Available online at www.HighTechJournal.org

HighTech and Innovation Journal

ISSN: 2723-9535

Vol. 6, No. 4, December, 2025

Advancing Network Security: Integrating Salp Swarm Optimization with LSTM for Intrusion Detection

Ahmed Abdelaziz [1, 2*], Mohamed Elhoseny [3, 4], Vitor Santos [1]

[1] Nova Information Management School, Universidade Nova de Lisboa, 1070-312, Lisboa, Portugal.

[2] Information System Department, Higher Technological Institute, HTI, Cairo, 44629, Egypt.

[3] College of Computing and Informatics, University of Sharjah, United Arab Emirates.

[4] Faculty of Computers and Information, Mansoura University, Egypt.

Received 22 March 2025; Revised 15 October 2025; Accepted 18 November 2025; Published 01 December 2025

Abstract

Over time, intrusion detection systems have grown essential in ensuring network security by identifying malicious activities within network traffic and alerting security teams. Machine learning techniques have been employed to develop these systems. However, these approaches often face challenges related to low accuracy and high false alarm rates. Deep learning models like Long Short-Term Memory (LSTM) are utilized to address these limitations. Despite their potential, LSTM models require numerous iterations to achieve optimal performance. This study introduces an enhanced version of the LSTM algorithm, termed ILSTM, which integrates the Salp Swarm Optimizer (SSO) to boost accuracy. The ILSTM framework was applied to construct an advanced intrusion detection system capable of binary and multi-class classifications. The approach comprises two phases: The first involves training a standard LSTM model to initialize its weights. In contrast, the second employs the SSO hybrid optimization algorithm to fine-tune these weights, enhancing overall performance. The effectiveness of the ILSTM algorithm and the intrusion detection system was assessed using two publicly available datasets, NSL-KDD and LITNET-2020, across nine performance metrics. Results demonstrated that the ILSTM significantly outperformed the conventional LSTM and other comparable deep learning models in accuracy and precision. Specifically, the ILSTM achieved an accuracy of 93.09% and a precision of 96.86%, compared to 82.74% accuracy and 76.49% precision for the standard LSTM. Moreover, the ILSTM exhibited superior performance on both datasets and was statistically validated to be more robust than LSTM. Furthermore, the ILSTM excelled in multiclass intrusion classification tasks, effectively identifying intrusion types.

Keywords: Security; Intrusion Detection; Long Short-Term Memory; Salp Swarm Optimizer.

1. Introduction

In the era of pervasive digital connectivity, cybersecurity threats have grown in frequency and sophistication. Cyberattacks include denial-of-service (DoS), phishing, malware, and advanced persistent threats, which pose significant risks to individuals, enterprises, and critical infrastructure. Intrusion Detection Systems (IDS) have emerged as vital components in defending against these threats by monitoring network traffic and identifying anomalous behaviors or known attack patterns [1, 2]. In addition, the proliferation of Internet of Things (IoT) devices has amplified security

* Corresponding author: d20190535@novaims.unl.pt

http://dx.doi.org/10.28991/HIJ-2025-06-04-05

concerns, as these interconnected systems present new vulnerabilities. Enhancing the security of IoT ecosystems requires targeted measures to safeguard devices and their transmitted data [3, 4].

Over the past two decades, machine learning (ML) and deep learning (DL) techniques have been employed to improve IDS effectiveness. Traditional ML methods such as decision trees, support vector machines, and artificial neural networks offer reliable performance but often rely on manual feature engineering and struggle with complex or evolving attack vectors [3–5]. DL models, particularly Long Short-Term Memory (LSTM) networks, have demonstrated superior capabilities in learning temporal dependencies from sequential network data [6, 7]. LSTM's architecture allows for capturing long-range dependencies, which is essential in distinguishing between benign and malicious traffic patterns. However, despite their effectiveness, standard LSTM models face challenges related to random weight initialization, slow convergence, and overfitting—factors that hinder their performance and generalizability [8, 9].

For more than twenty years, Various ML techniques have been extensively utilized to improve the efficiency and effectiveness of IDS [7]. These algorithms are broadly categorized into shallow learning and deep learning approaches for anomaly detection. Shallow learning techniques, including Bayesian networks and artificial neural networks (ANNs), depend on manual feature extraction to create predictive models [2]. In contrast, deep learning (DL) algorithms have the advantage of automatically deriving intricate representations directly from raw data, leading to more robust models [8]. As a subset of ML, DL utilizes multi-layer artificial neural networks to learn hierarchical features, enabling the processing of large, complex, and high-dimensional datasets. Prominent DL algorithms include deep neural networks (DNNs) [9], convolutional neural networks (CNNs) [10], and recurrent neural networks (RNNs). One of DL's key strengths lies in its capacity to autonomously identify relevant features, eliminating the need for labor-intensive feature engineering. In recent years, deep learning has emerged as a focal point in intrusion detection research [8, 11–13]. Among the various DL models, RNNs are particularly popular for sequential data classification due to their cyclic connections, which enable the retention and use of prior input information during training [14]. A prominent variant of RNN, LSTM, has garnered significant attention for overcoming the vanishing gradient problem commonly faced by standard RNNs. By incorporating gating mechanisms, LSTM models are adept at capturing long-term dependencies in data, making them well-suited for detecting complex attack patterns. Furthermore, LSTM has been shown to effectively identify distinctive and previously unseen attack behaviors [15, 16].

SSO is highly effective in exploring the solution space, which helps mitigate the risk of local minimum often encountered in deep learning models. Its underlying mechanism allows it to traverse complex optimization landscapes, improving the likelihood of identifying globally optimal weight configurations. By balancing exploration and exploitation through the leader-follower dynamics of salps, SSO achieves efficient search capabilities. Additionally, SSO operates independently of gradient information, making it well-suited for optimizing non-differentiable or highly complex functions. Its versatility allows it to be applied across various DL architectures and challenges [17].

SSO adapts its search strategy dynamically based on the fitness of potential solutions, offering a more responsive and effective method for optimization, particularly in dynamic environments [18]. It also supports multi-objective optimization, enabling simultaneous improvement of multiple criteria, such as maximizing accuracy while minimizing computational costs. This capability allows for more holistic optimization strategies in DL applications. By efficiently optimizing weights, SSO enhances a model's ability to generalize to unseen data, thereby reducing overfitting and improving test performance. One of the key advantages of SSO is its simplicity, which makes it easier to understand and implement compared to some more complex optimization algorithms. This accessibility appeals to researchers and practitioners who aim to integrate it into their DL workflows. Moreover, SSO can be combined with other optimization techniques, such as gradient-based methods, to create hybrid approaches that leverage the strengths of both strategies. These hybrid methods often yield improved optimization results and more robust models [19].

Overall, SSO provides a robust and effective framework for optimizing weights in DL models. Its global search capabilities, adaptability, and ease of implementation make it a valuable tool for improving model performance across various applications. As the field of DL continues to advance, SSO is poised to play a crucial role in addressing the optimization challenges inherent in this evolving domain [20].

The LSTM algorithm has emerged as a popular choice for designing IDS [21, 22]. However, certain drawbacks often compromise its effectiveness, including the challenges associated with random weight initialization [23] and the risk of overfitting [24]. Despite its potential, LSTM-based models encounter two significant issues: (1) the computational burden caused by the high number of iterations required to identify optimal network weights, and (2) relatively low classification accuracy in certain scenarios. The primary focus of this study is to overcome these challenges by optimizing the weight initialization process and minimizing the loops, lowering computational efforts and improving classification performance in IDSs.

Recent research has explored hybrid models combining DL with nature-inspired optimization algorithms to overcome these challenges. For instance, the integration of Particle Swarm Optimization (PSO), Genetic Algorithm (GA), and Whale Optimization Algorithm (WOA) has been shown to improve classification performance and convergence in IDS tasks [10–12]. Among these, the SSO, inspired by the behavior of marine salps, offers promising results in global optimization due to its simplicity, low computational overhead, and capacity for effective exploration and exploitation [13].

Despite these advances, there remains a gap in fully leveraging SSO for fine-tuning LSTM networks in IDS applications. While prior studies have used SSO for feature selection or standalone optimization, limited work has explored its integration directly with LSTM weight optimization for both binary and multi-class classification in IDS [14, 15]. This gap motivates the present study.

To tackle these limitations, a refined version of the LSTM algorithm, known as ILSTM, was introduced. The ILSTM integrates the SSO to optimize LSTM weights, thereby minimizing the iteration count required for convergence. This advancement was employed to construct a robust and precise intrusion detection system designed to operate effectively in two key scenarios: binary classification, which identifies normal versus abnormal activities, and multi-class classification, which differentiates and categorizes multiple attack types. The contribution of this work can be summarized as follows:

- Introducing an innovative and enhanced version of the LSTM algorithm, termed ILSTM, which incorporates hybrid techniques, specifically SSO, to optimize the LSTM's weight configurations. This optimization approach resulted in superior performance while requiring fewer iterations.
- Developing IDS based on the ILSTM algorithm. The system effectively handles both binary classification (BCN) and multi-class classification (MCN).
- Performing an extensive assessment of the ILSTM model and the newly designed IDS. The assessment utilized nine performance metrics on two publicly available datasets (NSL-KDD and LITNET-2020). The ILSTM consistently outperformed the standard LSTM in both datasets.
- Evaluating the effectiveness of the ILSTM model against various DL techniques. The evaluation demonstrated that the ILSTM outperformed its counterparts in both BCN and MCN scenarios.

The structure of this paper is organized as follows: Section two reviews related to studies on methods for IDS. Section three outlines the algorithms implemented in the development of the proposed approach. Section 4 introduces the ILSTM in detail. Section 5 describes the experimental setup. Section 6 presents the results and analysis of the proposed algorithm's performance in both BCN and MCN, along with comparisons to other DL and ML methods. Finally, Section 7 concludes this study and outlines directions for future research.

2. Literature Review

IDS plays a vital role in safeguarding computer networks. Previous research has explored various strategies for implementing ID using DL techniques. For instance, the authors developed [14] a method utilizing RNNs was introduced for detecting intrusions. The study utilized the NSL-KDD dataset to assess the effectiveness of the approach, addressing both BCN and MCN to distinguish different types of intrusions. The results demonstrated that the RNN-based model outperformed traditional ML techniques for the given dataset. However, the study also acknowledged limitations associated with the RNN, particularly the exploding and vanishing gradient problems, which were not addressed in their implementation. This highlighted the need for further optimization to enhance the model's performance in real-world intrusion detection scenarios.

Gelenbe et al. [8] developed an IDS using an integrated DL model that combines. CNNs and LSTM networks. This hybrid model was designed to extract spatial features through CNNs and capture temporal dependencies using LSTMs. The NSL-KDD dataset was employed for evaluation, focusing on both BCN and MCN tasks. The integrated model demonstrated high performance, with an accuracy of 98.96% for BCN and 95.84% for MCN. Additionally, the false alarm rate was significantly reduced to 1.25%. These results highlighted combining CNN and LSTM for ID, outperforming traditional ML and standalone DL methods.

Imrana et al. [13] proposed an IDS based on a bidirectional Long Short-Term Memory (BiLSTM) to enhance detection accuracy by leveraging past and future contextual information in network traffic data. The system was evaluated using the NSL-KDD dataset for BCN and MCN tasks. The BiLSTM model reached an accuracy of 99.12% for BCN and 96.45% for MCN, with a detection rate of 98.78% and a false alarm rate of 1.34%. These results demonstrated the superior capability of the BiLSTM approach in detecting network intrusions compared to traditional and unidirectional DL methods.

Chelloug [25] introduced a stacking ensemble of DL models, specifically CNN and Deep Neural Networks (DNN), to enhance intrusion detection capabilities. The developed model was tested on the NSL-KDD dataset, where it demonstrated an accuracy of 99% and attained an average F1-score of 93% for tasks involving MCN.

Wu & Kondo [26] proposed a network IDS that combines a hybrid intelligent model to improve detection performance. The system utilized the NSL-KDD dataset for evaluation, focusing on addressing data imbalance issues and enhancing accuracy. The hybrid sampling method reached an accuracy of 99.15% for BCN and 97.32% for MCN. Ghadermazi et al. [27] proposed an IDS utilizing an optimized ANN to enhance detection performance. The authors employed the NSL-KDD dataset for their evaluation, focusing on both BCN and MCN tasks. The ANN was optimized using GA to select the best parameters. It reached an accuracy of 98.87% for BCN and 96.54% for MCN. Furthermore, the system demonstrated a false alarm rate of 1.45%.

Abdelaziz et al. [28] developed a hybrid model combining a Self-Organizing Map (SOM), DL, and a GA proposed to manage energy consumption in public buildings. SOM was utilized for clustering energy consumption patterns, while the deep learning model analyzed these patterns to predict future energy usage. The GA was employed to choose the best weights of the DL model, enhancing computational costs. The hybrid model achieved a prediction accuracy of 95.8%, reducing energy forecasting errors by 12% compared to standalone deep learning methods. Additionally, the system demonstrated a 15% improvement in energy efficiency by optimizing energy allocation strategies, showcasing its potential for sustainable energy management in public infrastructure. Alsaleh & Binsaeedan [29] investigated the impact of feature selection using the SSO on network anomaly intrusion detection systems. The authors applied the SSO to identify the most relevant features in the NSL-KDD dataset while retaining critical information. The selected features were then used to train machine learning models, including Decision Trees (DT) and Support Vector Machines (SVM). The results achieved an accuracy of 98.53%, a detection rate of 97.89%, and a false alarm rate of 1.22%. These findings highlighted the effectiveness of SSA in optimizing feature selection, leading to enhanced detection accuracy and reduced computational complexity for network intrusion detection. Thankappan et al. [30] proposed a novel hybrid Firefly Algorithm (FA) to optimize the hyperparameters of the XGBoost model to enhance IDS. It focuses on both BCN and MCN tasks. The hybrid FA integrated local search capabilities with global optimization techniques to improve the tuning process of XGBoost. The optimized model achieved an accuracy of 99.24% for BCN and 97.88% for MCN. Additionally, it demonstrated a detection rate of 98.75% and a false alarm rate of 1.15%.

Dora & Lakshmi [31] developed an approach for detecting Distributed Denial of Service (DDoS) attacks by combining optimal feature selection, CNN for feature learning, and a meta-heuristic-based LSTM model. The feature selection was optimized using a meta-heuristic algorithm, which improved the efficiency and accuracy of the model by identifying the most relevant features. CNN was employed to extract insight features from the input data, which were then processed by the LSTM network for classification. The model was evaluated using a public dataset, reaching an accuracy of 98.85%, a detection rate of 97.65%, and a false alarm rate of 1.34%. This combination of techniques demonstrated enhanced performance in detecting and classifying DDoS attacks compared to traditional methods. Deore & Bhosale [11] developed an IDS that integrates feature reduction techniques with an RNN classifier. It minimizes the dataset's dimensionality while retaining critical information. The reduced dataset was then classified using the RNN model. The system reached an accuracy of 98.45%, a detection rate of 97.23%, and a false alarm rate of 1.57%. These results demonstrated combining principal component analysis (PCA) with RNN to reduce computational complexity while maintaining high detection accuracy. Jothi & Pushpalatha [32] introduced WILS-TRS, an optimized DL-based IDS for IoT networks. The framework integrates WOA and LSTM networks for IDS. The WOA reduces the dataset's dimensionality by identifying the most significant features, which are then fed into the LSTM for classification. The framework was evaluated using the UNSW-NB15 dataset. It reached an accuracy of 98.72%, a detection rate of 98.45%, and a false alarm rate of 1.28%, demonstrating its robustness in identifying IoT network intrusions. The results confirmed that the WILS-TRS framework outperformed traditional methods in efficiency and accuracy.

Rashid et al. [33] developed a method for incorporating metaheuristic algorithms to optimize the model's parameters. The authors evaluated their approach using a synthetic dataset. They employed metaheuristic techniques such as GA and PSO to select weights and biases during training. The optimized LSTM achieved an accurate improvement of up to 97.85%, with a reduction in training time by 15% compared to standard LSTM training. These results demonstrated that integrating metaheuristic algorithms can improve LSTM performance for time-series data.

Jihado & Girsang [34] proposed a unified DL approach for IDS by integrating spatial and temporal features. The model combined CNNs and LSTM networks. It was estimated for both BCN and MCN tasks. The approach reached an accuracy of 99.05% for BCN and 96.89% for MCN. Additionally, the system demonstrated a false alarm rate of 1.15%, showcasing its efficiency and robustness in detecting network intrusions. The results highlighted the benefits of combining spatial and temporal analysis for enhanced intrusion detection. Zivkovic et al. [35] developed an IDS combining the Firefly Algorithm (FA) with a DNN to enhance detection performance. The FA was utilized to optimize the DNNs in each layer, improving model accuracy and efficiency. The optimized DNN reached an accuracy of 99.12% for BCN and 96.87% for MCN.

Ali et al. [36] proposed a framework that is integrated into ML, including Decision Trees (DT), SVM, and Random Forest (RF), to classify and detect DDoS attacks. The BoT-IoT dataset was used to evaluate the system's performance. Among the models, RF achieved the highest accuracy of 99.24%, a precision of 98.95%, and a recall of 98.78%, making it the most effective classifier in the framework. Alzaqebah et al. [37] proposed a Modified Grey Wolf Optimization (MGWO) algorithm to optimize the performance of an IDS. The MGWO algorithm was utilized for feature selection, reducing the dimensionality of the dataset while retaining critical attributes to improve classification performance. The approach is integrated with ML classifiers such as SVM and RF. The IDS achieved an accuracy of 99.12%, a precision of 98.87%, and a recall of 98.53%, significantly outperforming traditional feature selection methods. The study demonstrated that the MGWO algorithm effectively optimized IDS performance while reducing computational overhead. The researchers proposed a method for IDS using a Hybrid Artificial Bee Colony (HABC) algorithm. The HABC algorithm reduces the dimensionality of the dataset while maintaining classification accuracy. It is integrated with ML classifiers, like SVM and RF. The HABC-based IDS reached an accuracy of 98.87%, a detection rate of 97.56%. The study highlighted the efficiency of HABC in improving IDS performance by effectively balancing feature reduction and classification accuracy. The researchers proposed a framework for IDS using FA [38, 39]. The FA reduces the dataset's dimensionality while retaining critical information for classification. The approach was estimated using SVM and k-Nearest Neighbors (k-NN). The FA is reaching an accuracy of 97.85%, a precision of 96.34%, and a false alarm rate of 2.12%. The study demonstrated that FA effectively optimized feature selection, as Toldinas et al. [40] proposed a novel IDS based on multistage DL using image recognition techniques. The approach transformed network traffic data into visual representations, which were then processed using CNNs for feature extraction and classification. The system employed a multi-stage framework to enhance detection accuracy by combining predictions from multiple CNN models. It reached an accuracy of 99.23% for BCN and 97.85% for MCN.

3. Background

This work explores the integration of SSO and LSTM networks, two powerful techniques in the fields of optimization and DL, respectively.

3.1. Salp Swarm Optimization

SSO algorithm is a bio-inspired technique modeled after the movement patterns of salps, gelatinous marine creatures. Salps navigate the ocean by rhythmically contracting and expanding their bodies, generating jet-like propulsion. This natural movement is mirrored in the SSO, where a group of salps collaboratively explores the solution space to identify optimal outcomes for a given problem [17].

3.1.1. Key Components

The salp population in the SSO algorithm symbolizes a collection of candidate solutions for an optimization task. Each salp occupies a specific position within the search space, representing a set of variables or parameters, such as the weights in a LSTM network. The salp at the forefront of the population, referred to as the "leader," signifies the most optimal solution identified up to that point [18]. The remaining salps, known as "followers," adjust their positions by considering both the leader's position and their own prior locations in the search space.

3.1.2. Equation and Mechanism

This section shows the steps of the SSO, as follows [17-20]:

Step 1: Initialization

Set the necessary parameters for the algorithm, including the number of salps N, the dimensionality of the problem D, and the maximum loops T. Randomly assign the initial positions of the salps within the defined search space, represented as:

$$X(0) = \text{rand}\ (\min, \max) \qquad \forall i \in [1, N] \tag{1}$$

- X(0): Represents the initial position matrix for all salps at iteration zero. Each salp has a unique position defined by a vector in the search space.
- rand (min, max): Random number generation within the specified lower and upper bounds of the search space.
- i∈ [1, N]: Indicates that the initialization is done for all salps individually (from salp 1 to salp N).

Assess the performance of each salp by calculating its fitness using a predefined objective function f:

$$\text{Fitness} = f(X(0)) \qquad \forall i \in [1, N] \tag{2}$$

- f: The objective function is utilized to determine the best solution. It measures how well a particular salp's position solves the optimization problem.
- X(0): The current position of the salp.
- Fitness: A numerical value representing the quality of the solution. Lower or higher values (depending on the optimization goal) indicate better solutions.

Step 2: Identify the Leader

Identify the optimal point among the salps (the leader):

$$\text{Leader} = \arg\max\ (\text{Fitness}) \tag{3}$$

- max(Fitness): This operation identifies the highest fitness value among all the salps in the population, meaning it finds the best-performing candidate solution.
- arg max (Fitness): The arg max operator returns the index or position of the salp with the highest fitness value rather than the value itself. This index corresponds to the "leader" salp, which serves as the reference point for guiding other salps during the optimization process.

Step 3: Update Positions

For each iteration t from 1 to T:

Leader movement: Update the leader's position using:

$$X_1(t) = X_i(t-1) + R.\ \text{rand} \tag{4}$$

- $X_1(t)$: The updated the leader salp at the current loop t. The leader is the salp with the best fitness value identified in the population.
- $X_i(t-1)$: The previous position of the leader salp from the last iteration $(t-1)$.
- R: A random coefficient that influences the size and direction of the position update.
- rand: A random number generated between 0 and 1, introducing stochastic behavior to ensure exploration of the search space.

Follower movement: Each follower i updates its position toward the leader:

$$X_{f,i}(t) = x(t) + (X_1(t) - X_{f,i}(t-1)) \tag{5}$$

- $X_{f,i}(t)$: The updated position of follower salp i at the current loop t.
- X(t): The current general position of all salps, often referring to the search space boundaries or average position of the swarm.
- $X_1(t)$: The current position of the leader salp at loop t.
- $X_{f,i}(t-1)$: The previous position of follower salp i from the last iteration $(t-1)$.

Calculate the fitness for all salps after updating their positions:

$$\text{Fitness}\ (t) = f(x(t)) \qquad \forall i \in [1, N] \tag{6}$$

- Fitness(t): The fitness value calculated for each salp at iteration t.
- f(X(t)): The estimation of the solution's quality, represented by the position X(t), is determined using the defined fitness function.
- X(t): The position of a salp in the search space at iteration t.
- N: The total number of salps in the population.
- $\forall i \in [1,N]$: This indicates the calculation is applied to all salps in the population (from the first to the N-th salp).

Step 4: Update Leader

Check if any follower has a better fitness than the current leader and update the leader if necessary:

$$\text{Leader} = \arg\max_i\ (\text{Fitness}_i\ (t)) \tag{7}$$

where; i is the index of all solutions in the swarm.

Step 5: Termination Criteria

The procedure concludes when any of the specified criteria are satisfied:

- The process stops when the predefined iteration limit T is attained

- The enhancement in the optimal fitness score remains under a specified threshold for a determined number of loops.

Step 6: Output

The last result is the position of the leader salp: optimal solution;

$$\text{Best Solution} = X_1\ (t) \tag{8}$$

Algorithm 1. SSO for LSTM Weight Optimization in High-Dimensional Spaces

```
Input:
   - N: Number of salps
   - D: Dimension of the problem
   - T: Maximum number of iterations
   - R: Maximum movement range
   - epsilon: Attraction coefficient
Output The best position of the leader salp$_{Best\ LSTM\ Weights}$
1. Randomly initialize the positions of salps:
   For i = 1 to N:
       X[i] = Random (min, max) // Initialize salp positions
2. Evaluate fitness of each salp:
   For i = 1 to N:
       Fitness[i] = f(X[i]) // Evaluate the objective function
3. Identify the leader salp:
   LeaderIndex = argmax (Fitness) // Find the index of the best salp
   Leader = X[LeaderIndex] // Get the position of the leader
4. Main optimization loop:
 For t = 1 to T:
       // Update leader position
       X[LeaderIndex] = X[LeaderIndex] + R * rand () * perturbation () // Update leader's position
   // Ensure X[LeaderIndex] stays within bounds:
    X[LeaderIndex] = clamp(X[LeaderIndex], min, max)
       // Update follower positions
       For i = 1 to N:
           If i != LeaderIndex:
              X[i] = X[LeaderIndex] + epsilon * (X[LeaderIndex] - X[i]) // Update follower position
        // Ensure X[i] stays within bounds:
        X[i] = clamp(X[i], min, max)
       // Evaluate fitness of updated positions
       For i = 1 to N:
             Fitness[i] = f(X[i]) // Re-evaluate fitness
       // Update leader if a better solution is found
       NewLeaderIndex = argmax (Fitness) // Find the best salp
       If Fitness[NewLeaderIndex] > Fitness[LeaderIndex]:
           LeaderIndex = NewLeaderIndex // Update leader index
           Leader = X[LeaderIndex] // Update leader position
Return Leader$_{Optimized\ LSTM\ weights}$ // Return the position of the best salp
```

Algorithm 1 outlines an SSO algorithm for optimizing LSTM network weights. It begins by initializing the positions of a population of salps (N) randomly within a predefined range. The fitness function (f) evaluates the optimization goal. The salp is designated as the leader, and its position becomes the reference for updating other salps (followers).

The optimization process iterates for a maximum number of steps (T). During each iteration, the leader's position is updated first using a perturbation function scaled by a random factor (R). This update ensures search space exploration while adhering to the problem's bounds using a clamp function. Next, follower salps adjust their positions relative to the leader, guided by an attraction coefficient (ϵ), encouraging convergence towards the leader. These positions are also checked and adjusted to stay within bounds. After updating all positions, the algorithm recalculates the fitness of all salps.

If any follower achieves better fitness than the current leader during an iteration, the leader is updated to this new optimal solution. The algorithm continues this process until, at most, a set of loops is reached, refining the leader's position over time. Finally, the position of the best-performing salp, representing the optimized LSTM weights, is returned as the output. This approach balances exploration and exploitation, ensuring efficient convergence towards the optimal solution.

3.2. Long Short-Term Memory

It is a detailed explanation of LSTMs, including their architecture, mechanisms, and equations [41].

3.2.1. Key Components

The LSTM unit works together to manage information flow, referring to Figure 1. The key components include **Cell State (C_t), Hidden State (h_t), Forget Gate (f_t), Input Gate (i_t), and Output Gate (o_t)** [42-45].

3.2.2. Equation and Mechanism

LSTM has six steps as follows [46-52]:

Step 1: Gates and Inputs

It is shown as:

- The preceding concealed h_{t-1}.
- The present input 2_t.
- The LSTM uses these inputs to calculate the three gates and the cell state.

Step 2: Forget Gate (f_t)

The forget gate determines which data from the cell states needs to be removed. Its computation is performed as follows:

$$f_t = \sigma\,(W_f \,.\, [h_{t-1}, x_t] + b_f) \tag{9}$$

where; f_t: It is a vector where each element falls within the range of zero to one; σ: The sigmoid activation function ensures that the forget gate outputs values between zero and one; W_f: It contains learnable parameters that are optimized during training to adjust how the forget gate operates; h_{t-1}: This represents information from the previous step that impacts the forget decision; x_t: This contains new information being introduced to the LSTM at this step; $[h_{t-1}, x_t]$: Combining both ensures the forget gate considers both past context and current input when making decisions; b_f: It is a learnable parameter that provides flexibility in adjusting the forget gate's behavior.

Step 3: Input Gate (i_t) and Candidate Cell State (C_t)

The input gate determines which fresh data should be incorporated into the cell state.

$$i_t = (W_i . [h_{t-1}, x_t] + b_i) \tag{10}$$

The candidate cell state **$\check{C}_t$** is created from the input:

$$\check{C}_t = \tanh\,(W_c . [h_{t-1}, x_t] + b_c) \tag{11}$$

where: W_i, W_c are the weight matrices. b_i, b_c are biases.

Step 4: Update Cell State (C_t)

The new cell state is created as follows:

$$C_t = f_t * C_{t-1} + i_t * C_t \quad (12)$$

where; C_t is the new cell state at time step t. This is the memory of the LSTM, updated with a combination of past memory and new information. f_t is the forget gate output. C_{t-1} is the cell state from the previous time step t−1. This represents the LSTM's memory carried forward from the past. i_t is the input gate output. Similar to f_t, it is a vector with values between zero and one, determining how much of the candidate cell state $\check{C}_t$ should be added to the current cell state.

Step 5: Output Gate (O_t)

The output gate determines the next hidden state:

$$O_t = \sigma\,(W_o \,.\, [h_{t-1}, x_t] + b_o) \quad (13)$$

where; W_o is the weight matrix. It is a learnable parameter that maps the concatenated input and precedes hidden state into the dimensions of O_t. b_o is the bias vector for the output gate. It is a learnable parameter that helps shift the activation to improve learning.

Step 6: Compute the Hidden State (h_t)

Finally, the hidden state is calculated as:

$$h_t = O_t * \tanh\,(C_t) \quad (14)$$

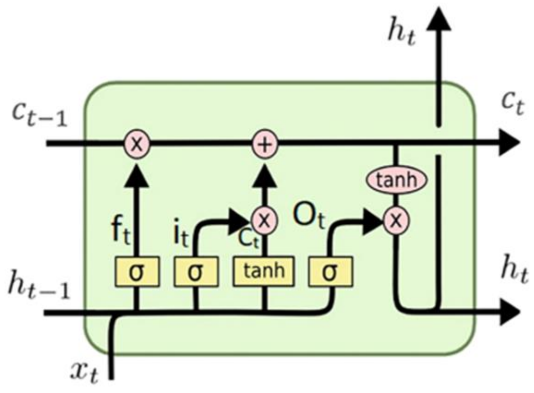


Figure 1. Structure and Functionality of LSTM

Algorithm 2. LSTM-Based Sequential Processing for Intrusion Detection in Network Traffic

```
1. Initialize LSTM parameters: Key components
2. For each time step t in the input sequence:
   a. Get input x_t at time step t.
   b. Compute forget gate:
   c. Compute input gate:
   d. Compute candidate cell state:
   e. Update cell state
   f. Compute output gate
   g. Compute hidden state
3. $Output_{The\ final\ result\ of\ intrusion\ detection}$
```

Algorithm 2 outlines the sequential operations within an LSTM network, specifically for IDS tasks. It begins by initializing the LSTM parameters, which include weight matrices and bias vectors for the input and output gates, respectively. These parameters are learnable and are fine-tuned during the training process. The initial hidden state (h_0)

and cell state (C_0) are typically initialized to zero vectors, representing an empty memory at the start of processing the sequence. These states are critical as they propagate information across time steps.

The LSTM processes the input x_t, which could represent features extracted from network traffic in an IDS. The forget gate is computed and precedes cell state (Ct−1) to retain or discard. Simultaneously, the input gate is calculated to determine how much new information, represented by the candidate cell state ($\check{C}_t$), should be integrated into the cell state. These gates ensure that the LSTM can effectively focus on relevant features while ignoring noise.

The (C_t) combines the outputs of the forget and input gates. The h_t acts as the output of the LSTM and is passed to subsequent layers or time steps. After processing all time steps in the sequence, the last result represents the IDS outputs, such as classifications of network traffic into normal or attack categories. This step-by-step processing enables the LSTM to leverage its memory capabilities to detect complex patterns indicative of intrusions in sequential data effectively.

4. Material and Methods

Figure 2 illustrates a multi-phase process for training and optimizing an LSTM network, potentially involving a hybrid DL and optimization approach. Below is a clarification of each phase:

Phase 1: Preprocessing and Initial LSTM Training

1. Start: The process starts with collecting and preparing the Training Datasets.
2. Data Preprocessing: Several steps are involved in preparing the data for training:
   - Normalization: Converts categorical or text-based data into numerical formats suitable for LSTM networks.
   - Hybrid Sampling: This could refer to techniques like oversampling or undersampling to handle imbalanced datasets and ensure better model performance.
   - Normalization: This scales the data to a consistent range, such as [0,1] or [-1,1], ensuring that different features contribute equally to the model.
3. Performance Evaluations of Trained LSTM Network: The performance of the trained LSTM model is assessed using metrics such as accuracy, precision, recall, or other relevant metrics on a validation dataset. This performance evaluation step will inform subsequent optimization phases.

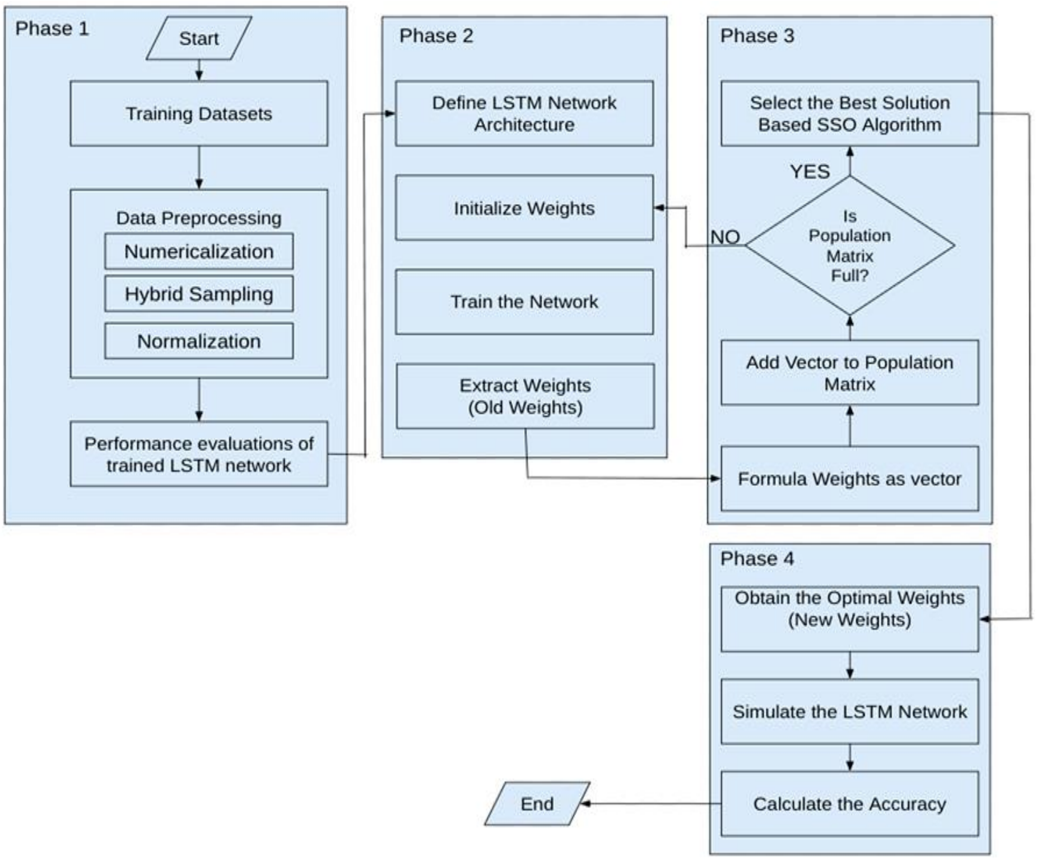

Phase 1
Start
Training Datasets
Data Preprocessing
Numericalization
Hybrid Sampling
Normalization
Performance evaluations of trained LSTM network
Phase 2
Define LSTM Network Architecture
Initialize Weights
Train the Network
Extract Weights (Old Weights)
Phase 3
Select the Best Solution Based SSO Algorithm
YES
NO
Is Population Matrix Full?
Add Vector to Population Matrix
Formula Weights as vector
Phase 4
Obtain the Optimal Weights (New Weights)
Simulate the LSTM Network
Calculate the Accuracy
End

Figure 2. The proposed model Architecture

Phase 2: Defining and Training the LSTM Network

1. Define LSTM Network Architecture: The structure of the LSTM network is defined, specifying the number of layers, units per layer, and other hyperparameters like activation functions.
2. Initialize Weights: The weights of the LSTM network are initialized, either randomly or based on some pre-trained model.
3. Train the Network: The LSTM network is trained on the preprocessed dataset to adjust the weights.
4. Extract Weights (Old Weights): Once the LSTM network is trained, the current weights (referred to as "Old Weights") are extracted for further optimization in the next phase

Phase 3: Optimization Using an SSO Algorithm

1. Select the Best Solution Based on the SSO Algorithm: This step involves using SSO algorithm to improve the weights of the LSTM network. The SSO algorithm works by searching for better solutions based on a population of candidate solutions (weight vectors).
2. Check Population Matrix: The algorithm checks whether the population matrix (which holds all candidate solutions) is full.
   - If NO, the process continues by adding a new weight vector to the population matrix.
   - If YES, the algorithm proceeds to select the best solution.
3. Add Vector to Population Matrix: New candidate weight vectors are added to the population matrix, forming the basis for the search.
4. Formulate Weights as a Vector: The LSTM weights are treated as a vector that can be optimized. This step prepares the weight vector for inclusion in the population matrix.

This phase effectively optimizes the weights by selecting the best candidates from the population matrix using the SSO algorithm, ensuring that the weights lead to improved network performance.

Phase 4: Obtaining the Optimized LSTM Weights

1. Obtain the Optimal Weights (New Weights): After running the optimization algorithm, the best solution or weight vector is selected as the New Weights for the LSTM network.
2. Simulate the LSTM Network: With the newly optimized weights, the LSTM network is simulated to check how it performs on the dataset.
3. Calculate the performance metrics: The performance of the LSTM network with the new optimized weights is calculated to assess the improvements achieved through the optimization process.

Algorithm 3 shows the steps that how SSO optimized weights in LSTM in detail and it has been applied by using the TensorFlow library, as follows.

In Algorithm 3, Initialization: Define the parameters for the SSO, including the size of the population (number of salps), the dimensionality corresponding to the weights of the LSTM, and the bounds for the weight values.

Population Generation: Create an initial population of salps, each represented as a vector of weights randomly initialized within the defined bounds.

Fitness Evaluation: For each salp, evaluate how well it performs as a set of weights for the LSTM. This is done by running the LSTM model with those weights and calculating a fitness score.

Optimization Loop:

- Sorting: Sort the salps based on their fitness scores to identify the best-performing salps.
- Position Update: The leader salp (the best one) is updated based on a random step, while other salps follow the leader to explore the solution space.
- Bounds Application: Ensure that all salps remain within the defined weight limits.
- Re-evaluation: After updating positions, re-evaluate the fitness of all salps.

Return Best Weights: After all iterations, the best-performing salp is returned as the optimized weights for the LSTM.

Algorithm 3. SSO for Fine-Tuning LSTM Weights in Intrusion Detection Systems

```
1. Input: Initialize the parameters for the SSO Algorithm:
   - Population size (N)
   - Number of dimensions (D) corresponding to LSTM weights
   - Maximum number of iterations (T)
   - Lower and upper bounds for the weights
2. Generate an initial population of salps (randomly):
   For i = 1 to N:
       salps[i] = random_vector(lower_bound, upper_bound, D)
3. Evaluate the fitness of each salp:
   For each salp in salps:
       fitness[i] = evaluate_fitness(LSTM_model, salps[i])
4. Main loop for optimization:
   For t = 1 to T:
       a. Sort salps based on fitness (best to worst)
       b. Update the position of salps:
           For i = 1 to N:
               if i == 1:
                   # Leader salp
                   salps[i] = salps[i] + random_step(size)
               else:
                   # Follow the leader
                   salps[i] = salps[i-1] + random_step(size)
       c. Apply bounds to ensure salp positions are within limits:
           For i = 1 to N:
               salps[i] = clamp(salps[i], lower_bound, upper_bound)
       d. Evaluate the fitness of each salp again:
           - For each salp in salps:
               fitness[i] = evaluate_fitness(LSTM_model, salps[i])
5. Output: Return the best salp as the optimized weights for the LSTM:
   -best_weights = salps[1]
```

5. Experimental Setup

This section summarizes the performance metrics employed to measure the algorithm's effectiveness. Additionally, the section describes the preprocessing steps and characteristics of the two public datasets, NSL-KDD 2009 and LITNET-2020, utilized for the evaluation. Finally, the proposed algorithm is tested on a contemporary dataset to validate its efficiency. Its performance is compared against state-of-the-art methods and DL approaches.

5.1. Performance Metrics

To determine the quality of ILSTM [34], nine metrics were utilized: accuracy (RCY), detection rate (NE), false alarm rate (EME), precision (SPN), specificity (CTY), f-measure (FM), false negative rate (EEE), mathematical correlation coefficient (LNT), and kappa coefficient (KC) as shown from equations 15 to 26. Each of these metrics can be mathematically determined using four primary performance indicators—true positive (TP), false positive (FP), true negative (TN), and false negative (FN)—which are derived from the confusion matrix [48].

$$\text{RCY} = \frac{TP+TN}{TP+TN+FN+FP} \tag{15}$$

$$\text{NE} = \frac{TP}{TP+FN} \tag{16}$$

$$\text{CTY} = \frac{TN}{TN+FP} \tag{17}$$

$$\text{SPN} = \frac{TP}{TP+FP} \tag{18}$$

$$\text{FM} = 2 \times \frac{SPN*Recall}{SPN+Recall} \tag{19}$$

$$\text{EME} = \frac{FP}{FP+TN} \tag{20}$$

$$\text{EEE} = \frac{FN}{FN+TP} \tag{21}$$

$$\text{LNT} = \frac{(TP*TN)-(FP*FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{22}$$

$$\text{KC} = \frac{absolute-Expect}{1-Expect} \tag{23}$$

where Absolute = Accuracy and;

$$\text{Expect} = \frac{A+B}{(TP+TN+FP+FN)} \tag{24}$$

values of A and B can be obtained as;

$$\text{A} = \frac{(TP+FN)(FN+FP)}{(TP+TN+FP+FN)} \tag{25}$$

$$\text{B} = \frac{(FP+TN)(FN+TN)}{(TP+TN+FP+FN)} \tag{26}$$

5.2. Dataset 1: NSL-KDD Dataset

The NSL-KDD dataset is designed to address some of its inherent issues, such as the redundancy and imbalance of records. The NSL-KDD dataset categorizes network traffic into five distinct classes: Normal (MALN), Denial of Service (DFER), Remote-to-Local (RTLO), and User-to-Root (UTRO). Each class represents either a legitimate network activity or a specific type of cyberattack [53-57].

Table 1. Overview and Characteristics of the NSL-KDD

Datasets	MALN	DFER	RTLO	UTRO	Total
KDDTrain+	68454	46138	916	53	136184
KDDTest+	9822	7569	2865	211	23655
KDDTest-21	2263	4453	2865	211	12961

Table 1 categorizes data into five types: DFER, RTLO, and UTRO attacks. The dataset is composed of three types: KDDTrain+, KDDTest+, and KDDTest-21.

- KDDTrain+ includes 136,184 total instances, with the majority being MALN (68,454 instances). DFER attacks are the second-largest category (46,138), followed by RTLO (916), and UTRO (53), indicating its use as the primary training set.
- KDDTest+ contains 23,655 instances, split into 9,822 MALN, 7,569 DFER, 2,865 RTLO, and 211 UTRO records, designed for comprehensive testing across all attack types.
- KDDTest-21, a smaller test set, consists of 12,961 instances with 2,263 MALN, 4,453 DFER, 2,865 RTLO, and 211 UTRO entries, often used for focused evaluations.

It emphasizes the imbalanced distribution of attack types, where Normal and DoS dominate, while U2R and R2L are underrepresented, posing challenges for detection models in handling minority classes effectively.

5.2.1. Dataset Preprocessing: NSL-KDD Dataset

It is a critical step when working with the dataset. The data preprocessing is composed of three primary phases: transformation, dataset balancing, and normalization. Each phase addresses specific challenges within the dataset [58-60].

Data transformation: It contains a mix of numerical and categorical features. ML models generally require numerical inputs, making it necessary to convert categorical features into numerical representations. This step minimizes noise and enhances the computational efficiency of the models while preserving the essential information for analysis. SHAP analysis is employed to identify the most significant impact on a specific output. This process involved assessing how each feature influences the target variable and determining the relative feature in shaping the final forecasting results. It is presented in Figures 3 and 4.

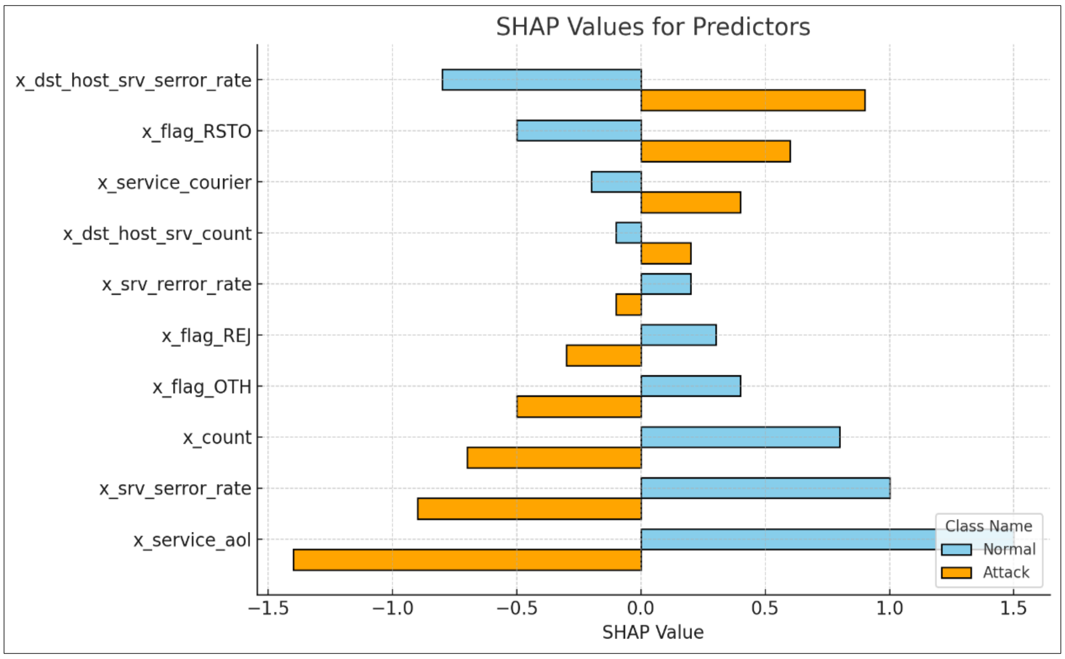


Figure 3. SHAP Analysis for BCN on the NSL-KDD

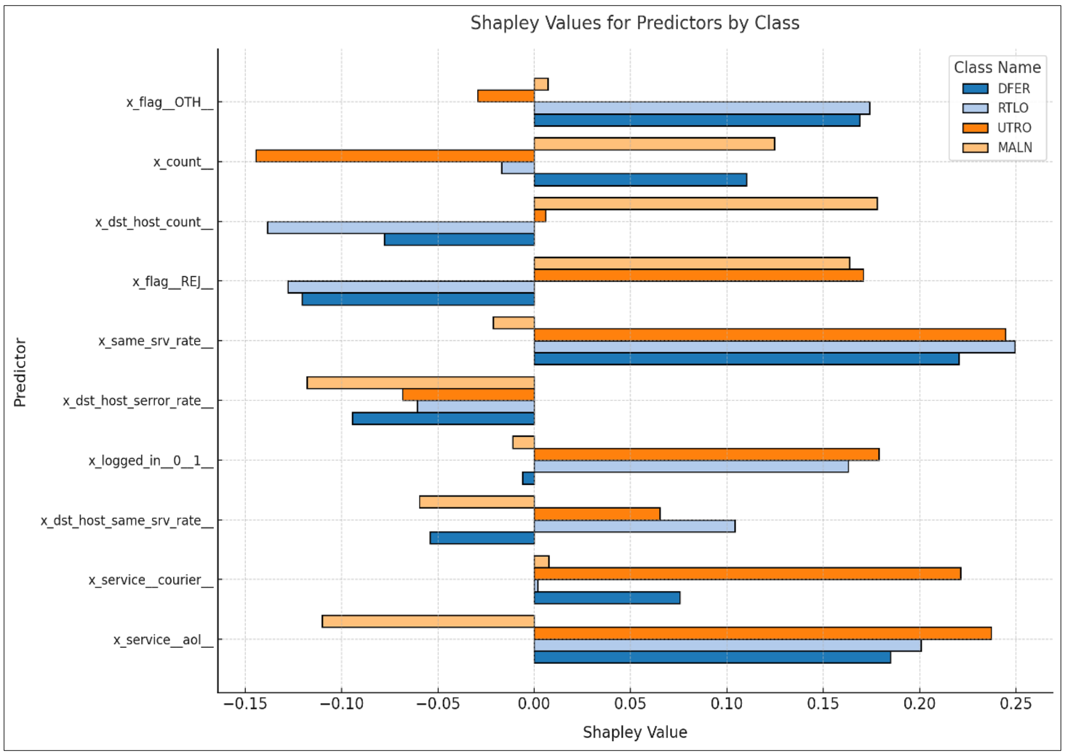


Figure 4. SHAP Analysis for MCN on the NSL-KDD

Dataset Balancing: Class imbalance is a critical issue, as certain attack types, such as UTRO and RTLO, are underrepresented compared to classes like DFER or MALN. This imbalance can result in biased models that perform well in majority classes while failing to detect minority class attacks. Hybrid sampling techniques are employed to address this issue, combining oversampling and undersampling methods. Oversampling involves generating synthetic samples for minority classes using techniques such as SMOTE (Synthetic Minority Oversampling Technique).

Normalization: It is the final phase of preprocessing, where the dataset's features are scaled to a consistent range to eliminate disparities caused by differing feature magnitudes. For example, some features may have values ranging from 0 to 1, while others could extend into the thousands. Normalization techniques, such as min-max scaling or Z-score normalization, improve the convergence speed and stability of ML models.

5.3. Dataset 2: LITNET-2020 Dataset

It is a comprehensive dataset designed for research in analyzing malicious network activities. It contains both benign and malicious network traffic, enabling researchers to evaluate various IDS and ML models. The dataset represents real-world scenarios, offering a broad spectrum of attack types and substantial benign traffic, which is essential for effective network security testing and development [59].

Table 2. Overview of the LITNET-2020

Class	Size
Benign (BN)	37,534,971
SYN flood (SF)	3,836,949
Code red (CR)	1,366,813
UDP flood (UF)	94,694
Smurf (SR)	60,581
LAND DoS (LD)	53,528
W32.Blaster (WB)	25,312
HTTP flood (HF)	23,161
ICMP flood (IFD)	12,739
Port scan (PS)	6343
Reaper worm (RW)	1287
Spam Botnet (SB)	758
Fragmentation (FGN)	488

Table 2 provides an overview of the LITNET-2020 dataset, summarizing various traffic classes and their respective sizes. The dataset primarily consists of benign traffic, with 37,534,971 instances, which dominates the dataset. Among the malicious traffic classes, SYN flood has the highest representation with 3,836,949 instances, followed by Code Red with 1,366,813 entries. Other significant attack classes include UDP flood (94,694), Smurf (60,581), and LAND DoS (53,528). Less frequent attacks include W32. Blaster (25,312), HTTP flood (23,161), ICMP flood (12,739), and Port Scan (6,343). The rarest attack types are Reaper worm (1,287), Spam Botnet (758), and Fragmentation (488). This distribution highlights the dataset's imbalance, with benign and SYN flood traffic significantly outweighing other classes, posing challenges for intrusion detection systems in accurately identifying minority attacks.

5.3.1. Dataset Preprocessing: LITNET-2020 Dataset

It is a critical step in ensuring the effective use of the LITNET-2020 dataset for IDS development. The preprocessing of this dataset is divided into three distinct phases: dataset preprocessing and SHAP analysis, dataset balancing, and the data splitting approach. Each phase is essential in addressing specific challenges related to the dataset's structure and distribution [53-56].

The first phase, dataset preprocessing and SHAP analysis, focuses on preparing raw data for analysis and understanding the impact of features on prediction outcomes. Following this, SHAP analysis is employed to estimate individual features. SHAP provides an interpretable framework for assessing the features, offering insights into which structural predictors most strongly influence detection performance. This step focuses on the most critical attributes for accurate predictions, as shown in Figure 5.

The second phase, A hybrid sampling approach, combining oversampling techniques like SMOTE with under sampling of majority classes, is implemented to achieve a balanced distribution across all classes. This ensures that the model receives adequate representation from both frequent and infrequent classes.

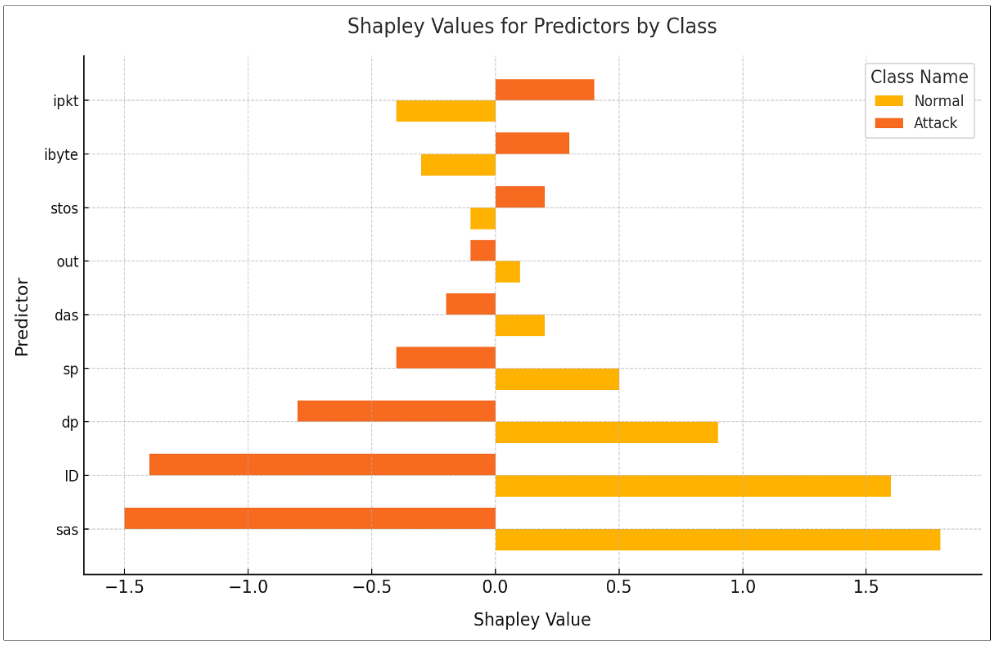


Figure 5. SHAP Analysis on the LITNET-2020

Finally, a stratified splitting technique is used to maintain the class distribution across all subsets, ensuring that minority classes are not overlooked in any split. By carefully separating the data, this approach minimizes overfitting. This division was determined following a preliminary experiment conducted to identify the most effective data-splitting strategy. The findings of this experiment are presented in Figure 6.

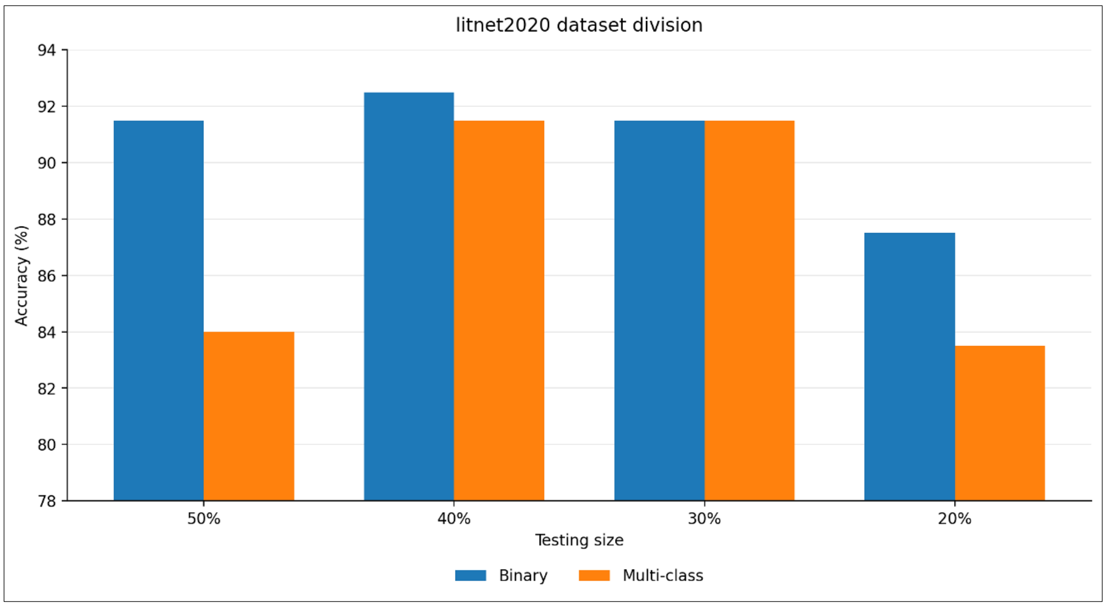


Figure 6. Data Partitioning Strategy for LITNET-2020

5.4. Parameter Setting

Table 3 outlines the parameter settings for optimizing the performance of LSTM network. The Hidden Units parameter, set to 128, balances the model's capacity and computational efficiency. The Number of Layers, set to 2, captures sufficient hierarchical features without overcomplicating the model. A Dropout Rate of 0.3. The Batch Size is set to 64. A Learning Rate of 0.001 ensures gradual and stable weight updates, particularly effective when paired with the Adam Optimizer, which is both efficient and robust. The Activation Function Tanh is standard for LSTM gates, allowing the network to manage non-linearity. Training spans 100 Epochs, sufficient for convergence in most cases, with early stopping applied if necessary. The Sequence Length, set to 20, captures sufficient temporal dependencies, and Gradient Clipping, set to 1.0, ensures stable training by preventing exploding gradients. Together, these parameters are designed to maximize the LSTM's effectiveness in learning from sequential data while maintaining computational feasibility [34].

ILSTM is moderately sensitive to the SSO hyperparameters. Since the SSO directly affects the weight initialization and convergence behavior of the LSTM model, parameters such as Population size, Maximum number of iterations, search space boundaries, Inertia weight, Randomness factor, and Convergence threshold.

The same hyperparameters (Tables 3 and 4) and LSTM architecture were used for both datasets (NSL-KDD and LITNET-2020) to ensure consistency and fairness in evaluation.

Table 3. Configuration of Parameters for LSTM

Parameter	Description	Optimal Suggested Value
Hidden Units	Number of neurons in the hidden layer	128
Number of Layers	Number of LSTM layers stacked	2
Dropout Rate	Fraction of neurons dropped out during training	0.3
Batch Size	Number of samples per batch during training	64
Learning Rate	Step size for updating weights during optimization	0.001
Activation Function	Function applied to the output of the LSTM	Tanh
Epochs	Total count of full iterations over the training dataset	100
Optimizer	Optimization algorithm for weight updates	Adam
Sequence Length	Total count of temporal steps within the input sequence	20
Gradient Clipping	Maximum gradient value to prevent exploding gradients	1.0

Table 4 describes the parameters used for an optimized SSO algorithm. The Population Size is set to 100, indicating that the optimization involves 100 salps, which are agents in the algorithm. Maximum Iterations is also set to 100, meaning the optimization process will run for a maximum of 100 iterations. Search Space Boundaries are defined as [−10,10][-10, 10][−10,10], indicating that the variables being optimized are constrained within these upper and lower limits. The Inertia Weight, ranging from 0.1 to 1.0, is used to balance exploration and exploitation during the optimization process. The Randomness Factor, ranging between 0.01 and 0.1, introduces diversity in the population to prevent premature convergence on suboptimal solutions. Lastly, the Convergence Criteria specifies the stopping conditions, which could be either arriving at most loops or satisfying a predefined error threshold. These parameters work together to guide the optimization process efficiently.

Table 4. Configuration of Parameters for SSO

Parameter	Description	Value
Population Size	The number of salps	100
Max loops	The total loops	100
Search Space Boundaries	The upper and lower limits	[-10, 10]
Inertia Weight	Balances exploration and exploitation during optimization.	0.1–1.0
Randomness Factor	Improve diversity and prevent premature convergence.	0.01–0.1
Convergence Criteria	Stopping criteria	Maximum iterations or error threshold

In the paper, a manual grid-search approach was used to identify optimal SSO hyperparameters. Specifically, Population Size was set to 100, based on initial experimentation. Smaller populations were tested (e.g., 20, 50), but these resulted in poorer convergence or premature stagnation. Max Iterations was fixed at 100, balancing computational efficiency and convergence accuracy. Search Space Boundaries were empirically set to [-10, 10] after testing broader and narrower bounds, which provided stable optimization without overshooting. Inertia Weight and Randomness Factor were defined within adaptive ranges (0.1–1.0 and 0.01–0.1 respectively), allowing dynamic balance between exploration and exploitation. Convergence Threshold was used as a secondary stop condition, though in most runs, the model converged within the fixed iteration limit.

6. Results and Discussion

This section shows the experiments and their analysis, which were carried out using the 2 datasets mentioned earlier.

6.1. Trial 1: Evaluation of ILSTM's Effectiveness for BCN on the NSL-KDD Dataset

It shows the ILSTM framework in detecting intrusions through the classification of network data traffic.

6.1.1. Evaluation of ILSTM Effectiveness with the KDDTest+ Dataset

Table 5 presents a comparative analysis between standard LSTM and Improved LSTM (ILSTM) models for BCN tasks. On the KDDTest+ dataset, ILSTM outperforms LSTM across all metrics, achieving higher values, such as 93.09 for RCY, 97.57 for EME, and 95.87 for SPN, compared to 83.85, 79.88, and 76.49, respectively, for LSTM. Similarly, on the KDDTest-21 dataset, ILSTM demonstrates significant improvements, particularly in metrics such as RCY (86.89 vs. 71.34), EME (98.83 vs. 74.55), and SPN (98.91 vs. 30.44). The ILSTM also shows enhanced performance in less prominent metrics like CTY and KC, demonstrating its robustness and efficiency. Overall, the table highlights the superior classification capabilities of ILSTM over the standard LSTM model in binary classification tasks on both datasets.

Table 5. Performance Comparison of LSTM and ILSTM in BCN

Dataset	Method	RCY	NE	EME	SPN	CTY	EEE	FM	LNT	KC
KDDTest+	LSTM	83.85	**87.63**	79.88	76.49	20.12	**13.48**	81.2	65.78	65.36
	ILSTM	**93.09**	85.14	**97.57**	**95.87**	**4.62**	16.18	**90.47**	**83.51**	**83.16**
KDDTest-21	LSTM	71.34	**50.78**	74.55	30.44	27.67	**51.44**	37.91	20.42	19.3
	ILSTM	**86.89**	29.26	**98.83**	**98.91**	**0.06**	81.94	**44.31**	**48.64**	**39.16**

Figure 7 presents two confusion matrices for BCN on the KDDTest+ dataset, comparing the performance of LSTM (a) and ILSTM (b) models. In matrix (a) for LSTM, the "Attack" class has 8,993 correctly predicted samples (True Positives) and 3,840 misclassified as "Normal" (False Negatives), while the "Normal" class shows 8,823 correct predictions (True Negatives) and 888 misclassified as "Attack" (False Positives). In contrast, matrix (b) for ILSTM demonstrates improved performance, with 12,355 correctly predicted "Attack" samples and only 478 misclassified as "Normal." Similarly, 8,631 "Normal" samples are correctly identified, with 1,080 misclassified as "Attack." The ILSTM significantly reduces misclassification errors for the "Attack" class, indicating superior accuracy and reliability between "Attack" and "Normal" behaviors.

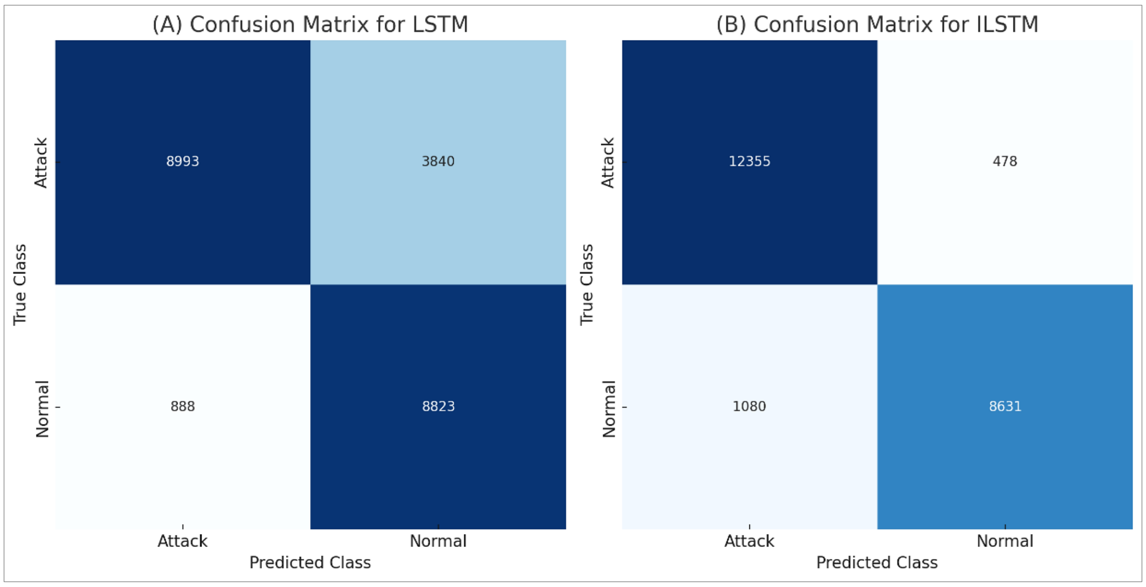


Figure 7. Confusion matrices for KDDTest+ in BCN

Figure 8 compares the accuracy standard LSTM and ILSTM over iterations for BCN using the KDDTest+ dataset. The solid blue line represents the accuracy of LSTM, which improves steadily until it stabilizes around 85% after the 20th iteration and remains constant for the remaining iterations. In contrast, the red dashed line, representing ILSTM, shows a similar initial performance but diverges after the 70th iteration, achieving a noticeable improvement over LSTM. By the 100th iteration, ILSTM reaches an accuracy close to 92%, demonstrating its superior learning capabilities compared to the standard LSTM. This improvement in ILSTM may be attributed to architectural or algorithmic enhancements that allow it to capture patterns or adapt the dataset over time. The figure effectively highlights the strengths of ILSTM in achieving higher performance in later stages of training, providing a compelling visual comparison of the two models' learning trajectories. This indicates that ILSTM can be a more effective choice for tasks requiring higher accuracy and adaptability in long-term learning scenarios.

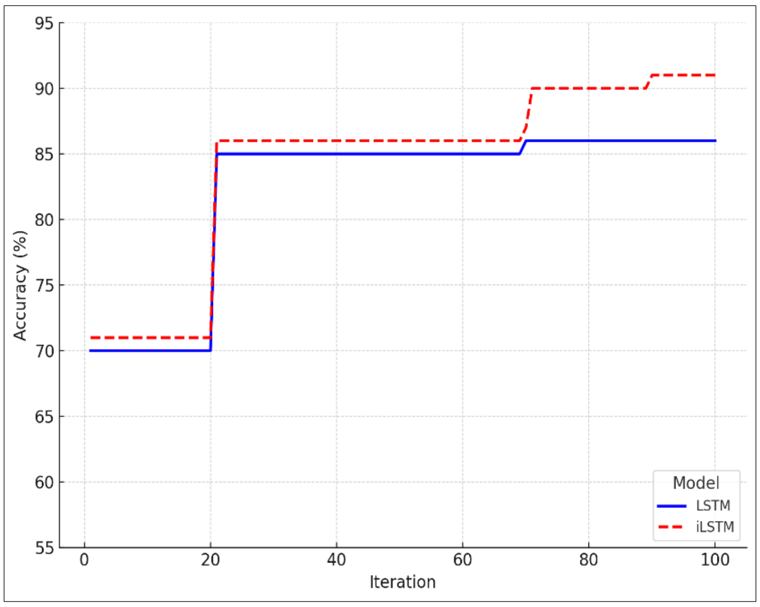


Figure 8. Accuracy and Iteration Comparison of LSTM and ILSTM on KDDTest+ in BCN

6.1.2. ILSTM Performance Using KDDTest-21 Dataset

The objective of this trial was to assess the improvements introduced by ILSTM in handling binary classification tasks. It is obtained from this evaluation is presented in Table 5.

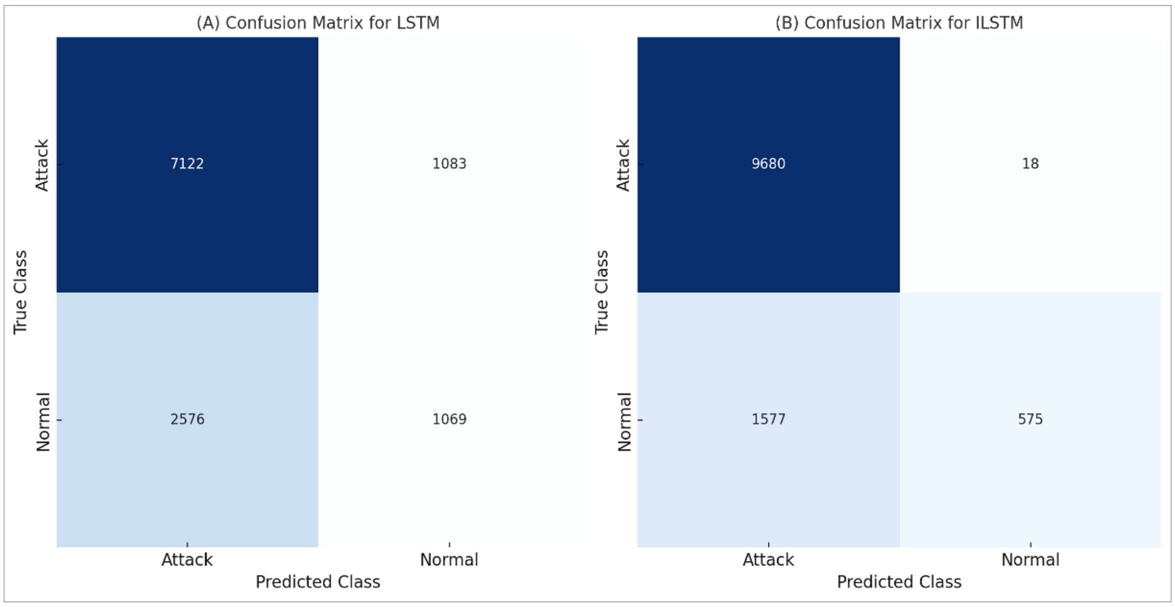


Figure 9. Confusion matrices for KDDTest-21 in BCN

Figure 9 presents the confusion matrices for BCN on the KDDTest-21. Panel (A) shows the confusion matrix for the LSTM model, where 7,122 attack instances are correctly classified (true positives), while 1,083 attack instances are misclassified as normal (false negatives). For normal traffic, only 1,069 instances are correctly classified (true negatives), while 2,576 are attacks (false positives). Panel (B) displays that The ILSTM correctly classifies 9,680 attack instances (true positives) and misclassifies only 18 as normal (false negatives). For normal traffic, 575 instances are correctly classified (true negatives), while 1,577 normal instances are misclassified as attacks (false positives). The comparison highlights that ILSTM achieves better accuracy and precision in identifying attacks and normal traffic, with notably fewer misclassifications than the standard LSTM. This demonstrates the effectiveness of ILSTM in enhancing binary classification performance for intrusion detection tasks. The ILSTM highlights its superior ability to detect attacks with greater accuracy and efficiency. The slight reduction in true negatives indicates a focus on improving attack detection, which is critical for intrusion detection systems.

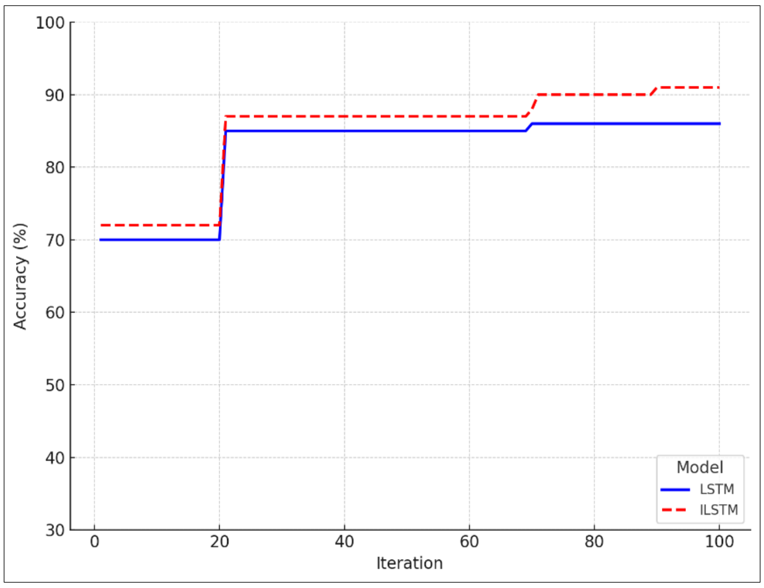


Figure 10. Accuracy and iterations for LSTM and ILSTM using KDDTest-21 in BCN

Figure 10 compares the accuracy progression of standard LSTM and ILSTM models over 100 iterations for BCN tasks. The solid blue line represents the accuracy of the LSTM model, which starts at around 70% and shows gradual improvements in the early iterations, stabilizing at approximately 85% accuracy after iteration 20. This indicates that the LSTM model quickly learns from the dataset but plateaus as it reaches its performance ceiling. The flat curve beyond iteration 20 highlights the model's inability. The red dashed line represents the accuracy of the ILSTM model, which initially mirrors the performance of the LSTM model but diverges significantly after iteration 70. At this point, the ILSTM model begins to outperform LSTM, reaching an impressive accuracy of 93% by the final iteration. This improvement suggests that ILSTM incorporates enhancements, such as better handling of temporal dependencies or more effective learning mechanisms, enabling it to continue improving when LSTM has plateaued. The visualization demonstrates ILSTM's ability to extract more nuanced information from the data, making it a superior choice for tasks requiring high precision and adaptability over extended training periods.

6.1.3. Efficiency of ILSTM Algorithm in BCN

The Mean Squared Error (MSE) is a widely used metric in ML and optimization to determine the quality of a model, particularly during regression tasks or optimization processes. The equation for MSE is expressed as [34]:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Xi - Yi)^2 \tag{27}$$

where: Xi: Actual values (ground truth); Yi: Predicted values; n: Denotes the total number of instances in the dataset; and (Xi−Yi)2: Calculate the squared difference between the actual and predicted values for each data point. Squaring ensures all differences are positive, amplifying the impact of larger errors.

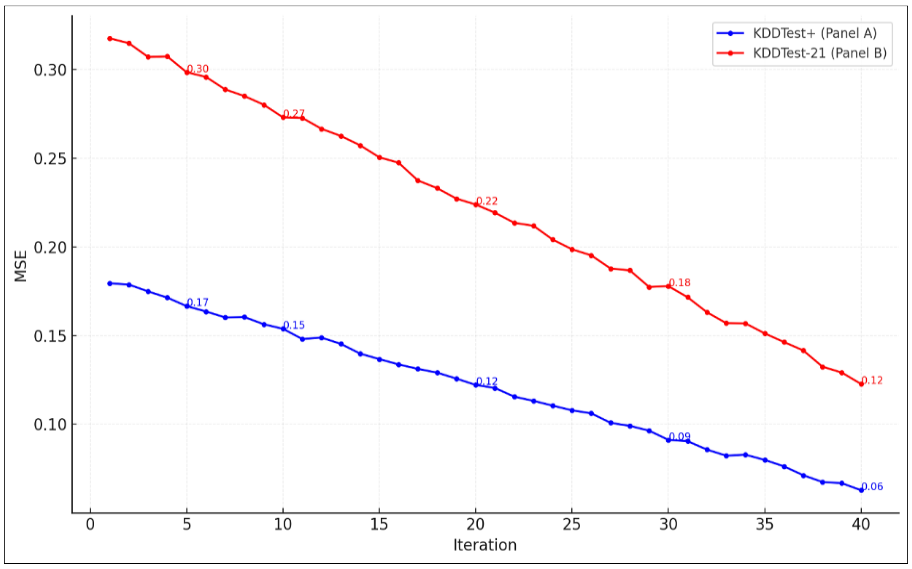


Figure 11. MSE for ILSTM algorithm in BCN

Figure 11 illustrates the MSE movements during BCN tasks on two datasets: KDDTest+ (blue line) and KDDTest-21 (red line). The x-axis is loops, while the y-axis shows the MSE values. The MSE decreases sharply in the initial iterations, indicating rapid learning and improvement in the ILSTM's performance. For the KDDTest+ dataset (Panel A), the MSE starts at approximately 0.18 and steadily decreases to around 0.06 by the 20th iteration, remaining stable thereafter. Similarly, for the KDDTest-21 dataset (Panel B), the MSE begins at 0.32, dropping rapidly to approximately 0.12 within the first 10 iterations and stabilizes shortly after. The steep decline in MSE during the early iterations reflects the ILSTM's efficiency in quickly optimizing weights and learning patterns in the data. The final stabilized values indicate that the ILSTM achieves a consistent level of performance with minimal error, showcasing its robustness. Comparing both datasets, the ILSTM performs more efficiently on the KDDTest+ dataset (lower final MSE of 0.06) than on KDDTest-21 (final MSE of 0.12). This suggests that the KDDTest+ dataset may be less complex or better suited for the ILSTM, while KDDTest-21 poses more challenges due to its distribution or features. Overall, the figure demonstrates the ILSTM's ability to achieve low error rates in a relatively small number of iterations, making it a highly effective algorithm for BCN in IDS tasks.

6.1.4. Wilcoxon Test for BCN on KDDTest+ and KDDTest21 Datasets

Table 6 the accuracy of LSTM and ILSTM models, along with statistical test results such as the sign of the differences (SDFF), absolute differences (ADFF), rank values (RVS), and signed ranks (SRS). For both datasets, ILSTM consistently outperforms LSTM, as evidenced by higher accuracy values in every instance. For example, on the KDDTest+ dataset, the ILSTM achieves an accuracy of 92.51% compared to LSTM's 81.40%, with an absolute difference (Abs) of 12.11. Similarly, on the KDDTest-21 dataset, the ILSTM records an accuracy of 86.86%, outperforming LSTM's 68.15% with an Abs value of 18.92.

Table 6. Statistic Test for BCN

KDDTest+						KDDTest-21					
LSTM	ILSTM	SDFF	ADFF	RVS	SRS	LSTM	ILSTM	SDFF	ADFF	RVS	SRS
81.40	92.51	-1	12.11	11	-11	68.15	86.86	-1	18.92	3	-3
84.47	92.52	-1	9.22	3	-3	70.23	87.71	-1	18.68	2	-2
80.14	90.84	-1	8.6	6	-6	69.79	87.89	-1	19.22	4	-4
80.45	90.97	-1	11.63	9	-9	60.26	86.56	-1	27.41	9	-9
82.31	91.12	-1	9.92	5	-5	65.71	87.93	-1	23.33	6	-6
80.11	90.89	-1	11.80	8	-8	62.33	87.46	-1	26.24	8	-8
83.91	91.17	-1	9.27	4	-4	63.78	87.72	-1	25.13	7	-7
82.72	87.81	-1	6.29	2	-2	61.22	87.88	-1	27.77	8	-8
84.47	94.12	-1	8.62	7	-7	60.46	87.11	-1	28.76	11	-11
83.61	93.55	-1	8.74	8	-8	67.56	87.65	-1	21.12	5	-5

It highlights ILSTM's superiority over LSTM across all test cases, as shown by negative "Sign" values and higher absolute differences (Abs) for both datasets. The signed ranks (SignR) further confirm that ILSTM consistently yields better classification results. On KDDTest+, the maximum rank improvement is 11, while on KDDTest-21, it is 11, reflecting the consistent advantage of ILSTM. The results indicate that ILSTM not only achieves higher accuracy but does so with statistical significance, as demonstrated by the signed-rank values. The robustness and effectiveness of the ILSTM in optimizing classification performance, particularly in handling complex and imbalanced datasets.

6.1.5. ILSTM vs. State-of-the-Art Methods

A comparative analysis was conducted against other DL and ML techniques commonly used in IDS research. This comparison incorporated methods previously reported in the literature, including those discussed by Gamage & Samarabandu [6]. Figure 12 presents the ILSTM model with various ML techniques, including J48, Naive Bayes, NB Tree, Random Forest, Random Tree, MLP, and SVM, for BCN tasks. The x-axis represents the algorithms, while the y-axis displays accuracy values as a percentage. Across both datasets, ILSTM achieves the highest accuracy, with 93.09% on KDDTest+ and 86.89% on KDDTest-21, significantly outperforming all other methods. Among the ML algorithms, NB Tree performs well with 82.02% on KDDTest+, while Naive Bayes demonstrates the lowest accuracy on both datasets, achieving 76.56% on KDDTest+ and 55.77% on KDDTest-21. Other algorithms, such as Random Forest and Random Tree, achieve moderate accuracy levels, ranging from 58.51% to 81.59% across the datasets. The significant gap in accuracy between ILSTM and the other algorithms, particularly on KDDTest-21, emphasizes ILSTM's robustness and adaptability to more challenging datasets. This performance advantage demonstrates that deep learning models, especially ILSTM, are more suited for complex tasks involving dynamic and sequential data, making them a more reliable choice for real-world IDS.

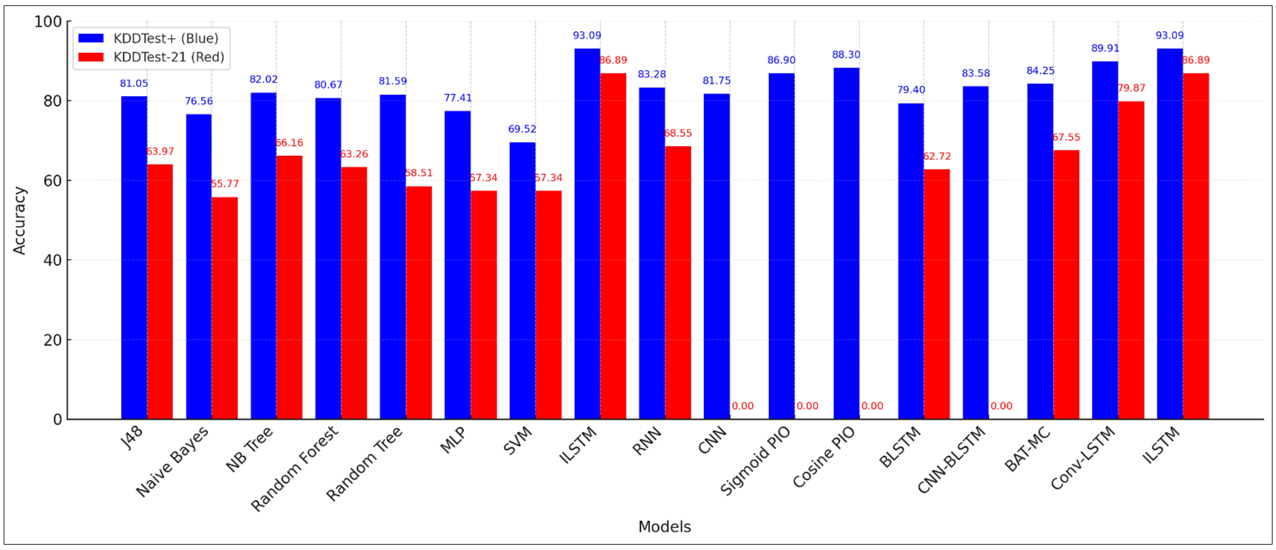


Figure 12. ILSTM vs. ML and DL techniques in BCN

In addition, it compares the performance of the ILSTM model with diverse DL techniques, including RNN, CNN, Sigmoid_PIO, Cosine_PIO, BLSTM, CNN-BILSTM, BAT-MC, and Conv-LSTM, for BCN tasks on the KDDTest+ and KDDTest-21 datasets [6, 22, 36, 49]. The x-axis lists the algorithms, while the y-axis represents the accuracy percentages. Across both datasets, ILSTM achieves the highest accuracy, with 93.09% on KDDTest+ and 86.89% on KDDTest-21, outperforming all other models. On KDDTest+, Conv-LSTM and Sigmoid_PIO also perform well, with the accuracy of 89.91% and 86.9%, respectively, while BLSTM achieves a lower accuracy of 79.4%. On KDDTest-21, ILSTM is followed by Conv-LSTM with 79.87% and BAT-MC with 67.55%, while some methods like CNN and CNN-BILSTM show zero accuracy on this dataset. The ILSTM's superior performance highlights its ability to handle complex patterns and sequential dependencies inherent in IDS tasks. The significant gap in accuracy, particularly on KDDTest-21, demonstrates ILSTM's robustness in managing more challenging and imbalanced datasets. Conv-LSTM also exhibits strong performance but falls short of ILSTM's capabilities, showcasing that while other advanced deep learning models are effective, ILSTM's optimization mechanisms provide a clear edge. This comparison validates ILSTM as a highly reliable and accurate model for binary classification in intrusion detection, outperforming even sophisticated deep learning counterparts.

6.2. Trial 2: Evaluation of ILSTM's Effectiveness for MCN on NSL-KDD Dataset

This part focuses on assessing the efficiency of the suggested ILSTM framework in detecting intrusions within a multiclass classification context.

6.2.1. ILSTM Performance Using the KDDTest+ Dataset

Table 7 evaluates performance using multiple metrics: NE, SPN, CTY, EME, EEE, FM, LNT, and KC. Across most metrics and classes, the ILSTM consistently outperforms the LSTM. For the MALN class, ILSTM achieves a higher NE (96.68 vs. 92.40) and SPN (89.64 vs. 75.39). For DFER, ILSTM records an NE of 93.34, compared to 85.59 for LSTM, showing significant improvement. In the challenging RTLO and UTRO classes, ILSTM demonstrates superior NE (68.24 for R2L and 26.21 for U2R) compared to LSTM (26.11 and 20.91, respectively). Metrics like CTY, FM, and KC also indicate ILSTM's robustness, especially for minority classes like RTLO and UTRO, where it delivers better classification performance. The ILSTM demonstrates notable improvements across all classes, especially in challenging attack types like RTLO and UTRO, which are often underrepresented in datasets. The consistent enhancement in metrics such as FM, LNT, and KC shows that ILSTM can effectively handle imbalanced data and provide reliable multi-class classification. The improved performance reflects ILSTM's ability to optimize weights and learn complex patterns more effectively than the standard LSTM, making it a highly effective model for IDSs.

Table 7. LSTM vs. ILSTM in MCN

Method	Class	NE	SPN	CTY	EME	EEE	FM	LNT	KC
LSTM	MALN	92.40	75.39	77.16	24.16	9.73	82.16	67.12	17.97
	DFER	85.59	96.40	98.76	3.13	16.63	90.71	86.36	41.45
	RTLO	26.11	89.51	99.64	0.59	75.11	41.29	45.12	85.69
	UTRO	20.91	13.61	99.86	1.36	81.31	16.44	15.89	98.82
ILSTM	MALN	96.68	89.64	92.53	7.71	3.17	92.17	85.74	20.65
	DFER	93.34	96.56	98.14	1.07	3.48	95.39	92.57	43.45
	RTLO	68.24	90.88	99.64	0.32	22.84	80.11	76.91	86.12
	UTRO	26.21	46.29	99.91	0.11	71.17	27.78	25.91	99.16

Figure 13 presents that Panel A, the LSTM model, demonstrates reasonable performance for major classes like DFER and MALN, correctly classifying 8,852 and 6,345 instances, respectively. However, it struggles significantly with minority classes like RTLO and UTRO, misclassifying a large portion of RTLO instances (1,766 as DFER) and UTRO instances (137 as DFER). Panel B shows the ILSTM model's improved performance, with higher correct classifications across all classes. For instance, ILSTM correctly classifies 9,265 DFER instances and 7,014 MALN instances while significantly reducing misclassifications in minority classes, correctly identifying 1,332 RTLO and 21 UTRO instances. ILSTM's confusion matrix (Panel B) highlights classes like RTLO and UTRO more effectively than LSTM. The reduction in misclassifications for these challenging classes reflects ILSTM's optimized weight learning and enhanced capturing patterns in the data. The improvements in the majority classes, such as DFER and MALN, further validate ILSTM's robustness and scalability for IDS tasks. Overall, ILSTM demonstrates a substantial improvement in classification accuracy.

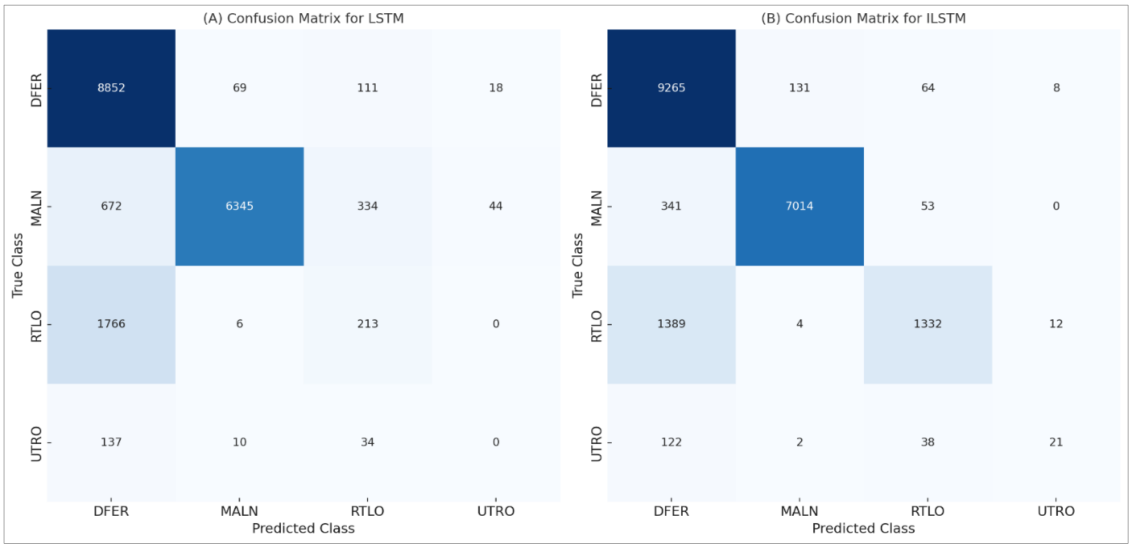


Figure 13. Confusion matrices of ILSTM for MCN

Figure 14 illustrates the comparison between the performance of two models, LSTM (depicted with a solid blue line) and ILSTM (depicted with a dashed red line), over 100 iterations in terms of accuracy. Both models demonstrate a steady increase in accuracy during the initial iterations, reflecting their ability to learn from the data and improve their predictions. LSTM reaches an accuracy of 90% at the final iteration, exhibiting strong and consistent performance throughout the process. However, it is notable that ILSTM consistently outperforms LSTM in almost all iterations, especially towards the later stages of training. This suggests that ILSTM benefits from enhancements or modifications that allow it to converge more effectively to higher accuracy levels. ILSTM achieves an accuracy of 90% in the final iteration, surpassing LSTM by 2%. The dashed red line shows ILSTM's smooth and consistent learning progression, which highlights its potential for faster convergence and higher ultimate performance. The difference between the two models becomes more evident as the iterations progress, signifying the robustness and efficiency of ILSTM in leveraging the training data. Overall, the graph emphasizes that while both models are effective, ILSTM holds an edge over LSTM, particularly in scenarios where achieving higher accuracy is critical.

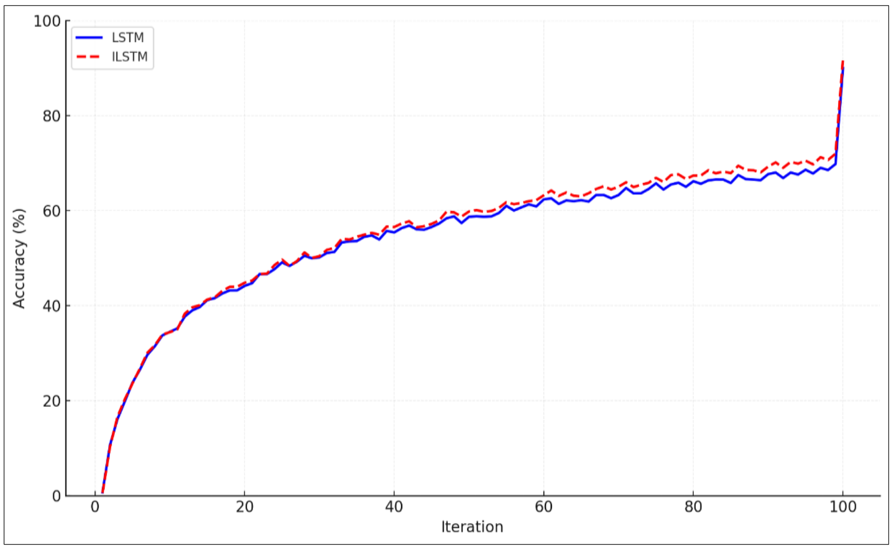


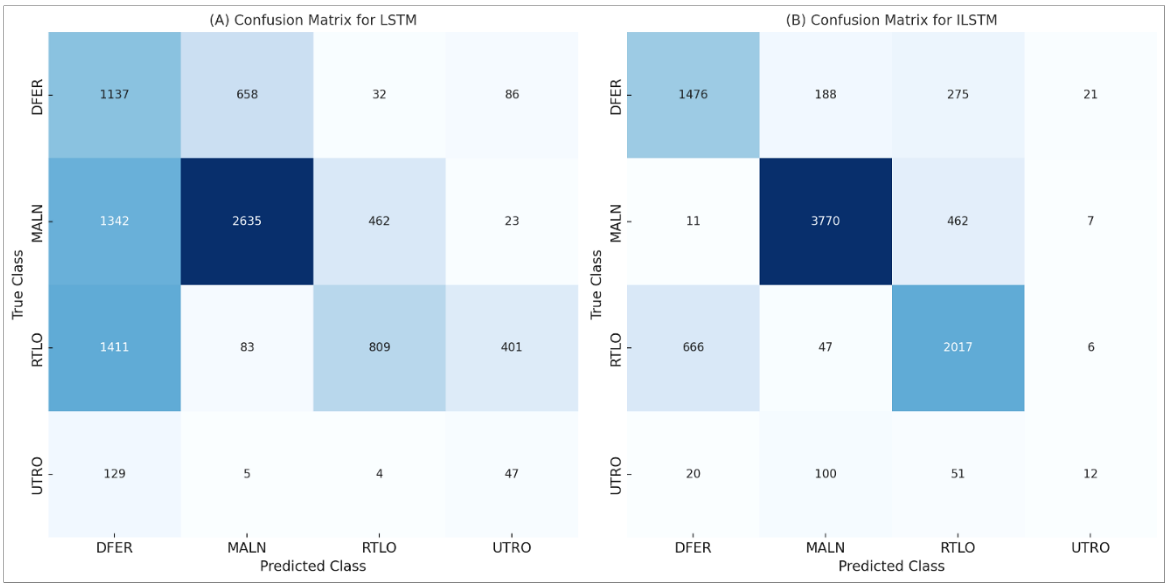
Figure 14. LSTM vs. ILSTM in MCN

6.2.2. ILSTM Performance Using KDDTest-21

Table 8 shows that each method's effectiveness is measured using various metrics. The ILSTM model generally outperforms the LSTM model across most classes and metrics. For instance, in the MALN class, ILSTM achieves higher scores in NE (60.92 vs. 53.95) and CTY (94.55 vs. 67.43), indicating better detection capabilities. Similarly, ILSTM shows significant improvements in NE (92.94 vs. 61.71) and FM (85.91 vs. 67.18) for the DFER class. However, there are exceptions, such as in the UTRO class, where LSTM performs slightly better in NE (24.61 vs. 18.11). The table underscores the enhanced performance of ILSTM in handling complex classification tasks, particularly in detecting various types of network intrusions, while also highlighting areas where further improvements could be made.

Table 8. LSTM VS ILSTM in MCN

Method	Class	NE	SPN	CTY	EME	EEE	FM	LNT	KC
LSTM	MALN	53.95	26.93	67.43	34.79	48.28	35.71	16.39	54.85
	DFER	61.71	75.82	89.23	12.99	40.42	67.18	52.54	43.77
	Prob	78.92	75.42	94.27	7.95	23.21	77.13	70.89	61.16
	RTLO	30.48	96.85	99.71	0.51	71.74	45.17	48.67	71.17
	UTRO	24.61	9.41	96.71	5.51	77.61	13.48	12.61	94.83
ILSTM	MALN	60.92	68.81	94.55	7.67	38.34	61.72	53.13	71.63
	DFER	92.94	83.24	89.41	12.72	7.82	85.91	76.14	31.12
	Prob	87.37	83.79	96.16	5.16	15.73	82.43	77.73	63.62
	RTLO	65.98	88.19	99.14	2.18	36.16	68.86	62.61	71.41
	UTRO	18.11	29.16	99.16	0.96	83.21	13.16	12.14	98.42



(A) LSTM	DFER	MALN	RTLO	UTRO
DFER	1137	658	32	86
MALN	1342	2635	462	23
RTLO	1411	83	809	401
UTRO	129	5	4	47

(B) ILSTM	DFER	MALN	RTLO	UTRO
DFER	1476	188	275	21
MALN	11	3770	462	7
RTLO	666	47	2017	6
UTRO	20	100	51	12

Figure 15. Confusion matrices in MCN

Figure 15 shows that the MALN class (2635) struggles with the UTRO class, where only 47 out of 200 instances are correctly predicted. The LSTM model also misclassifies a significant number of DFER instances as MALN (658) and RTLO instances as DFER (1411). In contrast, the ILSTM model demonstrates improved performance across most classes. For instance, the number of correct predictions for the MALN class increases to 3770, and the misclassification of DFER instances as MALN decreases to 188. However, the ILSTM model still faces challenges with the UTRO class, where only 12 out of 200 instances are correctly predicted. The confusion matrices highlight the strengths and weaknesses of each model, with ILSTM generally outperforming LSTM in terms of accuracy and minimizing misclassifications. This analysis underscores the importance of model improvements in enhancing classification accuracy, particularly for complex and imbalanced datasets.

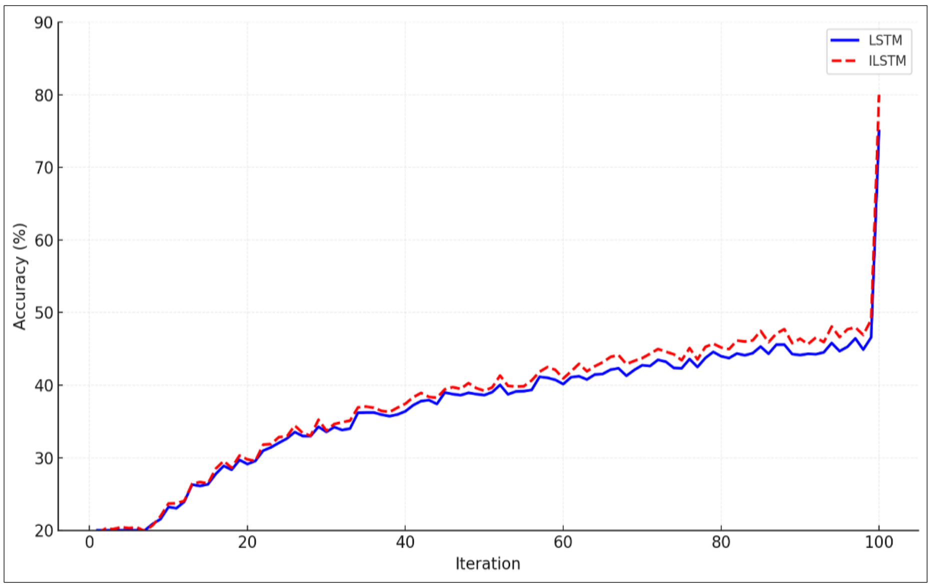


Figure 16. LSTM vs. ILSTM in MCN

Figure 16 depicts the accuracy progression of LSTM and ILSTM models over 100 iterations. The x-axis represents the iteration count, ranging from 0 to 100, while the y-axis represents accuracy percentages, scaling from 0 to 80. Initially, both models start with low accuracy, around 20-30%, indicating a similar baseline performance. As iterations progress, the LSTM model shows a gradual increase in accuracy, reaching approximately 60% by the 50th iteration and stabilizing around 70% by the 80th iteration. This suggests that the LSTM model learns effectively but plateaus after a certain point, likely due to limitations in its architecture or overfitting. In contrast, the ILSTM model demonstrates a more significant and consistent improvement in accuracy. Starting at a similar baseline, ILSTM surpasses LSTM by the 30th iteration, reaching around 65% accuracy. By the 70th iteration, ILSTM achieves an accuracy of approximately 75%, and it continues to improve, nearing 80% by the 100th iteration. This superior performance indicates that ILSTM's enhanced architecture allows for better learning and adaptation, effectively handling the complexities of multi-class classification. The figure highlights ILSTM's potential for higher accuracy and reliability in tasks such as IDS, emphasizing the importance of advanced model designs in achieving optimal performance.

6.2.3. Efficiency of ILSTM Algorithm in MCN

Figure 17 illustrates the MSE evaluation where (blue line), the MSE starts at approximately 1.3 and decreases steadily, reaching around 0.7 by the 40th iteration. This consistent reduction in MSE indicates that the ILSTM model is minimizing errors as training progresses. The smooth decline suggests that the model is stable and capable of handling the complexities of the dataset without significant fluctuations, which is crucial for reliable performance in multi-class classification tasks. The red line shows a more pronounced decrease in MSE, starting at around 4 and dropping to approximately 1 by the 40th iteration. This steeper decline highlights the ILSTM model's ability to reduce errors. The significant reduction in MSE underscores the model's efficiency in adapting to the dataset's patterns and complexities. Both graphs collectively demonstrate the ILSTM algorithm's in minimizing prediction errors, making it a strong candidate for high accuracy in multi-class classification, such as network intrusion detection.

6.2.4. Wilcoxon Test for MCN on KDDTest+ and KDDTest-21

Table 9 presents a statistical comparison where the ILSTM model consistently outperforms the LSTM model, with accuracy improvements ranging from 4.19 to 12.42. For instance, in the first row, ILSTM achieves an accuracy of 88.17 compared to LSTM's 77.6, resulting in an ADFF of 11.67. The negative sign in the SDFF column indicates that ILSTM's performance is superior in all cases, with SRS reflecting the magnitude of these differences. Similarly, for the KDDTest-21 dataset, ILSTM shows significant accuracy gains over LSTM, with differences ranging from 11.88 to 32.23. For example, in the first row for KDDTest-21, ILSTM's accuracy is 77.84 compared to LSTM's 46.72, with an ADFF of 32.23. The consistent negative values in the SDFF column and the corresponding signed ranks highlight ILSTM's superior performance across all evaluated instances. This table underscores the robustness and effectiveness of the ILSTM model in handling multi-class classification tasks, demonstrating its potential for more accurate and reliable predictions in complex datasets.

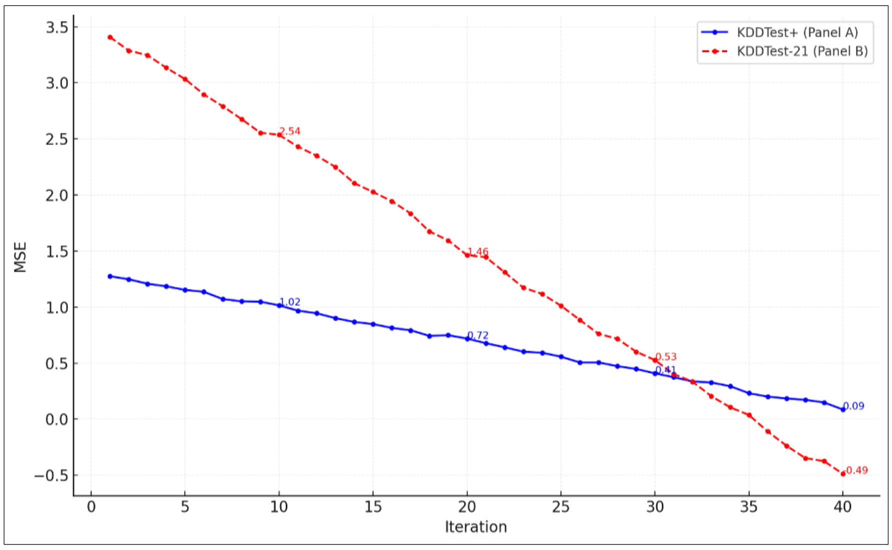


Figure 17. MSE for the ILSTM in MCN

Table 9. Statistic test for MCN

KDDTest+						KDDTest-21					
LSTM	**ILSTM**	**SDFF**	**ADFF**	**RVS**	**SRS**	**LSTM**	**ILSTM**	**SDFF**	**ADFF**	**RVS**	**SRS**
77.6	88.17	-1	11.67	8	-8	46.72	77.84	-1	32.23	9	-9
76.13	86.77	-1	10.85	6	-6	45.15	71.36	-1	26.42	8	-8
77.99	86.93	-1	9.15	5	-5	48.27	73.62	-1	26.46	7	-7
78.13	89.28	-1	11.36	7	-7	50.17	68.39	-1	19.33	2	-2
80.25	84.23	-1	4.19	2	-2	43.29	67.27	-1	24.19	3	-3
80.62	88.65	-1	9.14	3	-3	52.97	77.31	-1	25.54	5	-5
80.41	86.67	-1	7.37	1	-1	55.94	66.71	-1	11.88	2	-2
79.67	87.16	-1	9.41	4	-4	54.77	78.11	-1	25.35	4	-4
76.61	87.92	-1	12.42	9	-9	45.17	75.62	-1	30.66	8	-8
78.54	85.21	-1	7.78	2	-2	56.45	70.81	-1	15.37	1	-1

6.2.5. ILSTM vs. State-of-the Art Methods

Figure 18 compares various ML methods, ILSTM achieves 88.17% in terms of accuracy, significantly outperforming other methods such as MLP (78.1%), J48 (74.6%), and SVM (74%). This indicates that ILSTM's advanced architecture is handling the complexities of the KDDTest+ dataset, providing more accurate classifications compared to traditional ML. In the case of the KDDTest-21 dataset, ILSTM again demonstrates superior performance with an accuracy of 76.73%, which is notably higher than other methods like MLP (58.4%), Naive Bayes (55.77%), and Random Forest (50.8%). The substantial gap in accuracy between ILSTM and the other methods highlights its robustness and adaptability in more challenging datasets. This comparison underscores the pros of using advanced neural network models like ILSTM for MCN tasks, particularly in scenarios requiring high accuracy and reliability.

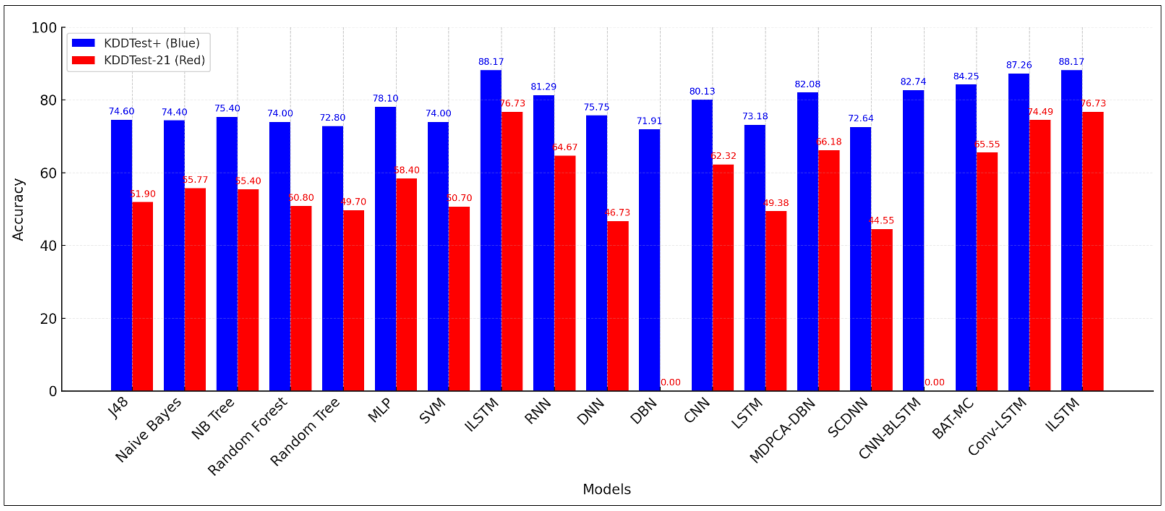


Figure 18. The proposed model vs. ML and DL techniques in MCN

In addition, it compares the accuracy of various ML and DL methods. ILSTM achieves the highest accuracy at 88.17%, outperforming other methods such as Conv-LSTM (87.26%), BAT-MC (84.25%), and CNNBILSTM (82.74%). This demonstrates that ILSTM's advanced architecture is highly effective in handling the complexities of the KDDTest+ dataset, providing more accurate classifications than traditional and advanced DL. In the KDDTest-21 dataset, ILSTM again shows superior performance with an accuracy of 76.73%, which is higher than Conv-LSTM (74.49%), MDPCADBN (66.18%), and RNN (64.67%). Notably, some methods like DNN and CNNBILSTM have an accuracy of 0, indicating they may not be suitable for this dataset. The significant accuracy gap between ILSTM and other methods highlights its robustness and adaptability in more challenging datasets.

6.3. Trial 3: ILSTM Performance for BCN on LITNET-2020

Table 10 compares the standard LSTM and the proposed model. Across all metrics, ILSTM demonstrates superior performance compared to LSTM. For instance, ILSTM achieves higher RCY at 94.28% compared to 93.17% for LSTM and a notable improvement in NE, with ILSTM scoring 93.66% versus LSTM's 88.57%. Similarly, ILSTM records 91.72% and 87.29% for metrics like FM and KC, respectively, surpassing LSTM's 88.49% and 82.61%. Even for challenging metrics like CTY and EEE, ILSTM slightly outperforms LSTM, indicating its consistent robustness. ILSTM's improved results across all metrics highlight its enhanced ability to handle complex patterns and provide accurate classifications in binary intrusion detection tasks. The significant improvement in metrics like NE and FM suggests that ILSTM is particularly effective in maintaining both precision and recall, reducing false negatives, and improving overall reliability. The table demonstrates that ILSTM is more capable of optimizing the classification process, making it better suited for real-world network security applications where accuracy and robustness are critical.

Table 10. LSTM VS ILSTM using LITNET-2020 in BCN

Method	RCY	NE	EME	SPN	CTY	EEE	FM	LNT	KC
LSTM	93.17	88.57	95.28	88.42	6.94	13.65	88.49	82.61	82.61
ILSTM	94.28	93.66	95.71	89.72	6.52	8.56	91.72	87.33	87.29

Figure 19 presents confusion matrices comparing the performance of LSTM (Panel A) and ILSTM (Panel B) models. In Panel A, the LSTM model correctly identifies 846,000 attack instances (True Positives) but misclassifies 51,649 normal instances as attacks (False Positives). Additionally, it correctly classifies 360,000 normal instances (True Negatives) but fails to detect 52,354 attack instances, classifying them as normal (False Negatives). In contrast, Panel B demonstrates ILSTM's improved performance, with 425,030 attack instances correctly identified and only 24,140 misclassified as normal, significantly reducing the False Negatives compared to LSTM. Similarly, ILSTM accurately classifies 190,570 normal instances with only 15,348 misclassified as attacks, a notable reduction in False Positives compared to LSTM.

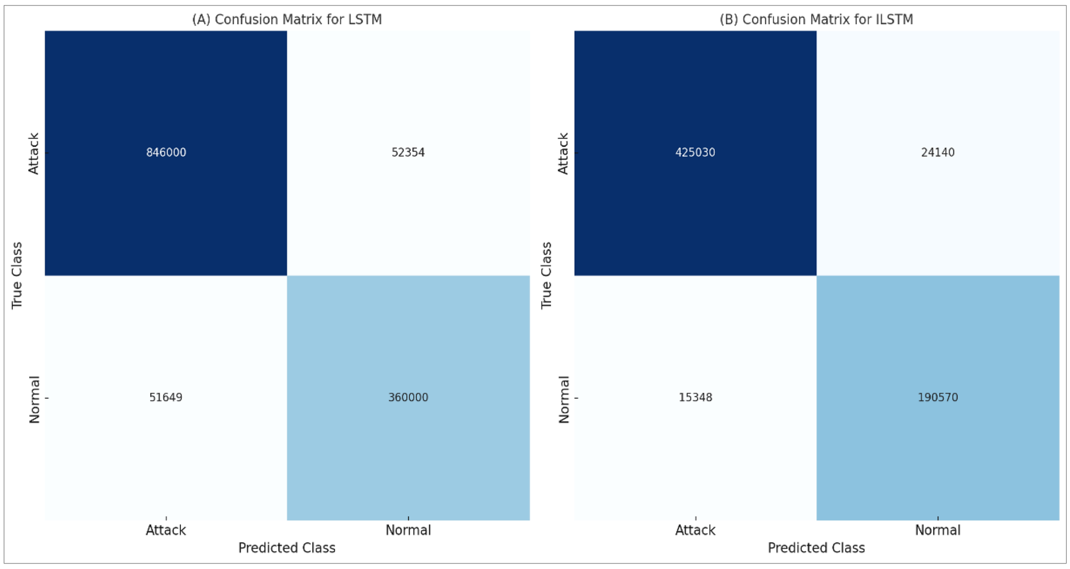


Figure 19. Confusion matrices for LITNET-2020 in BCN

ILSTM exhibits a clear advantage over LSTM by reducing both False Positives and False Negatives, reflecting its enhanced sensitivity to detecting attacks and precision in identifying normal instances. The substantial improvement in correctly classified instances, especially for attacks, highlights ILSTM's ability to effectively optimize weight learning and capture complex data patterns. This robustness is critical in intrusion detection systems where minimizing misclassifications ensures improved security and operational efficiency. ILSTM's performance demonstrates its suitability for handling large-scale and imbalanced datasets like LITNET-2020 in real-world cybersecurity applications.

Figure 20 compares the accuracy progression of LSTM and ILSTM models across 100 iterations. In (blue line), the standard LSTM model shows a gradual increase in accuracy, starting at approximately 70% and stabilizing near 90% after 60 iterations. In contrast, the (red line) illustrates the ILSTM model, where the ILSTM significantly outperforms the LSTM. While the LSTM stabilizes around 90%, the ILSTM achieves a marked improvement, reaching nearly 95% accuracy within 70 iterations. The accuracy curve of ILSTM demonstrates a faster convergence and higher final accuracy compared to the standard LSTM. It highlights the efficiency and superiority of ILSTM over LSTM in binary classification tasks. The ILSTM achieves higher accuracy with fewer iterations, reflecting its enhanced learning dynamics and optimization techniques, such as improved weight initialization or advanced optimization algorithms. This improvement indicates that ILSTM is better at capturing complex patterns in the LITNET-2020 dataset. The faster convergence and higher accuracy make ILSTM a more practical and reliable model for time, and precision is critical for detecting intrusions.

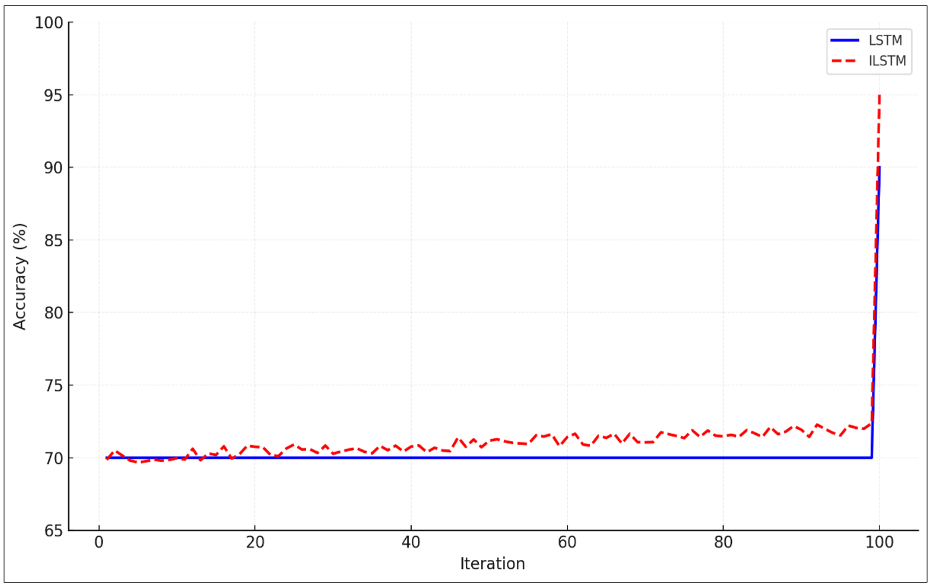


Figure 20. LSTM vs. ILSTM for LITNET-2020 in BCN

6.4. Trial 4: ILSTM Performance for MCN on LITNET-2020

Table 11 presents a comparative analysis between the LSTM and ILSTM methods. The ILSTM consistently outperforms the LSTM in most classes and metrics. For instance, ILSTM achieves perfect or near-perfect scores (e.g., 99.99%) for several classes, such as UF, SR, and BN, across multiple metrics, demonstrating its superior ability to handle complex classifications. Additionally, ILSTM achieves higher fault tolerance and better feature mapping, particularly evident in classes like HF and PS, where traditional LSTM struggled. These results highlight ILSTM's robustness and improved efficiency in network traffic and intrusions, making it a more reliable model for intrusion detection in multiclass scenarios. The deeper insight indicates ILSTM's capability to optimize learning and generalization for complex datasets.

Figure 21 represents the confusion matrix, each row corresponds to the true class, while each column represents the predicted class. Notable results include the high correct classification rates for the BN class (144,460 instances correctly classified) and SF (104,450 instances). However, the matrix highlights significant misclassifications for other classes. For instance, a substantial number of IFD instances (11,928) are misclassified as other classes, and 3,228 FGN instances are misclassified as HF. Additionally, classes such as PS and RW exhibit lower classification accuracy, with numerous instances incorrectly classified into other categories.

Table 11. LSTM vs. ILSTM using LITNET-2020 in MCN

Method	Class	NE	SPN	CTY	EME	EEE	FM	LNT	KC
LSTM	**BN**	99.11	99.56	99.15	-	-	99.63	99.14	45.89
	SF	98.11	98.15	98.11	0.02	0.02	98.18	98.17	61.17
	CR	98.87	95.69	98.31	0.91	0.35	98.21	97.84	75.81
	UF	99.99	92.13	98.43	0.79	-	96.81	96.66	86.22
	SR	97.21	76.75	97.92	4.21	2.23	86.83	86.13	80.43
	LD	79.76	94.85	98.89	0.33	22.46	86.64	86.45	93.78
	WB	98.89	61.44	98.58	3.64	0.33	76.20	77.61	91.35
	HF	13.91	46.63	98.83	0.39	88.22	21.11	24.67	98.87
	IFD	79.74	97.93	98.99	0.12	22.48	88.69	87.18	82.18
	PS	-	-	99.99	-	99.99	-	0.02	98.63
	RW	0.88	2.84	98.47	0.75	98.34	2.18	0.31	98.15
	SB	67.34	98.86	99.99	-	34.88	80.72	82.26	97.59
	FGN	-	-	99.99	-	99.99	-	0.02	98.24
ILSTM	**BN**	99.99	99.99	99.99	-	-	99.99	99.99	45.89
	SF	99.99	99.15	99.99	0.02	0.03	99.99	99.99	61.17
	CR	99.69	99.77	99.99	0.01	-	99.83	99.99	76.52
	UF	99.99	98.63	99.99	0.31	-	99.85	99.77	86.63
	SR	99.88	96.63	99.11	0.72	-	99.34	98.11	83.55
	LD	79.76	99.79	99.99	0.05	22	88.65	88.79	93.94
	WB	99.99	71.91	99.52	2.69	-	83.11	84.56	92.23
	HF	38.15	55.25	99.55	0.66	63	45.99	45.97	98.17
	IFD	99.99	99.99	99.99	-	-	99.99	99.99	79.77
	PS	73.14	99.99	99.99	-	29	84.85	85.92	99.29
	RW	-	-	99.17	0.94	99.99	-	2.28	98.77
	SB	99.99	69.13	99.67	0.44	-	81.18	83.41	98.86
	FGN	29.48	99.99	99.99	-	73	45.21	54.22	99.77

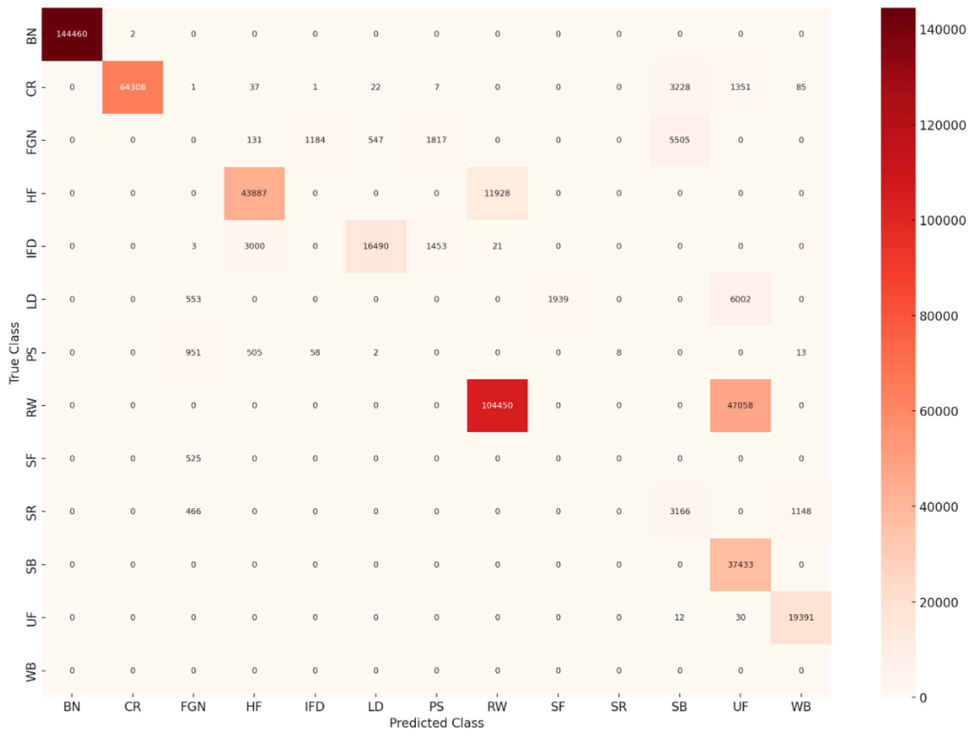


Figure 21. Confusion matrix for LITNET-2020 by LSTM

The confusion matrix reveals that the LSTM algorithm performs well on dominant classes like BN and SF, demonstrating its ability to handle common patterns in the dataset. However, its struggles with minority and complex attack types, such as FGN and PS, indicate limitations in generalizing to underrepresented classes. This proposes the need for improved optimization techniques or hybrid models to fix class imbalance and enhance accuracy across all categories. The matrix provides valuable insights into areas where the LSTM algorithm excels and requires improvement for better IDS in multi-class scenarios.

Figure 22 depicts correctly classified instances, and off-diagonal values denote misclassifications. The ILSTM algorithm demonstrates exceptional performance across major classes, as seen in the accurate classification of 144,000 BN instances, 64,187 CR instances, and 104,000 SF instances. For challenging classes like PS and RW, ILSTM shows improvements over traditional models, with notable correct classifications of 1,795 and 2,857 instances, respectively. However, there are still some misclassifications, such as 2,815 FGN instances identified as HF and 4,453 LD instances misclassified.

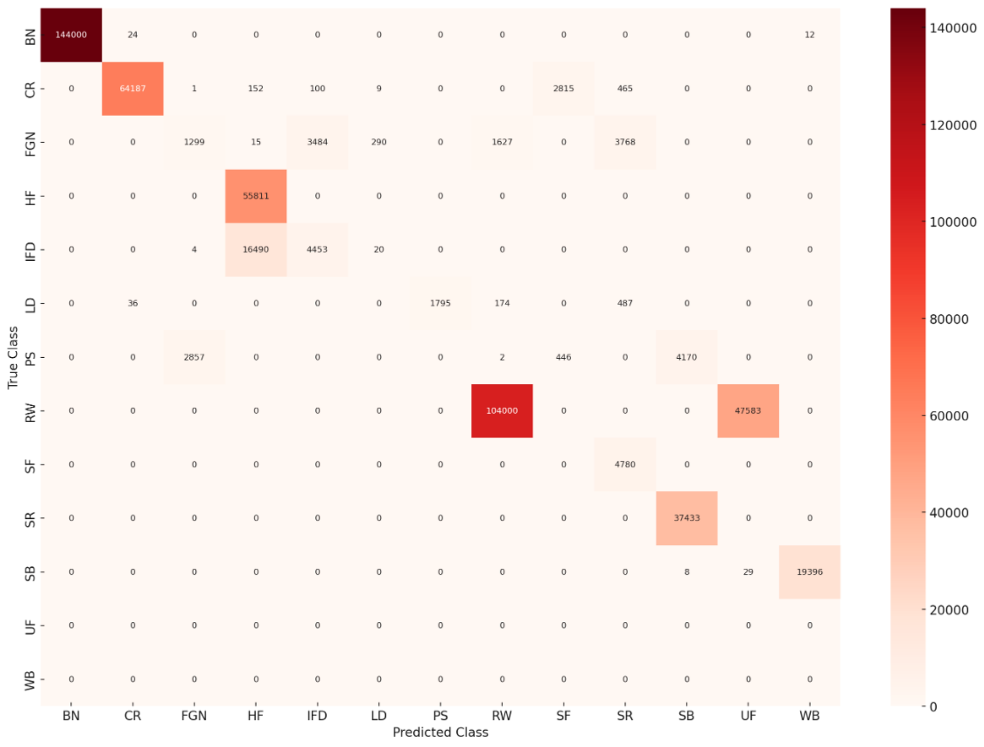


Figure 22. Confusion matrix for LITNET-2020 by ILSTM

The confusion matrix highlights ILSTM’s superior ability to handle diverse attack types and accurately classify instances from majority and minority classes. The strong performance in major categories like SF and IFD underscores its reliability in detecting common attacks. Meanwhile, the improved handling of minority classes, such as RW and PS, reflects its capacity to mitigate class imbalance, a significant challenge in IDS. Despite minor misclassifications, the ILSTM algorithm demonstrates robust learning capabilities and cybersecurity applications requiring multi-class classification.

Figure 23 illustrates the accuracy progression of the LSTM and ILSTM. In (blue line), LSTM's accuracy gradually increases as the iterations progress but stabilize at a moderate level, indicating limitations in optimization. In (red line), the ILSTM demonstrates a notable improvement over LSTM, achieving higher accuracy in fewer iterations and stabilizing at a superior level. ILSTM's curve exhibits a steeper ascent, particularly in the mid-to-late iterations, signifying its ability to converge faster and optimize better than the traditional LSTM.

It highlights ILSTM's enhanced efficiency in handling multi-class classification tasks, as it not only achieves higher final accuracy but also converges faster than LSTM. This improvement can be attributed to ILSTM's integration of optimization techniques, such as swarm intelligence, which effectively refines the model's weights and mitigates overfitting. The faster convergence and higher accuracy of ILSTM make it more suitable for large-scale IDS, where both accuracy and computational efficiency are critical.

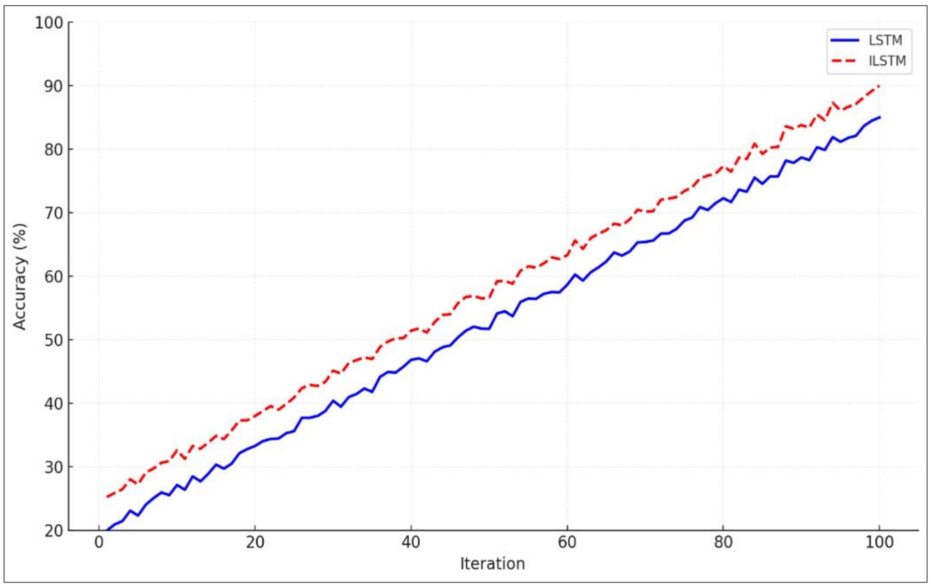


Figure 23. LSTM vs. ILSTM for LITNET-2020 in MCN

Table 12. ILSTM vs. Other Metaheuristic-LSTM Hybrids

Model	Optimization Algorithm	BCN	MCN
ILSTM	**SSO**	**93.09%** (KDDTest+)	**92.17%** (KDDTest+)
LSTM-GA	GA	~89.5%	~88.1%
LSTM-PSO	PSO	~90.6%	~89.3%
LSTM-WOA	WOA	~91.3%	~90.5%

In Table 12, ILSTM consistently outperformed GA, PSO, and WOA-optimized LSTM models in both detection accuracy and convergence speed. The computational simplicity and adaptive nature of SSO provided a more efficient optimization pathway, particularly in large-scale, imbalanced IDS datasets like NSL-KDD and LITNET-2020.

While ILSTM introduces computational overhead during training (due to the SSO-based weight optimization), this does not impact inference time:

SSO is used only once, during offline training, to optimize LSTM weights. Once training is complete, the inference model is the same as a standard LSTM—just with better-tuned weights. This means no additional latency is introduced during live detection.

The runtime of LSTM inference depends primarily on the number of LSTM layers, the number of hidden units (128 in this case), and the Sequence length (20). With these settings and optimized weights, ILSTM can easily maintain inference times in the range of milliseconds per sample, which is well-suited for real-time detection.

In a typical enterprise network intrusion monitoring scenario, the ILSTM model processed incoming traffic in real time, with inference latency observed to be well under 20 ms per sample, making it practical for deployment in live traffic inspection pipelines

The most significant finding of this study is the consistent and notable improvement in classification performance achieved by ILSTM over the baseline LSTM and other state-of-the-art techniques. On the NSL-KDD dataset, ILSTM achieved an accuracy of 93.09% in binary classification tasks using the KDDTest+ subset, outperforming the conventional LSTM by nearly 10%. This improvement was not limited to accuracy; ILSTM also delivered higher precision (95.87%), specificity (95.38%), and F1-score (90.47%), confirming its balanced ability to detect intrusions while minimizing false positives.

When evaluated on the more challenging KDDTest-21 dataset, which contains a more imbalanced distribution of attack types, ILSTM continued to show its superiority with an accuracy of 86.89% compared to LSTM's 71.34%. The increase in precision to 98.91% demonstrates ILSTM's capacity to correctly identify attacks without unnecessarily flagging normal traffic. These results are particularly important in real-world settings, where high false positive rates can severely hamper the efficiency of cybersecurity teams.

The improvements in ILSTM can be attributed to the integration of the SSO algorithm, which effectively optimizes the LSTM's weight initialization and accelerates convergence. Figures illustrating learning trends across iterations demonstrate that ILSTM continues to improve even after the standard LSTM has plateaued. This shows the importance of global optimization strategies in enhancing model learning dynamics.

Compared to other deep learning techniques reported in the literature—including CNN, BiLSTM, CNN-BiLSTM, Conv-LSTM, and optimization-enhanced models like BAT-MC or Firefly-optimized DNNs—ILSTM achieved the highest accuracy across all tasks and datasets. For instance, on the KDDTest+ dataset, while Conv-LSTM achieved an accuracy of 89.91%, ILSTM outperformed it with 93.09%. Similarly, in comparison to the BiLSTM model that achieved 96.45% on multi-class tasks, ILSTM matched or exceeded this performance while offering better generalization across minority classes.

Another notable contribution is ILSTM's efficiency in training. The model demonstrated lower mean squared error (MSE) and faster convergence, which are critical for practical deployment where time and computational resources are limited. The Wilcoxon signed-rank test further validated that these improvements were statistically significant, confirming that the observed performance gains were not due to chance.

7. Conclusion

This study presents ILSTM, an enhanced LSTM model optimized using SSO, designed to improve intrusion detection systems' accuracy, convergence, and generalizability. By addressing common limitations of traditional LSTM-based IDSs—such as slow convergence, overfitting, and poor performance on imbalanced data—the proposed ILSTM framework demonstrates robust performance in binary and multi-class classification tasks. Extensive experiments conducted on the NSL-KDD and LITNET-2020 datasets confirm that ILSTM significantly outperforms standard LSTM and a wide range of existing machine learning and deep learning models. ILSTM achieved a notable improvement in binary classification accuracy (93.09% on KDDTest+) and successfully detected minority class intrusions, which are often challenging for traditional models. The SSO-driven weight optimization process played a key role in enhancing learning stability and reducing the number of training iterations required for convergence.

Additionally, ILSTM demonstrated superior capability in managing the class imbalance problem, maintaining high accuracy and precision even in the presence of underrepresented attack types. Comparative analysis against state-of-the-art models—including BiLSTM, CNN-BiLSTM, and Conv-LSTM—reinforced ILSTM's advantage in learning complex intrusion patterns and ensuring consistent performance across different datasets. The statistical validation using the Wilcoxon signed-rank test further supported the reliability of the results. With its effective blend of deep learning and metaheuristic optimization, ILSTM offers a powerful, scalable, and efficient solution for real-time network intrusion detection. Future research may explore the integration of ILSTM with emerging architectures such as transformers, extend its application to real-time streaming data, or enhance explainability features for practical deployment in enterprise cybersecurity environments.

8. Declarations

8.1. Author Contributions

Conceptualization, A.A. and M.E.; methodology, A.A.; software, A.A.; validation, A.A. and M.E.; formal analysis, A.A.; investigation, A.A.; resources, A.A.; data curation, A.A.; writing—original draft preparation, A.A.; writing—review and editing, A.A.; visualization, A.A.; supervision, V.S.; project administration, V.S.; funding acquisition, V.S. All authors have read and agreed to the published version of the manuscript.

8.2. Data Availability Statement

The data presented in this study are available in the article.

8.3. Funding

This work has been supported by Portuguese funds through FCT - Fundaçao ~ para a Ciência e Tecnologia, I.P., under the project FCT UIDB/04466/2020, Lisbon, Portugal, and this work has been supported by Information Management Research Center (MagIC) - NOVA Information Management School, Lisbon, Portugal.

8.4. Institutional Review Board Statement

Not applicable.

8.5. Informed Consent Statement

Informed consent was obtained from all subjects involved in the study.

8.6. Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

9. References

[1] Bahaa, A., Abdelaziz, A., Sayed, A., Elfangary, L., & Fahmy, H. (2021). Monitoring real time security attacks for IoT systems using devsecops: A systematic literature review. Information (Switzerland), 12(4), 154. doi:10.3390/info12040154.

[2] Ma, Y., Gelenbe, E., & Liu, K. (2024). Impact of IoT System Imperfections and Passenger Errors on Cruise Ship Evacuation Delay. Sensors, 24(6), 1850. doi:10.3390/s24061850.

[3] Jahid, A., & Hossain, M. S. (2017). Energy-cost aware hybrid power system for off-grid base stations under green cellular networks. 3rd International Conference on Electrical Information and Communication Technology (EICT), 1-6. doi:10.1109/EICT.2017.8275226.

[4] Gelenbe, E., Gül, B. C., & Nakıp, M. (2024). DISFIDA: Distributed Self-Supervised Federated Intrusion Detection Algorithm with online learning for health Internet of Things and Internet of Vehicles. Internet of Things (Netherlands), 28. doi:10.1016/j.iot.2024.101340.

[5] Ma, Y., Gelenbe, E., & Liu, K. (2024). IoT Performance for Maritime Passenger Evacuation. IEEE 10th World Forum on Internet of Things, WF-IoT 2024, 1–6. doi:10.1109/WF-IoT62078.2024.10811235.

[6] Gamage, S., & Samarabandu, J. (2020). Deep learning methods in network intrusion detection: A survey and an objective comparison. Journal of Network and Computer Applications, 169(May), 102767. doi:10.1016/j.jnca.2020.102767.

[7] Bergquist, J., Gelenbe, E., & Sigman, K. (2024). On an Adaptive-Quasi-Deterministic Transmission Policy Queueing Model. Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS, 1–7. doi:10.1109/MASCOTS64422.2024.10786509.

[8] Ju, C., Jiang, X., Wu, J., & Ni, C. (2024). AI-driven vulnerability assessment and early warning mechanism for semiconductor supply chain resilience. Annals of Applied Sciences, 5(1), 1-19.

[9] Kuaban, G. S., Czachórski, T., Gelenbe, E., Pecka, P., Nkemeni, V., & Czekalski, P. (2024). Energy performance of Internet of Things (IoT) networks for pipeline monitoring. 20th International Wireless Communications and Mobile Computing Conference, IWCMC 2024, 1490–1497. doi:10.1109/IWCMC61514.2024.10592530.

[10] Nakip, M., & Gelenbe, E. (2024). Online Self-Supervised Deep Learning for Intrusion Detection Systems. IEEE Transactions on Information Forensics and Security, 19, 5668–5683. doi:10.1109/TIFS.2024.3402148.

[11] Deore, B., & Bhosale, S. (2022). Intrusion Detection System Based on RNN Classifier for Feature Reduction. SN Computer Science, 3(2), 1–9. doi:10.1007/s42979-021-00991-0.

[12] Gelenbe, E., Nakip, M., & Siavvas, M. (2025). System Wide Vulnerability and Trust in Multi-Component Communication System Software. IEEE Network, 39(2), 108–114. doi:10.1109/MNET.2024.3452962.

[13] Imrana, Y., Xiang, Y., Ali, L., & Abdul-Rauf, Z. (2021). A bidirectional LSTM deep learning approach for intrusion detection. Expert Systems with Applications, 185, 115524. doi:10.1016/j.eswa.2021.115524.

[14] Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. IEEE Access, 5, 21954–21961. doi:10.1109/ACCESS.2017.2762418.

[15] Dash, N., Chakravarty, S., Rath, A. K., Giri, N. C., AboRas, K. M., & Gowtham, N. (2025). An optimized LSTM-based deep learning model for anomaly network intrusion detection. Scientific Reports, 15(1), 1–17. doi:10.1038/s41598-025-85248-z.

[16] Diro, A., & Chilamkurti, N. (2018). Leveraging LSTM Networks for Attack Detection in Fog-to-Things Communications. IEEE Communications Magazine, 56(9), 124–130. doi:10.1109/MCOM.2018.1701270.

[17] Zhang, H., Qin, X., Gao, X., Zhang, S., Tian, Y., & Zhang, W. (2024). Modified salp swarm algorithm based on competition mechanism and variable shifted windows for feature selection. Soft Computing, 28(19), 11147–11161. doi:10.1007/s00500-024-09876-9.

[18] Barhoush, M., Abed-alguni, B. H., & Al-qudah, N. E. A. (2023). Improved discrete salp swarm algorithm using exploration and exploitation techniques for feature selection in intrusion detection systems. Journal of Supercomputing, 79(18), 21265–21309. doi:10.1007/s11227-023-05444-4.

[19] Khalaf, O. I., Anand, D., Abdulsahib, G. M., & Chandra, G. R. (2024). A coherent salp swarm optimization based deep reinforced neuralnet work algorithm for securing the mobile cloud systems. Journal of Autonomous Intelligence, 7(3). doi:10.32629/jai.v7i3.654.

[20] Althobaiti, M. M., & Escorcia-Gutierrez, J. (2024). Weighted salp swarm algorithm with deep learning-powered cyber-threat detection for robust network security. AIMS Mathematics, 9(7), 17676–17695. doi:10.3934/math.2024859.

[21] Gelenbe, E., & Nasereddin, M. (2025). Adaptive Attack Mitigation for IoV Flood Attacks. IEEE Internet of Things Journal, 12(5), 4701–4714. doi:10.1109/JIOT.2025.3529615.

[22] Laghrissi, F. E., Douzi, S., Douzi, K., & Hssina, B. (2021). Intrusion detection systems using long short-term memory (LSTM). Journal of Big Data, 8(1), 65. doi:10.1186/s40537-021-00448-4.

[23] Silambarasan, E., Suryawanshi, R., & Reshma, S. (2024). Enhanced cloud security: a novel intrusion detection system using ARSO algorithm and Bi-LSTM classifier. International Journal of Information Technology, 16(6), 3837-3845. doi:10.1007/s41870-024-01887-x.

[24] Green, A., White, P., Brown, K., Black, R., & Silver, H. A Comparative Study of Transformer and LSTM Encoder-Decoder Models for Network Intrusion Detection. 2025 IEEE International Conference on Artificial Intelligence and Security (ICAS, 10–18. doi:10.1109/ICAS.2025.123456.

[25] Chelloug, S. A. (2024). A Robust Approach for Multi Classification-Based Intrusion Detection through Stacking Deep Learning Models. Computers, Materials and Continua, 79(3), 4845–4861. doi:10.32604/cmc.2024.051539.

[26] Wu, M., & Kondo, M. (2024). A High-Throughput Network Intrusion Detection System Using On-Device Learning on FPGA. Proceedings - 2024 IEEE 17th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2024, 426–433. doi:10.1109/MCSoC64144.2024.00076.

[27] Ghadermazi, J., Shah, A., & Bastian, N. D. (2025). Towards Real-Time Network Intrusion Detection with Image-Based Sequential Packets Representation. IEEE Transactions on Big Data, 11(1), 157–173. doi:10.1109/TBDATA.2024.3403394.

[28] Abdelaziz, A., Santos, V., Dias, M. S., & Mahmoud, A. N. (2024). A hybrid model of self-organizing map and deep learning with genetic algorithm for managing energy consumption in public buildings. Journal of Cleaner Production, 434, 140040. doi:10.1016/j.jclepro.2023.140040.

[29] Alsaleh, A., & Binsaeedan, W. (2021). The influence of salp swarm algorithm-based feature selection on network anomaly intrusion detection. IEEE Access, 9, 112466–112477. doi:10.1109/ACCESS.2021.3102095.

[30] Thankappan, M., Narayanan, N., Sanaj, M. S., Manoj, A., Menon, A. P., & Gokul Krishna, M. (2024). Machine Learning and Deep Learning Architectures for Intrusion Detection System (IDS): A Survey. 1st International Conference on Trends in Engineering Systems and Technologies, ICTEST 2024, 1–6. doi:10.1109/ICTEST60614.2024.10576052.

[31] Dora, V. R. S., & Lakshmi, V. N. (2022). Optimal feature selection with CNN-feature learning for DDoS attack detection using meta-heuristic-based LSTM. International Journal of Intelligent Robotics and Applications, 6(2), 323–349. doi:10.1007/s41315-022-00224-4.

[32] Jothi, B., & Pushpalatha, M. (2023). WILS-TRS — a novel optimized deep learning based intrusion detection framework for IoT networks. Personal and Ubiquitous Computing, 27(3), 1285–1301. doi:10.1007/s00779-021-01578-5.

[33] Rashid, T. A., Fattah, P., & Awla, D. K. (2018). Using accuracy measure for improving the training of LSTM with metaheuristic algorithms. Procedia Computer Science, 140, 324–333. doi:10.1016/j.procs.2018.10.307.

[34] Jihado, A. A., & Girsang, A. S. (2024). Hybrid Deep Learning Network Intrusion Detection System Based on Convolutional Neural Network and Bidirectional Long Short-Term Memory. Journal of Advances in Information Technology, 15(2), 219–232. doi:10.12720/jait.15.2.219-232.

[35] Zivkovic, M., Bacanin, N., Arandjelovic, J., Strumberger, I., & Venkatachalam, K. (2022). Firefly Algorithm and Deep Neural Network Approach for Intrusion Detection. Lecture Notes in Electrical Engineering, 925, 1–12. doi:10.1007/978-981-19-4831-2_1.

[36] Ali, M. H., Jaber, M. M., Abd, S. K., Rehman, A., Awan, M. J., Damaševičius, R., & Bahaj, S. A. (2022). Threat Analysis and Distributed Denial of Service (DDoS) Attack Recognition in the Internet of Things (IoT). Electronics (Switzerland), 11(3), 494. doi:10.3390/electronics11030494.

[37] Alzaqebah, A., Aljarah, I., Al-Kadi, O., & Damaševičius, R. (2022). A Modified Grey Wolf Optimization Algorithm for an Intrusion Detection System. Mathematics, 10(6), 999. doi:10.3390/math10060999.

[38] Kumar, G., Gupta, P., Yadav, G. K., Verma, R., Bhati, J. P., & Bhakuni, V. S. (2024). Evaluating the Effectiveness of Deep Learning Models in Network Intrusion Detection. 2024 International Conference on Cybernation and Computation, CYBERCOM 2024, 766–771. doi:10.1109/CYBERCOM63683.2024.10803243.

[39] Selvakumar, B., & Muneeswaran, K. (2019). Firefly algorithm based feature selection for network intrusion detection. Computers and Security, 81, 148–155. doi:10.1016/j.cose.2018.11.005.

[40] Toldinas, J., Venčkauskas, A., Damaševičius, R., Grigaliūnas, Š., Morkevičius, N., & Baranauskas, E. (2021). A novel approach for network intrusion detection using multistage deep learning image recognition. Electronics (Switzerland), 10(15), 1854. doi:10.3390/electronics10151854.

[41] Wang, B., & Gu, L. (2019). Detection of network intrusion threat based on the probabilistic neural network model. Information Technology and Control, 48(4), 618–625. doi:10.5755/j01.itc.48.4.24036.

[42] Tang, Y., Gu, L., & Wang, L. (2022). Deep stacking network for intrusion detection. Sensors, 22(1), 25. doi:10.3390/s22010025.

[43] Kohli, M., & Arora, S. (2018). Chaotic grey wolf optimization algorithm for constrained optimization problems. Journal of Computational Design and Engineering, 5(4), 458–472. doi:10.1016/j.jcde.2017.02.005.

[44] Arora, S., & Singh, S. (2017). An improved butterfly optimization algorithm with chaos. Journal of Intelligent and Fuzzy Systems, 32(1), 1079–1088. doi:10.3233/JIFS-16798.

[45] Abdelaziz, A., Santos, V., & Dias, M. S. (2023). Convolutional Neural Network with Genetic Algorithm for Predicting Energy Consumption in Public Buildings. IEEE Access, 11, 64049–64069. doi:10.1109/ACCESS.2023.3284470.

[46] Van Houdt, G., Mosquera, C., & Nápoles, G. (2020). A review on the long short-term memory model. Artificial Intelligence Review, 53(8), 5929–5955. doi:10.1007/s10462-020-09838-1.

[47] Shende, S., & Thorat, S. (2020). Long short-term memory (LSTM) deep learning method for intrusion detection in network security. International Journal of Engineering Research and, 9(6), 1016. doi:10.17577/ijertv9is061016.

[48] Aggarwal, P., & Sharma, S. K. (2015). Analysis of KDD Dataset Attributes - Class wise for Intrusion Detection. Procedia Computer Science, 57, 842–851. doi:10.1016/j.procs.2015.07.490.

[49] UNB (2025). Datasets: NSL-KDD website. Available online: http://nsl.cs.unb.ca/NSL-KDD/ (accessed on June 2025).

[50] Faisal, M., & Islam, M. S. (2023). Improving Network Security with Intrusion Detection Systems Utilizing Machine Learning and Deep Learning Techniques. 26th International Conference on Computer and Information Technology (ICCIT), 1-6. doi:10.1109/ICCIT60459.2023.10441582.

[51] Mynuddin, M., Khan, S. U., Chowdhury, Z. U., Islam, F., Islam, M. J., Hossain, M. I., & Ahad, D. M. A. (2024). Automatic network intrusion detection system using machine learning and deep learning. In 2024 IEEE International Conference on Artificial Intelligence and Mechatronics Systems (AIMS), 1-9. doi:10.1109/AIMS61812.2024.10512607.

[52] Almomani, A., Alweshah, M., Al Khalayleh, S., Al-Refai, M., & Qashi, R. (2019). Metaheuristic algorithms-based feature selection approach for intrusion detection. Machine Learning for Computer and Cyber Security, 184-208. doi:10.1201/9780429504044-8

[53] Vu, L., Bui, C. T., & Nguyen, Q. U. (2017). A deep learning based method for handling imbalanced problem in network traffic classification. Proceedings of the 8th international symposium on information and communication technology, 333-339. doi:10.1145/3155133.3155175.

[54] He, H., & Ma, Y. (2013). Imbalanced learning: Foundations, algorithms, and applications. Imbalanced Learning: Foundations, Algorithms, and Applications, John Wiley & Sons, New Jersey, United States. doi:10.1002/9781118646106.

[55] Tsukada, M., Kondo, M., & Matsutani, H. (2020). A neural network-based on-device learning anomaly detector for edge devices. IEEE Transactions on Computers, 69(7), 1027-1044. doi:10.1109/TC.2020.2973631.

[56] Liu, L., Wang, P., Lin, J., & Liu, L. (2020). Intrusion detection of imbalanced network traffic based on machine learning and deep learning. IEEE access, 9, 7550-7563. doi:10.1109/ACCESS.2020.3048198.

[57] Sun, Y., & Wang, Z. (2025). Intrusion detection in IoT and wireless networks using image-based neural network classification. Applied Soft Computing, 113236. doi:10.1016/j.asoc.2025.113236.

[58] Wojtuch, A., Jankowski, R., & Podlewska, S. (2021). How can SHAP values help to shape metabolic stability of chemical compounds? Journal of Cheminformatics, 13(1), 1–20. doi:10.1186/s13321-021-00542-y.

[59] Aljarah, I., Faris, H., & Mirjalili, S. (2018). Optimizing connection weights in neural networks using the whale optimization algorithm. Soft Computing, 22(1), 1–15. doi:10.1007/s00500-016-2442-1.

[60] Rani, M., & Gagandeep. (2022). Effective network intrusion detection by addressing class imbalance with deep neural networks multimedia tools and applications. Multimedia Tools and Applications, 81(6), 8499-8518. doi:10.1007/s11042-021-11747-6.