TELKOMNIKA Telecommunication Computing Electronics and Control
Vol. 24, No. 5, October 2026, pp. 1597~1610
ISSN: 1693-6930, DOI: 10.12928/TELKOMNIKA.v24i5.27884

Enhancing SQL injection detection accuracy with fine-tuning LLaMa 3.2 1B using low-rank adaptation

Diash Firdaus[1], Idi Sumardi[2], Muhammad Ichwan[1], Maulana Seno Aji Yudhantara[1]
[1]Faculty of Informatics and Technology Industry, Institut Teknologi Nasional Bandung (ITENAS), Bandung, Indonesia
[2]Department of Informatics Engineering, Al-Ghifari University, Bandung, Indonesia

Article Info

Article history:

Received Feb 20, 2026
Revised Jun 11, 2026
Accepted Jun 30, 2026

Keywords:

Digital forensics
Large language model
LLaMA 3.2
Low-rank adaptation
SQL injection
Web application security

ABSTRACT

Structured query language injection (SQL injection/SQLi) remains a critical threat to web application security, particularly because it can bypass traditional detection systems through obfuscation. Existing approaches, including rule-based systems and classical machine learning (ML) models, often fail to capture the contextual semantics of com0plex SQL queries. This study proposes a lightweight and context-aware SQLi detection model by fine-tuning the large language model Meta artificial intelligence (LLaMA) 3.2–1B large language model using low-rank adaptation (LoRA). The model was trained using the SQL-Injection-Extend dataset obtained from Kaggle, consisting of 57,310 malicious SQLi samples and 52,208 benign user-input samples. The benign data comprise general user-generated text commonly submitted through web application interfaces. Experimental results demonstrate that the proposed model achieves an accuracy of 99.91%, precision of 99.96%, recall of 99.86%, and an F1-score of 99.91%, outperforming baseline models including convolutional neural network (CNN), long short-term memory (LSTM), and non-fine-tuned large language models (LLMs). These findings indicate that LoRA-based fine-tuning substantially enhances detection performance while maintaining computational efficiency, making the approach suitable for real-time deployment in modern web security systems. Furthermore, the integration of context-aware digital forensic reporting supports post-incident investigation and improves the interpretability of detection outcomes.

Corresponding Author:

Idi Sumardi
Department of Informatics Engineering, Alghifari University
140 Cisaranten Kulon Street, Soekarno-Hatta, Bandung, West Java 40293, Indonesia
Email: idisumardi@gmail.com

1. INTRODUCTION

Structured query language injection (SQL injection/SQLi) attacks are among the most dangerous threats in the web application ecosystem due to their ability to directly exploit databases, leading to fatal consequences such as sensitive data (PII) leakage, unauthorized data modification, and even server takeover through remote code execution (RCE) [1]. These destructive impacts not only disrupt technical operations but also trigger massive financial losses and violations of strict data privacy regulations. As emphasized in recent studies, SQLi remains a primary attack vector dominating the cyber threat landscape due to its effectiveness in penetrating weak application defenses and its ongoing exploitation by security researchers in pursuit of improved mitigation solutions [2], [3]. The primary issue no longer lies in simple, easily identifiable attacks but rather in attackers' ability to leverage these techniques to manipulate databases undetected by standard security systems. This threat is further intensified by technological evolution,

including the adoption of large language models (LLMs) and machine learning (ML) algorithms that can assist attackers in automatically generating diverse payload variations, thereby rendering conventional detection methods, such as signature-based detection, increasingly ineffective [4], [5].

Over the past two decades, security researchers and practitioners have developed various defense mechanisms; however, each exhibits fundamental limitations when confronted with modern attacks [6]. First, traditional approaches that rely on signature-based systems, such as classical web application firewalls (WAFs), operate by matching traffic against databases of known attack patterns. The critical weakness of this approach lies in its rigidity: even minor modifications to a payload can alter its signature, allowing new attacks to bypass detection and resulting in a high false-negative rate [7].

Numerous studies have previously attempted to address SQLi threats using ML and deep learning (DL) algorithms, achieving promising results but still facing structural limitations. Recent surveys indicate that classifiers like Naïve Bayes and support vector machine (SVM) are effective for standard payloads [8], [9]. They remain vulnerable to adversarial evasion techniques, where attackers subtly modify query syntax to bypass detection [10]. A previous study, Rosca *et al*. [5] introduced a ML–based framework for SQLi detection by developing a two-stage processing mechanism that integrates SQL query normalization with semantic feature extraction. As its primary contribution, the approach extracts eight semantic characteristics from SQL statements after transforming raw queries into generalized representations that are more suitable for ML applications. To support model development, the authors compiled a large-scale dataset containing 90,000 SQL queries, comprising 17,695 legitimate queries and 72,304 malicious queries. The dataset was generated through a combination of publicly available sources and synthetic samples produced using generative pre-trained transformer 4o (GPT-4o), aiming to mitigate privacy issues and address the limited availability of labeled SQLi data. Multiple ML algorithms were subsequently assessed using Azure ML Studio under various sampling configurations. Among the fifteen evaluated model–sampling combinations, a voting ensemble model incorporating extreme gradient boosting (XGBoost) and light gradient boosting machine (LightGBM) demonstrated the highest performance, achieving an accuracy of 96.86%, a weighted area under the receiver operating characteristic curve (AUC) of 98.25%, a weighted F1-score of 96.77%, and a Matthews correlation coefficient (MCC) of 89.89%. Nevertheless, despite the promising predictive performance, conventional ML approaches may still have difficulty capturing the contextual and semantic complexities associated with sophisticated and obfuscated SQLi attacks.

Subsequently, various studies have investigated SQLi detection techniques using diverse approaches, ranging from pattern-based methods to ML and DL. Falor *et al*. [11] demonstrated that a convolutional neural network (CNN)–based approach can achieve a high accuracy of 98.6%, highlighting CNNs' effectiveness in learning SQLi attack patterns, albeit at the cost of high computational overhead. In contrast, Raut *et al*. [12] adopted the Aho–Corasick pattern-matching algorithm, which is computationally lightweight but achieved only 91.3% accuracy and tends to fail in detecting advanced or obfuscated attacks. A more complex DL approach was introduced by [13] through a hybrid long short-term memory (LSTM)–CNN model, which reported the highest accuracy of 99.96%, though it requires substantial computational resources.

Conventional ML approaches continue to demonstrate competitive performance. Patil *et al*. [14] applied decision trees (DT), random forest (RF), and SVM, achieving accuracies of up to 96%, while emphasizing the importance of model explainability in enhancing trust in detection systems, despite challenges in real-time deployment. Similarly, Mustapha *et al*. [15] showed that RF and artificial neural networks (ANN) can reach 97% accuracy; however, evaluation on a single dataset limits the generalizability of their models. Furthermore, several studies have proposed hybrid and domain-specific approaches. Wang *et al.* [16] utilized a combination of CNN and bidirectional long short-term memory (BiLSTM), achieving an accuracy of 99.3%, although their work primarily focused on input injection rather than SQLi specifically. Although these studies report high detection accuracy, they share common limitations. The primary limitation lies in the semantic gap, namely the inability of these models to understand the context and intent behind SQL query structures, particularly when attacks are modified using sophisticated syntax obfuscation techniques.

The previously identified semantic gap underscores the urgency of adopting LLMs. Unlike traditional ML or DL approaches, LLMs are trained on massive corpora of source code and internet text, which endows them with deep contextual and semantic understanding. Rather than merely matching statistical patterns, LLMs are capable of "reading" SQL queries and determining whether the underlying logic is malicious, even when the syntax has been manipulated through sophisticated obfuscation techniques [17]. However, deploying large-scale LLMs such as GPT-4 or large language model Meta artificial intelligence (LLaMA) 70B for real-time intrusion detection is generally impractical due to high latency and substantial operational costs [18].

As an innovative solution, LLaMA 3.2 1B emerges as a game-changer in the category of small language models (SLMs). With only one billion parameters, this model is specifically designed for high

efficiency, making it lightweight enough to run on edge devices such as Raspberry Pi or resource-constrained servers, while still inheriting the architectural intelligence of larger LLaMA models [19].

To further optimize performance without overburdening computational resources, low-rank adaptation (LoRA) is employed as a key component in this study. Full fine-tuning of LLMs typically requires substantial memory and computational overhead, while also introducing the risk of catastrophic forgetting, where the model loses its previously learned general knowledge. LoRA addresses these limitations by updating only a small subset of parameters through low-rank matrix decomposition, reducing memory requirements by up to 60–70% and significantly accelerating the training process [20].

By leveraging these advantages, the combination of LLaMA 3.2–1B and LoRA enables efficient and context-aware analysis of user input, allowing security systems to detect SQLi attacks with low latency while maintaining high accuracy. In addition, the model provides contextual explanations that support digital forensic analysis, making it suitable for practical deployment in modern web security environments. Accordingly, this study aims to develop an intelligent, efficient, and context-aware SQLi detection system that addresses the limitations of existing approaches. Although previous studies have reported promising results using conventional ML and DL approaches for SQLi detection, many of these methods primarily focus on learning statistical patterns and often struggle to capture the contextual semantics of obfuscated attack payloads. Moreover, existing studies rarely investigate the use of lightweight LLMs combined with parameter-efficient fine-tuning techniques for SQLi detection. Therefore, the novelty of this study lies in the integration of LLaMA 3.2–1B and LoRA to develop a context-aware SQLi detection framework that balances semantic understanding, computational efficiency, and practical deploy ability. Furthermore, this work extends beyond attack classification by incorporating context-aware digital forensic reporting to support post-incident investigation and explainable cybersecurity analysis. The main contributions of this study are as follows: (i) the development of a context-aware SQLi detection model based on LLaMA 3.2–1B with LoRA fine-tuning; (ii) the implementation of a lightweight deployment strategy suitable for edge and resource-constrained environments through parameter-efficient adaptation; and (iii) the integration of real-time SQLi detection with context-aware digital forensic reporting to support explainable cybersecurity operations and post-incident analysis.

2. METHOD

Figure 1 illustrates the architecture of the SQLi detection model, depicting the workflow from the user submitting input through the application programming interface (API) gateway to its processing within the web application. The architecture positions the detection system as an additional security layer that analyzes incoming user input before it is executed by the database.

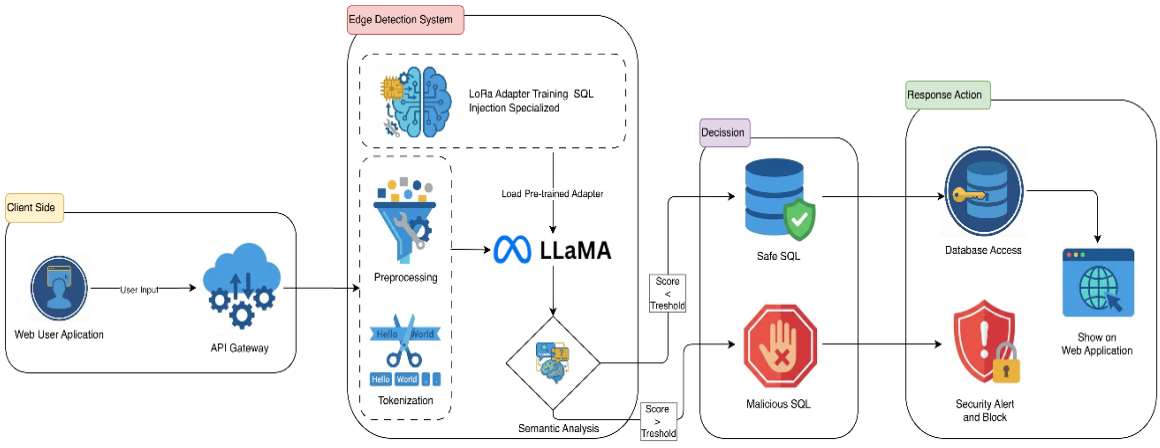

Figure 1. SQLi detection architecture

On the client side, users access the web application through a browser. When the website is loaded, users submit requests in the form of user input (e.g., form inputs or query strings) via an API. Subsequently, the edge detection system performs preprocessing and tokenization on the incoming user inputs before they are processed by the LLaMA 3.2–1B model, which has been fine-tuned using the LoRA method. The model analyzes the user input to determine whether it contains SQLi patterns. If the input is identified as malicious,

the request is blocked, and the system generates a notification indicating a detected SQLi attempt. Conversely, if the input is classified as benign, access to the database is granted, and the web application operates normally. Figure 2 illustrates the research methodology employed in this study. The research process begins with data collection from Kaggle and proceeds through several stages, culminating in the final phase of model evaluation.

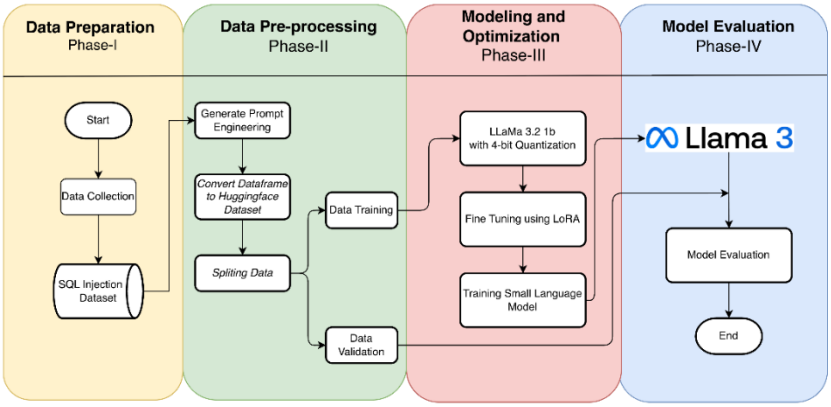

Figure 2. Research method

Phase I (data preparation): the data preparation phase aims to establish a dataset foundation that is relevant to the research problem. This phase begins with data collection, focusing on the cybersecurity domain, particularly SQLi attacks. The collected dataset includes a wide range of user input patterns, encompassing both benign and malicious inputs, to represent real-world attack scenarios encountered by modern web applications. By explicitly centering SQLi as the primary threat, this phase clearly defines the problem scope and threat model that will be learned by the language model in subsequent stages.

Phase II (data pre-processing): the data pre-processing phase is responsible for transforming raw data into a format suitable for model training. The collected dataset is loaded into the processing environment and converted from a dataframe format into a Hugging Face dataset to ensure compatibility with modern training pipelines. The dataset is then split into training and validation sets to enable fair evaluation and prevent overfitting. This separation also supports performance monitoring during training, ensuring that the model maintains strong generalization capabilities on previously unseen data.

Phase III (modeling and optimization): the modeling and optimization phase focuses on efficiently building and adapting the language model. The base model used in this study is LLaMA 3.2 with 1 billion parameters, which is optimized using 4-bit quantization (NF4) to reduce memory usage and computational overhead. Fine-tuning is performed using the LoRA method, allowing the model to be adapted to the SQLi detection domain without updating all model parameters. This approach results in a relatively lightweight language model with strong domain-specific capabilities while maintaining resource efficiency.

Phase IV (model evaluation): model evaluation aims to assess the performance and effectiveness of the trained model. In this stage, the model is evaluated using a dedicated evaluation dataset to measure its ability to accurately detect and handle SQLi cases. Performance is assessed using relevant evaluation metrics, including accuracy, precision, recall, and F1-score, to ensure that the model not only performs well on training data but also generalizes effectively to unseen data. The evaluation results provide a basis for determining the model's suitability for deployment in real-world application scenarios.

2.1. Data preparation

The dataset utilized in this study was obtained from the SQL-Injection-Extend repository on Kaggle [21]. To provide an overview of the data used for model training and evaluation, Table 1 presents representative examples of malicious SQLi payloads and benign user inputs. The malicious samples include various SQLi techniques such as time-based attacks, UNION-based attacks, and database function exploitation, whereas the benign samples consist of normal user-generated text commonly submitted through web applications. This diversity enables the model to learn contextual differences between malicious and non-malicious inputs. After the data preparation stage, the dataset was transformed into an instruction-based format suitable for supervised fine-tuning of LLMs.

Table 1. Sample of SQLi dataset

No	Sentence	Label
1	“or pg_sleep(_TIME_) --	1
2	select * from users where id = ‘1’ or @@1 = 1 union select 1, version() -- 1’	1
3	One of the other reviewers … after watching just 1 Oz episode you’ll be hooked ...	0
4	Petter Mattei’s … stunning film to watch ...	0

In terms of data distribution, the dataset contains 57,310 malicious sentences (Label = 1) and 52,208 benign sentences (Label = 0). This relatively balanced distribution supports robust training and evaluation of text classification models for automatic SQLi detection in database security research. Although the dataset is derived from Kaggle, it is formulated as a user input classification task, where malicious samples represent SQLi payloads and benign samples represent general user input text submitted through web forms. This approach reflects real-world scenarios in which SQLi attacks are embedded within user-provided input. However, although the benign samples represent general user-generated text, they may not fully capture the diversity and complexity of realistic web application inputs, such as login credentials, uniform resource locator (URL) parameters, API requests, and form submissions. Preprocessing was applied to introduce adversarial variations such as obfuscation and encoding to simulate realistic attack patterns. Therefore, future work will focus on validating the proposed approach using real-world datasets, including WAF logs, web server logs, and security information and event management (SIEM) data sources, to further improve model robustness and generalizability.

2.2. Preprocessing

The preprocessing pipeline is designed to systematically transform the raw dataset into a structured format suitable for instruction-based learning. First, (i) data loading is performed by importing the dataset from a comma-separated values (CSV) file and converting it into a Pandas dataframe for efficient processing. Next, (ii) data type standardization is applied by converting the sentence and label columns into string data types to ensure consistency and prevent type-related errors during training. Subsequently, (iii) label identification is conducted by extracting unique class labels directly from the dataset to maintain flexibility for potential extensions to multi-class classification tasks. The dataset is then divided into (iv) data splits into training and evaluation subsets using an 80:20 ratio, with a fixed random_state = 42 to ensure reproducibility while preserving class distribution. To support instruction-based learning, (v) prompt construction is carried out by transforming each data instance into a structured format consisting of task instructions, label options, input text, and the corresponding output label. Furthermore, (vi) tokenization is applied using the tokenizer associated with the LLaMA 3.2–1B model, where text inputs are converted into token IDs with appropriate padding and truncation to ensure uniform sequence length. This is followed by (vii) special token handling, where padding and end-of-sequence (EOS) tokens are incorporated to maintain proper sequence boundaries. Finally, (viii) evaluation data preparation is performed by generating unlabeled prompts from the evaluation dataset to support inference and performance testing. Through this pipeline, the original tabular dataset is effectively transformed into a model-ready format suitable for supervised fine-tuning of LLMs. In this transformation, each sample is reformulated as an instruction-following classification task consisting of a prompt, input text, and the corresponding label. Table 2 presents representative examples of the converted dataset format used during model training.

Table 2. Conversion of dataframe into hugging face dataset

No	Text
1	Classify the following input into one of these labels: [‘0’, ‘1’] Input: “or pg_sleep (TIME) -- Answer with only one of the labels above. Label: 1
5	Classify the following input into one of these labels: [‘0’, ‘1’] Input: select * from users where id = ‘1’ or @@1 = 1 union select 1, version() -- ‘ Answer with only one of the labels above. Label: 1
6	Classify the following input into one of these labels: [‘0’, ‘1’] Input: One of the other reviewers … watching just 1 Oz episode you’ll be hooked... Answer with only one of the labels above. Label: 0
10	Classify the following input into one of these labels: [‘0’, ‘1’] Input: Peter Mattei’s … is a visually stunning film to watch ... Answer with only one of the labels above. Label: 0

As shown in Table 2, the dataset consists of text samples that have been transformed into an instruction-based format for SQLi detection. Each data instance includes a clear classification instruction, a predefined set of labels [0, 1], and an input that represents user-provided text submitted through web application interfaces. These inputs vary between potentially malicious SQLi payloads and benign user inputs. The malicious samples exhibit common SQLi patterns such as time-based attacks, union-based queries, and system function exploitation. In contrast, the benign samples represent general user input, including natural language text commonly found in web forms such as search queries, comments, and descriptive inputs. This formulation reflects real-world scenarios where SQLi attacks are embedded within user-provided input rather than isolated SQL queries. By combining structured SQL payloads with general user input, the model is trained to distinguish malicious patterns from normal input behavior based on contextual understanding. By framing the task as an instruction-following problem, the dataset becomes suitable for supervised fine-tuning of LLMs, enabling robust evaluation of SQLi detection across diverse input patterns.

2.3. Modeling and optimization

In this study, the LLaMA 3.2–1B model was employed as the base language model for supervised fine-tuning. The model and tokenizer were loaded using the Hugging Face Transformers library. To reduce memory consumption and enable efficient training on limited hardware, NF4 was applied using the BitsAndBytes configuration with NF4 quantization and float16 computation. The model was automatically distributed across available devices using $device\_map = "auto"$. Additionally, the cache mechanism was disabled to support training stability, and the tokenizer padding token was aligned with the EOS token to ensure proper sequence handling during training. This configuration allows efficient fine-tuning of an LLM while maintaining reasonable computational requirements

The core of this system is the LLaMA 3.2–1B model, optimized using NF4 to ensure computational efficiency on limited hardware resources. The detailed environment and model setup are summarized in Table 3. This configuration allows the deployment of a SLM architecture with a significantly reduced memory footprint.

Table 3. Environment and model configuration

No	Component	Configuration
1	Base model	LLaMA 3.2–1B (Meta-LLaMA / LLaMA -3.2-1B)
2	Model type	Causal language model
3	Library	Hugging face transformers
4	Quantization method	NF4 (BitsAndBytes)
5	Quantization type	NF4
6	Double quantization	Disabled
7	Compute data type	Float16
8	Device mapping	Automatic ($device\_map = "auto"$)
9	Cache usage	Disabled ($use\_cache = False$)
10	Pretraining tensor parallelism	1
11	Tokenizer	AutoTokenizer
12	Padding token	EOS token
13	Framework	PyTorch

Table 3 summarizes the environment and model configuration used in this study. The base model employed is LLaMA 3.2–1B, which is a causal language model accessed through the Hugging Face Transformers library. To improve memory efficiency and enable training on limited computational resources, the model was loaded using NF4 with the BitsAndBytes framework. The NF4 quantization type and float16 computation were selected to balance performance and numerical stability, while double quantization was disabled to simplify the quantization process. The model was automatically distributed across available hardware using an automatic device mapping strategy. Cache usage was disabled to support stable training behavior, and the pretraining tensor parallelism parameter was set to one. For text processing, the AutoTokenizer was used, with the padding token aligned to the EOS token to ensure consistent input handling. Overall, the configuration presented in this Table 3 enables efficient fine-tuning of an LLM while maintaining manageable computational requirements.

Figure 3 illustrates the architecture of LoRA. To adapt the pre-trained language model to the SQLi detection task, this study employed LoRA, a parameter-efficient fine-tuning technique designed to reduce computational overhead. Instead of updating the entire set of model parameters, LoRA introduces trainable low-rank matrices into selected layers while preserving the original pre-trained weights. This mechanism

significantly decreases memory consumption and training costs while maintaining the model's ability to learn domain-specific knowledge. Consequently, LoRA offers an effective solution for deploying LLMs in environments with limited computational resources.

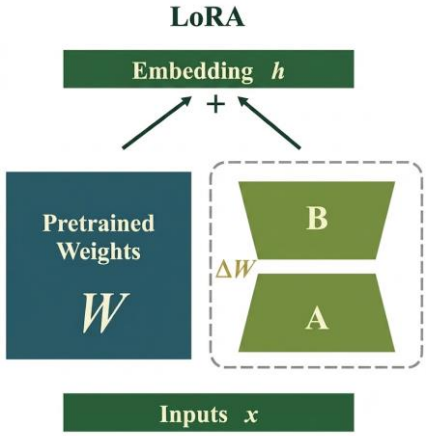

Figure 3. Architecture of LoRA [22]

LoRA enables parameter-efficient fine-tuning by injecting trainable low-rank matrices into selected layers of the base language model while keeping the original model weights frozen. In this study, LoRA is configured with a rank $r = 64$ and a scaling factor $\alpha = 16$, as summarized in Table 4. In addition, the overall training configuration, including learning rate, batch size, optimizer, and training strategy, is presented in Table 5 to ensure clarity and reproducibility. This approach minimizes the risk of catastrophic forgetting and accelerates the training process. Furthermore, comparative studies confirm that LoRA significantly reduces computational overhead compared to full fine-tuning, making it an ideal strategy for resource-constrained cybersecurity deployments. This efficiency is crucial when processing large-scale threat datasets in real-time environments [23].

Table 4. LoRA configuration for fine-tuning

No	Parameter	Value	Description
1	Fine-tuning method	LoRA	Parameter-efficient fine-tuning approach
2	Task type	Causal language modeling	Model predicts next token autoregressively
3	Rank (r)	64	Dimension of low-rank matrices
4	LoRA alpha	16	Scaling factor for LoRA updates
5	Bias	None	No bias parameters were trained
6	Target modules	Selected model layers	Modules adapted using LoRA
7	Trainable parameters	LoRA layers only	Base model weights frozen
8	Parameter-efficient fine-tuning (PEFT)	Hugging Face PEFT	Used for LoRA implementation

Table 5. Training hyperparameters

No	Parameter	Value
1	Epochs	1
2	Learning rate	2×10^{-4}
3	Per-device batch size	1
4	Gradient accumulation	8
5	Effective batch size	8
6	Optimizer	Paged AdamW (32-bit)
7	Weight decay	0.001
8	LR scheduler	Cosine
9	Warmup ratio	0.03
10	Max gradient norm	0.3
11	Gradient checkpointing	Enabled
12	Precision	FP16
13	Evaluation strategy	Steps (every 50 steps)

Table 4 presents the configuration of the LoRA method used in this study. The Table 4 details the key hyperparameters that control how LoRA adapts the base language model, including the rank, scaling factor, and dropout rate. These settings determine the capacity and stability of the adaptation while keeping the number of trainable parameters low. By applying LoRA only to selected target modules and freezing the original model weights, the fine-tuning process becomes more efficient in terms of memory usage and training time, making it suitable for LLM training on limited computational resources.

Table 5 presents the main training hyperparameters used in the fine-tuning process. These parameters define the optimization strategy, training duration, and memory management settings applied during model training. The Table 5 shows that the model was trained for a single epoch using a learning rate of 2×10^{-4}. A single epoch was selected because preliminary experiments showed convergence of evaluation loss without additional performance gains from longer training, while also reducing computational costs. Gradient accumulation and checkpointing were employed to improve training efficiency, while a cosine learning rate scheduler with warmup was used to stabilize the optimization process.

2.4. Model evaluation metrics

To assess the performance of the proposed model, we employed a confusion matrix and derived standard classification metrics: accuracy, precision, recall, and F1-score. These metrics provide a comprehensive view of the model's ability to minimize both false positives (benign queries flagged as attacks) and false negatives (attacks bypassing detection). The evaluation metrics are calculated using (1) to (4) [24]:

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

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

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

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

The primary performance indicators—accuracy, precision, recall, and F1-score—are computed to measure the model's overall performance. Within the confusion matrix, TP and TN denote positive and negative samples that are classified correctly, whereas FP and FN refer to instances in which the system produces incorrect predictions. The mathematical definitions of each metric are provided in (1)–(4).

3. RESULTS AND DISCUSSION

In this study, the experiments were conducted using a benchmark dataset derived from Kaggle, designed for SQLi detection. The dataset consists of both malicious SQLi inputs and benign text samples, which were used to train and evaluate the proposed model. Model training was performed using the LLaMA 3.2–1B architecture with the training configuration described in Table 4. The training and evaluation processes were monitored using loss values, where the training loss is presented in Figure 4, and the evaluation loss is shown in Figure 5. The results discussed in this section highlight the learning behavior of the model and its effectiveness in distinguishing between malicious and non-malicious inputs.

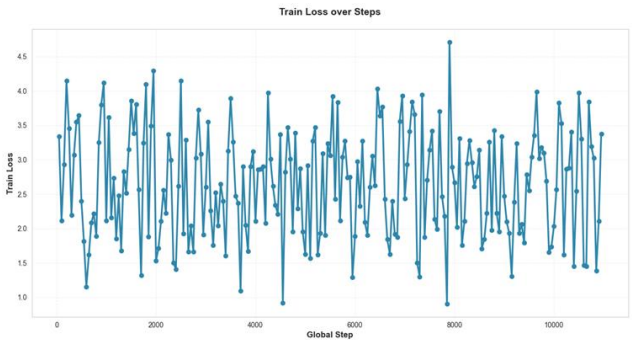

Figure 4. Training loss progression over global steps

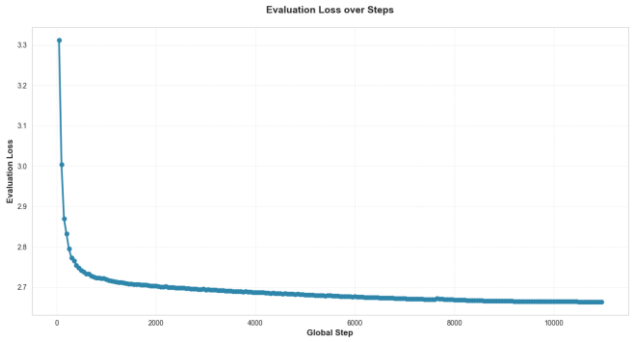

Figure 5. Evaluation loss progression over global steps

Figure 4 illustrates the trend of training loss throughout the fine-tuning process across global training steps. Variations in the loss values are observed during training, which is expected when stochastic optimization, small batch sizes, and gradient accumulation are employed. Despite these fluctuations, the overall loss remains within a stable range, indicating that the optimization process proceeds in a controlled manner. The highest training loss value of 5.1391 was recorded at global step 2 during the initial adaptation stage of the model. As training progressed, the loss generally declined and exhibited a more stable pattern. The minimum loss value of 0.4 was achieved at global step 7643, suggesting effective optimization during fine-tuning. Overall, the decreasing trend in training loss demonstrates the model's capability to learn relevant representations from the dataset without exhibiting signs of divergence or unstable behavior. This observation is consistent with the characteristics of LoRA-based fine-tuning performed using a limited number of epochs and a relatively small effective batch size.

Complementing the training loss**,** Figure 5 presents the evaluation loss on the validation set. Figure 5 presents the evaluation loss trend over global training steps. At the early stage of training, the evaluation loss shows a relatively high value, reflecting the model's limited generalization capability before sufficient learning has taken place. As training progresses, the evaluation loss decreases steadily, indicating improved model performance on the evaluation dataset. Based on the observed results, the maximum evaluation loss of 3.3124 occurs at global step 50, which corresponds to the initial phase of training. The evaluation loss continues to decline and reaches its minimum value of 2.6640 at global step 10,800, suggesting that the model achieves its best generalization performance toward the end of the training process. Overall, the consistent reduction and stabilization of the evaluation loss indicate that the model successfully learns relevant patterns from the training data without exhibiting signs of overfitting. This behavior demonstrates that the applied training configuration and LoRA-based fine-tuning strategy effectively support stable generalization on unseen data.

Based on the evaluation loss analysis, the model demonstrates stable and consistent generalization performance on the evaluation dataset. To further assess the classification performance in terms of prediction outcomes, the evaluation is continued using a confusion matrix, which provides a detailed view of true and false predictions across classes. The results of the confusion matrix are presented in Figure 6.

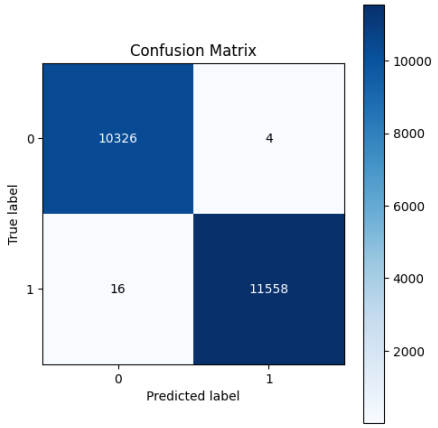

Figure 6. Confusion matrix of the fine-tuned LLaMA 3.2 model

Figure 6 provides a detailed overview of the classification outcomes generated by the proposed model on the evaluation dataset. The confusion matrix indicates that 10,326 benign samples were correctly identified as non-malicious (true negatives/TN), while 11,558 SQLi samples were accurately classified as malicious (true positives/TP). Misclassification cases were limited, consisting of four false positives and sixteen false negatives. These findings demonstrate the model's strong capability to differentiate between legitimate user inputs and SQLi attempts. Furthermore, the relatively low number of false alarms suggests that the proposed approach can effectively minimize unnecessary blocking of benign requests while maintaining high sensitivity toward malicious patterns. Overall, the confusion matrix confirms the reliability and robustness of the proposed framework for SQLi detection [25].

Table 6 presents a benchmark comparison between the proposed LLaMA 3.2–LoRA model and various traditional ML and DL baselines. The RF model achieves an accuracy of 97.48%, with a precision of 98.75%, a recall of 96.49%, and an F1-score of 97.61%, indicating strong and balanced performance. The XGBoost model records an accuracy of 96.96%, precision of 99.37%, recall of 94.89%, and an F1-score of 97.08%, showing high precision but slightly lower recall. Similarly, the DT model achieves 96.61% accuracy, 99.06% precision, 94.54% recall, and an F1-score of 96.75%. Among DL approaches, the CNN model attains an accuracy of 97.84%, precision of 99.52%, recall of 96.42%, and an F1-score of 97.94%, demonstrating competitive performance compared to traditional ML models. The LSTM model shows lower but consistent results, with an accuracy of 92.25%, precision of 92.84%, recall of 92.60%, and an F1-score of 92.72%. The baseline LLaMA 3.2–1B model without fine-tuning achieves an accuracy of only 50.21%, despite a high precision of 99.26%. However, its recall drops significantly to 5.78%, resulting in a low F1-score of 10.93%, indicating poor detection capability without task-specific adaptation. In contrast, the fine-tuned LLaMA 3.2–1B model using LoRA achieves the best overall performance, with an accuracy of 99.91%, precision of 99.96%, recall of 99.86%, and an F1-score of 99.91%. These results confirm that LoRA-based fine-tuning substantially improves the effectiveness of LLMs for SQLi detection.

Table 6. Comparison of model performance against baselines

No	Model	Accuracy	Precision	Recall	F1-score
1	RF	97.48%	98.75%	96.49%	97.61%
2	XGBoost	96.96%	99.37%	94.89%	97.08%
3	DT	96.61%	99.06%	94.54%	96.75%
4	CNN	97.84%	99.52%	96.42%	97.94%
5	LSTM	92.25%	92.84%	92.60%	92.72%
6	LLaMa 3.2 1B	50.21%	99.26%	5.78%	10.93%
7	Fine tuning LLaMa 3.2 1B (LoRa)	99.91%	99.96%	99.86%	99.91%

In addition to the experimental evaluation, the proposed model has been implemented in a web-based application, as illustrated in Figure 1. The web-based system allows users to submit SQL queries or textual inputs and receive real-time classification results indicating whether the input is malicious or benign. This implementation demonstrates that the fine-tuned model can be effectively integrated into practical security systems, such as WAFs or database security monitoring platforms. Based on Figure 1, the system architecture confirms the feasibility of deploying the proposed approach in real-world environments beyond offline experimentation. Figure 7 presents examples of the system's real-time detection capability using both malicious and benign inputs. The malicious example consists of a login request containing the SQLi payload `' OR 1=1 --`, whereas the benign example consists of a film review submitted as ordinary user-generated text. The results demonstrate that the proposed system is capable of accurately distinguishing malicious SQLi attempts from legitimate user inputs in practical deployment scenarios.

As a result, the system displays a notification indicating that the user is attempting a Boolean-based SQLi attack, and access to the system is denied. Furthermore, for digital forensic purposes, the model provides a contextual explanation identifying the detected attack along with recommended mitigation actions. The output of this digital forensic analysis can be observed in Figure 8. This capability supports not only real-time attack prevention but also post-incident investigation by assisting administrators in understanding the attack characteristics and appropriate response measures.

Figure 8 presents the digital forensic report generated by the proposed system for detected SQLi events. The report records detailed information for each detected attack, including the timestamp, source internet protocol (IP) address, user agent, target field, detected payload, attack type, recommended mitigation, and system action. The results show that the system is able to accurately identify various types of SQLi attacks, such as Boolean-based, Time-based, UNION-based, and Generic SQLi patterns. Each malicious payload is clearly highlighted, allowing security analysts to quickly understand the structure of the

attack. In all detected cases, the system automatically blocks the malicious request, as indicated by the BLOCKED status in the action column. Additionally, the forensic report provides contextual mitigation recommendations, such as the use of prepared statements, input sanitization, and the deployment of WAF. This comprehensive logging mechanism supports post-incident digital forensic analysis by enabling administrators to trace attack behavior, identify attack vectors, and apply appropriate security countermeasures. Overall, the results demonstrate that the proposed model not only detects SQLi attacks effectively but also supports forensic investigation and incident response. This context-aware reporting significantly aids security analysts in post-incident investigation and aligns with the growing need for explainable artificial intelligence (XAI) in cybersecurity [6].

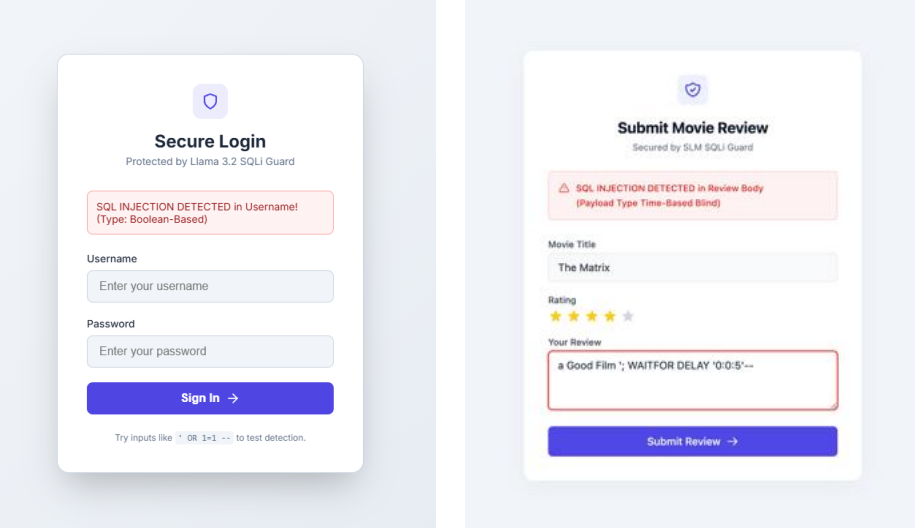

Figure 7. Real-time SQLi detection interface

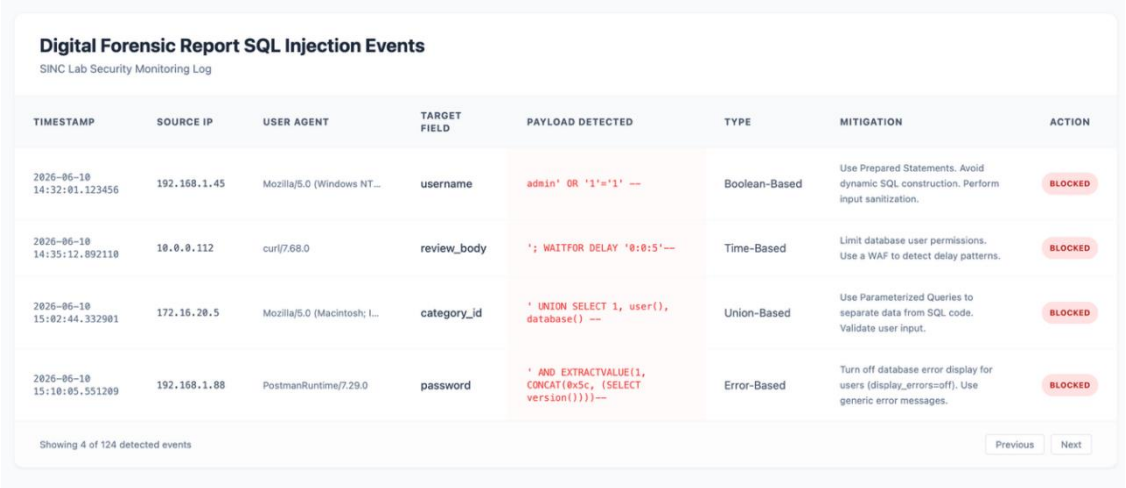

Figure 8. Context-aware digital forensic report generated by the model

4. CONCLUSION

This study demonstrates that integrating LoRA with the LLaMA 3.2–1B model substantially enhances the effectiveness of SQLi detection while maintaining computational efficiency. The proposed approach successfully combines context-aware semantic understanding with parameter-efficient fine-tuning, enabling accurate detection of malicious inputs under resource-constrained environments. Beyond classification performance, the model was successfully integrated into a web-based system capable of real-time SQLi detection, automatic request blocking, and the generation of context-aware digital forensic reports.

These capabilities improve both the practical usability and interpretability of SQLi detection systems in modern cybersecurity applications. Despite these promising findings, several limitations should be acknowledged. The dataset used in this study is primarily derived from Kaggle and may not fully capture the diversity and complexity of real-world web application traffic. In addition, the evaluation was conducted within a controlled experimental setting and has not yet been validated using production-level datasets such as WAF logs, web server logs, or SIEM data sources. Due to the computational demands associated with fine-tuning LLMs, this study employed a hold-out validation strategy instead of k-fold cross-validation. Future work will focus on addressing these limitations by incorporating more realistic datasets, performing broader validation studies through cross-validation and independent testing, and investigating deployment strategies for operational cybersecurity environments.

FUNDING INFORMATION

The authors declare that this research received no external funding. The article processing charge (APC) was supported by the Institute for Research and Community Service (LPPM), Institut Teknologi Nasional Bandung.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Diash Firdaus	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Idi Sumardi	✓	✓		✓	✓		✓		✓			✓		
Muhammad Ichwan	✓			✓	✓		✓			✓				
Maulana Seno Aji Yudhantara	✓								✓	✓	✓			

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The authors declare no conflict of interest.

DATA AVAILABILITY

The data and source code presented in this study are available on request from the corresponding author. The data are not publicly available because the data source involves private or sensitive information.

REFERENCES

[1] M. Al-Olaqi, A. Al-Gailani, and M. M. H. Rahman, "Comprehensive Study of SQL Injection Attacks Mitigation Methods and Future Directions," *Journal of Cyber Security and Risk Auditing*, vol. 2025, no. 4, pp. 347–365, 2025, doi: 10.63180/jcsra.thestap.2025.4.11.

[2] M. C. Wijaya, "Security Analysis of SQL Injection Attacks on Multimedia and Journal-Services Sites Using Concatenated Input Validation and Parsing Method (CIVP)," *Ingenierie des Systemes d'Information*, vol. 29, no. 5, pp. 1915–1924, Oct. 2024, doi: 10.18280/isi.290523.

[3] J. Choi, Y. A. Jung, and H. Ko, "Comparative Analysis of SQL Injection Defense Mechanisms Based on Three Approaches: PDO, PVT, and ART," *Applied Sciences (Switzerland)*, vol. 15, no. 23, p. 12351, Nov. 2025, doi: 10.3390/app152312351.

[4] N. Prasad, A. Diro, M. Warren, and M. Fernando, "A survey of cyber threat attribution: Challenges, techniques, and future directions," *Computers and Security*, vol. 157, p. 104606, Oct. 2025, doi: 10.1016/j.cose.2025.104606.

[5] C. M. Rosca, A. Stancu, and C. Popescu, "Machine Learning Models for SQL Injection Detection," *Electronics (Switzerland)*, vol. 14, no. 17, p. 3420, Aug. 2025, doi: 10.3390/electronics14173420.

[6] S. Szymoniak, J. Piątkowski, and M. Kurkowski, "Defense and Security Mechanisms in the Internet of Things: A Review," *Applied Sciences (Switzerland)*, vol. 15, no. 2, p. 499, Jan. 2025, doi: 10.3390/app15020499.

[7] F. K. Alarfaj and N. A. Khan, "Enhancing the Performance of SQL Injection Attack Detection through Probabilistic Neural Networks," *Applied Sciences (Switzerland)*, vol. 13, no. 7, 2023, doi: 10.3390/app13074365.

[8] A. Arnap and Kusrini, "Enhancing SQL Injection Attack Detection Using Naïve Bayes and SMOTE Method on Imbalanced Datasets," *Journal of Artificial Intelligence and Engineering Applications (JAIEA)*, vol. 4, no. 1, pp. 74–81, 2024, doi: 10.59934/jaiea.v4i1.559.
[9] H. Dehariya, P. Kumar, and M. Ahirwar, "A Survey on Detection and Prevention Techniques of SQL Injection Attacks," *International Journal of Computer Applications*, vol. 137, no. 5, pp. 9–15, Mar. 2016, doi: 10.5120/ijca2016908672.
[10] M. Alqhtani, D. Alghazzawi, and S. Alarifi, "Black-Box Adversarial Attacks Against SQL Injection Detection Model," *Contemporary Mathematics (Singapore)*, vol. 5, no. 4, pp. 5098–5112, Nov. 2024, doi: 10.37256/cm.5420245292.
[11] A. Falor, M. Hirani, H. Vedant, P. Mehta, and D. Krishnan, "A Deep Learning Approach for Detection of SQL Injection Attacks Using Convolutional Neural Networks," *Lecture Notes on Data Engineering and Communications Technologies*, vol. 91, pp. 293–304, 2022, doi: 10.1007/978-981-16-6285-0_24.
[12] S. Raut, A. Nikhare, Y. Punde, S. Manerao, and S. Choudhary, "A Review on Methods for Prevention of SQL Injection Attack," *International Journal of Scientific Research in Science and Technology*, pp. 463–470, Apr. 2019, doi: 10.32628/ijsrst196258.
[13] A. G. Kakisim, "A deep learning approach based on multi-view consensus for SQL injection detection," *International Journal of Information Security*, vol. 23, no. 2, pp. 1541–1556, Apr. 2024, doi: 10.1007/s10207-023-00791-y.
[14] S. Patil *et al.*, "Explainable Artificial Intelligence for Intrusion Detection System," *Electronics (Switzerland)*, vol. 11, no. 19, p. 3079, Sep. 2022, doi: 10.3390/electronics11193079.
[15] A. A. Mustapha, A. S. Udeh, T. A. Ashi, O. S. Sobowale, M. J. Akinwande, and A. O. Oteniara, "Comprehensive review of machine learning models for sql injection detection in e-commerce," *World Journal of Advanced Research and Reviews*, vol. 23, no. 1, pp. 451–465, Jul. 2024, doi: 10.30574/wjarr.2024.23.1.2004.
[16] X. Wang, J. Zhai, and H. Yang, "Detecting command injection attacks in web applications based on novel deep learning methods," *Scientific Reports*, vol. 14, no. 1, p. 25487, Oct. 2024, doi: 10.1038/s41598-024-74350-3.
[17] G. Palma, G. Cecchi, M. Caronna, and A. Rizzo, "Leveraging Large Language Models for Scalable and Explainable Cybersecurity Log Analysis," *Journal of Cybersecurity and Privacy*, vol. 5, no. 3, p. 55, Aug. 2025, doi: 10.3390/jcp5030055.
[18] J. Hong, N. V. Tu, and J. W. K. Hong, "A Comprehensive Survey on LLM-Based Network Management and Operations," *International Journal of Network Management*, vol. 35, no. 6, Nov. 2025, doi: 10.1002/nem.70029.
[19] F. Corradini, M. Leonesi, and M. Piangerelli, "State of the Art and Future Directions of Small Language Models: A Systematic Review," *Big Data and Cognitive Computing*, vol. 9, no. 7, p. 189, Jul. 2025, doi: 10.3390/bdcc9070189.
[20] S. Yu, N. Ran, and J. Liu, "Large-language models: The game-changers for materials science research," *Artificial Intelligence Chemistry*, vol. 2, no. 2, p. 100076, Dec. 2024, doi: 10.1016/j.aichem.2024.100076.
[21] "SQL-Injection-Extend." https://www.kaggle.com/datasets/alextrinity/sqlinjectionextend (accessed Jan. 30, 2026).
[22] Y. Han, J. Liu, A. Luo, Y. Wang, and S. Bao, "Fine-Tuning LLM-Assisted Chinese Disaster Geospatial Intelligence Extraction and Case Studies," *ISPRS International Journal of Geo-Information*, vol. 14, no. 2, p. 79, Feb. 2025, doi: 10.3390/ijgi14020079.
[23] F. Alghamdi and B. Ben Ammar, "Enhancing SQL Code Security and Maintainability: A Deep Learning Based Approach," *International Journal of Advances in Artificial Intelligence and Machine Learning*, vol. 2, no. 3, pp. 160–169, Nov. 2025, doi: 10.58723/ijaaiml.v2i3.515.
[24] D. Firdaus, R. Munadi, and Y. Purwanto, "DDoS Attack Detection in Software Defined Network using Ensemble K-means++ and Random Forest," in *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems, ISRITI 2020*, IEEE, Dec. 2020, pp. 164–169, doi: 10.1109/ISRITI51436.2020.9315521.
[25] J. Lin and D. Mohaisen, "From Large to Mammoth: A Comparative Evaluation of Large Language Models in Zero-Shot Vulnerability Detection," in *Proceedings 2025 Network and Distributed System Security Symposium*, Reston, VA: Internet Society, 2025, doi: 10.14722/ndss.2025.241491.

BIOGRAPHIES OF AUTHORS

Diash Firdaus received the B.Eng. degree in Informatics engineering from STMIK JABAR and the M.Eng. degree in Electrical Engineering from Telkom University, Bandung, Indonesia, in 2018 and 2021, respectively. He is currently a lecturer and the head of the SINC (Information and Communication Technology and Cybersecurity) Laboratory, Institut Teknologi Nasional (ITENAS), Bandung. His current research interests include machine learning, deep learning, multimodal large language models (MLLM) security, and the application of IoT in cybersecurity. He has authored numerous publications in the areas of DDoS detection, SQL injection, and AI-driven network security. He can be contacted at email: diashfirdaus@gmail.com.

Idi Sumardi received his B.Eng. from STMIK JABAR and M.Eng. degrees from the Bandung Institute of Technology (ITB), Bandung, Indonesia. He is currently a senior lecturer and the dean of the Faculty of Information Technology at Universitas Al-Ghifari (UNFARI), Bandung. His research interests include computer networks, information technology, and the application of machine learning in cybersecurity and healthcare. He has authored numerous publications in the areas of phishing website detection, large language models (LLM), and network quality of service (QoS). He can be contacted at email: idisumardi@gmail.com.

Muhammad Ichwan received the B.Eng. degree in Electrical Engineering from Institut Teknologi Nasional (ITENAS) and the M.Eng. degree in STEI from the Bandung Institute of Technology (ITB), Bandung, Indonesia. He is currently a senior lecturer in the Department of Informatics Engineering, Institut Teknologi Nasional (ITENAS), Bandung. His research interests include informatics systems, software engineering, and the application of artificial intelligence in pattern recognition. He has authored numerous publications and held intellectual property rights in the areas of enterprise architecture, academic information systems. He can be contacted at email: ichwan@itenas.ac.id.

Maulana Seno Aji Yudhantara is currently pursuing a Bachelor's degree in Informatics at Institut Teknologi Nasional Bandung (ITENAS), Bandung, Indonesia. As a final-year student, he is highly active as a practitioner and developer in the field of Artificial Intelligence. He currently serves as an AI engineer facilitator at a prominent technology bootcamp, guiding students in mastering industry-standard AI concepts. His primary research and practical interests encompass machine learning, deep learning, and computer vision. He is deeply passionate about building intelligent systems and applying AI engineering methodologies to solve real-world problems. He can be contacted at email: senoaji115@gmail.com.