Journal of Network and Systems Management (2026) 34:17
https://doi.org/10.1007/s10922-025-09987-0

Enhancing Network Resilience: A Hybrid Meta-Heuristic Intrusion Detection System Leveraging SSA-BOA-CNN

R. C. Jeyavim Sherin[1] · **K. Parkavi**[1]

Received: 3 December 2024 / Revised: 2 April 2025 / Accepted: 4 October 2025

Abstract

Intrusion Detection System (IDS) plays a pivotal role in safeguarding network security by identifying and mitigating malicious attacks. Deep Learning (DL) and Bio-Inspired (BI) algorithms has revolutionized IDS by significantly enhancing their ability to classify and predict network intrusions. Novel hybrid framework, SSA-BOA-CNN introduced in this study, leverages the strengths of Deep Convolutional Neural Networks (Deep CNNs) and bio-inspired metaheuristic algorithms to achieve superior performance in intrusion detection. The proposed framework employs the Salp Swarm Algorithm (SSA) for effective feature selection, ensuring the elimination of redundant and irrelevant data, thereby reducing computational complexity while preserving critical information. Concurrently, the Butterfly Optimization Algorithm (BOA) is utilized to optimize the hyperparameters of the Deep CNN, leading to improved model efficiency and accuracy. CICIDS2019 and CICIDS2018 datasets used for training and testing our model. Extensive experimental evaluations demonstrate that the SSA-BOA-CNN model achieves state-of-the-art performance metrics, including an accuracy rate of 99.16% and a high detection rate, significantly outperforming contemporary IDS methodologies. The hybrid approach effectively addresses key challenges in intrusion detection, such as feature dimensionality reduction, parameter optimization, and accurate classification of diverse attack patterns. This research novelty lies in the synergistic integration of SSA and BOA algorithms with Deep CNN, presenting a robust and efficient solution for enhancing IDS performance. This innovative methodology offers a practical and scalable framework for real-world network security applications, making a substantial contribution to the field of Cybersecurity.

Keywords Cybersecurity · Intrusion detection system · Deep learning · Malware detection · Nature inspired optimization

Extended author information available on the last page of the article

Published online: 28 October 2025

1 Introduction

The field of network computing has seen significant growth in recent years, fueled by rapid advancements in information technology. Key drivers include the rise of large-scale datasets and the increasing reliance on cloud computing infrastructure [1]. However, the growing number of Internet-connected devices, many of which have limited processing power and are designed for low energy consumption, introduces significant challenges to maintaining information security. Because of their increased vulnerability to cyberattacks, these devices require intrusion detection systems (IDS) that can recognize and notify users of possible threats instantly, enabling timely responses [2].

Researchers have improved malware detection through machine learning (ML) and deep learning (DL) techniques as Cybersecurity continues to garner attention [3]. With the growing dependence on network-based services for everyday communication and activities, ensuring the security of these systems has become increasingly critical. Effective network security is essential for mitigating risks and safeguarding interconnected systems from potential vulnerabilities [4]. Addressing intrusions remains a significant challenge, as such incidents can disrupt the availability, confidentiality, and integrity of sensitive information, posing substantial risks to network operations and administration [5]. As Cybersecurity threats continue to evolve, businesses and industries increasingly depend on information systems to guide their security strategies and protect digital assets [6]. The purpose of intrusion detection systems (IDSs) is to identify possible threats by detecting and analyzing abnormalities in a system. These systems monitor network activity for unusual patterns and recommend defensive measures to strengthen security [7].

The two main categories of IDS are anomaly-based and signature-based. IDS that rely on signatures or predetermined patterns of known threats to identify attacks have a high detection rate (DR), but they have trouble seeing novel or zero-day threats [8, 9]. However, anomaly-based intrusion detection systems keep an eye on network traffic for departures from typical patterns and send out notifications when anomalies are detected. While anomaly-based systems excel at detecting unknown threats, they have certain limitations, including a lower detection rate and a higher rate of false alarms (FAR) [10]. The rapid development of methods and procedures has introduced challenges related to noisy and redundant features. It is computationally challenging to manage many attributes in intrusion detection systems because it takes a lot of resources to maintain a high Detection Rate (DR) and a low False Alarm Rate (FAR) [11].

Enhancing the Detection Rate (DR) and reducing the False Alarm Rate (FAR) are the primary goals of IDS. To achieve these goals, researchers have incorporated knowledge-based detection methods, statistical analysis, and a range of machine learning techniques into IDS design [12]. Many IDSs are developed using a hybrid approach that combines machine learning with other detection technologies to enhance their effectiveness and accuracy [13]. Research on intrusion detection has demonstrated its applications across various fields, including artificial intelligence and anomaly detection, leveraging clustering techniques for network analysis [14, 15].

To address these issues, pre-processing techniques are used to remove unnecessary or irrelevant features from the dataset, optimizing the IDS to operate with a more focused and efficient set of attributes [16]. This work highlights a collaborative approach using Genetic Algorithms (GA) for feature extraction and selection. These algorithms optimize subsets of features after pre-processing, improving the system's performance [17]. Furthermore, optimization in intrusion detection often employs meta-heuristic techniques, and their efficiency is systematically evaluated to identify the most effective approach during the stages of feature selection and classification [18].

In order to improve accuracy and lower false alarm rates in Network Intrusion Detection Systems (NIDS), feature selection is essential. By isolating and prioritizing the most relevant features, these methods significantly improve the classification performance of NIDS models. As a result, the systems become more dependable and effective due to increased detection accuracy and decreased mistake rates [19]. Intrusion Detection Systems (IDSs) are designed to identify unauthorized activities within a network. However, their performance often suffers due to the challenges posed by large and irregular datasets, highlighting the need for more effective solutions [20]. Proper integration and positioning of IDSs within a network are crucial for efficiently monitoring and detecting potential security threats. This integration provides a clear framework for understanding the IDS's role in protecting the overall network environment [21, 22].

The rise of sophisticated cyber threats has exposed vulnerabilities in traditional defense mechanisms such as intrusion detection/prevention systems and firewalls. These systems primarily rely on heuristic methods and static attack patterns, limiting their ability to detect new and evolving threats [23]. The growing reliance on internet-based services has significantly increased the risk of cyberattacks. In response, research efforts have intensified, focusing on malware detection and intrusion prevention to tackle these ever-evolving security threats [24].

1.1 Motivation of the Study

Addressing important issues in improving the security of contemporary networked settings is the main goal of this research, especially in light of the rapidly expanding number of network devices and the related difficulties in handling a variety of network traffic. Even though intrusion detection systems (IDS) have advanced in many ways, many current methods find it difficult to strike a compromise between a high Detection Rate (DR) and a low False Alarm Rate (FAR), particularly when working with irregular and large-scale datasets. The necessity to get beyond these restrictions by utilizing cutting-edge feature selection and classification techniques is what motivated this work. This study is driven by the need to overcome these limitations by leveraging advanced methodologies for feature selection and classification. Specifically, we integrate metaheuristic algorithms with deep learning frameworks like Convolutional Neural Networks (CNN), to streamline feature selection, eliminate redundancy, and enhance classification performance. The overarching goal is to deliver a robust intrusion detection mechanism capable of addressing the inefficiencies inherent in current systems.

This study is driven by the growing need to efficiently identify zero-day attacks, minimize false positives, and handle high-dimensional network traffic while using the fewest possible resources. By employing sophisticated optimization and deep learning techniques, this study aims to propose an efficient and scalable solution that caters to the security needs of contemporary network environments. Our work seeks to bridge existing gaps in IDS by demonstrating improvements in detection accuracy and reducing false alarms while maintaining computational efficiency. The findings are expected to contribute significantly to advancing intrusion detection methodologies, ensuring a more secure and resilient networked infrastructure for evolving technological landscapes. This refined motivation highlights the practical and theoretical significance of our research, providing a strong foundation for addressing current challenges in network security.

1.2 Research Gap

The research gap targeted by the proposed system, which utilizes the SSA-BOA for feature selection alongside hybrid classification techniques in an IDS, revolves around the shortcomings of existing IDS solutions in meeting the specific challenges of network environments. Traditional IDS methods often struggle to achieve a balance between detection accuracy and computational efficiency, particularly in networks where devices are constrained by limited processing power and memory. Furthermore, conventional feature selection techniques are often inadequate for the dynamic and heterogeneous nature of network, where the importance of features can shift over time. Additionally, single-classifier systems may be insufficient in detecting a broad spectrum of both known and emerging threats, resulting in potential security gaps. The proposed system addresses these issues by employing SSA-BOA to refine the feature set, thereby enhancing efficiency, and by integrating hybrid classification techniques to boost detection accuracy and adaptability in the complex landscape of network environments.

1.3 Contributions of the Study

This study proposes novel methodologies to enhance the performance of Intrusion Detection Systems (IDS) in the context of complex and high-dimensional datasets. Specifically, the key contributions of this research are as follows:

- We propose hybrid novel framework, SSA-BOA-CNN, which synergistically integrates Deep Convolutional Neural Networks (Deep CNN) with metaheuristic optimization algorithms. The framework demonstrates exceptional efficacy in detecting Distributed Denial-of-Service (DDoS) attacks, particularly in unbalanced network traffic datasets, by leveraging synthetic sampling techniques for enhanced class balance.
- In this study we employed the Salp Swarm Algorithm (SSA) is utilized for efficient feature selection, enabling the elimination of irrelevant and redundant features while preserving essential data for intrusion detection. The Butterfly Optimization Algorithm (BOA) improves classification accuracy and processing

efficiency by fine-tuning the Deep CNN hyperparameters.

- To efficiently handle the difficulties presented by high-dimensional datasets, we used the integration of SSA and BOA in this study, which guarantees an optimum feature set. In addition to lowering computing costs, this method increases intrusion detection's accuracy and dependability.
- To demonstrate the SSA-BOA-CNN framework introduces a scalable and resource-efficient solution that is adaptable to dynamic network environments, making it appropriate for intrusion detection in real time. The architecture of the system is intended to handle large-scale datasets and diverse network traffic patterns without compromising on accuracy or efficiency. The proposed SSA-BOA-CNN framework has been rigorously evaluated using performance measures like accuracy, F1-score, precision, recall and to validate its effectiveness.

1.4 Organization of the Paper

This paper is structured as follows: Sect. 2 provides a comprehensive review of existing studies on network attack prediction. In Sect. 3, we present a detailed discussion of the SSA-BOA-CNN model, including an analysis of its architecture. Section 4 focuses on the experimental analysis and presents the results obtained from the proposed model. Finally, in Sect. 5, we explore potential future directions and conclude the paper with a summary of our findings.

2 Related Works

Feature selection (FS) is a valuable technique applied across statistics, data mining, machine learning, and pattern recognition domains [25]. Acknowledged for its efficacy, FS is a powerful method for eliminating redundant and unnecessary data. The provided summary encompasses pertinent research on intrusion detection employing deep learning algorithms, encompassing factors such as models, features, datasets, and performance metrics.

Kunhare et al. [26] pioneered a highly effective strategy for intrusion detection by leveraging hybrid classifiers integrated with meta-heuristic algorithms. Their methodology incorporated genetic algorithms for adept feature selection, enhancing the system's discriminative power. Furthermore, the research utilized an advanced hybrid classification strategy, merging decision tree and logistic regression algorithms to get higher detection accuracy. Grey Wolf Optimization (GWO) integration further optimized the selected optimal features. The Intrusion Detection System (IDS) significantly improved accuracy and detection rate. This innovative work demonstrated a heightened capability to identify and thwart network malicious attacks accurately.

Sethi et al. [27] suggested enhancing IDS efficiency to 90.48% across various datasets by incorporating a reinforcement learning approach. Recently, meta-heuristic algorithms have gained prominence for feature selection (FS). Significantly, Ant Colony Optimization (ACO) was employed for feature selection in both the KDD Cup'99 and NSL-KDD datasets, in conjunction with optimization algorithms such as PSO, Cuckoo Search, Firefly Algorithm, Bat Algorithm (BA), Genetic Algorithm

(GA), and Artificial Bee Colony (ABC). The results indicate that the suggested system, with a time complexity of, demonstrated promising outcomes of $O\left(n^3\right)$, attains a commendable accuracy level of 98.9%.

Alazzam et al. [28] proposed that particle swarm optimization (PSO) has demonstrated superior performance over genetic algorithm (GA) in the reduction of features in wireless sensor networks, establishing its efficacy in this domain. The application of PSO in these networks has resulted in the detection of 99.40% of the intended system. The increase in data dimensionality has made feature selection a crucial preprocessing step in Intrusion Detection Systems (IDSs). Applying a Feature Selection (FS) technique utilizing Support Vector Machines (SVM) revealed that only six out of 41 dimensions were necessary for analyzing the KDD Cup'99 dataset. This selective feature set resulted in a 1% increase in categorization accuracy.

Mebawondu et al. [29] developed Supervised Learning-based IDS geared towards safeguarding computer resources amidst the pervasive use of the internet. The methodology incorporated an information gain technique to evaluate the importance of each feature in discerning between normal and malicious network traffic. The strategy involved the use of a Multi-Layer Perceptron Neural Network, which effectively identified patterns in crucial traits. This neural network effectively determined whether the network traffic was normal or indicative of an intrusion. The system yielded important outcomes in detecting malicious activities within the network, offering a real-time solution for intrusion detection characterized by high accuracy.

Nagaraja et al. [30] employed a reduced feature set of 19 features from the same dataset, achieving an elevated classification accuracy of 99.95%. The contemporary trend in IDSs involves the widespread adoption of heuristic methods for feature selection, reflecting the current emphasis on efficient and targeted feature subset identification. Particle Swarm Optimization (PSO), evolutionary computation methodologies, and the Firefly algorithm have all been developed to select features in IDSs in recent years. Analyzing the KDD Cup'99 dataset is an example of the ability to perform preprocessing and analyze data to profile network traffic within an anomaly-based Network Intrusion Detection System (IDS). A comprehensive examination of twelve bio-inspired algorithms was conducted, delving into their complexity, application, scope, utility and limitations.

A concise synthesis of the literature is provided in Table 1, following a thorough examination of the pertinent studies.

The Related work highlights several gaps in existing studies, including limited use of advanced metaheuristic algorithms for optimal feature selection, and lower accuracy rates in classification tasks. Many models focus on specific datasets, such as Microsoft BIG-2015, which are smaller and less diverse compared to datasets like CICIDDoS2019, limiting generalizability. Furthermore, challenges remain in addressing scalability, interpretability, and the reduction of features, which may overlook critical information. This study seeks to overcome these shortcomings by leveraging advanced techniques for feature selection and classification to improve intrusion detection in network environments in terms of accuracy and usefulness.

Table 1 Summary of related work

Authors	Research focus	Approach	Dataset	Constraints
Gaber et al. [19]	Detecting malicious activities in IIoT network traffic.	PSO, BA and RF	WUSTL-IIOT-2021	Lack of specificity regarding the deep learning model, its feasibility for, handling data issues, and addressing interpretability and scalability challenges.
Smmarwar et al. [31]	Andriod Malware detection	XAI-AMD-DL	CICAndMal2019	This study did not utilize the most recent metaheuristic algorithms for optimal feature selection.
Kumar et al. [32]	Malware detection in IoMT Devices	CNN-INCA-MD	IoT malware dataset	Achieved accuracy of 96.98%.
Daniel et al. [33]	Malware detection framework tailored for cyber-physical systems (CPS)	GCN-FPA SOFS-OGCNMD	NSL-KDD-2015 CICIDS-2017	Achieved accuracy of 98% for binary classification
Gupta et al. [34]	AI-powered, zero-day malware detection system designed for Industrial Internet of Things (IIoT) environments.	D3WT-CNN-LSTM	Microsoft BIG-2015	This dataset is smaller compared to CI-CIDDoS2019 but is tailored to specific IoT and network traffic scenarios with a focus on malware-related activities.
Smmarwar et al. [35]	Enhance the Malware detection performance	WFS- RF-GreedySW-RF-DT-SVM	CIC-InvesAndMal2019	Models achieved accuracy rates of 91.80%, 91.32%, and 82.33% for static features, and 72.41%, 75.10%, and 62.07% for dynamic features.
Gupta et al. [36]	Analyzing different permissions, intents, and API call logs produced by Android apps after installation or following a device restart is the main goal of this method.	OEL-AMD	CICInvesAndMal2019	The models binary classification accuracy was 96.95%, while their category classification accuracy was 83.49%.
Albakri et al. [25]	Reduce computational costs, and enhance the model's performance.	RHSO- DL-ARAE	Andro-AutoPsy dataset	Performs malware detection using binary classification.
Lee et al. [37]	Combination of a deep hierarchical network and hybrid sampling approaches, with the goal of improving the precision of the intrusion detection	OSS-BSMOTE	NSL-KDD UNSW-NB15 CICIDS2017	Achieved the performance of 94.58% accuracy.

Table 1 (continued)

Authors	Research focus	Approach	Dataset	Constraints
Moizuddin et al. [38]	Dual-Phase methodology to fortify an intrusion detection system and safeguard organizational assets against unauthorized access.	ElasticNet Contractive Auto Encoder and GMGW	NSL-KDD BOT-IOT	These datasets are smaller compared to CICIDS 2019 but is tailored to specific IoT and network traffic scenarios
Almomani et al. [39]	Filter the chosen features that enhance the detection mechanism's functionality.	Wrapper Based-GA-PSO-GWO-FFA	UNSW-NB15	There are only thirty features chosen by this model.

3 Proposed SSA-BOA-CNN Model

This section discusses network traffic data, data preprocessing, SSA-BOA-CNN Fig. 1. This approach's unique pattern recognition characteristics include minimizing the dimension of data, extracting features sequentially, and categorizing within a single network. ML and DL algorithms can be degraded by excessively uneven network traffic data, particularly when sample counts are small.

In this section, a hybrid IDS for the network framework has been proposed. The suggested system is shown in Fig. 1 and is broken down into the following sections: Evaluation of performance, data preparation, feature selection employing SSA-BOA, classification and detection. The CICIDDoS2019, was chosen for the study. These datasets serve as the foundation for our research in DDoS attack detection.

Entering the data preparation stage, a set of transformations and normalization techniques were implemented to guarantee the quality and uniformity of the data. Null values, redundancies, and outliers were addressed, and the dataset was appropriately scaled using standard techniques. Additionally, a sampling method was employed to manage the dataset size and ensure a representative subset.

Following data preparation, the feature selection stage was initiated, incorporating two algorithms: SSA (Salp Swarm Algorithm) and BOA (Butterfly Optimization Algorithm). These algorithms were utilized to detect and prioritize the most impor-

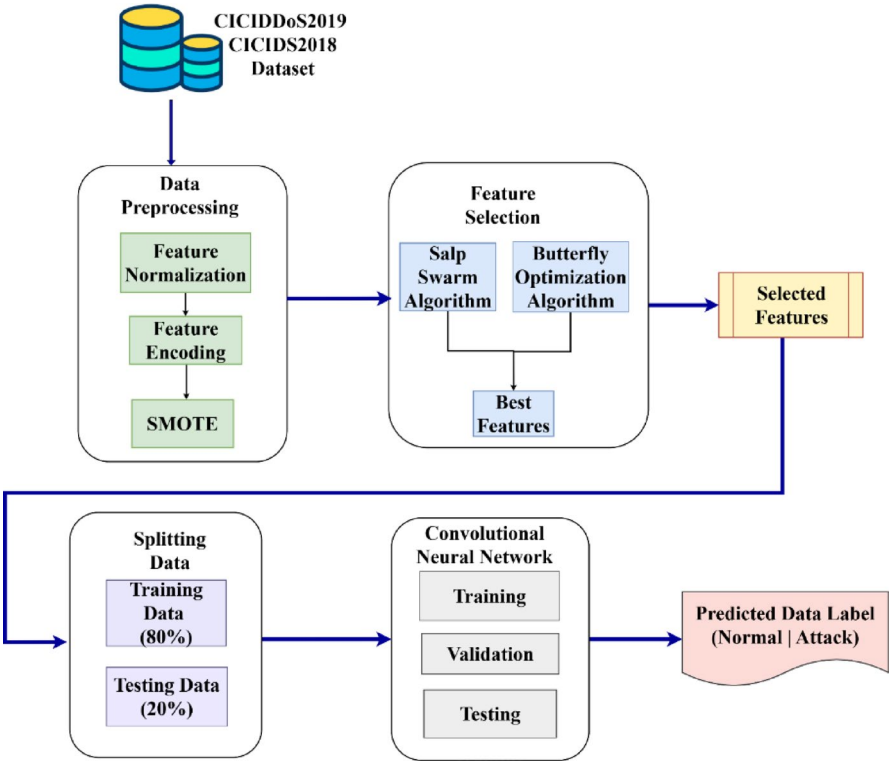


Fig. 1 SSA-BOA-CNN for detecting DDoS attacks

tant characteristics in the datasets, thereby increasing the efficacy of subsequent classification models. As a result of the feature selection process, training and testing sets were separated from the dataset. Data from 80% of the data was utilized in training, while data from 20% was allocated to testing.

CNN was used for classification and detection. This algorithm is chosen for its effectiveness in handling sequential and structural data, allowing for robust detection of normal and attack patterns. In the final step, the trained model was used to predict the nature of incoming data, classifying it as either normal or indicative of an attack. This comprehensive approach, combining data advanced classification algorithms, preprocessing, choosing features, and forms the proposed framework for enhancing DDoS attack detection.

3.1 Dataset

The CICIDDoS2019 dataset, crafted in 2019, stands out with its substantial 25 GB of labeled data, making it a pivotal asset for research in DDoS attack detection. We opted for this dataset due to its remarkable reliability and its fidelity to real-world DDoS attacks, providing a strong and authentic basis for our investigative endeavors. Table 2 illustrates the distribution of the dataset, offering insights into its composition and characteristics. Moreover, It is crucial to highlight that in the CICIDS2018 dataset, the occurrences of Web Attacks are confined to 928 instances [40]. This is a notable difference considering the substantial expansion in the overall dataset size when compared to the CICIDS-2017 IDS dataset. The dataset CIC-IDS2017 was developed employing a real-time network traffic data collected over five days and distributed across eight separate files. It includes both normal traffic and data associated with different kinds of attacks [41]. Specifically, the occurrences of Infiltration and Botnet assaults have surged by 4497 and 143 instances, respectively. A detailed breakdown of the amounts of samples for both datasets is provided in Table 3. When compared with the CICIDS2017 and CICIDS2018 datasets, recent attack scenarios are included in the CICIDDoS2019 dataset. Therefore, it is essential to quickly identify these threats. Hence, we utilized the CICIDDoS2019 dataset for our research.

The CICIDDoS2019 dataset [42] was obtained from the Canadian Institute for Cybersecurity (CIC). It is specifically designed for the detection of Distributed Denial of Service (DDoS) attacks. Modern reflection DDoS assaults, such as those using PortMap, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, SYN, NTP, DNS, and SNMP, are included in this dataset. This timeframe was characterized by the fol-

Table 2 Data of CICDDoS2019

Dataset	Label	Number of Data
CICIDDoS2019	Benign	10,154
	UDP	317,973
	Syn	122,357
	MSSQL	107,672
	LDAP	7138
	NetBIOS	3516
	Portmap	677
	UDPLag	16

Table 3 Content of CI-CIDS2018 and CICIDS2017 dataset

Dataset	Label	Number of Data
CICIDS2017	Benign	1,743,179
	DDoS	128,027
	DoS	252,661
	Botnet	1966
	Bruteforce	13,835
	Infiltration	36
	WebAttacks	2180
	PortScan	158,930
CICIDS2018	Benign	6,112,151
	DDoS	687,742
	DoS	654,301
	Botnet	286,191
	Bruteforce	380,949
	Infiltration	161,934
	WebAttacks	928

lowing attacks. DNS, LDAP, MSSQL, NetBIOS, SNMP, SSDP, UDP, UDP-Lag, WebDDoS, SYN, and TFTP are some of the DDoS attacks that were delivered during training, while PortScan, LDAP, MSSQL, UDP, UDP-Lag, SYN, and TFTP were used during testing. PortScan was run just once on the testing day, and the results are unknown for evaluating the proposed model due to the low traffic volume during testing. Table 4 lists the 77 features in this dataset.

3.2 Data Preprocessing

The preprocessing phase of data collection involves normalizing, reshaping, encoding labels, and splitting data. Furthermore, we excluded any samples from the dataset that had feature values containing NaN or INF. All characteristics in the datasets are standardized to guarantee consistency in the outcomes of characteristic quantification and to alleviate the influence of characteristics with a broad value range on the output of the model. Data normalization can improve the model's accuracy and expedite the solution process. When utilizing the Min-Max normalization technique, the collection of characteristics within the dataset is represented as $\boldsymbol{X}_1$, $\boldsymbol{X}_2$ and so forth. The following Eq. (1) represents the normalization equation for a particular data point $\boldsymbol{X}_i$ in the set:

$$\boldsymbol{X}_i{}' = \frac{\boldsymbol{X}_i - \boldsymbol{X}_{Min}}{\boldsymbol{X}_{Max} - \boldsymbol{X}_{Min}} \tag{1}$$

In the equation provided above, the variable symbolizes the data that has been standardized, while the variables $X_i{}'$ and $X_i{}'$ represent the highest and lowest values within the dataset. After preprocessing got the cleaned dataset Fig. 5 shows the distribution of cleaned CICIDDoS2019 Training Data. The division of the dataset, validation techniques employed, and strategies implemented to prevent overfitting. Synthetic Minority Over-sampling Technique (SMOTE) is a widely used technique for addressing datasets with class imbalance, particularly in the context of machine

Table 4 List of features in CICIDDoS2019 dataset

Sl. No	Feature	Sl. No	Feature
1	Protocol	40	Bwd Packets/s
2	Flow Duration	41	Max Packet Length
3	Total Length of Fwd Packets	42	Packet Length Variance
4	Total Backward Packets	43	Packet Length Std
5	Total Fwd Packets	44	RST Flag Count
6	Fwd Packet Length Min	45	FIN Flag Count
7	Fwd Packet Length Max	46	SYN Flag Count
8	Total Length of Bwd Packets	47	ACK Flag Count
9	Bwd Packet Length Max	48	PSH Flag Count
10	Fwd Packet Length Std	49	URG Flag Count
11	Bwd Packet Length Min	50	CWE Flag Count
12	Bwd Packet Length Std	51	Average Packet Size
13	Bwd Packet Length Mean	52	Down/Up Ratio
14	Flow Packets/s	53	Avg Bwd Segment Size
15	Flow Bytes/s	54	Avg Fwd Segment Size
16	Flow IAT Std	55	Fwd Avg Bytes/Bulk
17	Flow IAT Min	56	Fwd Header Length.1
18	Flow Packets/s	57	Fwd Avg Bulk Rate
19	Flow IAT Max	58	Fwd Avg Packets/Bulk
20	Flow IAT Mean	59	Bwd Avg Packets/Bulk
21	Fwd IAT Mean	60	Bwd Avg Bytes/Bulk
22	Fwd IAT Min	61	Subflow Fwd Packets
23	Fwd IAT Std	62	Bwd Avg Bulk Rate
24	Fwd IAT Max	63	Subflow Fwd Bytes
25	Bwd IAT Total	64	Init_Win_bytes_forward
26	Bwd IAT Mean	65	Subflow Bwd Bytes
27	Bwd IAT Max	66	Active Std
28	Bwd IAT Std	67	Active Min
29	Fwd PSH Flags	68	Active Mean
30	Bwd IAT Min	69	Idle Mean
31	Fwd URG Flags	70	Active Max
32	Bwd PSH Flags	71	Idle Max
33	Fwd Header Length	72	Idle Min
34	Bwd URG Flags	73	Idle Std
35	Fwd Packets/s	74	Bwd IAT Min
36	Bwd Header Length	75	Bwd IAT Max
37	Min Packet Length	76	Label
38	Average Packet Size	77	Inbound
39	Packet Length Variance		

learning classification tasks. Class imbalance occurs when one class (typically the minority class) is significantly underrepresented compared to the other classes in the dataset. This can lead to biased models that perform poorly in predicting the minority class [43]. In order to equalize the distribution of classes, SMOTE creates synthetic

samples for the minority class. The following ways shows how SMOTE works in our research.

1. Identify Minority Class: First, SMOTE identifies the minority class in the dataset. This is typically the class with fewer instances compared to the other classes.

2. Select a Minority Instance: For each minority class instance in the dataset, SMOTE selects one of its k nearest neighbors. The value of k is a parameter specified by the user.

3. Generate Synthetic Instances: SMOTE then generates synthetic instances along the line segment connecting the selected minority instance and its chosen neighbor(s) in the feature space. The synthetic instances are created by randomly selecting a point along the line segment.

4. Repeat: Until the target balance between the majority and minority classes is reached, steps two and three are repeated.

3.3 Proposed Salp Swarm Algorithm and Butterfly Optimization Algorithm (SSA-BOA)

3.3.1 Salp Swarm Algorithm (SSA)

The inspiration of the SSA algorithm draws from Salps hunting behavior in the ocean. Salps, akin to jellyfish, belong to the Salpidae family and exhibit group living where they form chain-like structures [44]. SSA is predominantly used for solving complex optimization problems in various domains, such as engineering, machine learning, and resource allocation. The Salp Swarm Algorithm (SSA), influenced by this natural phenomenon, is relatively straightforward to implement and requires minimal parameter adjustments. However, it is susceptible to premature convergence. In the SSA context, within a chain, the leader holds a pivotal role, guiding the movements of the other members. The calculation of the leader's position is determined by the following Eqs. 2 and 3.

$$X_n^1 = FPos_j - crn_1 \left[(U_pB_n - LoB_n)\, crn_2 + LoB_n \right] \; if \; crn_3 < 0.5 \tag{2}$$

$$X_n^1 = FPos_j + crn_1 \; \left[(U_pB_n - LoB_n)\, crn_2 + \; LoB_n \right] \; if \; crn_3 \; \geq \; 0.5 \tag{3}$$

Where $FPos_j$ is the food position and X_n^1 is the leader position. The upper bound is U_pB_n. The lower bound is LoB_n. The random number crn_2, which ranges from 0 to 1, is utilized to regulate the leader's mobility stride. The transition between two equations that update position is controlled by crn_3. The control parameter that adjusts the execution of SSA is denoted as crn_1. Equation 4 provides defines it:

$$crn_1 = 2e^{-(4l/Max\_Iteration)^2} \tag{4}$$

Where l represents the current iteration and Max_Iteration indicate the maximum number of iterations.

Where X_n^m denotes the mth follower in nth dimension. The following Eqs. 5 and 6 are used to update the leader position in SSA:

$$X_n^1 = w \times FPos_j + crn_1 \times \sin(rnd_2) \times \left| rnd_3 X_{Best}^{ite} - X_n^{ite} \right| \; if \; crn_3 \geq 0.5 \quad (5)$$

$$X_n^1 = w \times FPos_j - crn_1 \times \cos(rnd_2) \times \left| rnd_3 X_{Best}^{ite} - X_n^{ite} \right| \; if \; crn_3 < 0.5 \quad (6)$$

The introduction of the Levy flight function improves the second phase of SSA. Consequently, the position of the follower is established by Eq. 7.

$$X_n^m = 1/2\left(X_n^m + X_n^{m-1}\right) + 0.01 \times LF \times crn_4 \quad (7)$$

Algorithm 1 SSA Pseudocode

Step 1: Initial population as $i = 1, 2, \ldots, n$ taking into account the upper bound (ub) and lower bound (lb).
Step 2: Calculate the fitness function for each salp $X_n^{ite} = the\ best\ search\ agent$
Step 3: While($ite \leq ite_{max}$)
Step 4: Update $LF, crn_1, rnd_2, crn_4, rnd_3$ and crn_3
If ($ite == 1$)
If ($crn_3 \geq 0.5$)
Step 5: Apply Equation 5 to update the position of each Salp.
Else if ($crn_3 < 0.5$)
Employ Equation 6 to update position of each Salp
Step 6: Else if ($ite \geq 2$)
Apply equation 4 to update the position of each Salp
End if
$ite = ite + 1$
End while
Provide the optimal subset of features as the output.

1) Population Initialization: Initialize a population of Salps, where each Salp represents a potential solution in the search space. The search space is defined by upper bounds (ub) and lower bounds (lb) for each feature. Table 5 indicates the parameter initialization of the Salp Swarm Algorithm for feature selection.

2) Fitness Computation: Assess the fitness of each Salp by utilizing a fitness function. Revise the position of the best Salp best Salp (X_n^{ite}) discovered thus far.

3) Iterations: While the maximum number of iterations (ite_{max}) is not reached:

• Update control parameters: $LF, crn_1, rnd_2, crn_4, rnd_3$ and crn_3
• If it's the first iteration:
• Use Eq. 5 to update the Salp's position if crn_3 is greater than or equal to 0.5.
• If crn_3 is less than 0.5, update the salp position.
using Eq. 6.
• If it's not the first iteration:
• Revise the position of the Salp using Eq. 7.
• Increment the iteration counter (ite).

Optimal Feature Subset: Provide the optimal feature subset determined by the final positions of the Salps.

Table 5 Parameter initialization of salp swarm algorithm

Parameter	Value
Dimension	Number of features
Max_iteration	20
upper bound (ub)	5
Lower bound (lb)	−5
fobj	Fitness function in terms of the selected features

Table 5 contains the information about the parameters used in an optimization algorithm.

• Dimension: This parameter represents the number of features or variables in the problem being optimized or modeled. In other words, it indicates the dimensionality of the input data.

• Max_iteration: This parameter specifies the maximum number of iterations that the optimization algorithm or model training process will run. It acts as a stopping criterion to prevent the algorithm from running indefinitely.

• upper bound (ub): This parameter represents the upper bound or maximum value that each feature or variable can take. In this case, the upper bound is set to 5, meaning that no feature value can exceed 5.

• Lower bound (lb): This parameter represents the lower bound or minimum value that each feature or variable can take. In this case, the lower bound is set to −5, meaning that no feature value can be less than −5.

• fobj: This parameter represents the objective function or fitness function that the optimization algorithm or model is trying to optimize or minimize. It is typically a mathematical expression or a function that takes the selected features as input and outputs a scalar value representing the quality or fitness of the solution.

Based on the Salp Swarm Algorithm the selected features are.

SSA Based Feature Selection = *{Total Backward Packets*, *Total Length of Bwd Packets*, *Fwd Packet Length Min*, *Bwd Packet Length Std*, *Flow Packets/s*, *Flow IAT Std*, *Flow IAT Min*, *Fwd IAT Mean*, *Fwd IAT Max*, *Bwd IAT Total*, *Bwd IAT Std*, *Bwd IAT Min*, *Bwd PSH Flags*, *Bwd URG Flags*, *Bwd Header Lengths*, *Bed Packets/s*, *Max Packet Length*, *Packet Length Std*, *FIN Flag Count*, *RST Flag Count*, *ACK Flag Count*, *CWE Flag Count*, *Down/Up Ratio*, *Avg Fwd Segment Size*, *Fwd Header Length.1*, *Fwd Avg Packets/Bulk*, *Bwd Avg Bytes/Bulk*, *Bwd Avg Bulk Rate*, *Subflow Fwd Bytes*, *Subflow Bwd Bytes*, *Active Mean*, *Active Max*, *Idle Mean*, *Idle Max*, *Inbound}*.

3.3.2 Butterfly Optimization Algorithm (BOA)

The Butterfly Optimization Algorithm (BOA) is an optimization algorithm inspired by the collective behavior of butterflies in nature [45]. Designed for addressing optimization challenges, BOA strives to effectively explore the solution space and converge towards optimal solutions. The four primary steps of the BOA are covered in detail below.

Step 1 Initialization.

In the BOA phase, involving problem parameters and population initialization, all solutions are randomly generated. The vectors are used to depict the solutions with lengths equivalent to the dimension *d* of the problem.

There are two possible phases to the update mechanism: Local and global exploration. The butterfly i advances in the global search in the direction of the fittest butterfly g^*, which is represented as follows:

$$F_i^{t+1} = (r^2 \times g^* - x_i^t) \times pf_i \tag{8}$$

Where r in [0,1] is a random number. The updating movement in local search can be expressed as follows:

$$F_i^{t+1} = \left(r^2 \times x_j^t - x_i^t\right) \times pf_i \tag{9}$$

To generate the population, all solutions are found in a matrix, as shown in Eq. (10) below.

$$Population = \begin{bmatrix} x_1^1 & x_2^1 \ldots & x_d^1 \\ x_1^2 & x_2^2 \ldots & x_d^2 \\ x_1^N & x_2^N \ldots & x_d^N \end{bmatrix} \tag{10}$$

Step 2 Fitness value calculation.

This stage involves evaluating each solution in light of the optimization problem's objective function. The optimal solution is then allocated to g∗.

Step 3 Update the population.

In this specific phase, the Butterfly Optimization Algorithm (BOA) advances by updating all solutions using the fitness metrics that were acquired in Step 2. This method determines whether the search behavior should be directed locally or globally by generating a random number (r) and comparing it with population (p). When r is smaller than p, the butterfly starts to move globally as per Step 2. Otherwise, it shifts locally following Step 3. Afterward, if the newly acquired solution proves superior to the current one, it supplants the old solution. Finally, the parameter g* undergoes an update.

Step 4 Check the stop condition.

Until the maximum number of repetitions is achieved, steps 2 and 3 are repeated.

Algorithm 2 BOA Pseudocode

```
Input: CICIDDoS2019 dataset
Output: Feature subset (optimal features)
Step 1: Initialize the Population Matrix for the Butterfly Optimization Algorithm
        N Butterflies x_i = (i = 1, 2, ...,n)
Step2: While (iteration_butterfly ≤ max_iterations_butterfly)
    For each solution in the population do
      Calculate the solution's fitness value
      g* = best solution
      Generate r (random number within the range [0, 1])
      If (r < p)
        Update the solution using Equation 8
      Else
        Update the solution using Equation 9
      End If
    End for
    iteration_butterfly = iteration_butterfly + 1
  End While
Step 3: Return the optimal feature subset
```

In Algorithm 2, the fitness values are crucial for guiding the optimization process and identifying the optimal feature subset. The fitness of each solution in the population is evaluated based on its performance in selecting relevant features that maximize the classification accuracy or minimize the error rate for the given task. The impact of the feature subset on the overall model performance is specifically taken into account by the fitness function, which is commonly quantified using measures like accuracy, precision, recall, or F1-score.

At each iteration, the fitness of all solutions (i.e., butterfly positions) is assessed, and the best solution, denoted as g*, is updated accordingly. The solution with the highest fitness value is retained as the current optimal solution, and it serves as the basis for further updates. This process ensures that the optimization algorithm converges towards the most relevant feature subset, thereby improving the model's detection capabilities. Figure 5 indicates the selected features for the CICIDDoS2019 dataset using the SSA and BOA method. Better feature subset leads to higher classification accuracy and reduce the redundancy of feature selection.

Table 6 contains the information about the parameters used in an optimization algorithm.

The chosen features, as determined by the Butterfly Optimization Algorithm, are.

BOA Based Feature Selection = *{Fwd Packet Length Mean*, *Avg Fwd Segment Size*, *Flow IAT Max*, *Max Packet Length*, *Flow IAT Std*, *Fwd IAT Std*, *Protocol*, *Total Fwd Packets*, *Fwd IAT Mean*, *Flow IAT Mean*, *Fwd IAT Max*, *Fwd IAT Total*, *Fwd Header Length*, *Fwd Header Length.1*, *Flow Duration*, *Fwd Packets/s*, *Idle Max*, *Active Min*, *Total Backward Packets*, *Inbound*, *URG Flag Count*, *Active Max*, *Bwd Packets/s*, *Bwd Packet Length Max*, *Bwd IAT Max*, *Down/Up Ratio*, *Subflow Bwd Bytes*, *Bwd Header Length*, *Avg Bwd Segment Size*, *CWE Flag Count*, *Total Length of Bwd Packets*, *Subflow Bwd Packets*, *Fwd IAT Min*, *Flow IAT Min*, *Bwd IAT Total*, *Bwd IAT Mean*, *Bwd Packet Length Std*, *Bwd Packet Length Mean*, *Bwd IAT Std*, *Idle*

Table 6 Parameter initialization of butterfly optimization algorithm

Parameters	Value
Pop_Size	50
N (Number of dimensions)	10
Max_iter	20
Lb (Lower bound)	−10
Ub (Upper bound)	10

Mean, *Active Mean*, *RST Flag Count*, *Idle Min*, *Bwd IAT Min*, *Fwd PSH Flags*, *Idle Std*, *SYN Flag Count*, *Bwd Packet Length Min*, *Active Std*, *Fwd Avg Bytes/Bulk*, *Fwd Avg Packets/Bulk*, *Bwd Avg Bulk Rate*, *ECE Flag Count*, *Bwd URG Flags*, *PSH Flag Count*, *Fwd URG Flags*, *Bwd PSH Flags*, *Fwd Avg Bulk Rate*, *FIN Flag Count*, *Bwd Avg Packets/Bulk*, *Bwd Avg Bytes/Bulk}*.

3.4 Need of Hybrid Feature Selection

An efficient IDS must achieve the best accuracy using the provided information. In order to solve complex problems, hybrid approaches are designed. Finding the optimum feature subset from the original feature set is a crucial step in feature selection since it lowers the computational strain of storage, improves classification performance, and reduces the dimensionality of data processing. Table 9 shows the 55 best feature scores obtained using the SSA-BOA feature selection algorithm.

$\mathrm{SSA} \cup \mathrm{BOA} = \{$*'Fwd Packet Length Std'*, *'Average Packet Size'*, *'Total Length of Fwd Packets'*, *'Packet Length Std'*, *'Packet Length Variance'*, *'Min Packet Length'*, *'ACK Flag Count'*, *'Subflow Fwd Bytes'*, *'Fwd Packet Length Max'*, *'Fwd Packet Length Min'*, *'Packet Length Mean'*, *'Subflow Fwd Packets'*, *'Init_Win_bytes_forward'*, *'Fwd Packet Length Mean'*, *'Avg Fwd Segment Size'*, *'Flow IAT Max'*, *'Max Packet Length'*, *'Flow IAT Std'*, *'Fwd IAT Std'*, *'Protocol'*, *'Total Fwd Packets'*, *'Fwd IAT Mean'*, *'Flow IAT Mean'*, *'Fwd IAT Max'*, *'Fwd IAT Total'*, *'Fwd Header Length'*, *'Fwd Header Length.1'*, *'Flow Duration'*, *'Fwd Packets/s'*, *'Idle Max'*, *'Active Min'*, *'Total Backward Packets'*, *'Inbound'*, *'URG Flag Count'*, *'Active Max'*, *'Bwd Packets/s'*, *'Bwd Packet Length Max'*, *'Bwd IAT Max'*, *'Down/Up Ratio'*, *'Subflow Bwd Bytes'*, *'Bwd Header Length'*, *'Avg Bwd Segment Size'*, *'CWE Flag Count'*, *'Total Length of Bwd Packets'*, *'Subflow Bwd Packets'*, *'Fwd IAT Min'*, *'Flow IAT Min'*, *'Bwd IAT Total'*, *'Bwd IAT Mean'*, *'Bwd Packet Length Std'*, *'Bwd Packet Length Mean'*, *'Bwd IAT Std'*, *'Idle Mean'*, *'Active Mean'*, *'RST Flag Count'*, *'Idle Min'*, *'Bwd IAT Min'*, *'Fwd PSH Flags'*, *'Idle Std'*, *'SYN Flag Count'*, *'Bwd Packet Length Min'*, *'Active Std'*, *'Fwd Avg Bytes/Bulk'*, *'Fwd Avg Packets/Bulk'*, *'Bwd Avg Bulk Rate'*, *'ECE Flag Count'*, *'Bwd URG Flags'*, *'PSH Flag Count'*, *'Fwd URG Flags'*, *'Bwd PSH Flags'*, *'Fwd Avg Bulk Rate'*, *'FIN Flag Count'*, *'Bwd Avg Packets/Bulk'*, *'Bwd Avg Bytes/Bulk'}*

3.5 Convolutional Neural Network

CNNs belong to a category of deep neural networks that were originally utilized for analyzing visual images, comprehending video content, identifying voices, and processing spoken language. The convolution function maintains the spatial relationship within the input data by assessing the attributes of the kernel function [46]. CNN functions as an automated feature extractor, highly proficient in processing and analyzing complex, high-dimensional datasets. The optimal structural configuration is employed to automatically adjust the values of the kernel. The depth of the layer determines the scale of the feature map. Before convolution, the supplemental nonlinear function is used to create feature maps. According to Eq. 11, the nonlinear activation could resemble a ReLU [47]. The resulting layer is initiated using softmax in

a multi-level neural layer known as the Fully Connected Layer. The following layer are connected to those of the previous layer. Table 7 indicates the parameters used in CNN model for IDS. The cross-entropy loss function, which is frequently utilized for classification problems, is represented by Eq. 12. It measures the difference between the true labels y_i and the predicted probabilities $\widehat{y}_i$.

$$\boldsymbol{ReLu} = \boldsymbol{max}(0, \boldsymbol{x}) \tag{11}$$

$$\boldsymbol{Loss} = -\sum_{\boldsymbol{i}=1}^{\boldsymbol{output\_size}} \boldsymbol{y_i} * \log\left(\widehat{\boldsymbol{y}}_{\boldsymbol{i}}\right) \tag{12}$$

Where:

- y_i represents the true class label for the i^{th} output class.
- $\widehat{y}_i$ denotes the predicted probability for the i^{th} output class.
- output_size is the number of possible output classes (or categories).
- $\log\left(\widehat{y}_i\right)$ is the natural logarithm of the predicted probability for the i^{th} class.

The loss is calculated as the negative sum of the true labels weighted by the log of the predicted probabilities, and the sum is calculated over all classes. This formulation encourages the model to maximize the likelihood of the correct class by minimizing the loss during the training process. In simpler terms, the loss function penalizes the model for incorrect predictions, where the penalty increases as the actual class label deviates from the expected likelihood. This makes the model more likely to improve the accuracy of its predictions as the training progresses. Figure 2 illustrates the source code for creating layers in a Convolutional Neural Network (CNN) model for the CICIDDoS2019 dataset.

The input layer of the CNN model is made to handle data with a shape (55, 1, 1) indicating a sequence length of 55 with elements possessing a height and width of 1. Following this, a 2D convolutional layer is instantiated with 120 filters, each sized 2×2. The first convolutional layer (Conv2D) is responsible for capturing fundamental, low-level patterns within the data, such as correlations and interactions between

Table 7 Parameters used in CNN model for IDS

Layer (type)	Output Shape	Param #
Input_1 (InputLayer)	(None, 55, 1, 1)	0
Conv2d (Conv2D)	(None, 55, 1, 120)	600
Conv2d_1 (Conv2D)	(None, 55, 1, 60)	64,860
Conv2d_2 (Conv2D)	(None, 55, 1, 30)	28,830
Flatten (Flatten)	(None, 2370)	0
Dense (Dense)	(None, 15)	35,565
Output: Fully connected layer	1	
Total params: 129855 (507.25 KB)		
Trainable params: 129855 (507.25 KB)		
Non-trainable params: 0 (0.00 Byte)		

```
def create_cnn_model() -> keras.Model:
    # Creating layers
    inputs = keras.layers.Input(shape=(79, 1, 1))
    w = keras.layers.Conv2D(120, 2, activation='relu', padding="same")(inputs)
    w = keras.layers.Conv2D(60, 3, activation='relu', padding="same")(w)
    w = keras.layers.Conv2D(30, 4, activation='relu', padding="same")(w)
    w = keras.layers.Flatten()(w)
    outputs = keras.layers.Dense(15, activation='softmax')(w)
    cnn_model = keras.Model(inputs=inputs, outputs=outputs, name='cnn')

    # Compile layers
    cnn_model.compile(loss='sparse_categorical_crossentropy',
                      metrics=['sparse_categorical_accuracy'],
                      optimizer='adam')

    return cnn_model
```

Fig. 2 Creating Layers in CNN Model for CICIDDoS2019 Dataset

adjacent features (e.g., Flow Duration and Packet Length), thereby initiating the feature extraction process. An activation function for rectified linear units (ReLUs) is applied to introduce non-linearity into the system. Utilizing 'padding = same' ensures that the input is padded with zeros to maintain consistent spatial dimensions in the output. The mathematical Eqs. 13 and 14 can be employed to denote each hidden layer incorporating a ReLu activation function:

$$g(x) = f(x^T w + b) \tag{13}$$

$$g(x) = g_i(g_i - 1)(\dots(g_1(x))) \tag{14}$$

Subsequently, another 2D convolutional layer is created with 60 filters, each sized 3×3. Second Convolutional (Conv2d_1) captures more complex patterns or interactions between the features. Similar to the previous layer, ReLU activation and 'same' padding are employed, and the result is updated and stored. This process is repeated with a third 2D convolutional layer, now with 30 filters of size 4×4, maintaining the same activation function and padding strategy. The inclusion of pooling layers is integral to CNN architecture as they facilitate down sampling, reducing the complexity of the network and mitigating overfitting. Two prevalent pooling techniques utilized in CNN systems are Max-Pooling and Average Pooling.

Pooling layers are designed to condense the information obtained from convolutional layers, thus aiding in capturing the most salient features while discarding less relevant details. Using Max-Pooling, the most prominent features are retained within each region by selecting the maximum value from the specified window. Alternatively, Average Pooling provides a more generalized representation of the features present by calculating average values within the window.

By incorporating pooling layers, the CNN can effectively downsample the feature maps, increasing the model's capacity to generalize to new inputs while lowering computational cost. This undersampling process contributes to preventing overfitting

by promoting spatial invariance and reducing the sensitivity to small variations in the input data.

Therefore, the strategic inclusion of pooling layers, whether through Max-Pooling or Average Pooling, plays a vital role in enhancing the efficiency and effectiveness of CNN systems, contributing to their robustness and generalization capabilities. In the Max-Pooling operation, denoted as Eq. 13, the output value for a specific region (R) is obtained by selecting the maximum activation value among all neurons within that region. This approach effectively condenses the information in the region by emphasizing the most significant feature present, thus serving as a representative output for the entire region.

$$\Upsilon_i = \left(\underset{i \in R}{max} x_i\right) \tag{15}$$

A flattening layer is then introduced, crucial for transitioning from the 3D output of convolutional layers to a 1D array, facilitating connection with subsequent fully connected layers. Following flattening, a fully connected dense layer consisting of 15 units is added. The softmax activation function is chosen denoted in Eq. (15), particularly suitable for multi-class classification tasks. The resulting output is stored in the output layer. Table 8 indicates the hyper parameter for our model.

$$softmax\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{16}$$

As a final step, the model is compiled with sparse categorical cross-entropy as the loss function, sparse categorical accuracy as the performance metric, and Adam optimizer for parameter updates. Performance of the SSA-BOA-RF (Random Forest) using CICIDDoS2019 Dataset and CICIDS2018 Dataset is shown in Tables 10 and 11. Performance of the proposed model SSA-BOA-CNN is shown in Table 12 and Table 13 using CICIDDoS2019 dataset and CICIDS2018 dataset. There is a learning rate of 0.001.

Table 8 Hyper parameter for our model

Parameter	Value
Epoch & Batch Size	100/1024
Activation Function	ReLU/Softmax
Optimizer	Adam
Loss Function	Sparse-categorical_crossentropy
No of Hidden Layers	4
Hidden Neuron Size	60/30

4 Results and Discussion

The SSA-BOA-LSTM-CNN model was initialized using Python 3.8.5 and trained using a computer with an Intel Core i7-10700 K CPU and 32GB RAM. To implement a deep learning model, TensorFlow 2.4.1 library[1] was used. The Scikit-learn library[2] was used to select features and classify them. Assessing the effectiveness of an Intrusion Detection System (IDS) entails evaluating crucial aspects, including efficiency, user-friendliness, cost-effectiveness, speed, CPU and memory demands, and scalability. Effectiveness refers to the system's ability to detect and prevent intrusions accurately. Ease of use encompasses factors like integration with other products, investigation capabilities, and reporting features. Cost considerations involve both the initial investment and ongoing maintenance expenses. Speed, CPU, and memory requirements impact the system's efficiency. Scalability measures the system's ability to adapt to growing demands. A comprehensive evaluation considers these factors to ensure a balanced and efficient IDS implementation.

Evaluation of IDS is efficient when it is able to predict exact outcomes. Statistical classification of problems is represented by confusion matrices or error matrices. There are four potential outcomes presented in a table for a particular event. True Negative (TN) and True Positive (TP) represent accurate identifications of benign and malicious behavior, labeled as normal and attack, respectively. In false positives and false negatives, normal behavior is incorrectly classified as hostile or benign behavior. The elevated False Positive (FP) rate is significantly impacted by system performance, while the high False Negative (FN) rate exposes the system to vulnerabilities from intrusions. Consequently, an effective Intrusion Detection System (IDS) should exhibit low FN and FP rates, coupled with high True Negative (TN) and True Positive (TP) rates. R refers to Recall and P refers to Precision. Recall assesses the model's ability to correctly identify all the actual positive instances, measuring how many of the true positives were successfully detected. Precision evaluates the accuracy of the model's positive predictions, determining the proportion of predicted positive instances that are actually positive. Equations 16, 17, 18, 19, 20 and 21 provided the performance metrics derived from the confusion matrix for accurately predicting IDS outcomes.

$$True\ Negative\ Rate\ (TNR) = \frac{TN}{FP + TN} \times\ 100 \tag{17}$$

$$True\ Positive\ Rate\ (TPR) = \frac{TP}{FN + TP} \times\ 100 \tag{18}$$

$$Accuracy = \frac{TN + TP}{TP + TN + FN + FP} \times\ 100 \tag{19}$$

[1] https://www.tensorflow.org/.

[2] https://scikit-learn.org/stable/.

$$F1\ Score = 2 \times \frac{(R \times P)}{R + P} \times 100 \tag{20}$$

$$Recall = \frac{TP}{TP + FN} \times 100 \tag{21}$$

$$Precision = \frac{TP}{TP + FP} \times 100 \tag{22}$$

Table 9 presents the best feature scores for the CICIDDoS2019 dataset, determined based on the fitness value. Features with higher fitness values are selected for inclusion. Figure 5 displays the 55 features selected for the CICIDDoS2019 dataset using the SSA-BOA method.

Table 14 shows the performance comparison of proposed model with other existing models. Figure 3 visually represents the proportions of these labels. In the CICIDDoS2019 dataset, the label distribution is as follows: UDP (55.82%), Benign (1.78%), LDAP (1.25%), MSSQL (18.90%), Portmap (0.28%), Syn (21.48%), and UDPLag (0.02%). Figure 4 visually represents is as follows: Benign (27.81%), DoS (26.55%), DDoS (23.55%), and Bruteforce (22.09%).

Figure 6 illustrates the comparison of the best fitness values based on the generation between the SSA and BOA algorithms.

Figure 7 (a) and Fig. 7 (b) illustrates how the proposed SSA-BOA-CNN model performed over a 100 epochs for training with CICIDDoS2019 dataset. This graph provides a thorough overview of the model's performance throughout a particular number of training epochs, providing insight into its accuracy of 99.16% and loss during the training process. It shows the validation losses for SSA-BOA-CNN. However, when SSA-BOA was applied to the proposed model, validation losses were lower. With 100 epochs of validation, SSA-BOA-CNN achieved a validation loss of 0.0272. The research prior to this ignored feature selection strategies, which might reduce the number of features in the training phase.

Figure 8 (a) and Fig. 8 (b) illustrates how the proposed SSA-BOA-CNN model performed over a 100 epochs for testing with CICIDS2018 dataset. As a result of this graph, provides a thorough.

analysis of the model's performance across a predetermined number of epochs during the training process, providing insight into its accuracy of 99.10% and loss during the training process. It shows the validation losses for SSA-BOA-CNN. However, when SSA-BOA was applied to the proposed model, validation losses were lower. With 100 epochs of validation, SSA-BOA-CNN achieved a validation loss of 0.0260.

The CICIDDoS2019 confusion matrix is shown in Fig. 9. Our model predicted the attacks accurately with default threshold 0.5. The model accurately predicted 2549 instances for the 'UDP' class with only 1 misclassification. For the 'Sym' class, 1661 instances were correctly predicted, but there was 1 misclassification. The model successfully predicted 25,100 instances for the 'MSSQL' class, with 2 misclassifications. Similarly, for the 'Benign' class, there were 865 correct predictions and 2 instances misclassified. The 'LDAP' class had 139 instances correctly predicted, but there

Table 9 Best features score for CICIDDoS2019 dataset

Feature	Score
Fwd IAT Total	1.79E+13
Flow Duration	1.79E+13
Fwd Header Length	1.38E+13
Fwd Header Length.1	1.38E+13
Idle Max	7.07E+12
Fwd IAT Max	6.95E+12
Flow IAT Max	6.94E+12
Bwd IAT Total	6.52E+12
Idle Mean	5.24E+12
Bwd IAT Max	4.98E+12
Idle Min	3.68E+12
Fwd IAT Std	2.94E+12
Flow IAT Std	2.65E+12
Bwd IAT Std	2.43E+12
Fwd IAT Mean	2.16E+12
Idle Std	1.72E+12
Flow IAT Mean	1.60E+12
Bwd IAT Mean	1.43E+12
Bwd Header Length	1.41E+12
Fwd Packets/s	4.34E+11
Active Max	1.40E+11
Active Std	7.28E+10
Active Mean	4.50E+10
Active Min	2.64E+10
Packet Length Variance	2.01E+10
Total Length of Bwd Packets	6.97E+09
Subflow Bwd Bytes	6.97E+09
Flow IAT Min	4.27E+09
Fwd IAT Min	3.81E+09
Init_Win_bytes_forward	2.54E+09
Bwd Packets/s	1.72E+09
Total Length of Fwd Packets	2.65E+08
Subflow Fwd Bytes	2.65E+08
Bwd Packet Length Max	1.53E+08
Average Packet Size	1.45E+08
Min Packet Length	9.07E+07
Fwd Packet Length Min	9.06E+07
Fwd Packet Length Mean	8.69E+07
Avg Fwd Segment Size	8.69E+07
Packet Length Mean	8.66E+07
Fwd Packet Length Max	8.17E+07
Max Packet Length	7.99E+07
Bwd Packet Length Std	4.89E+07
Bwd Packet Length Mean	4.08E+07
Avg Bwd Segment Size	4.08E+07
Packet Length Std	1.03E+07
Total Fwd Packets	8.86E+06

Table 9 (continued)

Feature	Score
Subflow Fwd Packets	8.86E+06
Fwd Packet Length Std	7.56E+06
Bwd Packet Length Min	6.84E+06
Total Backward Packets	2.74E+06
Subflow Bwd Packets	2.74E+06
Bwd IAT Min	1.21E+06
Protocol	8.14E+05
ACK Flag Count	4.34E+05

Table 10 Performance of the SSA-BOA-RF (Random Forest) using CICIDDoS2019 dataset

Label	Precision	Recall	F1-score	Support
0	0.99	1.00	1.00	2067
1	0.92	0.90	0.91	1364
2	0.97	0.99	0.95	21,441
3	0.87	0.90	0.89	700
4	0.23	0.11	0.15	119
5	1.00	1.00	1.00	24,611
6	1.00	0.98	1.00	63,594
7	0.50	0.35	0.41	34
accuracy			0.97	113,930
macro avg	0.81	0.78	0.79	113,930
weighted avg	0.97	0.97	0.97	113,930

Table 11 Performance of the SSA-BOA-RF (Random Forest) using CICIDS2018 dataset

Label	Precision	Recall	F1-score	Support
0	0.93	0.99	0.96	124,776
1	1.00	1.00	1.00	47,233
2	1.00	1.00	1.00	81,682
3	1.00	1.00	1.00	90,904
4	1.00	1.00	1.00	71,682
5	0.93	0.68	0.79	30,445
6	0.94	0.69	0.80	180
accuracy			0.98	403,563
macro avg	0.97	0.91	0.93	403,563
weighted avg	0.98	0.98	0.97	403,564

were 2 misclassifications. For the 'Portmap' class, the model achieved 79,756 correct predictions, but there were 3 misclassifications. Lastly, for the 'UDPLag' class, 49 instances were correctly predicted with only 1 misclassification.

Figure 10 shows a graphical depiction of the confusion matrix for the CICIDS2018 dataset. The model correctly predicted 120,239 instances as the 'Benign' class, 95,154 instances for the 'DoS' class, and 101,531 instances for the 'DDoS' class. However, there were 2 misclassifications for the 'DDoS' class. For the 'Bruteforce' class, the model made 114256 correct predictions, but there were 21 instances misclassified as other classes.

Table 12 Performance of the proposed model SSA-BOA-CNN using CICIDDoS2019 dataset

Label	Precision	Recall	F1-score	Support
0	0.94	0.99	0.98	2901
1	0.35	0.84	0.50	901
2	0.99	0.96	0.98	33,555
3	0.96	0.97	0.96	1042
4	0.70	0.94	0.81	152
5	1.00	0.99	1.00	36,872
6	1.00	1.00	1.00	95,416
7	0.67	0.57	0.62	56
accuracy			0.99	170,895
Macro avg	0.83	0.91	0.85	170,895
Weighted avg	0.99	0.99	0.99	170,895

Table 13 Performance of the proposed model SSA-BOA-CNN using CICIDS2018 dataset

Label	Precision	Recall	F1-score	Support
0	0.95	0.99	0.97	134,776
1	0.49	0.85	0.50	57,233
2	0.99	0.97	0.98	91,682
3	0.96	0.97	0.96	100,904
4	0.77	0.94	0.85	91,682
5	1.00	0.99	1.00	32,445
6	1.00	1.00	1.00	180
accuracy			0.99	493,564
Macro avg	0.82	0.90	0.88	493,564
Weighted avg	0.99	0.99	0.99	493,564

The results in Table 15 show how well each model performed on each of these metrics. For instance, the Proposed SSA-BOA-CNN model achieved a precision of 0.9867, recall of 0.9952, AUC-ROC of 0.9985, F1-Score of 0.9868, and accuracy of 0.9916. Similarly, other models such as RNN-AD, GRU, GA, DNN, DAE, BiLSTM and SVM are evaluated based on these metrics. In the context of the classification task, these measures taken together offer insights into each model's efficacy and dependability. Figure 11 illustrates the efficacy analysis of the proposed SSA-BOA-CNN with different deep learning and machine learning models.

The performance of each model across these metrics is shown by the values in Table 16. For instance, the Proposed SSA-BOA-CNN model achieved a precision of 0.9923, recall of 0.9952, AUC-ROC of 0.9931, F1-Score of 0.9860, and accuracy of 0.9910. Similarly, other models such as RNN-AD, GRU, GA, DNN, DAE, BiLSTM and SVM are evaluated based on these metrics. Together, these measures offer information on each model's dependability and efficacy within the framework of the classification task. Figure 13 illustrates the efficacy analysis of the proposed SSA-BOA-CNN with different deep learning and machine learning models.

Table 14 Performance comparison of the proposed SSA-BOA-CNN with existing models

Sl.No	ReferenceS	Author & Year	Algorithm	Dataset	Features	Accuracy
1	[6]	Kumar et al. & 2025	DVQ-VAE	CICIDS2017	-	98.92%
2	[27]	Sethi et al. & 2023	ACO + FDRL	ISOT-CID, NSL-KDD	41	98.9%
3	[26]	Kunhare et al. & 2022	GWA + LR + DT	NSL-KDD	20	99.44%
4	[38]	Moizuddin et al. & 2022	GMGWA + ENAE	NSL-KDD, BoT-IoT	20	99.9%
5	[28]	Alazzam et al. & 2020	PIO	KDDCUP99, NSL-KDD, UNSW-NB15	7, 5, 5	99.40%
6	[30]	Nagaraja et al. & 2020	UTTAMA	KDDCUP99	19	99.95%
7	[29]	Mebawondu et al. & 2020	ANN + MLP	UNSW-NB15	41	76.96%
8	[37]	Lee et al. & 2022	OSS-BSMOTE	NSL-KDD UNSW-NB15 CICIDS2017	-	94.58%
9	[32]	Kumar et al. & 2024	CNN-INCA-MD	IoT malware dataset	-	96.98%
10	[35]	Smmarwar et al. & 2022	WFS- RF-GreedySW-RF-DT-SVM	CIC-InvesAnd-Mal2019	-	91.80%
11	[36]	Gupta et al. & 2022	OEL-AMD	CICInvesAnd-Mal2019	-	96.95%
12	[23]	Kumar et al. & 2024	DLTHF	SDN Dataset	-	98.54%
13	[41]	Momand et al. & 2024	ABCNN	Edge-IoTset IoTID20 ToN_IoT CIC-IDS2017	-	98.02%
14	*	**This Paper**	**SSA-BOA-CNN**	**CICDDoS2019 CICIDS2018**	**77 80**	**99.16% 99.10%**

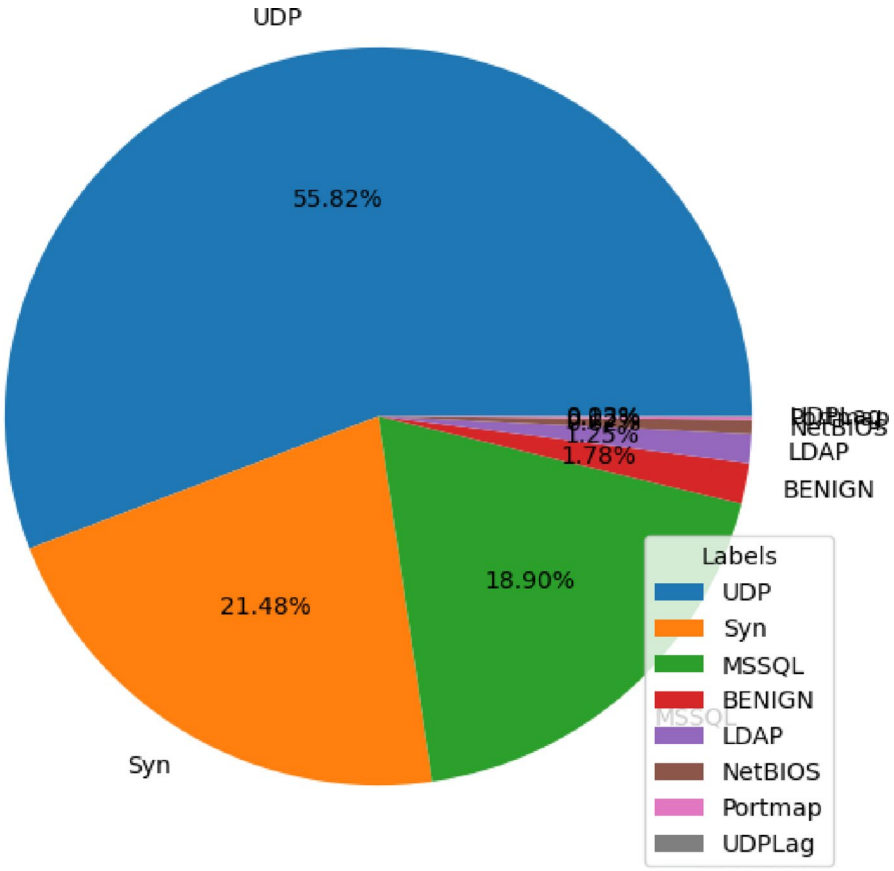


Fig. 3 Distribution of Labels for CICIDDoS2019 Dataset

5 Conclusion

This research provides an innovative method combining the SSA-BOA feature selection technique with a hybrid methodology that incorporates a Convolutional Neural Network (CNN) for dataset classification. The results demonstrate that SSA-BOA effectively selects an optimal subset of 55 features from an initial pool of 77, which significantly enhances the performance of the CNN. This research introduces the SSA-BOA-CNN system, a deep learning-based algorithm designed for DDoS attack detection. The adoption of SSA-BOA optimizes feature selection and improves CNN classification accuracy. Furthermore, the study explores the influence of class imbalance on key performance metrics, including precision, accuracy, recall, and F1 score. The proposed SSA-BOA-CNN approach achieves an accuracy of 99.16%, outperforming several contemporary metaheuristic algorithms.

In future we plan to optimize the algorithm to improve computational efficiency by reducing its complexity, optimizing hyperparameters, and exploring the use of hardware accelerators such as GPUs or TPUs. Additionally, we aim to evaluate the

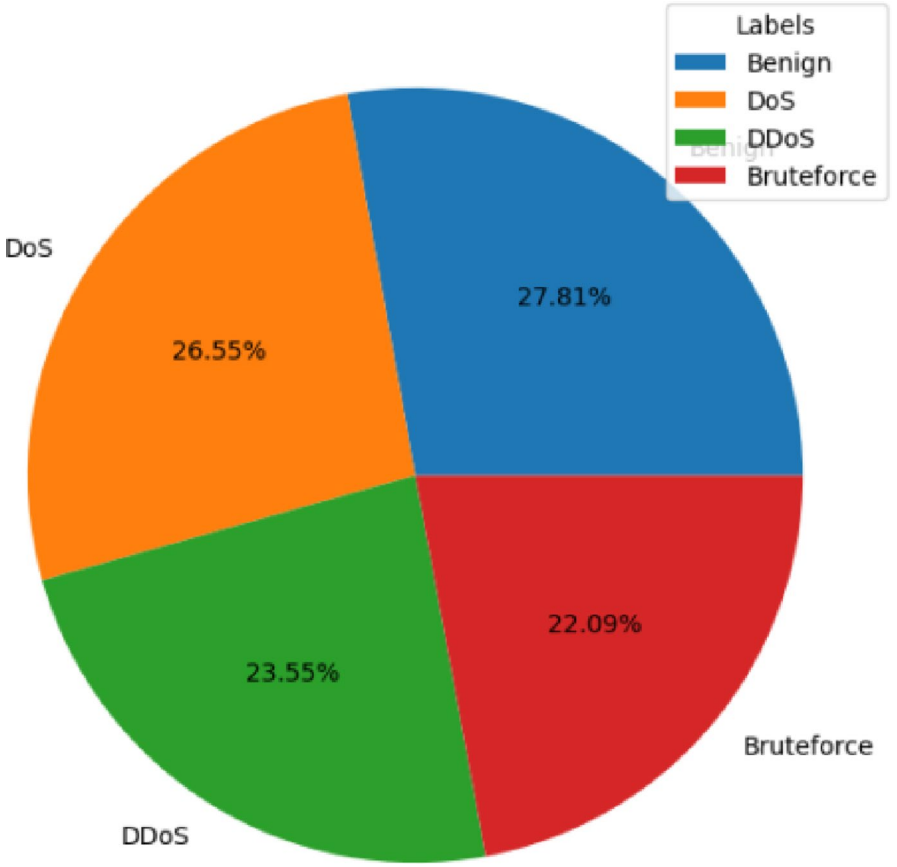


Fig. 4 Distribution of Labels for CICIDS2018 Dataset

model on a broader range of real-world datasets, including newer DDoS datasets and those from different network environments, to assess its generalizability and robustness. Further research will also involve combining and analyzing existing datasets to identify the most relevant features amidst the diverse challenges posed by various environments.

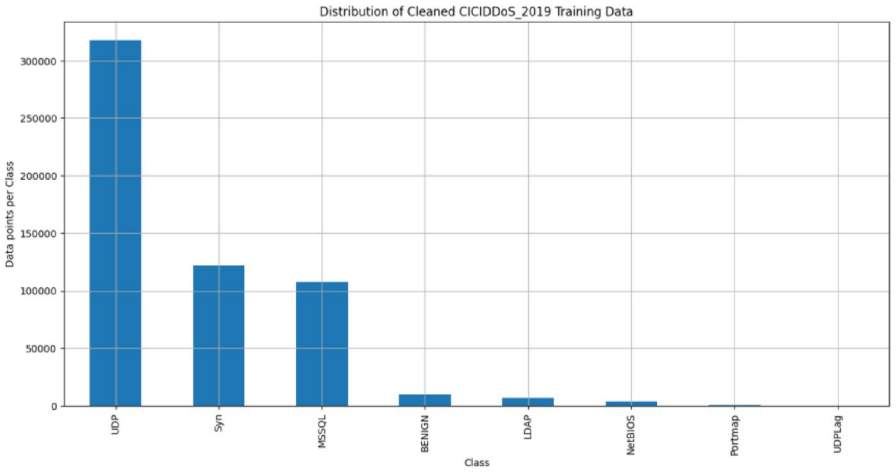


Fig. 5 Distribution of cleaned CICIDDoS2019 Training Data

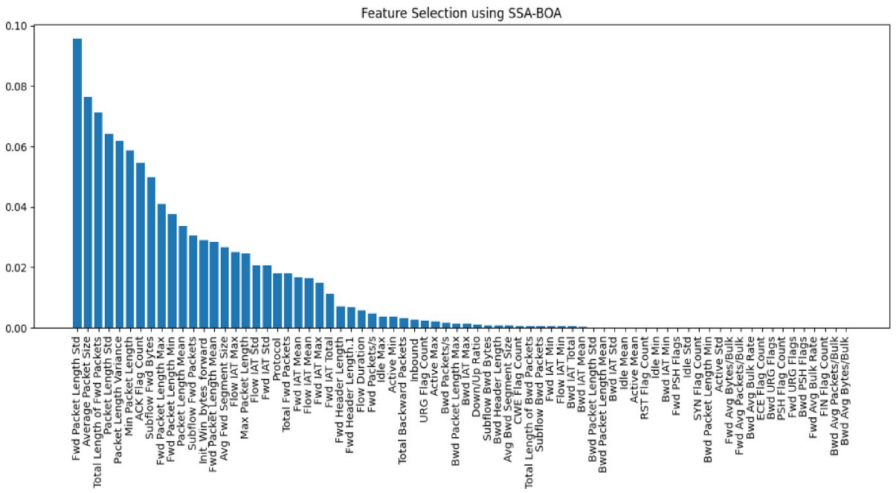


Fig. 6 Selected Features for CICIDDoS2019 Dataset using SSA-BOA method

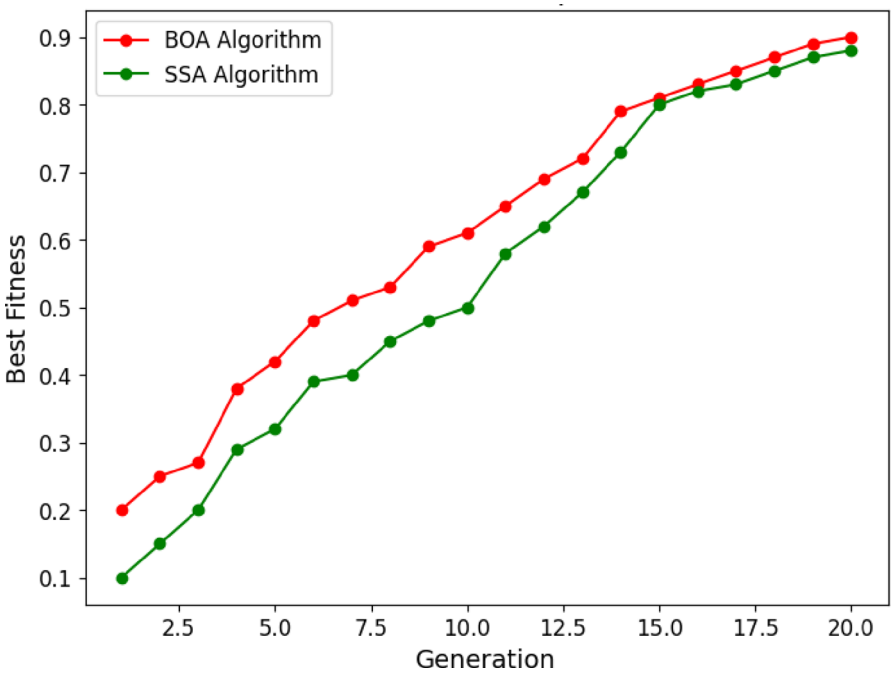
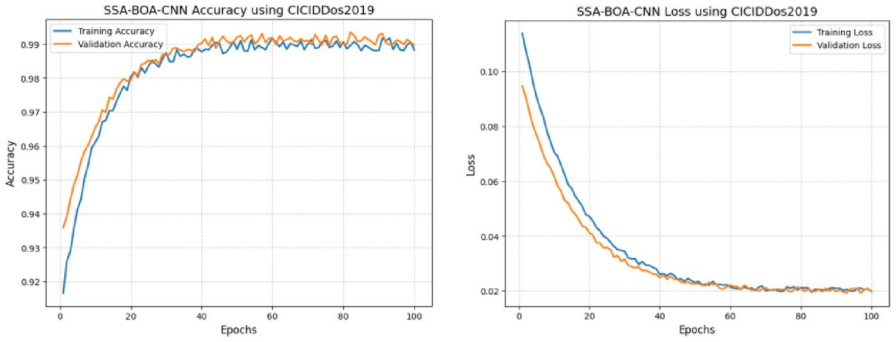
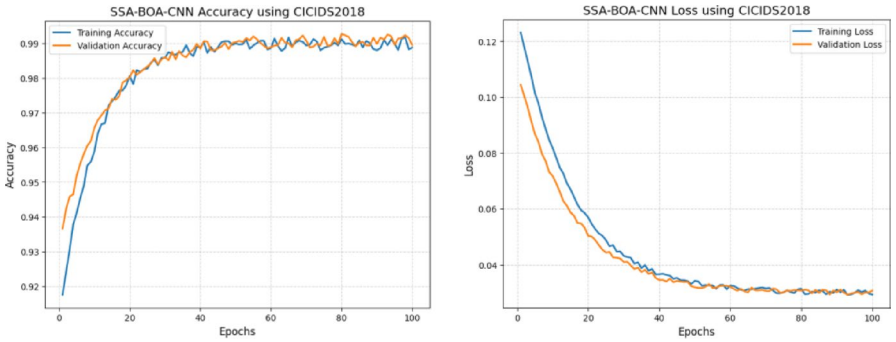


Fig. 7 Comparison graph for Best Fitness



(a) (b)

Fig. 8 (**a**) SSA-BOA-CNN IDS Model Accuracy and (**b**) Loss using CICIDDoS2019 Dataset



(a) (b)

Fig. 9 (**a**) SSA-BOA-CNN IDS Model Accuracy and (**b**) Loss using CICIDS2018 Dataset

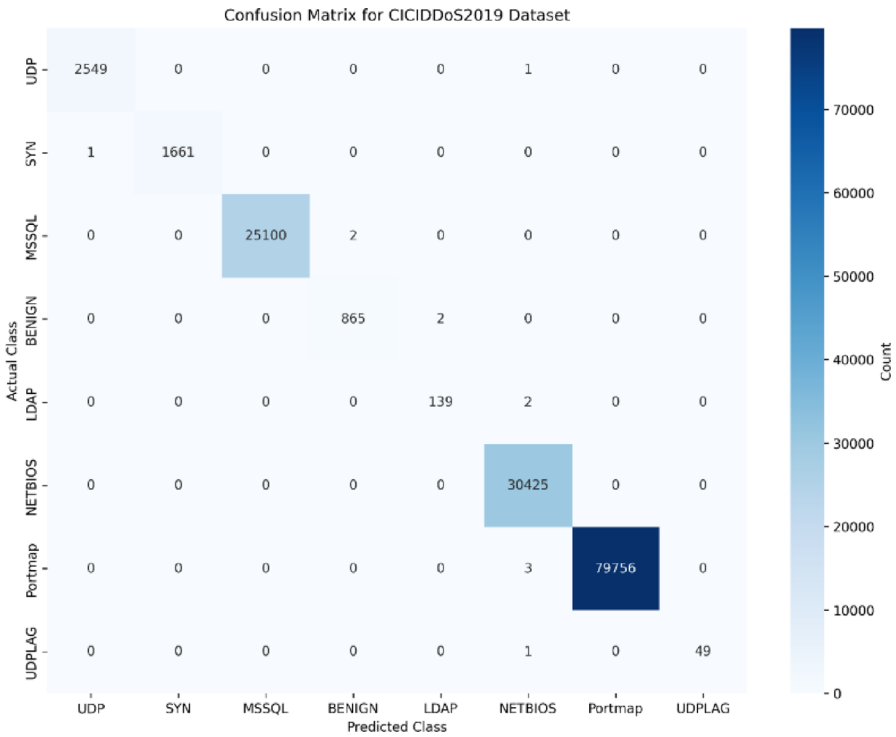


Fig. 10 Confusion Matrix for CICIDDoS2019 Dataset

Table 15 Efficacy analysis of the proposed SSA-BOA-CNN with different methods using CICIDDoS2019 dataset

Method	Precision	Recall	AUC-ROC	F1-Score	Accuracy
Proposed SSA-BOA-CNN	0.9867	0.9952	0.9985	0.9868	0.9916
RNN-AD	0.9253	0.9881	0.9788	0.9548	0.9537
GRU	0.9342	0.9842	0.9774	0.9474	0.9654
GA	0.9445	0.9755	0.974	0.9732	0.9795
DNN	0.9261	0.9657	0.9697	0.9556	0.9312
DAE	0.9678	0.9785	0.9867	0.9589	0.9524
BiLSTM	0.9638	0.9885	0.9776	0.9688	0.9728
SVM	0.9687	0.9531	0.9738	0.9671	0.9682

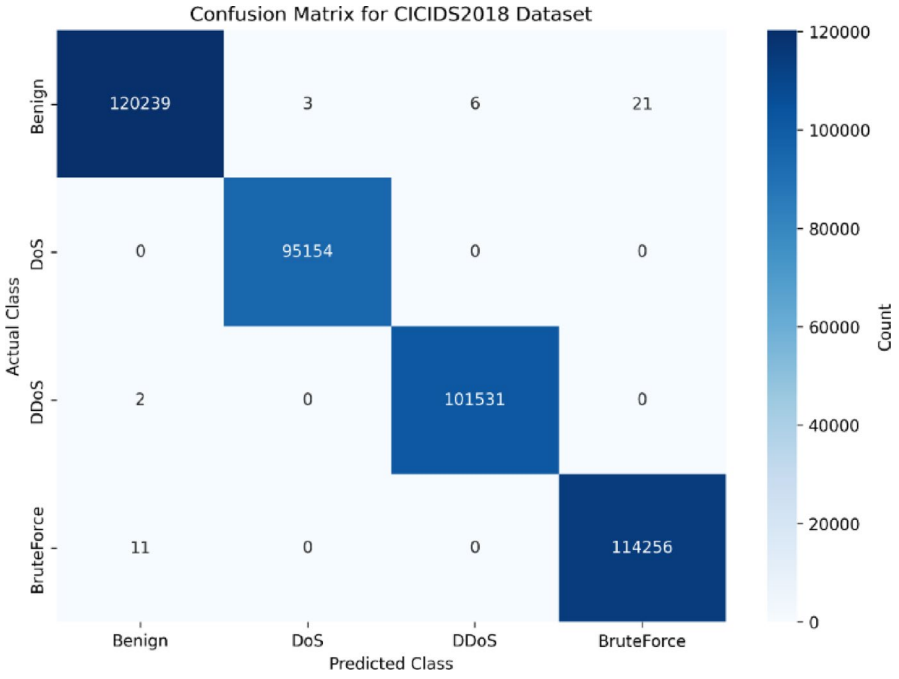


Fig. 11 Confusion Matrix for CICIDS2018 Dataset

Table 16 Efficacy analysis of the proposed SSA-BOA-CNN with different methods using CICIDS2018 dataset

Method	Precision	Recall	AUC-ROC	F1-Score	Accuracy
Proposed SSA-BOA-CNN	0.9923	0.9952	0.9931	0.9860	0.9910
RNN-AD	0.9248	0.9856	0.9753	0.9545	0.9595
GRU	0.9348	0.9839	0.9782	0.9428	0.9604
GA	0.9445	0.9755	0.9740	0.9732	0.9795
DNN	0.9252	0.9662	0.9688	0.9565	0.9395
DAE	0.9670	0.9705	0.9817	0.9598	0.9662
BiLSTM	0.9608	0.9814	0.9756	0.9629	0.9707
SVM	0.9674	0.9513	0.9755	0.9647	0.9654

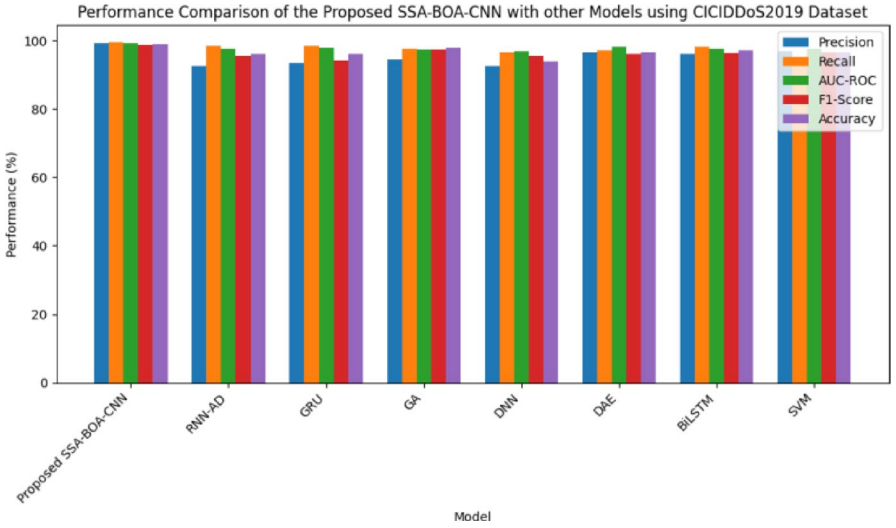


Fig. 12 Efficacy Analysis of the Proposed SSA-BOA-CNN with Different Methods using CICIDDoS2019 Dataset

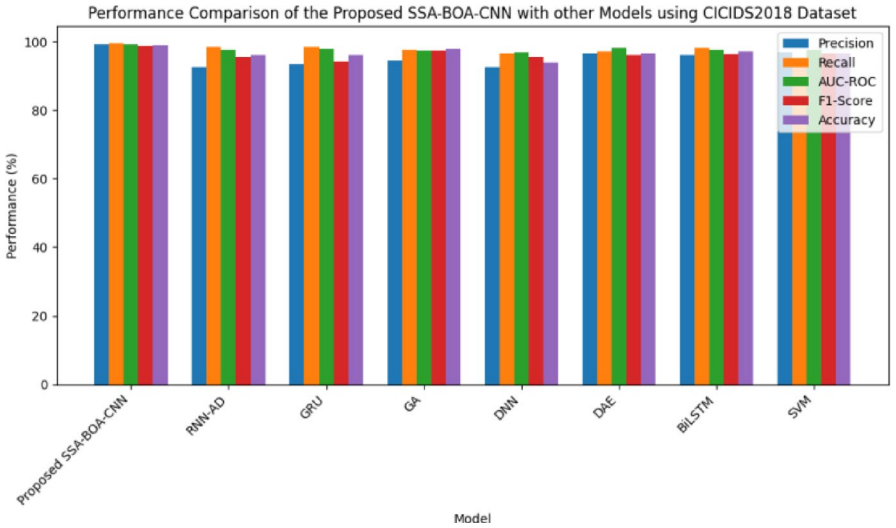


Fig. 13 Efficacy Analysis of the Proposed SSA-BOA-CNN with Different Models using CICIDS2018 Dataset

Acknowledgements The authors gratefully acknowledge Vellore Institute of Technology, Chennai, India for their support and resources in conducting this research.

Author Contributions J.S. conceived the idea, J.S. and P.K. designed the experiments, J.S. and P.K. analysed the results. J.S. wrote the article; P.K. supervised and reviewed the manuscript. All authors reviewed the manuscript.

Data Availability Supplementary data for experiments, we used the CICIDDoS2019 dataset, which can be accessed on online at https://www.unb.ca/cic/datasets/ddos-2019.html and CICIDS2018 dataset, which can be accessed on online at https://www.unb.ca/cic/datasets/ids-2018.htmlSource code can be found online at https://github.com/Sherin756/Leveraging_IDS.

Code Availability Source code can be found online at https://github.com/Sherin756/Leveraging_IDS.

Declarations

Competing Interests The authors declare no competing interests.

References

1. Abd Elaziz, M., Al-qaness, M.A.A., Dahou, A., Ibrahim, R.A., El-Latif, A.A.A.: Intrusion detection approach for cloud and IoT environments using deep learning and Capuchin Search Algorithm, *Adv. Eng. Softw.*, vol. 176, no. December 2023, (2022). https://doi.org/10.1016/j.advengsoft.2022.103402
2. Ajagbe, S.A., Awotunde, J.B., Florez, H.: Ensuring intrusion detection for IoT services through an improved CNN. SN Comput. Sci. (2023). https://doi.org/10.1007/s42979-023-02448-y
3. Smmarwar, S.K., Gupta, G.P., Kumar, S.: Android malware detection and identification frameworks by leveraging the machine and deep learning techniques: A comprehensive review. Telemat Inf. Rep **14**, 100130 (2024). https://doi.org/10.1016/j.teler.2024.100130

4. Liu, L., Wang, P., Lin, J., Liu, L.: Intrusion detection of imbalanced network traffic based on machine learning and deep learning. IEEE Access **9**, 7550–7563 (2021). https://doi.org/10.1109/ACCESS.2020.3048198
5. Chou, D., Jiang, M.: A survey on Data-driven network intrusion detection. ACM Comput. Surv. (2022). https://doi.org/10.1145/3472753
6. Kumar, P., Javeed, D., Islam, A.K.M.N., Robert, X., Luo: DeepSecure: A computational design science approach for interpretable threat hunting in cybersecurity decision making, *Decis. Support Syst.*, vol. 188, no. November p. 114351, 2025, (2023). https://doi.org/10.1016/j.dss.2024.114351
7. Diwan, T.D., et al.: Feature entropy Estimation (FEE) for malicious IoT traffic and detection using machine learning. Mobile Information Systems (2021). https://doi.org/10.1155/2021/8091363
8. Saba, T., Rehman, A., Sadad, T., Kolivand, H., Bahaj, S.A.: Anomaly-based intrusion detection system for IoT networks through deep learning model. Comput. Electr. Eng. **99**, 107810 (2022). https://doi.org/10.1016/j.compeleceng.2022.107810
9. Hnamte, V., Nhung-Nguyen, H., Hussain, J., Hwa-Kim, Y.: A Novel Two-Stage Deep Learning Model for Network Intrusion Detection: LSTM-AE, *IEEE Access*, vol. 11, no. March, pp. 37131–37148, (2023). https://doi.org/10.1109/ACCESS.2023.3266979
10. Ahmim, A., Maazouzi, F., Ahmim, M., Namane, S., Ben Dhaou, I.: Distributed Denial of Service Attack Detection for the Internet of Things Using Hybrid Deep Learning Model, *IEEE Access*, vol. 11, no. August, pp. 119862–119875, (2023). https://doi.org/10.1109/ACCESS.2023.3327620
11. G.S.R., E.S., Azees, M., Rayala, C.H., Vinodkumar, Parthasarathy, G.: Hybrid optimization enabled deep learning technique for multi-level intrusion detection. Adv. Eng. Softw. **173**, 103197 (2022). https://doi.org/10.1016/j.advengsoft.2022.103197
12. Halbouni, A., Gunawan, T.S., Habaebi, M.H., Halbouni, M., Kartiwi, M., Ahmad, R.: CNN-LSTM: Hybrid Deep Neural Network for Network Intrusion Detection System, *IEEE Access*, vol. 10, no. August, pp. 99837–99849, (2022). https://doi.org/10.1109/ACCESS.2022.3206425
13. Hnamte, V., Hussain, J.: DCNNBiLSTM: An efficient hybrid deep Learning-Based intrusion detection system. Telemat Inf. Rep **10**, 100053 (2023). https://doi.org/10.1016/j.teler.2023.100053
14. Sanju, P.: Enhancing intrusion detection in IoT systems: A hybrid metaheuristics-deep learning approach with ensemble of recurrent neural networks. Journal of Engineering Research , 100122 (2023). https://doi.org/10.1016/j.jer.2023.100122
15. Mahalingam, A., Perumal, G., Subburayalu, G., Albathan, M.: ROAST-IoT: A Novel Range-Optimized Attention Convolutional, (2023)
16. Ibor, A.E., Okunoye, O.B., Oladeji, F.A., Abdulsalam, K.A.: Novel hybrid model for intrusion prediction on cyber physical systems' communication networks based on Bio-inspired deep neural network structure. J. Inf. Secur. Appl. **65**, 103107 (2022). https://doi.org/10.1016/j.jisa.2021.103107
17. Mahboob, A.S., Shahhoseini, H.S., Ostadi, M.R., Moghaddam, Yousefi,, S.: A coronavirus herd immunity optimizer for intrusion detection system. 2021 29th Iran. Conf. Electr. Eng. ICEE 2021. **pp 579–585** (2021). https://doi.org/10.1109/ICEE52715.2021.9544165
18. Mohammad, A.H., Alwada'n, T., Almomani, O., Smadi, S., ElOmari, N.: Bio-inspired hybrid feature selection model for intrusion detection. Comput. Mater. Contin **73**(1), 133–150 (2022). https://doi.org/10.32604/cmc.2022.027475
19. Gaber, T., Awotunde, J.B., Folorunso, S.O., Ajagbe, S.A., Eldesouky, E.: Industrial internet of things intrusion detection method using machine learning and optimization techniques. Wireless Communications and Mobile Computing (2023). https://doi.org/10.1155/2023/3939895
20. Akshay Kumaar, M., Samiayya, D., Vincent, P.M.D.R., Srinivasan, K., Chang, C.Y., Ganesh, H.: A Hybrid Framework for Intrusion Detection in Healthcare Systems Using Deep Learning, *Front. Public Heal.*, vol. 9, no. January, pp. 1–18, (2022). https://doi.org/10.3389/fpubh.2021.824898
21. Alkahtani, H., Aldhyani, T.H.H., Al-Yaari, M.: Adaptive Anomaly Detection Framework Model Objects in Cyberspace. Applied Bionics and Biomechanics (2020). https://doi.org/10.1155/2020/6660489
22. Saheed, Y.K., Arowolo, M.O.: Efficient cyber attack detection on the internet of medical Things-Smart environment based on deep recurrent neural network and machine learning algorithms. IEEE Access **9**, 161546–161554 (2021). https://doi.org/10.1109/ACCESS.2021.3128837
23. Kumar, P., Jolfaei, A., Najmul, A.K.M., Islam: An enhanced Deep-Learning empowered Threat-Hunting Framework for software-defined Internet of Things, *Comput. Secur.*, vol. 148, no. September p. 104109, 2025, (2024). https://doi.org/10.1016/j.cose.2024.104109

24. Smmarwar, S.K., Gupta, G.P., Kumar, S.: Research trends for malware and intrusion detection on network systems: A topic modelling approach. In: Gupta, B.B. (ed.) Advances in Malware and Data-Driven Network Security, pp. 19–40. IGI Global, Hershey, PA, USA (2022). https://doi.org/10.4018/978-1-7998-7789-9.ch002
25. Albakri, A., Alhayan, F., Alturki, N., Ahamed, S., Shamsudheen, S.: Metaheuristics with deep learning model for cybersecurity and android malware detection and classification. Applied Sciences (2023). https://doi.org/10.3390/app13042172
26. Kunhare, N., Tiwari, R., Dhar, J.: Intrusion detection system using hybrid classifiers with meta-heuristic algorithms for the optimization and feature selection by genetic algorithm, *Comput. Electr. Eng.*, vol. 103, no. May p. 108383, 2022, (2021). https://doi.org/10.1016/j.compeleceng.2022.108383
27. Vadigi, S., Sethi, K., Mohanty, D., Das, S.P., Bera, P.: Federated reinforcement learning based intrusion detection system using dynamic attention mechanism. J. Inf. Secur. Appl. **78**, 103608 (2023). https://doi.org/10.1016/j.jisa.2023.103608
28. Alazzam, H., Sharieh, A., Sabri, K.E.: A feature selection algorithm for intrusion detection system based on pigeon inspired optimizer. Expert Systems with Applications (2020). https://doi.org/10.1016/j.eswa.2020.113249
29. Mebawondu, J.O., Alowolodu, O.D., Mebawondu, J.O., Adetunmbi, A.O.: Network intrusion detection system using supervised learning paradigm. Sci. Afr **9**, e00497 (2020). https://doi.org/10.1016/j.sciaf.2020.e00497
30. Nagaraja, A., Uma, B., kumar Gunupudi, R.: UTTAMA: An intrusion detection system based on feature clustering and feature transformation. Found. Sci. **25**(4), 1049–1075 (2020). https://doi.org/10.1007/s10699-019-09589-5
31. Smmarwar, S.K., Gupta, G.P., Kumar, S.: XAI-AMD-DL: An explainable AI approach for android malware detection system using deep learning. Proc. - 2023 IEEE World Conf. Appl. Intell. Comput. AIC 2023. 423–428 (2023). https://doi.org/10.1109/AIC57670.2023.10263974
32. Smmarwar, S. K., Gupta, G. P., Kumar, S., Malware Detection Framework Based on Iterative Neighborhood Component Analysis for Internet of Medical Things BT - Biomedical Engineering Science and Technology, Singh, B. K., Sinha, G. R., and Pandey, R. (eds.), Cham: Springer Nature Switzerland, pp. 98–106. (2024)
33. Daniel, A., Deebalakshmi, R., Thilagavathy, R., Kohilakanagalakshmi, T., Janakiraman, S., Balusamy, B.: Optimal feature selection for malware detection in cyber physical systems using graph convolutional network. Comput. Electr. Eng. **108**, 108689 (2023). https://doi.org/10.1016/J.COMPELECENG.2023.108689
34. Smmarwar, S.K., Gupta, G.P., Kumar, S.: AI-empowered malware detection system for industrial internet of things. Comput. Electr. Eng. **108**, 108731 (2023). https://doi.org/10.1016/j.compeleceng.2023.108731
35. Smmarwar, S.K., Gupta, G.P., Kumar, S.: A Hybrid Feature Selection Approach-Based Android Malware Detection Framework Using Machine Learning Techniques BT - Cyber Security, Privacy and Networking, D. P., Agrawal, N., Nedjah, B. B., Gupta, Martinez Perez, G. (eds.), Singapore: Springer Nature Singapore, pp. 347–356. (2022)
36. Smmarwar, S.K., Gupta, G.P., Kumar, S., Kumar, P.: An optimized and efficient android malware detection framework for future sustainable computing. Sustain. Energy Technol. Assessments **54**, 102852 (2022). https://doi.org/10.1016/j.seta.2022.102852
37. Lee, B.S., Kim, J.W., Choi, M.J.: Experimental comparison of hybrid sampling methods for an efficient NIDS. APNOMS 2022–23rd Asia-Pacific Netw. Oper. Manag Symp. Data-Driven Intell. Manag Era Beyond 5G. **pp 1–4** (2022). https://doi.org/10.23919/APNOMS56106.2022.9919939
38. Moizuddin, M.D., Jose, M.V.: A bio-inspired hybrid deep learning model for network intrusion detection. Knowledge-Based Syst **238**, 107894 (2022). https://doi.org/10.1016/j.knosys.2021.107894
39. Almomani, O.: SS symmetry Detection System Based on PSO, GWO, FFA and, *A Featur. Sel. Model Netw. Intrusion Detect. Syst. Based PSO, GWO, FFA GA Algorithms*, vol. 33, no. 32, pp. 1–22, (2020)
40. Geng, Z., Li, X., Ma, B., Han, Y.: Improved Convolution neural network integrating attention based deep sparse auto encoder for network intrusion detection. Appl. Intell. **55**(2), 1–17 (2025). https://doi.org/10.1007/s10489-024-05872-6
41. Momand, A., Jan, S.U., Ramzan, N.: Attention-Based convolutional neural network for intrusion detection in IoT networks. Wirel. Pers. Commun. **136**(4), 1981–2003 (2024). https://doi.org/10.1007/s11277-024-11260-7

42. Saheb, M.C.P., Yadav, M.S., Babu, S., Pujari, J.J., Maddala, J.B.: A Review of DDoS Evaluation Dataset: CICDDoS2019 Dataset BT - Energy Systems, Drives and Automations, J. R., Szymanski, C. K., Chanda, P. K., Mondal, Khan, K. A. (eds.), Singapore: Springer Nature Singapore, pp. 389–397. (2023)
43. Hammad, M., Hewahi, N., Elmedany, W.: MMM-RF: A novel high accuracy multinomial mixture model for network intrusion detection systems. Computers & Security (2022). https://doi.org/10.1016/j.cose.2022.102777
44. Aljehane, N.O., et al.: Golden jackal optimization algorithm with deep learning assisted intrusion detection system for network security. Alexandria Engineering Journal **86**, 415–424 (2024). https://doi.org/10.1016/j.aej.2023.11.078
45. Nasir, M.H., Khan, S.A., Khan, M.M., Fatima, M.: Swarm intelligence inspired intrusion detection Systems — A systematic literature review. Comput. Networks **205**, 108708 (2022). https://doi.org/10.1016/j.comnet.2021.108708
46. Akgun, D., Hizal, S., Cavusoglu, U.: A new DDoS attacks intrusion detection model based on deep learning for cybersecurity. Comput. Secur **118**, 102748 (2022). https://doi.org/10.1016/j.cose.2022.102748
47. Selvapandian, D., Santhosh, R.: Deep learning approach for intrusion detection in IoT-multi cloud environment. Autom. Softw. Eng. **28**(2), 1–17 (2021). https://doi.org/10.1007/s10515-021-00298-7

Authors and Affiliations

R. C. Jeyavim Sherin[1] · K. Parkavi[1]

✉ K. Parkavi
parkavi.k@vit.ac.in

[1] School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, India