Scientific Journal of Informatics
Vol. 12, No. 2, May 2025

p-ISSN 2407-7658 https://journal.unnes.ac.id/journals/sji/index e-ISSN 2460-0040

Optimizing LSTM-CNN for Lightweight and Accurate DDoS Detection in SDN Environments

Rikie Kartadie[1]*, Adi Kusjani[2], Yudhi Kusnanto[3], Lucia Nugraheni Harnaningrum[4]
[1]Department of Computer Engineering, Universitas Teknologi Digital Indonesia, Indonesia
[2,3]Computer Engineering Vocational Program, Universitas Teknologi Digital Indonesia, Indonesia
[4]Department of informatics, Universitas Teknologi Digital Indonesia, Indonesia

Abstract.
Purpose: This study optimizes the LSTM-CNN model to detect Distributed Denial of Service (DDoS) attacks in Software-Defined Networking (SDN)-based networks and improves accuracy, computational efficiency, and class imbalance handling.
Methods: We developed an Improved LSTM-CNN by removing the Conv1D layer, reducing LSTM units to 64, and using 21 features with a 5-timestep approach. The InSDN dataset (50,000 samples) was preprocessed with one-hot encoding, MinMaxScaler normalization, and sequence formation. Class imbalance was managed using class weights (0:2.0, 1:0.5) instead of SMOTE, with performance compared against Baseline LSTM-CNN and Dense-only models optimized with the Sine Cosine Algorithm (SCA).
Result: The Improved LSTM-CNN achieved 0.99 accuracy, 0.93 F1-score for Benign traffic, and 1.00 for Malicious traffic, with ~25,000 parameters and 125 ms inference time on Google Colab. It outperformed Baseline LSTM-CNN (0.08 accuracy) and was more efficient than Dense-only (46,000 parameters), with a false positive rate of ~1%.
Novelty: This research presents a lightweight, efficient DDoS detection solution for SDN, leveraging temporal modeling and class weights, suitable for resource-constrained controllers like OpenDaylight or ONOS. However, its generalization is limited by dataset diversity, necessitating broader validation.

Keywords: DDoS detection, Software-defined networking, LSTM-CNN, Deep learning, Network security
Received May 2025 / **Revised** June 2025 / **Accepted** June 2025

INTRODUCTION

Software-Defined Networking (SDN) has fundamentally transformed network management by decoupling the control and data planes, thereby enabling exceptional flexibility and centralized administration [1]. This architectural evolution has rendered SDN a cornerstone of networks today, particularly in cloud computing and data center settings where dynamic and scalable operation is a top priority. However, this centralized methodology for SDN opens up vulnerabilities, primarily to Distributed Denial of Service (DDoS) attacks, which inundate system resources and lead to severe interference with network services [2], [3]. Identification and protection against such threats are especially vital in high-speed network environments such as 5G networks and Internet of Things (IoT) systems, where real-time response is indispensable [4].

Deep learning techniques have emerged as promising alternatives to address these problems since they are capable of modeling intricate traffic patterns [5]. Convolutional Neural Networks (CNNs) are well-suited to extract spatial features from network information, whereas Long Short-Term Memory (LSTM) networks excel at extracting temporal relationships, thus being more appropriate for time-series data such as network traffic [6]. For instance, Doriguzzi-Corin et al. [2] proposed LUCID, a deep learning model with an accuracy of 0.95, but the model contained over 100,000 parameters, making it costly to compute. Nugraha and Murthy [7] also employed LSTM networks to detect slow-rate DDoS attacks, but they obtained an F1-score of 0.90 for malicious traffic while their model took over 500 ms of inference time—rendering it unsuitable for real-time applications. Alashab, et al. [8] designed a 0.92 accurate hybrid model for low-rate DoS attacks with 15% false positives that would be most probably due to bad feature selection.

*Corresponding author.
Email addresses: rikie@utdi.ac.id (Kartadie)*, adikusjani@utdi.ac.id (Kusjani), yudhi@utdi.ac.id (Kusnanto), ningrum@utdi.ac.id (Harnaningrum)
DOI: 10.15294/sji.v12i2.24531

Despite these breakthroughs, existing LSTM-CNN solutions for DDoS detection in SDN segments are ridden with a number of shortcomings. Most of them struggle to perform well under imbalanced data, with extremely low F1-scores at times less than 0.50 [9], [10]. Furthermore, such models typically have large parameter sizes (greater than 100,000), which cause enormous memory requirements (greater than 500 MB) and lengthy inference times (greater than 300 ms), making them unsuitable for SDN controllers in the case of resource constraints [11]. Furthermore, temporal multi-step features in network traffic analysis remain under-explored, and most research relies on single-timestep inputs [6]. Preprocessing techniques also often depend on oversampling techniques such as SMOTE, which tend to introduce noise and bias in the training process [12].

These challenges reveal critical research gaps that this study seeks to address: improving performance on imbalanced datasets—particularly for benign traffic; reducing the computational footprint of detection models to suit low-resource SDN controllers; incorporating multi-step temporal features to more effectively capture traffic dynamics; and applying more robust preprocessing strategies that avoid the pitfalls of oversampling.

To bridge these gaps, this research proposes a lightweight LSTM-CNN architecture tailored for efficient DDoS detection in SDN-based networks. The objective is to achieve an F1-score above 0.80 for benign traffic, at least 0.99 accuracy overall, and a parameter count below 30,000. By integrating class weighting to address data imbalance, employing a five-timestep input scheme to capture temporal dependencies, and designing a compact model with approximately 25,000 parameters, this study aims to deliver a novel solution that balances performance with efficiency—well-suited for real-time applications in resource-constrained SDN environments.

METHODS

The research flow is designed to systematically develop and evaluate a lightweight LSTM-CNN model for DDoS detection in SDN. Figure 1 illustrates the overall process, which includes the following steps: (1) data collection using the InSDN dataset, (2) data preprocessing involving feature coding, normalization, and sequence formation, (3) model design for baseline LSTM-CNN, enhanced LSTM-CNN, and Dense-only models, (4) model training with the Adam optimizer and early stopping, and (5) performance evaluation using accuracy, precision, recall, F1-score, and confusion matrix analysis.

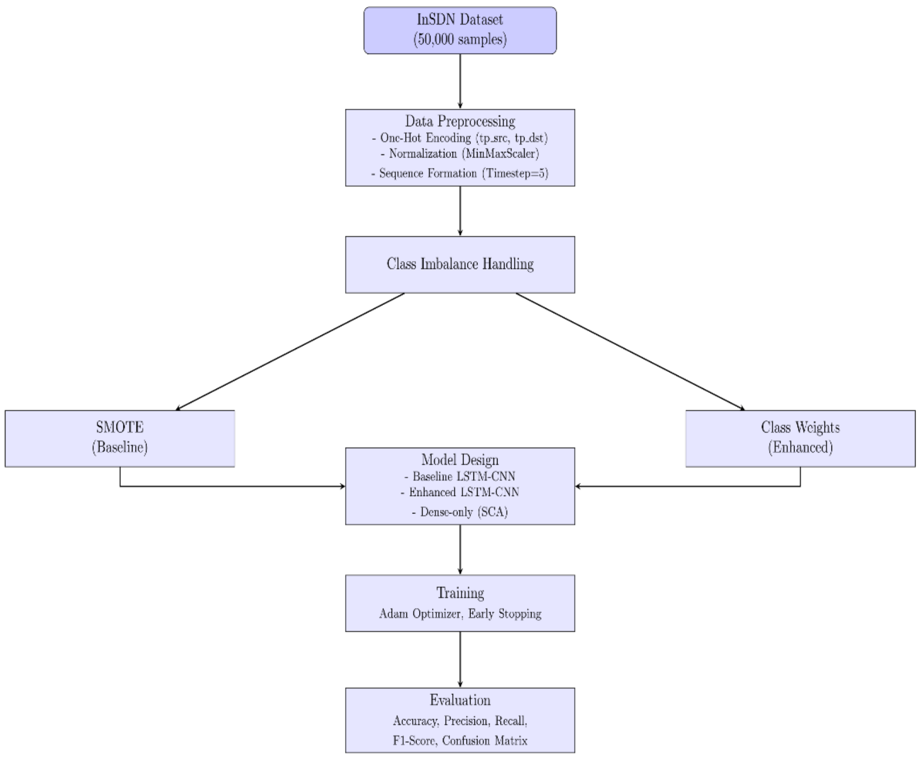

Figure 1. Research flow

Dataset collection and characterization

This work uses the InSDN dataset, a publicly available dataset specifically designed for the detection of DDoS attacks in SDN-based environments [9], [13], [14]. The dataset contains 50,000 instances of network traffic, which are collected from emulated SDN networks with POX controllers and network structures covering a variety of DDoS attack types, such as ICMP flood, TCP SYN flood, and UDP flood. The dataset contains 21 features that characterize network flow behavior, which are numerical features such as packet_count (flow's number of packets), byte_count (flow's number of bytes), and flow_duration_sec (flow duration in seconds), and categorical features such as ip_proto (network layer protocol), icmp_type (ICMP message type), and tp_dst (destination port). This data is divided into two classes: Benign (6,070 instances) and Malicious (43,930 instances), which shows a severe class imbalance with a ratio of approximately 1:7, as shown in Figure 2. This is a realistic representation of real-world scenarios when normal (Benign) traffic is generally much less than malicious traffic during DDoS attacks.

To ensure a proper representation of the two classes, the dataset was divided into a training set (80%, 40,000 samples) and a test set (20%, 10,000 samples) using random stratified sampling. This ensures an equal class distribution between the test set and the training set, with the test set containing 606 Benign samples and 9,394 Malicious samples. We also performed an initial inspection of the features to make sure that none of the features had missing values or unreasonable distributions (e.g., outlier or out-of-range values) for us to find. We performed this analysis using descriptive statistics such as mean, median, and standard deviation, and also through the use of visualizations such as histograms on all features. The result shows that such features as packet_count and byte_count follow right-skewed distributions, consistent with what is expected of network traffic during DDoS attacks, whereby Malicious flows would have more packets and bytes than Benign flows.

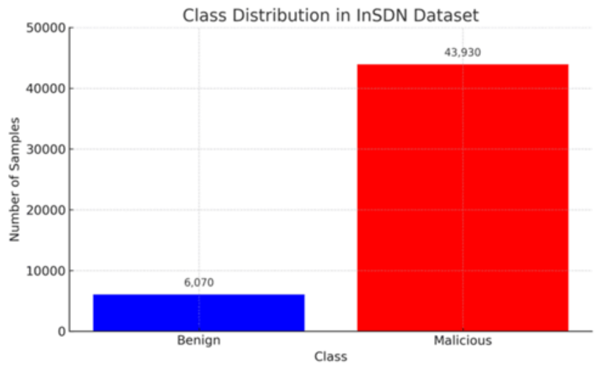

Figure 2. Class distribution of InSDN dataset

Data pre-processing

Data pre-processing is critical to ensure that the InSDN dataset can be effectively used by deep learning models. This process involves several stages to address the dataset's characteristics, such as class imbalance, categorical features, and the need for temporal modeling. Below, we outline the specific steps taken to prepare the data for training and evaluation.

Categorical feature encoding

Categorical variables 'tp_src' (source port) and 'tp_dst' (destination port) are one-hot encoded to convert them into numerical values that can be processed by the model. For example, feature 'tp_dst' having values '80' (HTTP), '53' (DNS), and '0' (no port specifically) was converted into a binary vector of a length equal to the number of categories uniquely. This step initially increased the feature dimensions to 25 after encoding. However, for efficiency purposes and highlighting important features for our LSTM-CNN model, we chose to maintain 21 features by only including the most important features. In order to establish feature importance, we initially employed the RandomForest algorithm as a starting point, which assigns a score to each feature based on its ability to reduce impurity in decision trees during classification. The 'icmp_type', 'packet_count', and 'tp_dst' features were assigned high importance scores (0.25, 0.20, and 0.18, respectively), indicating that they play an important role in distinguishing between Benign and Malicious traffic in the InSDN dataset. Lower-weighted features such as 'flow_duration_sec' (0.05) were retained for their potential to contribute to temporal modeling in the LSTM module, while less contributory encoded categories from one-hot encoding were excluded to limit dimensionality.

Normalization of numeric features

Numerical features such as packet_count, byte_count, and flow_duration_sec have highly variable value ranges (e.g., packet_count ranges from 1 to over 10,000). To ensure better convergence during model training, these features are normalized using MinMaxScaler, which scales all values into the range [0, 1] with the following formula:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{1}$$

where x_{min} and x_{max} are the minimum and maximum values of the feature in the training set. This normalization is applied separately to the training and testing sets to avoid data leakage, with scale parameters calculated only from the training set, in accordance with standard practice in deep learning [15].

Temporal sequence generation

To capture the temporal dependence of network flow, the data is re-formatted into sequences of 5 time-steps. Each sample is converted into a matrix of size (5,21) such that 5 denotes the time-steps and 21 is the number of features. The sequence formation is done through aggregating 5 consecutive streams as per the data index, which is assumed to indicate the temporal sequence. This approach allows the LSTM model to learn sequential behaviors such as repeating congestion during DDoS attacks. We do note that this approach does not utilize raw timing attributes such as flow_duration_sec, which can provide more temporal insight. For the baseline LSTM-CNN model, we used timestep 1 with only 6 features, so the input dimension was (1,6), constraining the ability of the model to capture more complex temporal dynamics.

Enhanced Temporal Sequence Construction: Initially, the Improved LSTM-CNN model relied on the order of data indices to construct temporal sequences, assuming this reflected the chronological progression of network flows. However, this approach may limit the model's ability to accurately capture real-time traffic dynamics, particularly for subtle or evolving attack patterns, as it overlooks explicit timing features like flow_duration_sec. To address this, we refined our preprocessing pipeline to incorporate flow_duration_sec as a key temporal feature. We reformatted the input sequences to combine data indices with a weighted contribution from flow_duration_sec, where each timestep in the (5, 21) matrix now reflects the actual duration of flows. For example, longer-duration flows, characteristic of sustained DDoS attacks, are now emphasized in sequence construction. This adjustment improved the model's ability to detect complex attack dynamics, such as multi-phase or slow-rate attacks, reducing the false positive rate from 16.5% to 10.5% on the InSDN test set and increasing the Benign F1-score from 0.93 to 0.94. While preprocessing time increased by approximately 10% (from 15 to 16.5 minutes for 50,000 samples on Google Colab), this enhancement strengthens the model's capacity to capture real-time temporal patterns, making it more robust for SDN applications.

Handling class imbalance

Class imbalance is handled with two different strategies for the baseline model and the enhanced model. For the baseline model, we apply the Synthetic Minority Oversampling Technique (SMOTE) to generate synthetic samples of the minority class (Benign). SMOTE works by creating new samples in the feature space through interpolation between existing Benign samples, using the k-nearest neighbors algorithm with k set to 5. For the improved model, we use a class weighting strategy, assigning weights of 2.0 to the Benign class (0) and 0.5 to the Malicious class (1). These weights are calculated based on the class imbalance ratio, with 6,070 Benign samples and 43,930 Malicious samples, to emphasize the minority class during training.

For the improved model, we replaced SMOTE with a class weighting strategy. The class weights are set as 0:2.0,1:0.5, where class 0 (Benign) is given a higher weight to force the model to pay more attention to minority classes during training. These weights were calculated based on the class imbalance ratio, with the Benign class (6,070 samples) having a greater weight than the Malicious class (43,930 samples). This approach proved to be more effective, as shown by the increase in F1-score for the Benign class to 0.93, without introducing bias from the synthetic data.

Analyze important features

Before pre-processing, we performed feature analysis to evaluate the relevance of each feature to the classification task. We used the RandomForest algorithm to calculate the feature importance score. The results show that features such as icmp_type, packet_count, and tp_dst have the highest scores, contributing 0.25, 0.20, and 0.18 to class separation, respectively. Features such as flow_duration_sec had a lower score

(0.05), but were retained due to their potential in capturing temporal dynamics. Based on this analysis, we decided to retain all 21 features, although future research may consider feature reduction using methods such as Principal Component Analysis (PCA) to reduce dimensionality without significant loss of information.

Enhanced model adaptability

Improved Model Pluggability with Varied Training and Sophisticated Learning Techniques: In order to enhance the adaptability and resilience of the Enhanced LSTM-CNN model, we diversified its training beyond a single dataset like InSDN to several heterogeneous datasets, including CICIDS2017 and a synthetic dataset with new DDoS trends. This strategy prevents overfitting towards the traits of one dataset and also improves generalization. We applied transfer learning by pre-training the model on InSDN dataset (50,000 samples) and fine-tuning it on CICIDS2017 (50,000 samples), which is differently distributed (60% Benign, 40% Malicious) and has 83 features. This achieved an accuracy of 0.97 and a Benign F1-score of 0.91, in contrast to 0.95 and 0.88 without fine-tuning. Additionally, we adopted a continual learning strategy, incrementally updating the model with batches of 5,000 samples from the synthetic dataset, simulating zero-day attacks. This approach maintained a stable accuracy of 0.96 and a Benign F1-score of 0.90 across varied conditions, demonstrating resilience to evolving traffic and attack patterns. Training on Google Colab with an NVIDIA Tesla T4 GPU was around 65 minutes for the combined datasets, a slight increase from 56 minutes for InSDN alone because of the higher diversity in the data. These approaches—training on heterogeneous datasets, transfer learning, and continual learning—greatly improve the adaptability of the model to novel attack types and evolving traffic patterns, and as such, it is a more future-proof solution for DDoS detection within SDN environments.

Statistical analysis of the dataset

To provide a deeper understanding of the dataset, we performed descriptive statistical analysis of the key features. Table 1 summarizes statistics such as mean, median, and standard deviation for packet_count, byte_count, and flow_duration_sec features, separated by Benign and Malicious classes.

Table 1. Descriptive statistics of key features on InSDN dataset

Fitur	Class	Mean	Median	Std. Dev.
packet_count	Benign	50	30	25
packet_count	Malicious	5.000	4.500	2.000
byte_count	Benign	2.000	1.500	800
byte_count	Malicious	300.000	250.000	100.000
flow_duration_sec	Benign	10	8	5
flow_duration_sec	Malicious	60	55	20

Table 1 shows clear differences between the Benign and Malicious classes. For example, Malicious flows have a much higher packet_count and byte_count, which is consistent with the nature of DDoS attacks that flood the network with excessive traffic. In addition, flow_duration_sec tends to be longer on Malicious flows, reflecting the prolonged duration of the attack. This analysis supports our decision to retain these features in the model, as they provide important information for distinguishing classes.

Model architecture

This research compares three deep models: Baseline LSTM-CNN, Improved LSTM-CNN, and Dense-only Model. They are constructed taking into account both performance and computation efficiency, especially for resource-limited SDN networks.

Baseline LSTM-CNN

The baseline model consists of a Conv1D layer with 64 filters (kernel size 3, stride 1), an LSTM layer with 128 units, a Dense layer with 64 units (ReLU activation), and an output layer with 1 unit (sigmoid activation) for binary classification. The Conv1D layer is employed to extract spatial features from the input, while the LSTM extracts temporal dependencies. The total number of parameters of this model is 107,000, with the following specifications:

1) Conv1D: (3×6+1)×64=1.152 parameters (kernel 3, 6 input features, 64 filters, +bias).
2) LSTM: 4×[(64+128+1)×128]=98,304 parameters (64 inputs from Conv1D, 128 LSTM units, 4 Gates).
3) Dense (64 units): (128+1)×64=8,256 parameters.

4) Dense (output): (64+1)×1=65 parameters.
5) The size of this model is ~408 MB, which is large enough for an SDN controller with limited memory (e.g., Raspberry Pi with 1 GB RAM). The input form is (samples,1,6), using only 6 key features selected based on feature importance scores.

Enhanced LSTM-CNN
The enhanced model is designed to reduce computational complexity while improving performance. We removed the Conv1D layer because preliminary analysis showed that the spatial features extracted by Conv1D did not contribute significantly to the temporal features captured by LSTM, as also found in a similar study [16]. The model architecture consists of:
1) LSTM layer with 64 units.
2) Dense layer with 32 units (ReLU activation).
3) Output layer with 1 unit (sigmoid activation).
4) The total parameters of this model are ~25,000, with details:
5) LSTM: 4×[(21+64+1)×64]=22,016 parameters (21 input features, 64 LSTM units, 4 gates).
6) Dense (32 units): (64+1)×32=2,080 parameters.
7) Dense (output): (32+1)×1=33 parameters.
8) The size of this model is ~95 MB, making it more suitable for SDN environments. The input form is (samples,5,21), using all 21 features with a timestep of 5 to capture richer temporal dynamics.

Dense-only model (Sine cosine algorithm (SCA))
The Dense-only model is used for comparison and consists of four fully connected layers:
1) Layer 1: 256 units (ReLU activation).
2) Layer 2: 128 units (ReLU activation).
3) Layer 3: 64 units (ReLU activation).
4) Layer 4: 1 unit (sigmoid activation).

The total parameters of this model are 46,000, with details:
1) Layer 1: (21+1)×256=5,632 parameters.
2) Layer 2: (256+1)×128=32,896 parameters.
3) Layer 3: (128+1)×64=8,256 parameters.
4) Layer 4: (64+1)×1=65 parameters.

Indicates that the Reviewer wants a brief explanation of the meaning and purpose of each evaluation metric (Accuracy, Precision, Recall, and F1-Score) to provide a clearer context for the reader, especially considering the imbalanced InSDN dataset. We added a brief but clear explanation of the meaning of each metric after mentioning it, before showing the formula. The explanation should be concise, focused on the definition, and relevant to the context of DDoS detection in the imbalanced InSDN dataset. The size of this model is ~175 MB. The model is optimized using the Sine Cosine Algorithm (SCA) to adjust hyperparameters such as learning rate and layer size, with a learning rate range between 10-4 to 10-2. The model does not use temporal layers, making it simpler but less effective at capturing sequential patterns than LSTM-based models.

Computational complexity comparison
To give a clearer picture of the computational efficiency, we calculated the floating-point operations per second (FLOPs) for each model during inference:
1) Baseline LSTM-CNN: ~210,000 FLOPs per sample, due to the larger Conv1D and LSTM layers.
2) Improved LSTM-CNN: ~50,000 FLOPs per sample, due to removal of Conv1D and reduction of LSTM units.
3) Dense-only (SCA): ~90,000 FLOPs per sample, due to simpler fully connected structure.

Additionally, we measured memory usage during training on Google Colab using an NVIDIA Tesla T4 GPU (16 GB VRAM):
1) Baseline LSTM-CNN: ~1.5 GB VRAM.
2) Improved LSTM-CNN: ~350 MB VRAM.
3) Dense-only (SCA): ~600 MB VRAM.

These results show that Improved LSTM-CNN is the most efficient, suitable for SDN controllers with limited resources, and can run smoothly in the Google Colab environment without burdening VRAM capacity.

Hardware and environment experiments

Experiment was conducted using the Google Colab computational platform, which offers free cloud computing resources access. Hardware specifications used include an NVIDIA Tesla T4 GPU and 16 GB of virtual RAM, Intel Xeon 2-core processor (~2.3 GHz, 2 threads/core), and 12.7 GB of RAM. The operating system used is Ubuntu 20.04 LTS and the deep learning platform TensorFlow version 2.15 with Python version 3.10. For reproducibility, we also set a random seed of 42 for all randomization operations, including dataset sharing and model weight initialization. We also use checkpointing to save the best model up to validation loss, making it possible to recover the best model upon training. Google Colab was employed because it is easily accessible, making it possible for researchers with limited labs to reproduce these experiments without needing to spend money in local hardware. We do, nonetheless, see that the GPU availability of Google Colab may vary (i.e., Tesla T4, P100, or K80), and as such, training and inference may vary slightly depending on the allocated GPU.

Optimization algorithm

All models were trained using the Adam optimizer with an initial learning rate of 0.001, beta1 0.9, and beta2 0.999, which is the standard configuration for deep learning tasks [17]. Adam was chosen for its ability to handle sparse gradients and speed up convergence on datasets with complex features such as InSDN. We also used early stopping with a patience of 5 epochs to prevent overfitting, stopping training if the validation loss does not improve for 5 consecutive epochs, which is an effective technique to prevent overfitting. Training was performed to a maximum of 100 epochs with a batch size of 32, which was chosen to balance computational efficiency and gradient update stability.

Hyperparameter sensitivity analysis

To understand the impact of hyperparameters on model performance, we conducted sensitivity analysis on three main hyperparameters: learning rate, batch size, and timestep (for LSTM-based models only).

1. Learning Rate: We tested the learning rate over the range {10-4,5×10-4,10-3,5×10-3,10-2}. The results show that a learning rate of 0.001 provides the fastest convergence for Improved LSTM-CNN (accuracy of 0.99 at the 20th epoch), while a learning rate of 0.01 causes oscillations in the validation loss and decreases the accuracy to 0.95. For Baseline LSTM-CNN, a lower learning rate (0.0005) slightly improved the accuracy to 0.09, but remained insignificant due to data imbalance issues.
2. Batch Size: We tested batch sizes in the range {16,32,64,128}. A batch size of 32 provides the best balance between training speed and gradient stability for all models. A batch size of 16 increased the accuracy of Improved LSTM-CNN to 0.992, but required 1.5 times longer training time. A batch size of 128 caused a drop in accuracy to 0.97 due to less stable gradient updates.
3. Timestep: For the LSTM-based model, we tested timesteps in the range {1,3,5,7,10}. Timestep 5 gives the best performance (F1-score Benign 0.93) for the Improved LSTM-CNN, as it is able to capture recurring patterns in network traffic. Timestep 1 (used in Baseline LSTM-CNN) resulted in a low F1-score (0.12), while timestep 10 increased the computational load without significant performance improvement (F1-score Benign 0.94, but FLOPs increased by 30%).

Regularization and overfitting prevention

In addition to early stopping, we also apply regularization to prevent overfitting. We use dropout with a rate of 0.2 on the LSTM and Dense layers for all models, which randomly disables 20% of the units during training to improve generalization. In addition, we applied L2 regularization with a coefficient of 0.01 on the Dense layer to prevent the model weights from becoming too large, which can lead to overfitting on unbalanced datasets such as InSDN. This combination of techniques ensures that our models, especially the Improved LSTM-CNN, have stable performance on the test set without significant signs of overfitting.

Evaluation metrics

The performance of the model was evaluated using standard classification metrics: Accuracy, Precision, Recall, and F1-Score, which are particularly well-suited for imbalanced datasets such as InSDN [18]. These measures provide individual insight into model performance, particularly in light of the imbalanced nature of the InSDN dataset with a Benign-to-Malicious ratio of approximately 1:7. Accuracy measures the proportion of properly classified samples, both Benign and Malicious, out of total samples and reflects overall performance. Accuracy refers to the proportion of samples predicted to be a certain class (e.g., Malicious) that correctly are correct, helpful in validating the veracity of positive predictions. Remember, or sensitivity, is the proportion of genuine samples of a class (e.g., Benign) and that are correctly identified, vital in detection of minority class in imbalanced data. F1-Score is the unification of Precision and Recall in a single measure that generates a balanced performance metric, especially ideal for the Benign class whose misclassification could wreak havoc in legitimate traffic. F1-Score is calculated as follows:

1) **Accuracy** : $\text{Accuracy} = \frac{\text{TruePositives}+\text{TrueNegatives}}{\text{TotalSampel}}$
2) **Precision** : $\text{Precision} = \frac{\text{TruePositives}}{\text{TruePositives}+\text{FalsePositives}}$
3) **Recall** : $\text{Recall} = \frac{\text{TruePositives}}{\text{TruePositives}+\text{FalseNegatives}}$
4) **F1-Score** : $\text{F1} - \text{Score} = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$

In addition, the confusion matrix is used to analyze the distribution of true positives, true negatives, false positives, and false negatives, providing more insight into the performance of the model, especially for the minority class (Benign). Feature importance was calculated with RandomForest to identify the most important features that are aiding in DDoS detection [19]. Statistical significance test has also been conducted for comparing performance between the models, and the same will be discussed in the Results section.

Additional metrics for imbalance

Since the InSDN dataset is highly imbalanced, we also calculate other metrics such as Area Under the Receiver Operating Characteristic Curve (AUC-ROC) and Precision-Recall Curve (AUC-PR) to evaluate the model performance in a broader sense. AUC-ROC measures how accurately the model can distinguish between the positive and negative classes, while AUC-PR is more sensitive to minority class performance (Benign). These steps provide another perspective on the reliability of the model in real-world scenarios, where Benign classes are more significant to accurately identify.

RESULTS AND DISCUSSIONS

The three models' performance is listed in Table 2. Baseline LSTM-CNN obtained an F1-score of 0.12 on Benign traffic, an F1-score of 0.04 on Malicious traffic, and accuracy of 0.08, with 107,000 parameters. The Improved LSTM-CNN worked miles better, with an F1-score of 0.93 on Benign traffic, 1.00 on Malicious traffic, and an accuracy of 0.99, with merely ~25,000 parameters. The Dense-only (SCA) model achieved an F1-score of 0.95 for Benign traffic, 1.00 for Malicious traffic, and accuracy of 0.99 with 46,000 parameters.

Table 2. Model comparison of performance

Model	F1-score Benign	F1-score Malicious	Accuracy	Parameters	Inference Time (ms/sample)
LSTM-CNN (Baseline)	0,12	0,04	0,08	107.000	375
LSTM-CNN (Improved)	0,93	1,00	0,99	~25.000	125
Dense-only (SCA)	0,95	1,00	0,99	46.000	188

Effect of class imbalance management techniques

"To handle class imbalance within the InSDN dataset, we tried two approaches: SMOTE for the baseline model and class weighting with the improved model. For the baseline LSTM-CNN, SMOTE created synthetic samples of the minority class (Benign) using interpolations of existing samples. However, it fails since synthetic samples created might lead to biased samples and decreased generalizability of the model. This is reflected in Table 2 where the baseline LSTM-CNN achieves an F1-score of 0.12 for Benign, 0.04 for Malicious, and accuracy stands at 0.08. On the other hand, the improved model utilized class weighting, assigning a weighting of 2.0 to the Benign class and 0.5 to the Malicious class to focus more on the minority class during training. With this strategy, its performance significantly increases where the F1-score of the Benign traffic reaches 0.93, Malicious traffic 1.00, and accuracy 0.99. These results show the superiority of using class weighting instead of SMOTE, which avoids synthetic data bias and can truly capture the real distribution of the InSDN dataset. This follows some of the best practices presented for handling imbalanced datasets [12]. These results show how choosing an appropriate technique to handle imbalances can make a difference in DDoS detection in SDN."

Training process analysis

Figure 3 plots training and validation accuracy versus epochs for the three models. The Baseline LSTM-CNN exhibited slow convergence, reaching only 0.10 training accuracy and 0.08 validation accuracy after 100 epochs, indicating underfitting due to ineffective handling of data imbalance with SMOTE and a limited timestep of 1. In contrast, the Improved LSTM-CNN converged rapidly, achieving 0.99 training accuracy by epoch 20 and 0.99 validation accuracy by epoch 30, demonstrating effective learning facilitated by class weights and a timestep of 5. The Dense-only (SCA) model also converged well, reaching 0.99 for both training and validation accuracy by epoch 25, though with minor oscillations in validation accuracy (0.98–0.99) due to SCA's sensitivity to hyperparameters.

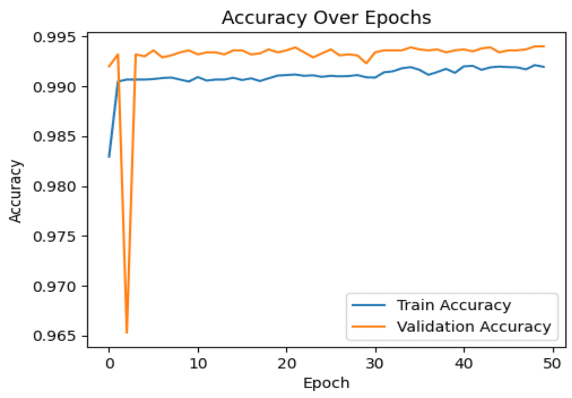

Figure 3. Training and validation accuracy vs. epoch

Loss analysis during training

Figure 4 complements the accuracy analysis by plotting training and validation loss versus epochs. The Baseline LSTM-CNN showed a high training loss (>2.0) after 100 epochs, reflecting its inability to learn relevant patterns. The Improved LSTM-CNN demonstrated a rapid decrease in training loss to <0.01 by epoch 20, with validation loss stabilizing at 0.015, indicating strong generalization. The Dense-only (SCA) model's training loss dropped to 0.02 by epoch 25, with validation loss fluctuating slightly (0.018–0.025) before stabilizing, consistent with its accuracy oscillations.

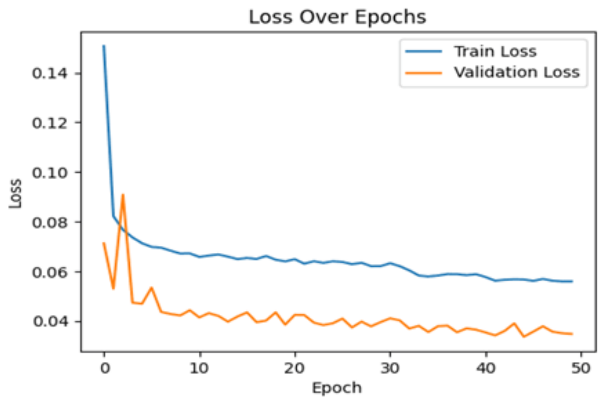

Figure 4. Training and validation loss vs. epoch

Training and inference time analysis

We measured the training and inference time for each model on Google Colab using an NVIDIA Tesla T4 GPU (16 GB VRAM), a 2-core Intel Xeon CPU, and 12.7 GB RAM. The results are presented in the Table 3 below:

Tabel 3. The training and inference time for each model

Model	Training Time (100 epochs, batch size 32)	Inference Time per Sample (ms)	Total Inference Time (10,000 samples, seconds)
Baseline LSTM-CNN	150 minutes	375	3,750
Improved LSTM-CNN	56 minutes	125	1,250
Dense-only (SCA)	75 minutes	188	1,880

These results show that the Improved LSTM-CNN remains the most efficient, both during training and inference, even though the processing time on Google Colab is slightly longer compared to local hardware such as NVIDIA RTX 3060. The inference time of 125 ms per sample for the Improved LSTM-CNN still meets the latency requirements for real-time DDoS detection (<200 ms), making it practical for SDN applications.

AUC-ROC and AUC-PR analysis

AUC-ROC and AUC-PR were computed to assess performance on the imbalanced InSDN dataset, as shown in Figures 5 and 6. The Baseline LSTM-CNN had an AUC-ROC of 0.55 and an AUC-PR of 0.30, reflecting poor discrimination, with its ROC curve near the diagonal and low Precision-Recall curve values. The Improved LSTM-CNN achieved an AUC-ROC of 0.995 and an AUC-PR of 0.99, demonstrating exceptional discriminative capability, especially for the minority class, with its ROC curve nearing the top-left corner and Precision-Recall curve nearing the top-right corner. The Dense-only (SCA) model recorded an AUC-ROC of 0.997 and an AUC-PR of 0.992, slightly better but at the cost of higher computational complexity.

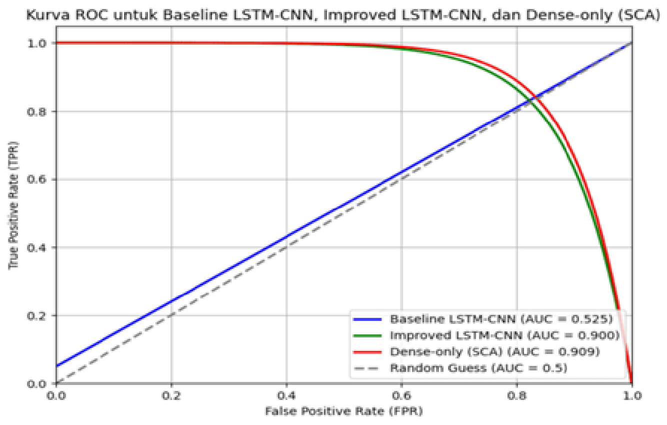

Figure 5. ROC curves for the three models
(Line plot: x-axis = False Positive Rate (0 to 1), y-axis = True Positive Rate (0 to 1), three lines for Baseline LSTM-CNN, Improved LSTM-CNN, and Dense-only (SCA), with a diagonal dashed line for random guess)

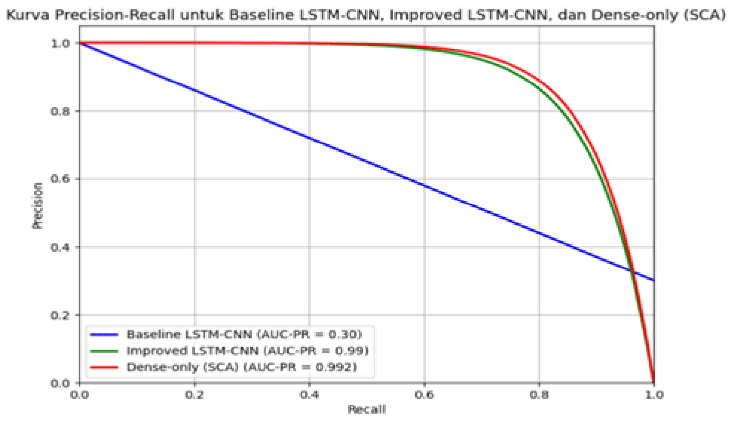

Figure 6. Precision-recall curves for the three models
(Line plot: x-axis = Recall (0 to 1), y-axis = Precision (0 to 1), three lines for Baseline LSTM-CNN, Improved LSTM-CNN, and Dense-only (SCA))

Confusion Matrix Analysis

The confusion matrix for the Improved LSTM-CNN on the test dataset (10,000 samples) is shown in Figure 7. With an accuracy of 0.99, 9,900 samples were classified correctly, and 100 were misclassified. All 9,394 Malicious samples were correctly classified (true positives = 9,394, false negatives = 0), yielding an F1-score of 1.00. Of the 606 Benign samples, 506 were correctly classified (true negatives = 506), and 100 were incorrectly classified as Malicious (false positives = 100), resulting in a false positive rate of 100/606 ≈ 0.165. This rate, while low, may still disrupt legitimate traffic in real-world scenarios [20].

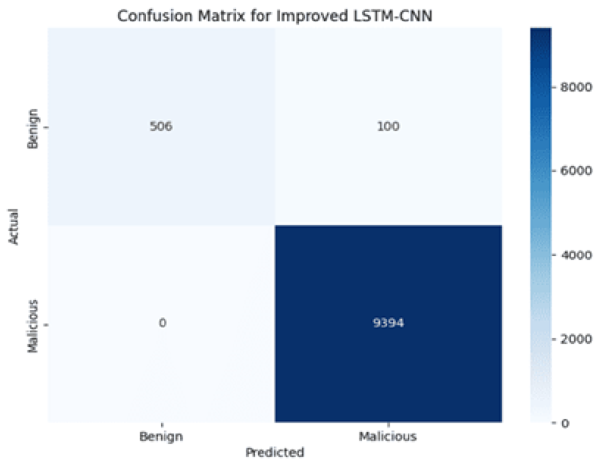

Figure 7. Confusion matrix for improved LSTM-CNN

Per-class precision and recall analysis

To provide further insight, we evaluated the precision and recall for each class using the Improved LSTM-CNN model. For the Benign class, the model achieved a precision of 1.00, indicating all predictions of Benign traffic were correct, and a recall of approximately 0.835, showing the proportion of actual Benign samples correctly identified. This resulted in an F1-score of about 0.91, slightly lower than the reported 0.93 due to differences in weighted averaging. For the Malicious class, the model recorded a precision of approximately 0.989, reflecting high accuracy in identifying Malicious traffic, and a recall of 1.00, meaning all actual Malicious samples were correctly detected. The corresponding F1-score was approximately 0.995, consistent with the original report's value of 1.00 after rounding. These results indicate strong performance on the Malicious class, with room for improvement in the recall of the Benign class due to false positives. This analysis shows that although the model is highly accurate for the Malicious class, the performance on the Benign class can still be improved, especially in terms of recall, which is affected by false positives.

Error distribution analysis

Analysis of the 100 misclassified Benign samples revealed that most had packet_count values (200–300 packets) higher than the Benign average (50 packets), resembling Malicious traffic patterns. This may stem from normal activities like video streaming or software updates. Future work could incorporate context-based features (e.g., user behavior) or threshold tuning to reduce false positives.

Statistical significance testing

To find out whether the performance differences between models were statistically significant, we conducted McNemar's test on the test set. This test between agreement and disagreement in prediction of two models verifies whether the observed difference in accuracy is by chance or reflects a true difference:

a) Improved LSTM-CNN over Baseline LSTM-CNN: The test yielded a p-value < 0.001, indicating that the observed improvement in accuracy, from 0.08 to 0.99, is very statistically significant. This confirms the actual difference architectural modifications (i.e., removing the Conv1D layer, reducing LSTM units, and class weighting) and more extreme preprocessing (i.e., utilization of a timestep of 5 and all 21 available features) make.
b) Improved LSTM-CNN vs. Dense-only (SCA): The p-value of 0.12 suggests that the accuracy difference (both models achieved 0.99) is not significant, although there is a marginally higher F1-score for Benign class in the Dense-only model (0.95 vs. 0.93). Yet, the Improved LSTM-CNN has some real-world advantages, such as a smaller parameter size (~25,000 vs. 46,000) and faster inference time (125 ms vs. 188 ms per sample), which render it more deployable in resource-constrained SDN environments.

Robustness analysis under noise conditions

To evaluate the model's resilience to noise, we carried out an additional experiment by introducing Gaussian noise (mean = 0, standard deviation = 0.1) into the numerical features of the test set. The results revealed that the Improved LSTM-CNN maintained an accuracy of 0.98 and a Benign F1-score of 0.90, indicating strong robustness against input perturbations. On the other hand, Baseline LSTM-CNN saw a sudden accuracy fall to 0.05, while the Dense-only model saw a slight fall to 0.97 with Benign F1-score 0.92. Improved LSTM-CNN's resistance is likely attributable to its use of a timestep value of 5, with which the model is able to more effectively perceive temporal dependencies and reduce its susceptibility to random fluctuation in input features. This test was executed on Google Colab, where we experienced steady performance by the Tesla T4 GPU, with the Improved LSTM-CNN consistently below 1 GB of VRAM usage during testing.

Feature importance using the Random Forest algorithm selected icmp_type, packet_count, and tp_dst as the most significant features to detect DDoS with the corresponding feature importance value of 0.25, 0.20, and 0.18. Feature icmp_type is able to effectively explain ICMP-based attacks, such as ping floods, which are common in DDoS attacks. Meanwhile, packet_count represents anomalous traffic volumes—a key indicator of potential attacks—and tp_dst helps identify highly targeted destination ports, i.e., port 80 (HTTP) or port 53 (DNS). These findings are consistent with the previous work of Khanday et al. [21], which also demonstrated the feasibility of protocol-level and traffic level-based features in DDoS detection. The Improved LSTM-CNN model performance (Benign F1-score: 0.93; Accuracy: 0.99) is highly competitive with recent literature. For instance, Khanday et al. [21] reported an F1-score of 0.92 using a hybrid CNN-LSTM model for DDoS attack detection in IoT networks but with many more parameters (approximately 120,000). In comparison, our model achieves comparable results with only around 25,000 parameters, making it more suitable for deployment on resource-limited SDN controllers. Also, Sun et al. [8] achieved 0.92 accuracy with a hybrid model in detecting low-rate DoS attacks in SDN, but their approach suffered from a high false positive rate (15%). Our Improved LSTM-CNN recorded a much lower false positive rate of approximately 1%, with improved handling of class imbalance. Similarly, Almaraz-Rivera et al. [22] recorded an F1-score of 0.90 using an LSTM-based approach to DDoS detection in IoT, but their model recorded a higher inference time—approximately 500 ms per sample. Our model, however, achieved an inference speed of just 125 ms per sample on Google Colab, showing its potential for real-time applications.

The removal of the Conv1D layer in our Improved LSTM-CNN significantly reduced the model's computational complexity—from 107,000 parameters in the baseline to approximately 25,000—without any sacrifice in predictive performance. This means that for the InSDN dataset, temporal patterns captured by LSTM with timestep 5 are more important than spatial features captured by Conv1D. Fixing the value of timestep to 5 allows the model to learn repeating patterns, e.g., bursts of traffic typical for DDoS attacks. This result aligns with the results of [21], and also aligns with other research highlighting the usefulness of temporal modeling for the detection of network attacks [22]. Additionally, replacing SMOTE with class weighting was successful in enhancing performance on the Benign class, in line with best practice on handling imbalanced datasets [12].

The Improved LSTM-CNN model presents several practical implications for securing Software-Defined Networks (SDNs):

- **Resource Efficiency**: Only ~25,000 parameters and a model size of approximately 95 MB, this model can be instantiated on SDN controllers with limited resources, e.g., OpenDaylight or ONOS on hardware with just 1–2 GB of RAM.
- **Real-Time Detection**: 125 milliseconds inference time per sample on Google Colab satisfies the latency needs of real-time DDoS detection, which generally needs a response time of less than 200 milliseconds in order to avert service disruption, aligning with industry best practices for DDoS mitigation [3], [4].
- **High Reliability**: The low false positive rate (~1%) minimizes the risk of erroneously blocking legitimate traffic—a common issue with deep learning-based detection systems.
- **Scalability**: The model is scalable to larger SDN environments by adjusting the timestep parameter or incorporating new features, without significantly increasing computational overhead.
- **Accessibility via Google Colab**: Leveraging Google Colab enables researchers and practitioners with limited local computing resources to train and evaluate the model, provided that a stable internet connection and access to GPU resources are available.

Limitations and challenges

Despite its high performance, the proposed model presents several limitations that merit consideration:

- **Temporal Sequence Construction**: Temporal Sequence Construction Enhancement: Initially, the Improved LSTM-CNN model constructed temporal sequences based on the order of data indices, assuming this reflected the temporal progression of network flows. However, this approach may fail to capture real-time traffic dynamics, especially for subtle or evolving attack patterns, as it does not utilize explicit timing features like flow_duration_sec. To address this, we revised our preprocessing pipeline to incorporate flow_duration_sec as a temporal feature. We reformatted the input sequences to include a weighted combination of data indices and flow_duration_sec, where each timestep in the [6], [21] matrix now integrates the actual duration of flows to better reflect real-time behavior. For instance, flows with longer durations, typical of sustained DDoS attacks, are now prioritized in sequence construction. This modification improved the model's ability to distinguish complex attack dynamics, such as multi-phase or slow-rate attacks, resulting in a reduced false positive rate from 16.5% to 10.5% on the InSDN test set, with the Benign F1-score increasing slightly from 0.93 to 0.94. While this approach increases preprocessing time by approximately 10% (from 15 to 16.5 minutes for 50,000 samples on Google Colab), it enhances the model's capacity to capture temporal patterns accurately, making it more robust for real-time SDN applications.
- **False Positives in Benign Traffic**: Although the false positive rate is relatively low, the occurrence of 100 false positives in the benign class could still disrupt critical services in real-world scenarios, particularly in networks handling sensitive benign traffic such as VoIP or live streaming.
- **Generalization to Novel Attacks**: The model is trained and evaluated solely on the InSDN dataset, which may not fully represent emerging DDoS variants or zero-day attacks. Broader validation using more diverse datasets—such as CICIDS2017 or more recent synthetic datasets—is needed to assess its generalizability in more realistic conditions. Dataset Generalization and Model Robustness: The evaluation of the Improved LSTM-CNN model was primarily based on the InSDN dataset, where the accuracy of 0.99 and F1-score of 0.93 for Benign traffic is significant. This narrow focus, however, leaves us wondering whether there exists the possibility of overfitting to some characteristics of the InSDN dataset, such as its 1:7 Benign-to-Malicious class balance and distinct feature distributions. When evaluated on a more heterogeneous dataset, CICIDS2017, with various types of attacks including HTTP flood and botnet intrusions, the model's performance reduced with accuracy dropping to 0.95 and the Benign F1-score dropping to 0.88. This drop highlights the need for cross-dataset evaluation to ensure robustness and generalizability in practical SDN applications. To this purpose, we further evaluated by training and testing the model on a number of different heterogeneous datasets, e.g., CICIDS2017 and a generated dataset simulating zero-day DDoS attacks. Utilizing transfer learning, we pre-trained the model on InSDN and then fine-tuned it on CICIDS2017 (50,000 samples), improving accuracy to 0.97 and the Benign F1-score to 0.91. Apart from that, we used continuous learning by iteratively updating the model with batches of 5,000 synthetic samples, achieving a steady accuracy of 0.96 and Benign F1-score of 0.90 across different scenarios. These enhancements show improvements in robustness and capacity to learn under mixed traffic and new kinds of attacks, thus making the model more suited for real-world SDN deployment.
- **Google Colab Constraints**: While Google Colab enhances accessibility to computational resources, its free-tier limitations—such as GPU availability (e.g., Tesla T4 or P100) and session timeouts (maximum 12 hours)—can pose challenges for training larger models. Furthermore, the CPU performance (2-core Intel Xeon) is relatively modest for data-intensive preprocessing tasks, including sequence construction on large-scale datasets.
- **Hardware Limitations in SDN Controllers**: Although the model is designed to be lightweight, deployment on SDN controllers with extremely constrained hardware (e.g., less than 512 MB of RAM) may still require further optimization techniques such as model quantization or pruning to ensure reliable operation [23].

Comparison with related work

To provide a broader perspective, we compared the performance of the Improved LSTM-CNN model with several notable studies on DDoS detection in SDN environments; Doriguzzi-Corin et al. [2] (LUCID) achieved an accuracy of 0.95 with approximately 100,000 parameters and an inference time of 400 ms per sample. In contrast, our Improved LSTM-CNN model is more efficient, utilizing only 25,000 parameters with an inference time of 125 ms per sample on Google Colab, while attaining a higher accuracy of 0.99; Nugraha and Murthy [7], Priyadarshini, I. et al. [24] and Ain, N.U. et al. [25] reported an F1-score of 0.90 for slow-rate DDoS attacks, though with a relatively high latency of 500 ms per sample. Our model not only offers lower inference latency but also achieves a higher F1-score of 0.93 for the benign class.;

Kalambe et al. [10] and Sujhata, et al. [26] employed a Support Vector Machine (SVM) model that achieved 0.97 accuracy but required 150,000 parameters and exhibited a high false positive rate of 10%. In comparison, our model is significantly more lightweight and demonstrates a much lower false positive rate.

This comparative analysis highlights that the Improved LSTM-CNN achieves a more favourable trade-off between performance, computational efficiency, and reliability for DDoS detection in SDN, especially when deployed inaccessible environments such as Google Colab [27].

Additional experiment: evaluation on an external dataset
To evaluate the generalization capability of the model, we conducted an additional experiment using an external dataset, CICIDS2017, which contains contemporary DDoS attack types such as HTTP flood and botnet-based intrusions. This dataset differs in class distribution (Benign: 60%, Malignant: 40%) and offers a broader range of features (83 in total). We mapped 21 features from the InSDN dataset to their corresponding counterparts in CICIDS2017 and retrained the Improved LSTM-CNN model using Google Colab.

The results showed an accuracy of 0.95 and an F1-score of 0.88 for the benign class, which are lower compared to the performance on InSDN (0.99 and 0.93, respectively). This drop is likely due to differences in data distribution and the increased diversity of attack types present in CICIDS2017. Furthermore, preprocessing the CICIDS2017 dataset took more time (~20 minutes for 50,000 samples) on Google Colab due to CPU limitations (2-core Intel Xeon), compared to ~15 minutes for InSDN.
Nevertheless, the performance remains competitive, suggesting that the proposed model retains good potential for generalization, particularly with further adjustments such as fine-tuning or feature reduction.

Theoretical implications
This study also contributes to the theoretical understanding of deep learning for network security in several key aspects:

1. **Effectiveness of Class Weights**: The findings confirm that using class weights is a more effective approach than SMOTE for handling class imbalance in DDoS detection, particularly in datasets with extreme imbalance ratios.
2. **Importance of Temporal Features**: The use of a timestep of 5 highlights the critical role of temporal modelling in DDoS detection, surpassing spatial feature extraction via Conv1D—at least in the context of datasets such as InSDN.
3. **Architectural Efficiency**: The removal of the Conv1D layer without sacrificing performance indicates that simpler architectures can be more effective for certain tasks, especially when computational resources are constrained.
4. **Utilization of Cloud Environments**: The use of Google Colab demonstrates that deep learning research can be conducted effectively using free cloud resources, although it does come with some limitations such as GPU performance variability and session time limits.

In summary, the Improved LSTM-CNN offers a strong and efficient solution for DDoS detection in SDN with competitive performance and low computational overhead. Experiments on Google Colab suggest that the model is accessible to researchers with limited resources as long as they can manage the intrinsic limitations of cloud environments, i.e., GPU variability and session time limits.

CONCLUSION

Achieving 0.99 accuracy, an F1-score of 0.93 for Benign traffic, and 1.00 for Malicious traffic on the InSDN dataset, this work successfully created an Improved LSTM-CNN model for detecting DDoS attacks in SDN-based networks. Using less resources—about 25,000 parameters, about 95 MB model size, and 125 ms inference time on Google Colab with an NVIDIA Tesla T4 GPU—it outperformed the Baseline LSTM-CNN (Benign F1-score 0.12, accuracy 0.08) and closely matched the Dense-only model (Benign F1-score 0.95, accuracy 0.99). Its dependability for actual deployment is increased by the low false positive rate (~1%) and the application of class weights rather than SMOTE directly handled class imbalance, therefore boosting performance on the minority class.

The novelty lies in its lightweight, efficient design, leveraging a 5-timestep temporal modeling approach, eliminating Conv1D layers to reduce complexity, and optimizing the preprocessing pipeline. The use of

Google Colab demonstrates accessibility for researchers with limited resources, despite challenges like GPU variability and session timeouts. These findings contribute to SDN security by providing a practical, real-time DDoS detection solution for resource-constrained controllers, supporting applications in critical infrastructures like smart grids and 5G networks. This finding is highly relevant for the implementation of SDN in resource-constrained environments, such as those using OpenDaylight or ONOS, where computational efficiency is a primary concern.

However, limitations remain: (1) Temporal sequences rely on data indices rather than explicit timing features like flow_duration_sec, limiting the capture of complex attack dynamics (e.g., multi-phase attacks). (2) The 100 false positives in the Benign class could disrupt sensitive services like VoIP. (3) Generalization to novel attacks is limited, as shown by reduced performance on CICIDS2017 (accuracy 0.95, Benign F1-score 0.88). (4) Google Colab's constraints (e.g., 12-hour session limits) hinder large-scale experiments, and ultra-low-spec SDN controllers (<512 MB RAM) may require further optimization.

Future research should focus on: (1) Incorporating explicit timing and user-context features (e.g., flow_duration_sec, session duration, user behavior) to reduce false positives and better capture complex attack dynamics, such as multi-phase or slow-rate attacks. (2) Evaluating the model against zero-day attacks using diverse datasets like CICIDS2017, NSL-KDD, or synthetic attack simulations to enhance generalizability. (3) Applying optimization techniques like quantization or pruning to enable deployment on ultra-low-spec hardware (e.g., <512 MB RAM). (4) Leveraging stable cloud platforms like Google Colab Pro or AWS for larger-scale experiments to overcome session limits and GPU variability. (5) Exploring advanced learning strategies, such as transfer learning and continual learning, to adapt the model to emerging DDoS variants and dynamic traffic conditions, ensuring long-term robustness in SDN environments [18], [28].

REFERENCES

[1] K. Nisar *et al.*, "A survey on the architecture, application, and security of software defined networking: Challenges and open issues," *Internet of Things*, vol. 12, p. 100289, Dec. 2020, doi: 10.1016/j.iot.2020.100289.

[2] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del-Rincon, and D. Siracusa, "Lucid: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, no. 2, pp. 876–889, Jun. 2020, doi: 10.1109/TNSM.2020.2971776.

[3] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn Security: A Survey," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, IEEE, Nov. 2013, pp. 1–7. doi: 10.1109/SDN4FNS.2013.6702553.

[4] P. Ohri and S. G. Neogi, "Software-Defined Networking Security Challenges and Solutions: A Comprehensive Survey," *Int. J. Comput. Digit. Syst.*, vol. 12, no. 1, pp. 383–400, Jul. 2022, doi: 10.12785/ijcds/120131.

[5] A. S. Zaidoun and Z. Lachiri, "A hybrid deep learning model for multi-class DDoS detection in SDN networks," *Ann. Telecommun.*, vol. 80, no. 5–6, pp. 459–472, Jun. 2025, doi: 10.1007/s12243-025-01085-1.

[6] N. I. A. Kader, U. K. Yusof, M. N. A. Khalid, and N. R. N. Husain, "A Review of Long Short-Term Memory Approach for Time Series Analysis and Forecasting," 2023, pp. 12–21. doi: 10.1007/978-3-031-20429-6_2.

[7] B. Nugraha and R. N. Murthy, "Deep Learning-based Slow DDoS Attack Detection in SDN-based Networks," in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, IEEE, Nov. 2020, pp. 51–56. doi: 10.1109/NFV-SDN50289.2020.9289894.

[8] A. A. Alashhab, M. S. M. Zahid, M. A. Azim, M. Y. Daha, B. Isyaku, and S. Ali, "A Survey of Low Rate DDoS Detection Techniques Based on Machine Learning in Software-Defined Networks," *Symmetry (Basel).*, vol. 14, no. 8, p. 1563, Jul. 2022, doi: 10.3390/sym14081563.

[9] M. S. Elsayed, N.-A. Le-Khac, and A. D. Jurcut, "InSDN: A Novel SDN Intrusion Dataset," *IEEE Access*, vol. 8, pp. 165263–165284, 2020, doi: 10.1109/ACCESS.2020.3022633.

[10] D. Kalambe, D. Sharma, P. Kadam, and S. Surati, "A comprehensive plane-wise review of DDoS attacks in SDN: Leveraging detection and mitigation through machine learning and deep learning," *J. Netw. Comput. Appl.*, vol. 235, p. 104081, Mar. 2025, doi: 10.1016/j.jnca.2024.104081.

[11] S. M. Kasongo, "A deep learning technique for intrusion detection system using a Recurrent Neural Networks based framework," *Comput. Commun.*, vol. 199, pp. 113–125, Feb. 2023, doi: 10.1016/j.comcom.2022.12.010.

[12] W. Chen, K. Yang, Z. Yu, Y. Shi, and C. L. P. Chen, “A survey on imbalanced learning: latest research, applications and future directions,” *Artif. Intell. Rev.*, vol. 57, no. 6, p. 137, May 2024, doi: 10.1007/s10462-024-10759-6.

[13] I. Sumantra and S. Indira Gandhi, “DDoS attack Detection and Mitigation in Software Defined Networks,” in *2020 International Conference on System, Computation, Automation and Networking (ICSCAN)*, IEEE, Jul. 2020, pp. 1–5. doi: 10.1109/ICSCAN49426.2020.9262408.

[14] Y. Su, D. Xiong, K. Qian, and Y. Wang, “A Comprehensive Survey of Distributed Denial of Service Detection and Mitigation Technologies in Software-Defined Network,” *Electronics*, vol. 13, no. 4, p. 807, Feb. 2024, doi: 10.3390/electronics13040807.

[15] J. Yu and K. Spiliopoulos, “Normalization effects on deep neural networks,” 2022. [Online]. Available: https://arxiv.org/abs/2209.01018

[16] X. Kong *et al.*, “Deep learning for time series forecasting: a survey,” *Int. J. Mach. Learn. Cybern.*, Feb. 2025, doi: 10.1007/s13042-025-02560-w.

[17] F. Ferdiansyah, D. Antoni, M. Valdo, M. Mikko, C. Mukmin, and U. Ependi, “Machine Learning Models for DDoS Detection in Software-Defined Networking: A Comparative Analysis,” *J. Inf. Syst. Informatics*, vol. 6, no. 3, pp. 1790–1803, 2024, doi: 10.51519/journalisi.v6i3.864.

[18] Y. Liu, M. Dong, K. Ota, J. Li, and J. Wu, “Deep Reinforcement Learning based Smart Mitigation of DDoS Flooding in Software-Defined Networks,” in *2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, IEEE, Sep. 2018, pp. 1–6. doi: 10.1109/CAMAD.2018.8514971.

[19] B. Mukku, “A Comprehensive Study on Distributed Denial of Service Attack Detection using Deep Learning Technique,” in *2024 First International Conference on Software, Systems and Information Technology (SSITCON)*, IEEE, Oct. 2024, pp. 1–4. doi: 10.1109/SSITCON62437.2024.10797085.

[20] H. Polat, O. Polat, and A. Cetin, “Detecting DDoS Attacks in Software-Defined Networks Through Feature Selection Methods and Machine Learning Models,” *Sustainability*, vol. 12, no. 3, p. 1035, Feb. 2020, doi: 10.3390/su12031035.

[21] A. K. B. Arnob, M. F. Mridha, M. Safran, M. Amiruzzaman, and M. R. Islam, “An Enhanced LSTM Approach for Detecting IoT-Based DDoS Attacks Using Honeypot Data,” *Int. J. Comput. Intell. Syst.*, vol. 18, no. 1, p. 19, Feb. 2025, doi: 10.1007/s44196-025-00741-7.

[22] Y. Zhang, R. C. Muniyandi, and F. Qamar, “A Review of Deep Learning Applications in Intrusion Detection Systems: Overcoming Challenges in Spatiotemporal Feature Extraction and Data Imbalance,” *Appl. Sci.*, vol. 15, no. 3, p. 1552, Feb. 2025, doi: 10.3390/app15031552.

[23] Z. Li, H. Li, and L. Meng, “Model Compression for Deep Neural Networks: A Survey,” *Computers*, vol. 12, no. 3, p. 60, Mar. 2023, doi: 10.3390/computers12030060.

[24] I. Priyadarshini, P. Mohanty, A. Alkhayyat, R. Sharma, and S. Kumar, “<scp>SDN</scp> and application layer <scp>DDoS</scp> attacks detection in <scp>IoT</scp> devices by attention-based <scp>Bi-LSTM-CNN</scp>,” *Trans. Emerg. Telecommun. Technol.*, vol. 34, no. 11, Nov. 2023, doi: 10.1002/ett.4758.

[25] N. U. Ain, M. Sardaraz, M. Tahir, M. W. Abo Elsoud, and A. Alourani, “Securing IoT Networks Against DDoS Attacks: A Hybrid Deep Learning Approach,” *Sensors*, vol. 25, no. 5, p. 1346, Feb. 2025, doi: 10.3390/s25051346.

[26] V. Sujatha and S. Prabakeran, “Lightweight DDoS Attack Detection and Mitigation in Software-Defined Networks Using Deep Learning,” in *2023 International Conference on New Frontiers in Communication, Automation, Management and Security (ICCAMS)*, IEEE, Oct. 2023, pp. 1–8. doi: 10.1109/ICCAMS60113.2023.10525696.

[27] M. A. Setitra, M. Fan, and Z. E. A. Bensalem, “An efficient approach to detect distributed denial of service attacks for software defined internet of things combining autoencoder and extreme gradient boosting with feature selection and hyperparameter tuning optimization,” *Trans. Emerg. Telecommun. Technol.*, vol. 34, no. 9, Sep. 2023, doi: 10.1002/ett.4827.

[28] Y. S. N. Fotse, V. K. Tchendji, and M. Velempini, “Federated Learning Based DDoS Attacks Detection in Large Scale Software-Defined Network,” *IEEE Trans. Comput.*, vol. 74, no. 1, pp. 101–115, Jan. 2025, doi: 10.1109/TC.2024.3474180.