps: loading the SQL injection dataset, cleaning and preprocessing the dataset using NLP techniques, converting input text data into image features with a size of 244×244 pixels using the proposed Deep TF-IDF model, and finally classifying SQL injection attacks using the proposed LW-MNV2 model, as shown in Figure 2.

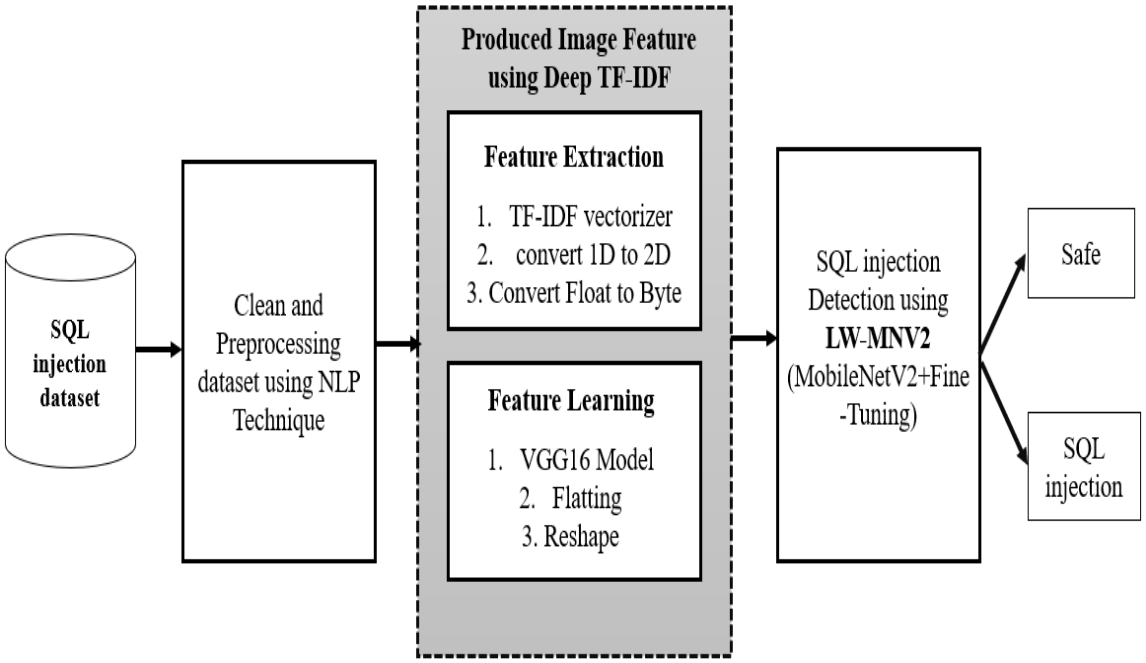


Figure 2: Framework of the proposed LWD-SQLIS.

4.1 SQLi Attack Dataset

The SQLi dataset, accessible to the public, was obtained from Kaggle [26]. It includes 30919 data samples overall and mostly satisfies the experimental conditions. Table 2 illustrates the details of the SQLi detection Dataset and the number of samples.

Table 2: SQLi detection dataset and number of samples

Label	Explanation	Number of samples
1	Sentence of SQL injection	11330
0	Sentence of non-SQL injection	10589

4.2 Cleaning and Preprocessing the Dataset

During preprocessing, the proposed LWD-SQLIS system uses tokenization to extract tokens from SQL injection attack statements. One of the NLP techniques [27] is to reduce noise and extract significant information from input words. This is important because input data quality substantially affects the machine learning and deep learning systems' SQL injection detection performance. Tokenization is problematic since SQL injection attack statements are frequently space-free. This is because tokenization relies on sentence word distances, which cannot be calculated without spaces. This is solved by putting a space between every pair of words in the phrase before tokenization.

Tokenization is a multi-step process that starts with loading the dataset, transferring the raw input into a series of tokens, and finding patterns for different operations. These comprise mathematical operations ['+', '-', '*', '/', '%', '++', '--], logical operations like ['all', 'any', 'some', 'like', 'in', 'Between', 'not], relational operations like ['=', '>', '\', '>=', '<=', '!>', '!>', '!<', '!<'], and punctuation marks like ['.', '?', '!', '^', ':', ';', '_', '(', ')', '[', ']', """]. Next, every unprocessed word in the text is converted to lowercase, and each word is carefully examined against these predetermined patterns. When a match is found, we preserve the resulting tokens and replace the corresponding pattern with a "word" identifier and also replace the number digits with term spaces. Table 3 illustrates the cleaning and tokenization process to prepare each input sentence for the next step.

Table 3: An example of cleaning and tokenization on the SQLi dataset

#	Original sentence	Clean sentence
1	“select * from users where id = '1' or \.<\ union select 1,@@VERSION -- 1””	“select * from users where id = ' number ' or \ . < \ union select number ,@@version -- number ””
2	“? or 1 = 1 –“	“? or number = number –“
3	“select * from users where id = 1 or " ( ]" or 1 = 1 – 1”	“select * from users where id = number or " ( ] " or number = number – number”
4	“admin' or 1 = 1#”	“admin ' or number = number #”
5	“AND 1 = utl_inaddr.get_host_address ((SELECT DISTINCT (table_name) FROM (SELECT DISTINCT (table_name) , ROWNUM AS LIMIT FROM sys.all_tables) WHERE LIMIT = 5)) AND 'i' = 'I”	“and number = utl _ inaddr . get _ host _ address ((select distinct (table _ name) from (select distinct (table _ name) , rownum as limit from sys . all _ tables) where limit = number)) and ' i ' = ' I”
--	---	---

4.3 Proposed Deep TF-IDF Model

This section presents the development of a TF-IDF-based feature extraction model that transforms input text into feature images. Word weights are represented as images for every input sentence instance instead of a one-dimensional vector to be suitable for the pass-through LW-MNV2 classification model. As a result, we propose a deep TF-IDF model by combining TF-IDF and VGG16. The Deep TF-IDF model consists of two main steps:

4.3.1 Extract TF-IDF Features

This step includes three sub steps:

- Apply TF-IDF

A digital statistical method called Term Frequency-Inverse Document Frequency (TF-IDF) is used to characterize a language's significance within a corpus [22]. The frequency of words utilized and the frequency of words in the text could be balanced when using TF-IDF for evaluation. The value of every phrase token from the word corpus serves as the input for the TF-IDF method. The TF-IDF methods are then used to calculate the value. TF-IDF value retrieval is achieved by combining two functions: inverse document frequency (IDF) & term frequency (TF) [28].

1- Term frequency (TF): The function in question provides the overall count of token appearances inside a content record. It is the proportion of a token's frequency of occurrence to all the tokens in the content record. As in Equ.1:

$$TF_{Ii,j} = \frac{n_{i,j}}{\sum_k n_{i,j}} \quad (1)$$

2- Use the Inverse Data Frequency (IDF) function to determine the frequency of odd tokens within a record document. Equation (2) shows that the tokens that sometimes appear in a document get a higher IDF score. As in Equ.2:

$$DF(w) = \log\left(\frac{N}{DF_i}\right) \quad (2)$$

3- A word in the corpus's TFIDF (w) value may be obtained by combining the two functions. As in Equ.3

$$w_{i,j} = TF_{i,j} \times \log\left(\frac{N}{DF_i}\right) \quad (3)$$

Where, wi,j= number of times i appears in j.

TFi,j= count of records that include i.

N= The total number of records.
The outputs of the TF-IDF convert input text into numeric values, as shown in Table 4, which reflect the weight of each word in the input sentence.

Table 4: Examples of attack features for SQLi using the TF-IDF algorithm

No.	"calle "	"Valencia"	⋯	"hpbt "	"jnmf"	"hrgu"
1	0.475476	0.942759	⋯	0.000000	0.000000	0.000000
2	0.000000	0.000000	⋯	0.000000	0.000000	0.000000
3	0.679251	0.000000	⋯	0.000000	0.000000	0.000000
4	0.000000	0.000000	⋯	0.000000	0.000000	0.000000
⋮	⋮	⋮	⋯	⋮	⋮	⋮

- Convert TF-IDF Values from 1D to 2D Matrix.

The proposed LW-MNV2 deep learning model requires two-dimensional input, particularly images. In this step, the model will transform the one-dimensional vector the TF-IDF produces into a two-dimensional matrix.

- Convert Float to Byte

This step converts the float values within the 2D matrix into byte format to ensure that all values in the matrix range from 0 to 255. To convert from Old Value to New Value, follow these steps:
1- Determine five parameters: Old Min Value=-1, Old Max Value=1, New Min Value = 0, New Max Value =255
2- Compute Old Range as (Old Max Value – Old Min Value)
3- Compute New Range as (New Max Value – New Min Value)
4- Apply Eq. 4:

$$\text{New Value} = \text{Int}\left(\frac{OldValue - OldMinValue \times NewRange}{OldValue} + NewMinValue\right) \quad (4)$$

Where, Old Min Value= -1, New Min Value=0, and New Range = New Max Value – New Min Value.

4.3.2 Learning Features

The second step in the Deep TF-IDF is to convert the 2d matrix with integer values into feature images using VGG16 networks. [29] Introduced the VGG16 Convolutional Neural Network (CNN) model for image classification applications. It is composed of sixteen convolutional layers that are completely coupled. The architecture's topology remains consistent, with 2 × 2 kernels for pooling layers and 3 × 3 kernels for convolutional layers, each with a stride of 2. This model uses fully connected layers for prediction and convolutional and max-pooling layers for feature extraction from input images [30]. As shown in Figure 3, the suggested Deep TF-IDF framework extracts features from a 2D input matrix using a portion of the VGG16 architecture. It excludes the fully connected layers and uses the 13 convolutional layers with 3×3 kernel filters. Instead of wholly linked layers, Deep TF-IDF uses a flattening procedure to produce a one-dimensional vector that we reshape into 224x224 feature images.

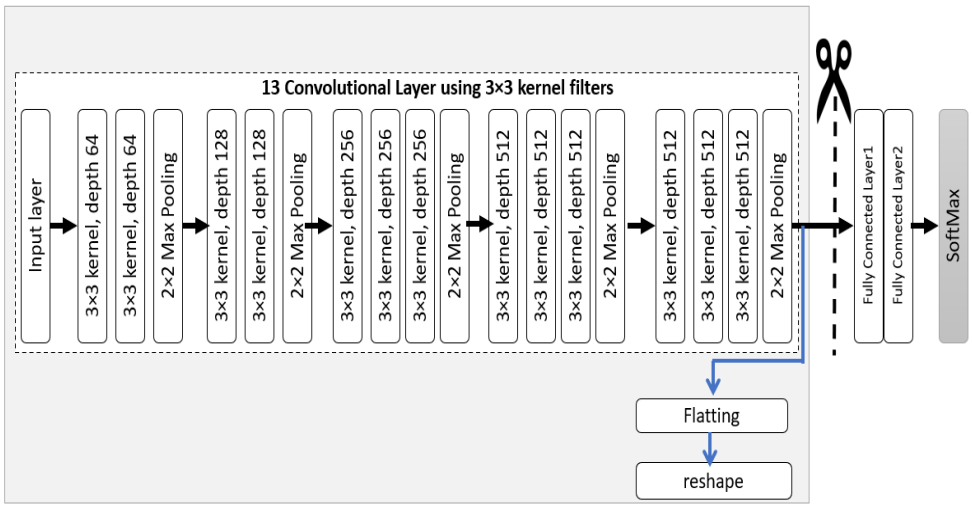


Figure 3: VGG16 network feature extraction model.

Examples of the outputs of the Deep TF-IDF are presented in Figure 4. In this figure, each input sentence is transformed into a visualized TF-IDF feature represented as a 2D matrix, which is then processed through VGG16 filters to produce 244×244 feature images. The white points in these images refer to the key features' locations. Additionally, the figure demonstrates how the Deep TF-IDF analyzes the distribution feature within both the TF-IDF and the resulting feature images, offering valuable insights into the spatial probability of these features.

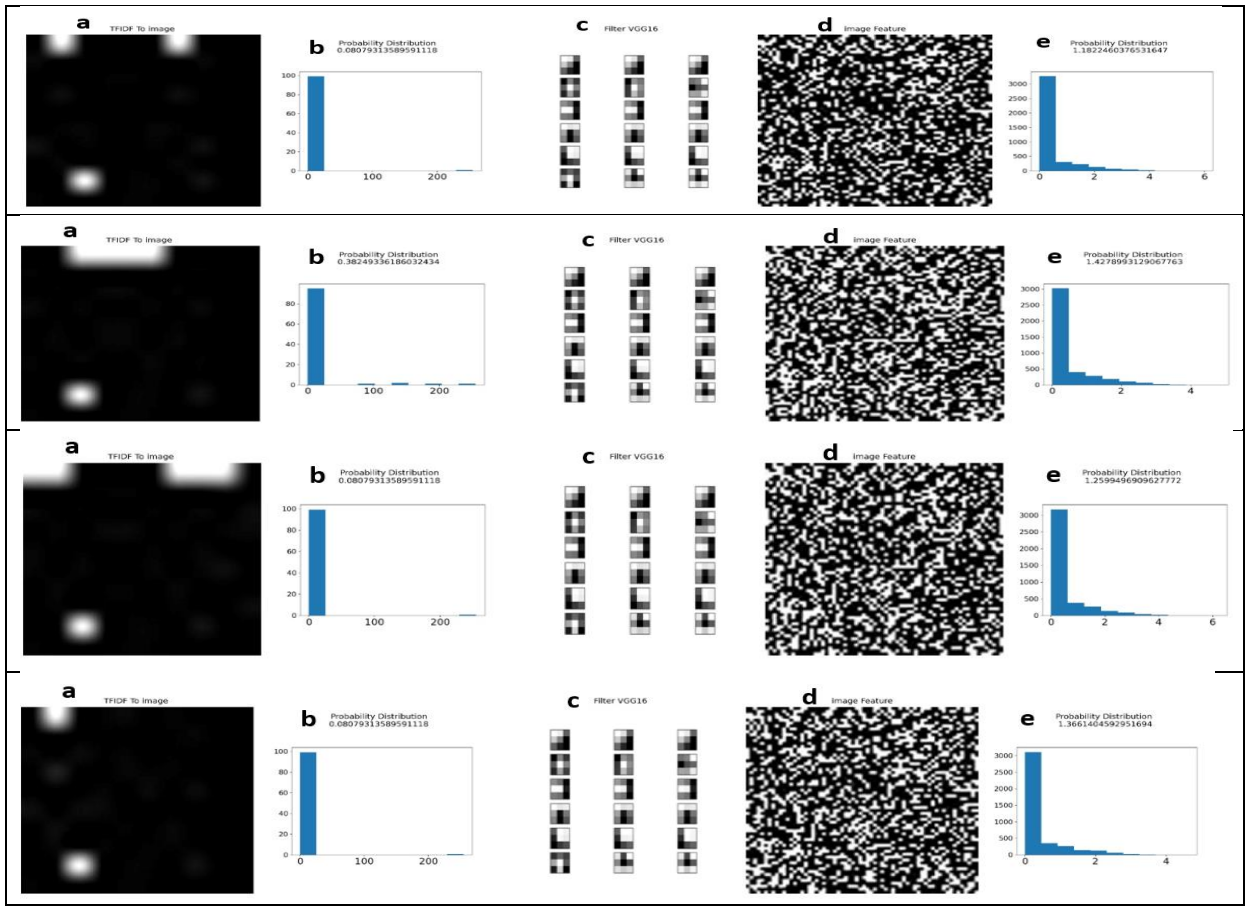


Figure 4: Results of the Deep TF-IDF feature extraction model: a) TF-IDF to image; b) Probability distribution of TF-IDF; c) Filter shape of VGG16; d) Image feature; e) Probability of image feature

4.4 Proposed Lightweight Deep Learning

The proposed lightweight deep learning model, LW-MNV2, utilizes an integrated MobileNetV2 architecture with a fine-tuning technique. Figure 5 presents the methodology outlined in the proposed model. The LW-MNV2 begins with processing input images with a size of 224×244 pixels, producing a binary classification output that determines whether the input belongs to the (safe) class or an (SQL Injection attack). The CNN architecture known as MobileNetV3 is designed to function effectively on mobile devices [31]. Its connections are made between the bottleneck layers, distinguishing it from other CNN topologies. Furthermore, deep folds are used by the intermediate expansion layer to eliminate nonlinear characteristics. The first 32 convolution layers in the MobileNetV2 architecture are followed by 19 "bottleneck layers." The MobileNetV2 design provides several advantages over the earlier deep learning systems. The system is complex to train on small datasets, and overfitting the visual classification task becomes a problem. The MobileNetV2 design is quick and effective, optimizing memory usage with a small margin of error, and it blocks this effect, avoiding over-fitting, as shown in Figure 5 [32].

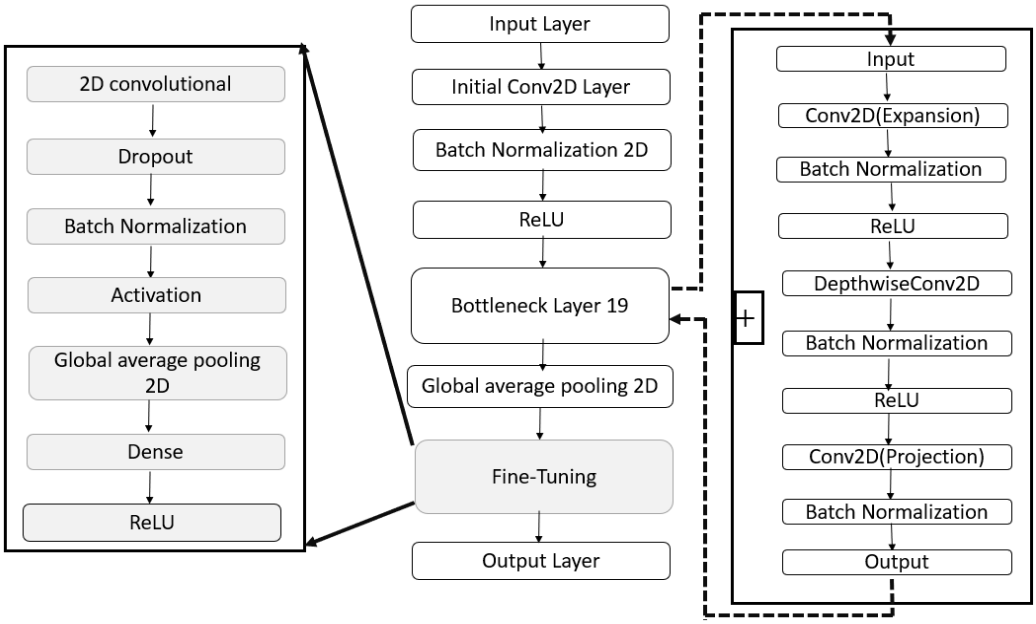


Figure 5: Frameworks of the proposed LW-MNV2 model for SQLi attack detection

A straightforward and efficient method for building a quick and dependable convolutional neural network (CNN) model is the fine-tuning approach [33]. Thus, the whole set of linked layers in MobileNetV2 is frozen in LW-MNV2, and a fine-tuning approach is used to maximize computing efficiency and accuracy. LW-MNV2 started with a 32-filter, 3x3 kernel-sized convolution layer, then batch normalization and ReLU activation. It has nineteen repeated bottleneck layers and employs depthwise separable convolution to improve performance and lower computation burden. One 1x1 convolution (expansion layer) with "batch normalization" and "Relu activation," one 3x3 "depthwise convolution" (depthwise convolution with batch normalization and Relu activation), and one last 1x1 "convolution' (projection layer) with batch normalization comprise each bottleneck layer. Additionally, this model uses a fine-tuning layer and global average pooling for classification.

5. IMPLEMENTATION AND RESULT

The LWD-SQLIS implementation was run on a Windows 10 operating system with a hardware configuration consisting of a Core i7-8550U CPU@ 1.80 GHz and 20.0GB of RAM, running at a speed of 2.00 GHz. The LW-MNV2 network was pre-trained on a SQL injection dataset comprising approximately 309191 samples. This study assessed the model utilizing 70, 10, and 20 sets for training, validation, and testing, respectively. While k-fold cross-validation is frequently recommended for thorough performance assessment, the dataset utilized was sufficiently large (30,919 samples), and the class distribution was maintained throughout the partitioning. This decision aligns with prior research that recognizes hold-out validation as appropriate for practical, application-focused AI systems, where computational efficiency and deployment feasibility are critical [34][35]. Details of the distribution are illustrated in Table 5.

Table 5: Details of the splitting SQLi attack dataset

Explanations	Number of samples	70%Training	10%Validation	20%Testing
SQL sentence	11330	7931	1133	2266
Non-SQL sentence	19589	13713	1959	3917
Total samples	30919	21644	3092	6183

We used a set of metrics to evaluate the performance of the LW-MNV2 classification model and compared it with the original MobileNetV2 to select the best model. The metrics used include accuracy, recall, f1-score, and precision as in Equ.5-8 [33].

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

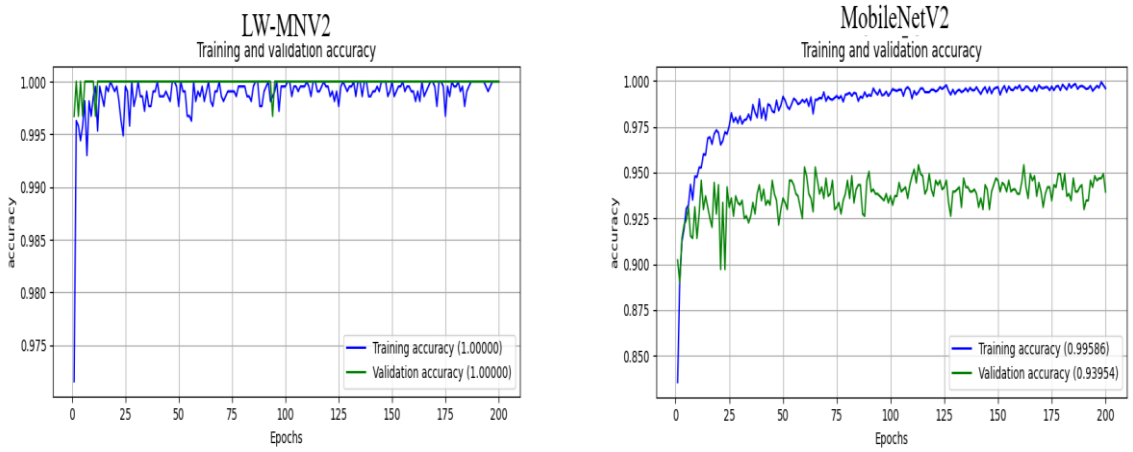
$$precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

Where TP represents the number of true positives, TN represents the number of true negatives, FP indicates the number of false positives, and FN indicates the number of false negatives.

In cases where the actual state is YES, a classification model's output yields YES in True Positive (TP) cases, NO in False Positive (FP) cases, NO in True Negative (TN) cases, and YES in False Negative (FN) cases where the model yields YES outcomes if the actual state is NO [33].

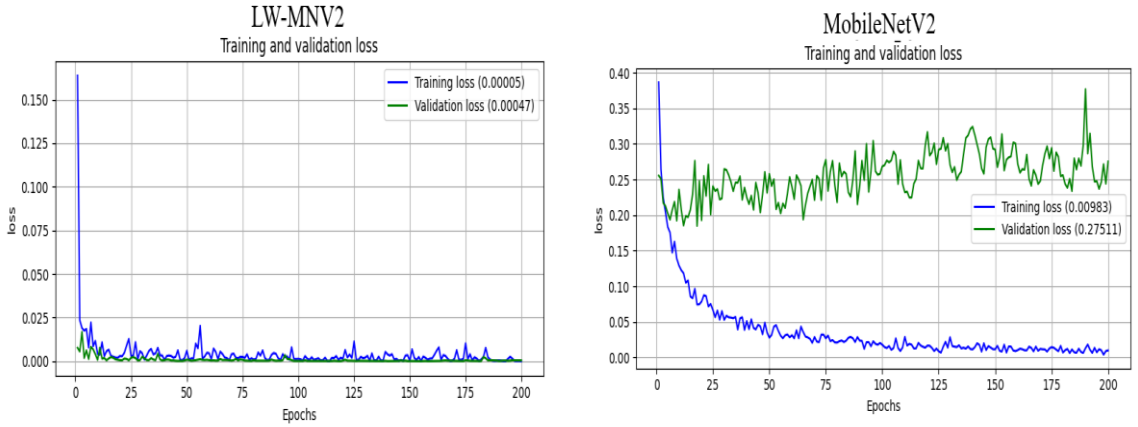
Figure 6 shows the accuracy values for the LW-MNV2 and MobileNetV2 models throughout the training and validation process over 200 epochs. The LW-MNV2 outperforms the original model, achieving optimal accuracy (100%) for training and validation. In contrast, the original MobileNetV2 reached (99.586%) accuracy in training and (93.95%) validation.



(a) **(b)**

Figure 6: Accuracy results for training and validation with 200 epochs: a) Accuracy of the proposed LW-MNV2 model; b) Accuracy of the Mobile NetV2 Model

Figure 7 displays the loss values for both the LW-MNV2 and MobileNetV2 models during testing and validation over 200 epochs. The LW-MNV2 outperforms the original model, achieving a minimum loss of (0.00005) in training and (0.00047) in validation. In comparison, the original MobileNetV2 recorded a loss of (0.00983) in training and (0.27511) in validation.



(a) **(b)**

Figure 7: Loss results for training and validation with 200 epochs: a) Loss of proposed LW-MNV2 mobile; b) Loss of Mobile NetV2 model.

To guarantee equitable and dependable assessment, various methods were instituted to avert overfitting and data leaking during model building. The incorporation of dropout layers (rate = 0.3), early stopping, and adaptive learning rate scheduling alleviated these concerns, preventing the model from retaining training patterns during the training process. The uniformity of the training loss and validation curves throughout epochs validated the model's stable convergence and robust generalizability.

Figure 8 presents the testing confusion matrix for the LW-MNV2 and MobileNetV2 models. The LW-MNV2 model shows the best performance, with fewer FP (6) and FN (3), while achieving higher TP (2260) and TN (3914). On the other hand, MobileNetV2 shows FP (66) and FN (3), TP (2200), and TN (3914). This means that LW-MNV2 is more effective in correctly identifying positive cases with fewer errors.

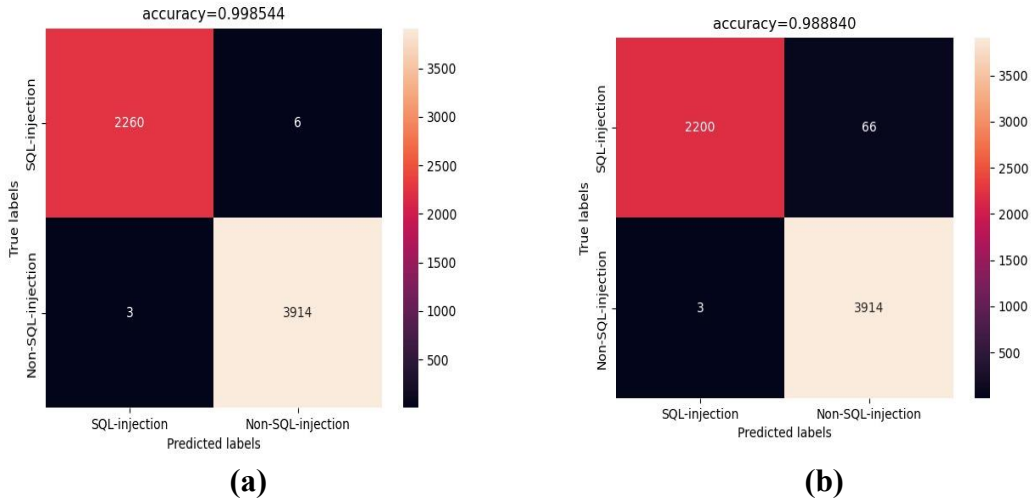


Figure 8: Confusion matrix of testing classification models: a) Proposed LW-MNV2; b) Mobile NetV2

Figure 9 compares the LW-MNV2 and MobileNetV2 models using criteria such as accuracy, recall, precision, and f1-score. The LW-MNV2 outperforms with a f1-score of 99.80%, accuracy of 99.85%, recall of 99.87%, precision of 99.74%, overtaking MobileNetV2 with a 98.88% accuracy, 99.86% recall, 97.08% precision, and 98.45% f1-score on the test. This comparison demonstrated that LW-MNV2 shows superior performance in detecting SQL injection attacks, supported by a deep neural network approach that enhances its detection capabilities.

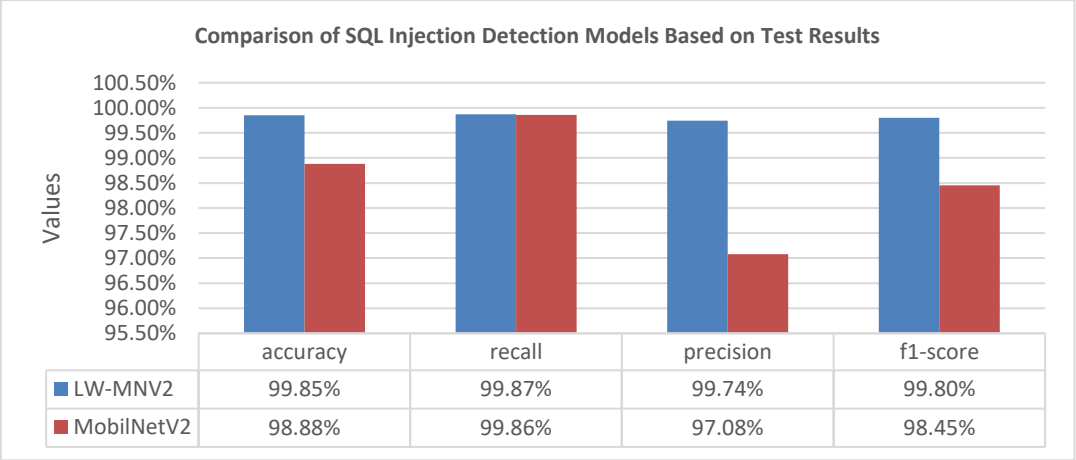


Figure 9: Comparison between LW-MNV2 and mobilenetV2 models in terms of testing results

The proposed model showed excellent performance on all of the stated evaluation criteria, striking an optimal compromise between lightweight construction and performance efficiency, rendering it ideal for sensitive applications. Table 6 shows a summary of how the lightweight models MobileNetV2 and the suggested LW-MNV2 model compare in terms of performance. The findings clearly show that the new model significantly reduced the number of parameters (by approximately 32%) and also decreased the time required for training, model size, and inference time.

Table 6: Comparison of MobilNetV2 and the Proposed LW-MNV2 in Lightweight Efficiency Analysis.

Models	Number of parameters (M)	Training time (s)	Model size (MB)	Inference Time (s)
MobilNetV2	3.7	865	14.2	5.5
LW-MNV2	2.5	650	9.5	3.5

These results were achieved through the model design modifications, such as using depthwise separable convolutions, minimizing channel expansion, and having fewer bottleneck layers. With these changes, the model was still able to extract features efficiently. These improvements have made the LW-MNV2 model better for situations with limited resources and applications that need to respond quickly. The comparison between the proposed LWD-SQLIS and some related works, as shown in Table **7**, highlights its pioneering approach. Unlike previous models, LWD-SQLIS transforms input queries from 1D text vectors into a 2D matrix representing image features using the proposed Deep TF-IDF model. This matrix is then processed through the lightweight LW-MNV2 deep learning model, resulting in superior SQL injection with near-optimal accuracy. This approach, evaluated against critical parameters such as dataset, feature extraction technique, and deep learning models, clearly demonstrates the LWD-SQLIS advantage over existing methods.

Table 7: Comparison between proposed LW-SQLIS and some related works. A '_' appears, meaning similar experimental findings are unavailable.

No. of Ref.	SQL injection Dataset Name	Feature Extraction	Types of Deep Learning Model	Testing Results	Model size (MB)	Inference Time (s)
[6]	Kaggle dataset with 30919 samples	TF-IDF	SQLNN deep neural network	accuracy =96.16%, precision =97.28% recall = 92.23%, F1-score = 94.68%.	–	–
[7]	Collected a new dataset with 10,052 samples	TF-IDF	integrates Text CNN, LSTM and the attention	accuracy=99.57%	–	–
[15]	3951-record Kaggle dataset	TF-IDF	CNN	Accuracy =94.84% Precision =85.67% Recall =96.56%	–	–
[17]	HTTPParams dataset with 10852 samples	-	Hybrid LSTM and BERT	Accuracy =97.3% Precision =96.3% Recall =96.2% F1-Score =95.8%	–	–
[18]	4,000 normal and 4,000 SQLI samples	Word2Vec	CNNs	Accuracy = 98.2% Precision =97.4% Recall =99% F1-Score =98.2%	–	26
[22]	Three Kaggle datasets with total samples respectively, (4200), (33861), (30648)	TF-IDF	Bi-LSTM	Accuracy =99% Precision =99% Recall =99% F1-Score =99%	–	–
Our Work	Kaggle dataset with 30919 samples	Proposed Deep TF-IDF	Proposed LW-MNV2	Accuracy =99.85% Precision =99.74% Recall =99.87% F1-Score =99.80%	9.5	3.5

Table 7 illustrates that the proposed LWD-SQLIS model has enhanced performance relative to prior deep learning-based SQL injection detection models. Although previous research attained considerable accuracy (between 94% and 99%), the majority failed to disclose computational efficiency parameters, such as model size or inference time, which are essential for real-time web security systems. The LW-MNV2 model attains an accuracy of 99.85% while maintaining a lightweight architecture of merely 9.5 MB and a rapid inference time of 3.5 seconds, underscoring its appropriateness for implementation in resource-constrained settings. This verifies that the suggested strategy improves detection performance while preserving computing cost, addressing the limitations noted in previous studies.

6. Conclusions

For back-end databases to stay safe, their integrity, privacy, and availability must be protected by accurately detecting SQL injection threats. This work presents a lightweight deep learning system for detecting SQL injection attacks, called LWD-SQLIS, which is built on a publicly available Kaggle dataset including 30,919 samples. It presents an innovative deep feature extraction method utilizing TF-IDF and VGG16, converting a 1D text vector into a 2D image feature. It also uses the lightweight classification model LW-MNV2, which combines MobileNetV2 and fine-tuning methods. When tested with conventional performance criteria, it did the best with 99.85% accuracy, 99.74% precision, 99.86% recall, and 99.80% score. The LWD-SQLIS system performed well. Future studies will investigate more intricate structures for transfer learning, expansive datasets, and practical implementation to augment the system's robustness.

References

[1] R. F. Sidik, S. N. Yutia, and R. Z. Fathiyana, "The effectiveness of parameterized queries in preventing SQL injection attacks at Go," in *Proc. Int. Conf. Enterprise Ind. Syst.*, vol. 270. Singapore: Springer Nature, 2023, pp. 204–213, doi: 10.2991/978-94-6463.

[2] V. Abdullayev and A. S. Chauhan, "SQL injection attack: Quick view," *Mesopotamian J. Cybersecurity*, pp. 30–34, 2023, doi: 10.58496/MJCS/2023/006.

[3] E. Peralta-Garcia, J. Quevedo-Monsalbe, V. Tuesta-Monteza, and J. Arcila-Diaz, "Detecting structured query language injections in web microservices using machine learning," *Informatics*, vol. 11, no. 2, art. no. 15, 2024, doi: 10.3390/informatics11020015.

[4] M. H. Furhad, R. K. Chakrabortty, M. J. Ryan, J. Uddin, and I. H. Sarker, "A hybrid framework for detecting structured query language injection attacks in web-based applications," *Int. J. Electr. Comput. Eng.*, vol. 12, no. 5, pp. 5405–5414, 2022, doi: 10.11591/ijece.v12i5.pp5405-5414.

[5] A. W. Marashdih, Z. F. Zaaba, and K. Suwais, "Predicting input validation vulnerabilities based on minimal SSA features and machine learning," *J. King Saud Univ. – Comput. Inf. Sci.*, vol. 34, no. 10, pp. 9311–9331, 2022, doi: 10.1016/j.jksuci.2022.09.010.

[6] W. Zhang *et al.*, "Deep neural network-based SQL injection detection method," *Security Commun. Netw.*, art. no. 4836289, 2022, doi: 10.1155/2022/4836289.

[7] H. Sun, Y. Du, and Q. Li, "Deep learning-based detection technology for SQL injection research and implementation," *Appl. Sci.*, vol. 13, no. 16, art. no. 9466, 2023, doi: 10.3390/app13169466.

[8] M. N. A. Kumar and B. Sujatha, "Early detection and mitigation methods for SQL injection attacks using adaptive free algorithm," *J. Positive School Psychol.*, vol. 6, no. 8, pp. 6315–6326, 2022.

[9] **[9]** J. Liu and C. He, "Online detection of SQL injection attacks based on ECA rules and dynamic taint analysis," *J. Comput. Appl.*, vol. 43, no. 5, pp. 1534–1541, 2023.

[10] D. Kar, S. Panigrahi, and S. Sundararajan, "SQLiGoT: Detecting SQL injection attacks using graph of tokens and SVM," *Comput. Secur.*, vol. 60, pp. 206–225, 2016, doi: 10.1016/j.cose.2016.04.005.

[11] Z. Guo *et al.*, "SQL injection detection method based on N-gram and TF-IDF," in *Proc. Int. Seminar Comput. Sci. Eng. Technol. (SCSET)*. IEEE, 2023, pp. 204–207, doi: 10.1109/SCSET58950.2023.00053.

[12] S. Zhang, Y. Li, and Q. Jiang, "Feature ratio method: A payload feature extraction and detection approach for SQL injection attacks," in *Proc. 3rd Asia-Pacific Conf. Commun. Technol. Comput. Sci. (ACCTCS)*. IEEE, 2023, pp. 172–175, doi: 10.1109/ACCTCS58815.2023.00019.

[13] M. Wang and C. Wang, "Detection of SQL injection attack based on improved TF-IDF algorithm," in *Proc. Int. Conf. Mech. Robot.*, vol. 12331. SPIE, 2022, pp. 885–892, doi: 10.1117/12.2652203.

[14] M. Alshammari, "Deep learning approaches to SQL injection detection: Evaluating ANNs, CNNs, and RNNs," in *Proc. Int. Conf. Math. Stat. Phys. Comput. Sci. Educ. Commun.*, vol. 12936. SPIE, 2023, pp. 131–138, doi: 10.1117/12.3012620.

[15] A. Falor *et al.*, "A deep learning approach for detection of SQL injection attacks using convolutional neural networks," in *Proc. Data Analytics Manag.*, vol. 2. Singapore: Springer, 2022, pp. 293–304, doi: 10.1007/978-981-16-6285-0_24.

[16] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Deep learning architecture for detecting SQL injection attacks based on RNN autoencoder model," *Mathematics*, vol. 11, no. 15, art. no. 3286, 2023, doi: 10.3390/math11153286.

[17] Y. Liu and Y. Dai, "Deep learning in cybersecurity: A hybrid BERT–LSTM network for SQL injection attack detection," *IET Inf. Secur.*, art. no. 5565950, 2024, doi: 10.1049/2024/5565950.

[18] D. Chen, Q. Yan, C. Wu, and J. Zhao, "SQL injection attack detection and prevention techniques using deep learning," *J. Phys.: Conf. Ser.*, vol. 1757, no. 1, art. no. 012055, 2021, doi: 10.1088/1742-6596/1757/1/012055.

[19] N. Gandhi *et al.*, "A CNN-BiLSTM based approach for detection of SQL injection attacks," in *Proc. Int. Conf. Comput. Intell. Knowl. Econ. (ICCIKE)*. IEEE, 2021, pp. 378–383, doi: 10.1109/ICCIKE51210.2021.9410675.

[20] N. Thalji *et al.*, "AE-Net: Novel autoencoder-based deep features for SQL injection attack detection," *IEEE Access*, vol. 11, pp. 135507–135516, 2023, doi: 10.1109/ACCESS.2023.3337645.

[21] P. Tang *et al.*, "Detection of SQL injection based on artificial neural network," *Knowl.-Based Syst.*, vol. 190, art. no. 105528, 2020, doi: 10.1016/j.knosys.2020.105528.

[22] I. Ghozali *et al.*, "A novel SQL injection detection using Bi-LSTM and TF-IDF," in *Proc. 7th Int. Conf. Inf. Netw. Technol. (ICINT)*. IEEE, 2022, pp. 16–22, doi: 10.1109/ICINT55083.2022.00010.

[23] I. S. Crespo-Martínez *et al.*, "SQL injection attack detection in network flow data," *Comput. Secur.*, vol. 127, art. no. 103093, 2023, doi: 10.1016/j.cose.2023.103093.

[24] M. H. A. Al-Maliki and M. N. Jasim, "Review of SQL injection attacks: Detection to enhance website security from client-side attacks," *Int. J. Nonlinear Anal. Appl.*, vol. 13, no. 1, pp. 3773–3782, 2022, doi: 10.22075/ijnaa.2022.6152.

[25] M. T. Muslihi and D. Alghazzawi, "Detecting SQL injection on web application using deep learning techniques: A systematic literature review," in *Proc. 3rd Int. Conf. Vocational Educ. Electr. Eng. (ICVEE)*. IEEE, 2020, pp. 1–6, doi: 10.1109/ICVEE50212.2020.9243198.

[26] S. Sajid, "SQL injection dataset," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/sajid576/sqlinjection-dataset

[27] R. Liu and W. Zhang, "A detection methodology for SQL injection attacks based on the TF-IDF-CHI algorithm," in *Proc. Int. Conf. Algorithms High Performance Comput. Artif. Intell. (AHPCAI)*, vol. 12941. SPIE, 2023, pp. 155–160, doi: 10.1117/12.3011777.

[28] L. Yu, S. Luo, and L. Pan, "Detecting SQL injection attacks based on text analysis," in *Proc. 3rd Int. Conf. Comput. Eng. Inf. Sci. Appl. Technol. (ICCIA)*. Atlantis Press, 2019, pp. 95–101, doi: 10.2991/iccia-19.2019.14.

[29] A. A. M. Ibrahim and J. R. Tapamo, "Transfer learning-based approach using new convolutional neural network classifier for steel surface defects classification," *Sci. Afr.*, vol. 23, art. no. e02066, 2024, doi: 10.1016/j.sciaf.2024.e02066.

[30] Z. P. Jiang *et al.*, "An improved VGG16 model for pneumonia image classification," *Appl. Sci.*, vol. 11, no. 23, art. no. 11185, 2021, doi: 10.3390/app112311185.

[31] M. Sandler *et al.*, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 4510–4520.

[32] Y. Kaya and E. Gürsoy, "A MobileNet-based CNN model with a novel fine-tuning mechanism for COVID-19 infection detection," *Soft Comput.*, vol. 27, no. 9, pp. 5521–5535, 2023, doi: 10.1007/s00500-022-07798-y.

[33] R. Poojary, R. Raina, and A. K. Mondal, "Effect of data augmentation on fine-tuned CNN model performance," *IAES Int. J. Artif. Intell.*, vol. 10, no. 1, pp. 84–92, 2021, doi: 10.11591/ijai.v10.i1.pp84-92.

[34] V. W. Lumumba *et al.*, "Comparative analysis of cross-validation techniques in machine learning models," *Amer. J. Theor. Appl. Stat.*, vol. 13, no. 5, pp. 202–214, 2024, doi: 10.11648/j.ajtas.20241305.13.

[35] S. M. Sadoon, M. A. Hasson, and B. H. Ali, "Insect pest management in artificial intelligence: Enhancing insect detection and classification," *Int. J. Intell. Eng. Syst.*, vol. 18, no. 7, pp. 257–270, 2025, doi: 10.22266/ijies2025.0831.18.