



Optimized common features selection and deep-autoencoder (OCFSDA) for lightweight intrusion detection in Internet of things

Uneneibotejit Otokwala¹ · Andrei Petrovski¹ · Harsha Kalutarage¹

Published online: 30 April 2024
© The Author(s) 2024

Abstract

Embedded systems, including the Internet of things (IoT), play a crucial role in the functioning of critical infrastructure. However, these devices face significant challenges such as memory footprint, technical challenges, privacy concerns, performance trade-offs and vulnerability to cyber-attacks. One approach to address these concerns is minimising computational overhead and adopting lightweight intrusion detection techniques. In this study, we propose a highly efficient model called optimized common features selection and deep-autoencoder (OCFSDA) for lightweight intrusion detection in IoT environments. The proposed OCFSDA model incorporates feature selection, data compression, pruning, and deparameterization. We deployed the model on a Raspberry Pi4 using the TFLite interpreter by leveraging optimisation and inferencing with semi-supervised learning. Using the MQTT-IoT-IDS2020 and CIC-IDS2017 datasets, our experimental results demonstrate a remarkable reduction in the computation cost in terms of time and memory use. Notably, the model achieved an overall average accuracies of 99% and 97%, along with comparable performance on other important metrics such as precision, recall, and F1-score. Moreover, the model accomplished the classification tasks within 0.30 and 0.12 s using only 2KB of memory.

Keywords Intrusion detection · Feature selection · Internet of things · Computational cost

1 Introduction

The IoT refers to the interconnection of everyday objects with the Internet. Roy et al. [49] define it as the connection of uniquely identifiable, diverse embedded devices. This connection has led to remarkable growth in the IoT over the years, transcending various facets of human life and finding applications in critical infrastructures, smart homes, healthcare facilities, industries, and even the environment. It has revolutionized the acquisition, processing, and use of data. However, this widespread adoption and application has also introduced some associated risks. These risks stem from an initial lack of emphasis on security during manufacturing, as manufacturers were mostly driven by market consider-

ations and competition [10, 35]. In addition, IoT systems have become targets because they are susceptible to diverse security issues, thus raising significant privacy concerns for end-users in sectors such as critical infrastructures and smart homes [8]. Controls and strategies such as encryption, access control, trust policies, and management controls have been developed to counter these attacks. However, the advent of Intrusion Detection Systems (IDS), especially AI-based IDS, has marked a significant stride in the future. IDS can detect and prevent malicious activities efficiently and effectively, thus representing an additional layer of defence against unauthorized intrusion. They can be categorized as either network-based (NIDS) or host-based, scrutinizing network traffic and system baselines to detect anomalies. Conventional IDS are unsuitable for IoT because of their unique characteristics, as noted by [49]. This is because IoT and other embedded systems have limited hardware resources such as Random Access Memory (RAM) and flash memory [66]. These limitations impose constraints on its ability to resolve pertinent security issues effectively. For instance, the standard technique of data feature analysis could escalate the computational costs of IoT devices. In addition, the incapacity could further be compounded by the redundancy

✉ Uneneibotejit Otokwala
u.otokwala@rgu.ac.uk

Andrei Petrovski
a.petrovski@rgu.ac.uk

Harsha Kalutarage
h.kalutarage@rgu.ac.uk

¹ School of Computing, Robert Gordon University, Aberdeen, Scotland, UK

of certain features in a dataset, which may impact overall classification outcomes. This underlines why a Lightweight Intrusion Detection System (LIDS) is better tailored for IoT and other embedded systems.

The deployment of Lightweight Intrusion Detection Systems (LIDS) offers several benefits, including reduced computational overhead, reduced power consumption, and optimized computing, communication, and storage capacities. According to [48], LIDS is a small but potent and flexible software tool with a modest system footprint that can be promptly integrated as a permanent security solution. The authors further emphasize that given the evolving nature of security scenarios, a LIDS should be adaptable to seamlessly integrate into the existing network security infrastructure, especially when its distinctive challenges in terms of protocol and structure are considered. In fact, the need for effective LIDS for the IoT is unique. This is because of the network protocols and other properties, such as IEEE 802.15.4, 6LoWPAN, RPL, and CoAP, used in the IoT network. These properties make the IoT very distinct from traditional network protocols. In addition, these properties also add to the constraints on its intrusion detection mechanisms [19].

This research introduces a novel LIDS. However, it is imperative to acknowledge the recent increase in scholarly research focusing on applying Machine Learning (ML) techniques for detecting security breaches across diverse devices. This heightened interest stems from the notable efficacy of the ML techniques in uncovering intrusions. Nevertheless, it is crucial to recognize that these approaches incur considerable computational overhead when implemented verbatim. Notably, deploying ML methods on resource-constrained devices, particularly within the IoT paradigm, presents formidable challenges. The intricacies of applying conventional and ML-based intrusion detection systems underscore the need for a tailored approach. Consequently, this study comprehensively addresses the research questions, taking cognisance of the complexities intrinsic to implementing intrusion detection systems.

1. RQ1: How can a lightweight Intrusion Detection System be achieved for resource-constrained devices such as the IoT with efficient and effective classification ability without significantly compromising accuracy?
2. RQ2: How can the proposed model be made resilient against adversarial attacks such as label poisoning?
3. RQ3: How will the result compare with the output of the benchmark dataset and existing studies regarding computation cost?

To address these research questions, this study proposes an Optimized Feature Selection and Deep Neural Network

model for effective lightweight intrusion detection in IoT. It incorporates a number of distinct approaches to achieve the proposed model. The approaches include feature selection, data compression, and quantization. Feature selection is a dimensionality reduction technique that aids in condensing the original feature space by eliminating redundant and less relevant features. This process enhances the efficiency of ML models by focusing on the most informative and significant features, thereby improving model performance and interpretability [60]. Subsequently, the selected features were compressed using the neural network's Long Short-Term Memory Autoencoder (LSTM-AE). The compression technique further reduces the dimensionality of the selected data, minimizes redundancy, and preserves data density and structure [24, 51]. Thereafter, the output of the autoencoder is further optimized through quantization and inferencing to achieve an effective Lightweight Intrusion Detection System. Consequently, this study presents the following novel contributions:

1.1 Contribution

1. We present a novel optimized common features selection and deep-autoencoder (OCFSDA) model for lightweight intrusion detection in the IoT. The proposed model can reduce CPU computational cost, measured in terms of memory usage and computation time, when applied to two benchmark datasets.
2. The proposed model's resilience was also evaluated against adversarial attacks such as label poisoning attacks, and the results show the robustness of the proposed model.
3. In addition to reducing memory footprint and execution time, the proposed model was also able to achieve effective results across different metrics of measurement, especially when compared with other related works on lightweight intrusion detection systems.

1.2 Scope of work

The scope of this study is limited to using existing benchmark datasets to achieve lightweight intrusion detection. To accomplish the scope, emphasis was placed on using feature selection and an LSTM-autoencoder with semi-supervised and supervised learning to implement this work. In addition, for inferencing, the proposed model is deployed on a Raspberry Pi for evaluation.

1.3 Structure of paper

The rest of this paper is structured as follows: Sect. 2 introduces the relevant literature. Section 3 provides an overview of the preliminaries, elucidating the background activities of the methodology. Section 4 elucidates the methods employed for the proposed model. Section 5 discusses and presents the results in relation to the outputs and proposal. Section 6 concludes the paper and outlines potential future work.

2 Related work

Over the years, researchers have developed diverse methodologies for lightweight intrusion detection in the IoT domain. While some techniques have proven effective, others have become outdated due to evolving threats. Therefore, this study split the related works between feature selection, machine learning, and quantisation approaches.

2.1 Feature selection technique for LIDS

Features do help in effective classification of data, however, redundant features can compromise model accuracy. Consequently, numerous researchers have advocated various feature selection methodologies to reduce data dimensions, striving for both lightweights and enhanced classification proficiency. As an illustration, Wang et al. [61] introduced a feature selection framework tailored for high-dimensional datasets. Their approach involved amalgamating outcomes from the chi-square test, maximum information coefficient, and XGBoost feature selection techniques to ascertain feature rankings. Similarly, Neumann et al. [36] presented an ensemble feature selection approach aimed at mitigating biases inherent in individual feature selection methods. In a related development, [4] proposed a hybrid approach for lightweight intrusion detection. This approach combines feature selection with a heterogeneous ensemble of intelligent classifiers. Meanwhile, Jan et al. [19] proposed a method that uses feature selection based on the packet arrival rate of the NSL-KDD dataset to achieve a lightweight intrusion detection system (IDS). The authors fitted a Support Vector Machine (SVM) model to the data, focusing on detecting Denial of Service (DoS) attacks. In another study, Li et al. [28] introduced the Hierarchical and Dynamic Feature Extraction Framework (HDFEF) for network intrusion detection in IoT. They addressed the challenge of effectively representing features in network flows by designing a hierarchical network model that dynamically adjusts the distribution of feature representations for correlated network packets using an attention mechanism. This approach achieved a classification rate of 99.7% by generating discriminant vectors from multiple mappings. Li et al. [29] proposed

a Lightweight Intrusion Detection System that implemented feature selection with a random mutation hill-climbing optimization approach to select the best features. Furthermore, Moukhafi et al. [33] proposed a model that combines the Genetic Algorithm (GA) and SVM algorithms. The GA algorithm was used to select a subset of features obtained through particle swarm optimization (PSO).

In the same vein, Manal et al. [2] proposed an intrusion detection method based on dividing a dataset into subsets based on the attack type. Using an information gain filter, a set of optimal features is selected and combined for model fitting. Furthermore, Halim et al. [12] in their work proposed a Genetic Algorithm (GA)-based feature selection method called GA-based Feature Selection (GbFS) for the reduction of data dimensionality and improvement in intrusion detection. Osanaiye et al. [40] in their work, proposed a feature selection approach that combines three filter methods, namely, Gain ratio, Chi-squared, and ReliefF, in a cluster-based heterogeneous Wireless Sensor Network. Similarly, Subbiah et al. [58] also proposed a novel framework for an Intrusion Detection System (IDS) that uses the Boruta feature selection with grid search random forest (BFS-GSRF) algorithm. In addition, Li et al. [27] in their work considered a deep learning approach that incorporated the random forest technique for creating a training set involving feature selection and grouping. Following this training process, they used an auto-encoder model to predict attacks, consequently reducing detection time and enhancing prediction accuracy. Similarly, Tao et al. [59] proposed a lightweight intrusion detection system based on feature selection and weight using a genetic algorithm. The authors conducted continuous optimization of the SVM parameters to achieve optimal values. In a related development, Jaw and Wang [20] highlighted the challenges of creating trustworthy IDSs, such as accuracy, novel attack detection, labelled training data, and time consumption. Following that, they proposed a Hybrid Feature Selection (HFS) with an ensemble classifier, combining CfsSubsetEval, genetic search, and a rule-based engine to reduce model complexity and enhance learning algorithm performance.

2.2 Machine learning-based lightweight intrusion detection approach

There are a number of studies bordering on using machine learning for dimensionality reduction in order to achieve lightweight intrusion detection. For instance, Mendonca et al. [32] proposed using deep learning-based intelligent systems for Lightweight Intrusion Detection Systems (IDS) in the IoT. The approach uses Sparse Evolutionary Training (SET) and involves weight modification and converting the resulting sparse matrix into different vectors. Lahasan B. and Sama [22] proposed an optimized deep autoencoder model

for a Lightweight Intrusion Detection System in IoT. Their approach incorporated a two-layer optimizer that selected features, training instances, and latent neurons simultaneously. Additionally, Wang et al. [63] presented a Knowledge Distillation approach based on Triplet Convolution Neural Network (KD-TCNN) to enhance the speed and efficiency of anomaly detection in IoT. Furthermore, Shone et al. [55] introduced a non-symmetric autoencoder (NDAE) model, which basically focused on the encoding phase of the Long Short-Term Memory (LSTM) without the decoding phase. Their approach aimed at unsupervised feature learning, and they further introduced a stacked form of NDAEs using the KDD and NSL-KDD datasets. In the same vein, Kim et al. [21] developed a deep autoencoder model for outlier detection, using normal IoT traffic as training data to identify instances that deviate from the learned pattern as malicious.

In another study, Mushtaq et al. [34] presented a two-stage intrusion detection system using an LSTM-autoencoder. Their method involved using the LSTM-autoencoder to identify relevant features and subsequently using a bidirectional LSTM for malicious classification. Using the NSL-KDD dataset, their model achieved an overall accuracy of 89%. Similarly, Xu et al. [65] proposed an IoT intrusion detection system based on an LSTM autoencoder. The authors leveraged the LSTM autoencoder to capture time series features and exploit its feature learning capabilities for classification. Hanafi et al. [13] introduced a new Intrusion Detection System (IDS) model that combines the Improved Binary Golden Jackal Optimization (IBGJO) algorithm with a Long Short-Term Memory (LSTM) network. Naung et al. [57] implemented a lightweight Intrusion Detection System based on machine learning on a Raspberry Pi. Their approach used a correlation-based feature selection (CFS) algorithm to reduce the number of features and employed a J48 classifier for classification, achieving improved performance in terms of overall accuracy and detection speed. In addition, Zhao et al. [69] proposed an approach for IoT LIDS using a lightweight deep neural network (LNN). The approach employs Principal Component Analysis (PCA) to reduce feature dimensionality and uses specialized structures for effective feature extraction at a low computational cost. [38] proposes “Realguard,” a Deep Neural Network-based network intrusion detection system (NIDS) for IoT devices. Mendonca et al. [32] proposed a deep learning model to predict cybersecurity threats. This approach uses a sparse evolutionary training (SET) based prediction model to analyze and detect attacks.

2.3 Quantisation

Quantisation is another technique for reducing the dimension of data, and several studies have introduced it to reduce data dimensionality for IoT intrusion detection. One approach proposed by [54] utilizes quantization techniques

in a Quantized Autoencoder (QAE) model tailored for IoT intrusion detection. Two variants, QAE-u8 and QAE-f16, were designed to make resource-intensive AI models suitable for Edge devices. According to the authors, these models identified various attack types, achieving improved accuracy. Another study by [53] introduced Vector Space Bag of Words (VSBW) for feature extraction and employed Boosted Variance Quantization Neural Networks (BVQNNs) for intrusion detection. The approach also incorporates an MH-RSO algorithm to enhance intrusion anticipation. Evaluation of standard IoT datasets demonstrated the effectiveness of this methodology in bolstering IoT system security. Additionally, [44] proposed a Vector Space Bag of Words (VSBW) combined with Boosted Variance Quantization Neural Networks (BVQNNs) for intrusion detection. The approach integrates the Multi-Hunting Reptile Search Optimization (MH-RSO) algorithm, enhancing intrusion anticipation. Evaluation of standard IoT datasets showed the ability of the methodology to improve IoT security. Furthermore, [62] proposed DL-BiLSTM, a lightweight IoT intrusion detection model that combines DNNs and BiLSTMs for feature extraction. The model employs IPCA for feature reduction and dynamic quantization for computational efficiency, achieving improved detection performance while maintaining lower complexity.

These studies have proposed various approaches to achieve lightweight intrusion detection in resource-constrained devices such as the IoT. However, a prevalent trend in the studies primarily entails deploying the models in edge and cloud environments, while many others implemented their approach on Windows systems. Interestingly, some results could be improved upon and suitable for the IoT ecosystem considering that the devices could be deployed in remote terrains. Therefore, in this study, we have undertaken a comprehensive approach that combines the three approaches to achieve effectiveness and reduce the size of our model while maintaining data density and structure.

3 Preliminaries

Several steps were taken to achieve our proposed model, including dimensionality reduction techniques (feature selection, LSTM autoencoder-based data compression), quantization, pruning, deparameterization, a Bayesian optimization technique, and inferencing.

3.1 Feature selection

Assuming a function f_x which was applied on a dataset T with the following features, $X_1, X_2, X_3, \dots, X_n$ to find the information content relative to the target variable. This approach is helpful in the selection of relevant features and in

the generation of a minimal set of features devoid of redundancies. Feature selection is generally helpful because it enhances generalization and reduces overfitting. However, the danger of using a single feature selection method is the risk of generating suboptimal feature subsets that could impact the detection performance of the learning method [15, 47]. Therefore, we adopted an ensemble of multiple feature selection techniques to rank the features. Our approach leverages the strengths of each technique to create a more robust and effective feature subset.

This study employed feature selection techniques specifically tailored to intrusion detection. We focused on Information Gain, Chi-Squared, and Gini Index methods. These selections were made based on their direct relevance to addressing issues such as (i) disorder or anomalies within traffic patterns, (ii) the critical need for precise data classification in the traffic flow, and (iii) the ability to measure the distribution of values in a feature while considering the probability of a randomly selected instance being misclassified.

3.1.1 Feature selection based on information gain

Information Gain quantifies the information a feature provides to predict the target variable. In intrusion detection, this methodology offers benefits by identifying unique and significant features crucial for accurate detection. In information theory, higher uncertainty equates to lower information content. Therefore, by assigning weights to features and emphasizing the most relevant ones, Information Gain helps reduce the complexity of the dataset. The calculation of the information gain for each feature relies on entropy, represented as $E()$, and these have been extensively discussed in some relevant literature, including [5]. Equations 1 and 2 present the fundamental mathematical principles behind calculating information gain. Nevertheless, we utilized relevant libraries in this study to generate and rank the features.

$$IG(S, x) = E(S) - E(S|x) \quad (1)$$

where $IG(S, x)$ represents the information gain for the dataset, S ; x represents a random variable; $E(S)$ represents the entropy of S ; $E(S|x)$ represents the conditional entropy of S given x .

$$E(S) = - \sum_{i=1}^n P_i \log_2 P_i \quad (2)$$

In information theory, entropy, as defined by Eq. 2, serves as a measure of the uncertainty associated with a random variable. This uncertainty is derived from the probabilities assigned to various outcomes of the random variable. By

comparing the entropy with the conditional probability given a specific feature, the learning algorithm can discern the significance of a feature.

3.1.2 Feature selection based on chi-square

The Chi-Square method is a statistical algorithm utilized for the binning of numerical variables based on the χ^2 statistic to detect significant variations within the dataset. Its primary objective is to assess the association between independent (predictor) and dependent categorical features (response). A higher Chi-Square value suggests a more pronounced reliance on the feature on the response, thereby enhancing its suitability for model training. In the context of a classification task involving a categorical target variable and continuous independent variables, the Chi-Square technique is a feasible feature selection option. A Chi-square test can ascertain whether a feature influences the class label [45]. While Eq. 3 has provided the basic mathematics behind feature ranking using Chi-Squared; in this case, however, relevant libraries were used to rank the features.

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i} \quad (3)$$

where c represents the degree of freedom; O represents the observed values; E represents the expected values.

The degree of freedom (c), observed values (O) and the expected values (E) are parameters used to calculate the Chi-squared statistic, which is then used to measure the extent of association between features and the target variable. The Chi-squared statistic is calculated by comparing the observed frequencies in the contingency table with the expected frequencies, and it quantifies the extent to which the observed frequencies deviate from what would be expected under the assumption of independence. Features with higher Chi-squared values are considered more relevant for classification.

3.1.3 Feature selection based on Gini-Index

The Gini Index was proposed by Breiman in 1984 and has since been widely used in various algorithms, including feature selection, with favorable classification results. It is a method that calculates the decrease in heterogeneity (d_{ij}) at a specific node (j) during a split, and it measures the impurity at nodes. The Gini Index is particularly suitable for binary and continuous numeric values [31].

Assuming a right split as R , a right Gini node is represented as G_R , and a left split is denoted as L , with a left Gini node represented as G_L . Then, the Gini Index (G) and the

decrease in impurity (d_{ij}) at a single node can be calculated as shown in Eq. 4.

$$d_{ij} = G - \left(\frac{N_L}{N} G_L + \frac{N_R}{N} G_R \right) \quad (4)$$

where N represents the number of units in the dataset; N_L and N_R indicates the number of units in the left and right nodes after the split, respectively.

The measure of the feature importance (FI) for the ranking of a feature, K_i in a tree, t , is as defined in Eq. 5 [50].

$$FI_{K_i}(t) = \sum_{j \in J} d_{ij} I(K_i \text{ splits at node } j) \quad (5)$$

3.2 Data compression

Data compression is another approach for reducing the dimensionality of data. It is a feature extraction method that reduces the number of features by replacing the existing features with fewer artificial features, a combination of the existing features. There are several feature extraction techniques, and they include Factor Analysis (FA), Principal Component Analysis (PCA), Singular Value Decomposition (SVD), Linear Discriminant Analysis (LDA), t-distributed Stochastic Neighbor Embedding (t-SNE), Auto-encoder, etc. All the approaches have their advantages and disadvantages, and techniques like PCA lead to the curse of dimensionality because of the need to explain the variance with each principal component. In this study, however, we employed the LSTM-AE approach for data compression to reduce the dimension of our common features selected data. Autoencoders are flexible and adaptable to different types of data and tasks. They can also handle both linear and non-linear relationships in the data. Moreover, autoencoders excel at discerning intricate and abstract features that may evade detection by alternative methodologies. The LSTM-AE architecture comprises two basic components: the encoder and the decoder. According to [6, 23], an LSTM-Autoencoder is a type of Recurrent Neural Network (RNN) that performs its operations in two steps: accepting input sequence prediction and then reconstructing the input. First, it compresses the input data into a lower-dimensional representation using the encoder, and then it recreates the original input from the lower-dimensional representation using the decoder. Because this study primarily focuses on dimensionality reduction to achieve lightweight intrusion detection, we restricted this work to the encoder component. Mathematically representing an LSTM-AE data compression model can be complex because it involves multiple layers containing LSTM cells. Each LSTM cell is equipped with three gates:

the input gate (i_t), output gate (o_t), and forget gate (f_t), which regulate the flow of information within the cell. The input gate determines how much new information should be added to the cell state, the forget gate determines how much of the previous state should be ignored, and the output gate controls the amount of the current state to be outputted. A simple pictorial representation of the LSTM-AE is shown in Fig. 1 with hyperbolic tangent (\tanh) [64] and *sigmoid* [17] as activation functions. The functions *tanh* and *sigmoid* enable LSTM to control the flow of information and sequential data effectively. In this way, the network can selectively remember or forget information over time using the input, forget, and output gates.

The cell state of an LSTM-AE is updated using Eq. 6.

$$C_t = f_t \times C_{t-1} + i_t \times g_t \quad (6)$$

where C_t is the cell state at time, t , f_t is the forget gate at time, t , i_t is the input gate at time, t , and g_t is the input modulation at time, t . The output of an LSTM cell is computed using Eq. 7.

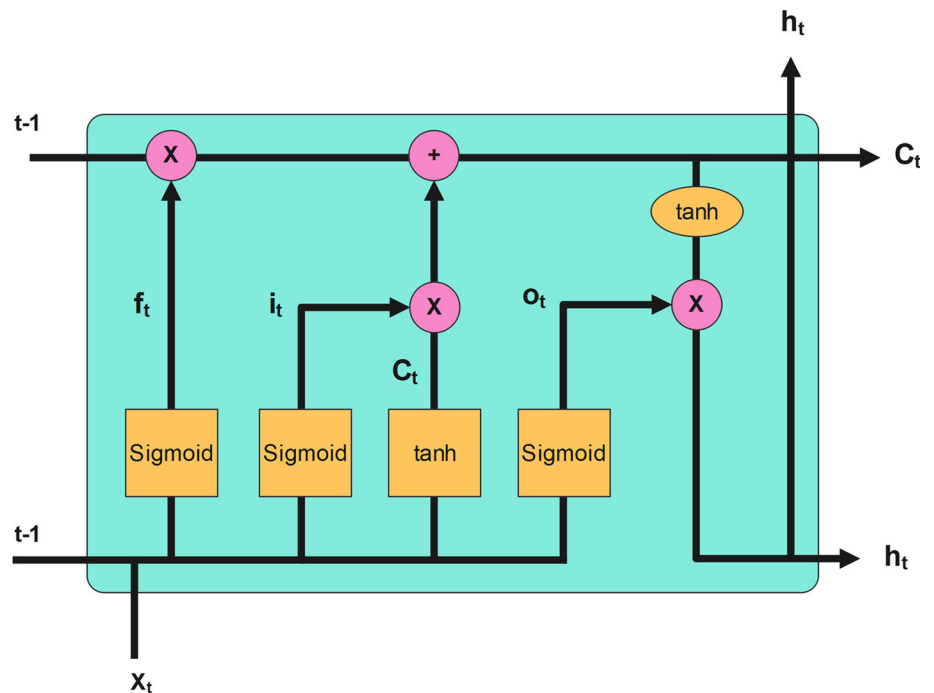
$$h_t = o_t \times \tanh(C_t) \quad (7)$$

where h_t is the output of the cell at time, t , o_t is the output gate at time, t , and *tanh* is the hyperbolic tangent activation function.

3.3 Bayesian optimisation for hyperparameter tuning

Bayesian optimization is an effective approach for hyperparameter tuning in deep learning models. It leverages Bayesian inference and optimization techniques to explore and discover the optimal hyperparameter configurations efficiently. Compared with traditional methods such as grid search, Bayesian optimization addresses the time and computational complexity associated with tuning many hyperparameters. One advantage of Bayesian optimization is its ability to consider varying hyperparameter requirements for accurate predictions on different datasets. This adaptability suits our work well because it allows for exploring different hyperparameter configurations while maximizing model performance. In this study, we focused on tuning the following hyperparameters: learning rate, batch size, number of layers, number of neurons, regularization, and activation functions, as they are crucial for achieving optimal model performance. More importantly, by finding the most suitable hyperparameter configurations, we aim to minimize cost and loss through weight adjustment in deep learning, thus aligning with the fundamental principle of optimizing deep learning models. Overall, Bayesian optimization provides a systematic and

Fig. 1 LSTM Autoencoder showing the encoder and the decoder



efficient framework for hyperparameter tuning in deep learning because it helps unlock better model performance while minimizing computational complexities.

3.4 Evaluation metrics

The metrics used to evaluate our proposed model in this study are Accuracy, Precision, Recall, F1-score, and Receiver Operating Characteristics (ROC). These multi-dimensional evaluation frameworks were used to thoroughly assess the efficacy, performance, and generalization capabilities of the model.

4 Experimental design

This section demonstrates the viability of our proposed OCFSDA lightweight intrusion detection model. The steps used to achieve the proposed model are shown in Fig. 2. Two benchmark datasets were used for the implementation.

4.1 Dataset

In this section, we provide an overview of the process. Initially, the selected features derived from the feature selection procedure are further condensed to reduce dimensionality. The condensed data are subsequently employed for training using a fully connected sequential model. This study utilizes two openly accessible IoT network datasets: the MQTT-IoT-IDS2020 dataset [14] and the CICIDS2017 dataset [52].

These datasets were transformed into binary class datasets. Algorithm II outlines the steps involved in this stage of the research.

4.1.1 MQTT-IoT-IDS2020 dataset

The MQTT-IoT-IDS2020 dataset was generated by simulating the Message Queuing Telemetry Transport (MQTT) protocol. The protocol is extensively employed in machine-to-machine communication within the IoT domain. The simulation encompassed various components such as sensors, a broker, a simulated camera, and an attacker. The simulation captured five scenarios: normal operation, aggressive scan, UDP scan, Sparta SSH brute-force, and MQTT brute-force attack. Pcap files were generated and stored, and features were extracted. The dataset authors claim to have extracted three levels of feature abstraction from the pcap files: packet features, unidirectional flow features, and bidirectional flow features. This study used the bidirectional flow feature dataset and consolidated the target class into binary (benign and malicious). Therefore, the dataset comprised 225,711 observations and 68 features. The class distribution was as follows: 128,025 instances of malicious traffic and 97,681 instances of benign traffic.

4.1.2 CIC-IDS2017 dataset

The creation of the CIC-IDS2017 dataset by the Canadian Institute of Cybersecurity addressed the lack of benchmark datasets that offer insights into traffic diversity, volumes,

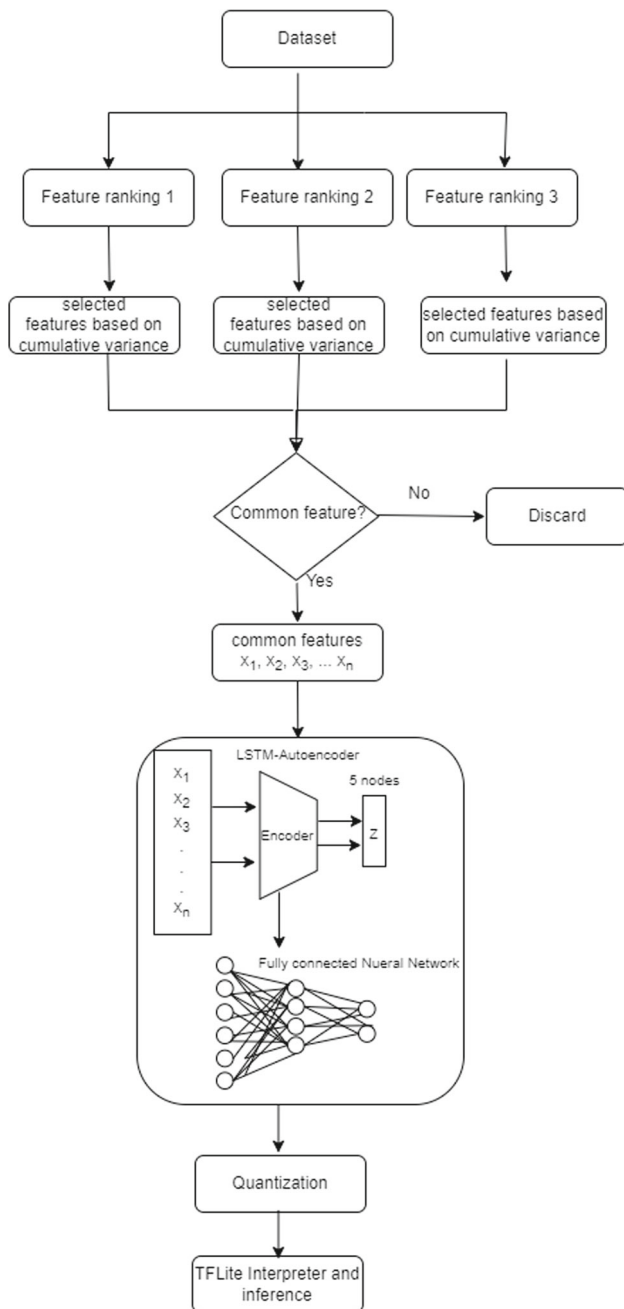


Fig. 2 Steps used to achieve the proposed model

known attacks, and anonymized packet payloads, all reflecting real network infrastructure trends. The dataset was partitioned into two segments. Part 1 encompassed four machines responsible for executing various attacks, whereas Part 2 involved ten machines susceptible to vulnerabilities. The dataset itself had 78 features and 286,467 observations. However, ten of these features either had values of zero or very marginal values, which, after normalization, resulted in NaN values. Consequently, these ten features were excluded from the dataset. This led to the distribution of the

target variable as follows: The “PortScan” instance, representing the attack class, comprised 158,930 observations, while the “Benign” class accounted for 127,537 observations.

4.1.3 Data preprocessing

In the data preprocessing stage, we primarily focus on data cleaning and getting the data ready for further actions. This includes tasks such as converting categorical variables and protocols into one-hot encoding. In addition, we transformed the target class, condensing the multiclass into a binary format, where ‘Benign’ is represented as 0 and the ‘Malicious’ class as 1. Furthermore, we applied min-max normalization to the independent variables. Min-max normalization is a data preprocessing technique that transforms numerical features to a common scale. The essence is to rescale the data so that all features have the same minimum and maximum values, typically within the range of [0, 1]. This helps to preserve the original distribution of the data.

Indeed, high-dimensional data often leads to a deterioration in the performance of learning algorithms, impacting both their accuracy and training duration [37, 67]. This phenomenon, commonly referred to as the “curse of dimensionality,” underscores the importance of feature ranking in selecting concise and informative feature subsets. Such subsets play a pivotal role in enhancing the efficacy of learning algorithms by mitigating the adverse effects associated with high-dimensional data.

4.1.4 Feature ranking and common features subset

In this stage, the features selected from Sect. 3.1 were used following data preprocessing. Using the two datasets (MQTT-IoT-IDS2020 and CIC-IDS2017), the features were ranked in descending order of importance as illustrated in Tables 11 and 12 (“Appendix A and B”). The procedure is outlined in Algorithm 1. The Common Features subset pertains to features shared among the three feature selection techniques, as depicted in Fig. 3. The selection process occurs after ranking the features, which then results in the selection of 17 features, as shown in Table 1. The selection of common features is determined by computing the cumulative variance. Before selection, the cumulative variance of the feature values was calculated, initially starting with 10 values and progressively increasing to 15, 20, and so forth. A threshold was identified when further value additions ceased to impact the cumulative variance. This threshold designates the point at which the cumulative variance plateaus, indicating that additional feature values no longer contribute to any variance increase (i.e., 0 variance).

Consider a dataset T , which contains features $(X_1, X_2, X_3, \dots, X_n)$. The features were ranked using Information Gain, Chi-squared, and Gini-index feature selection tech-

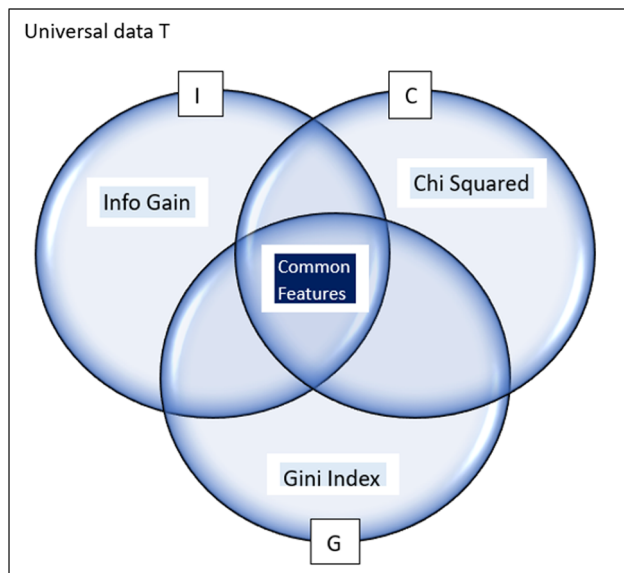


Fig. 3 Features and the area showing common features—intersection of features

Table 1 Common features for MQTT-IoT-IDS2020 and CIC-IDS2017 datasets

S/N	MQTT-IoT-IDS2020 Common Features	CIC-IDS2017 Common Features
1	Total.Length.of.Fwd.Packets	Packet Length Mean
2	Subflow.Fwd.Bytes	Packet Length Std
3	Average.Packet.Size	Average Packet Size
4	Bwd.Packet.Length.Mean	Init_Win_bytes_backward
5	Fwd.Header.Length.l	Init_Win_bytes_forward
6	Fwd.Header.Length	Fwd Packet Length Mean
7	Bwd.Packet.Length.Max	Avg Fwd Segment Size
8	Init_Win_bytes_forward	Fwd Packet Length Max
9	Fwd.Packet.Length.Mean	Avg Bwd Segment Size
10	Avg.Fwd.Segment.Size	Bwd Packet Length Max
11	Fwd.Packet.Length.Max	Bwd Packet Length Min
12	Total.Fwd.Packets	Flow Duration
13	Subflow.Fwd.Packets	Flow IAT Max
14	Init_Win_bytes_backward	Max Packet Length
15	act_data_pkt_fwd	PSH Flag Count
16	Bwd.IAT.Total	Fwd Packet Length Min
17	Fwd.Packet.Length.Std	Min Packet Length

niques. Let I , C , and G be subsets of the ranked features of the dataset T . It could be deduced that $I \subseteq T$, $C \subseteq T$, and $G \subseteq T$ are valid. Furthermore, Let i , c , and g represent any of the features ($X_1, X_2, X_3, \dots, X_n$) in the subsets. This therefore, imply that $i \in I$, $c \in C$, and $g \in G$. The common features are the intersecting features in I , C , and G . The representation of the set and subsets are shown in Fig. 3.

The area indicated as common features in Fig. 3 is the point of interest because it is the region shared by all subsets. The identification of the common features was done to shrink the original subsets into a smaller feature set to allow for further customising and processing of the selected features.

Algorithm 1 feature selection approaches

```

1: Input: Labeled — data,  $T_d$ , Number of iterations  $N_t$ 
2: Output: Common feature,  $C_f$ 
3: for  $i = 1$  in  $f_n$  : do
4:   rank features  $IG_R \leftarrow f(x)(f_i, f_{ii}, f_{iii}, \dots, f_n)$ 
5:   rank features  $CS_R \leftarrow f(y)(f_i, f_{ii}, f_{iii}, \dots, f_n)$ 
6:   rank features  $GI_R \leftarrow f(g)(f_i, f_{ii}, f_{iii}, \dots, f_n)$ 
7: end for
8: order ( $IG_R, CS_R, GI_R$ ) order magnitude in descending order
9: for  $a_i$  in ( $IG_R, CS_R, GI_R$ ) do
10:   compute  $Cum_{var}IG \leftarrow Cumulative_{var}(IG_R)$ 
11:   compute  $Cum_{var}CS \leftarrow Cumulative_{var}(CS_R)$ 
12:   compute  $Cum_{var}GI \leftarrow Cumulative_{var}(GI_R)$ 
13:   while  $Cum_{var}(i) + Var(a_i) \leq Cum_{var}(i)$  do  $\triangleleft$  where  $i$ 
       takes  $IG_R, CS_R$ , &  $GI_R$ 
14:      $Cum_{var}(i) \leftarrow Var(a_{i+1})$ 
15:      $A_i \leftarrow Cum_{var}(i)$   $\triangleleft$  save selected features
16:   end while
17: end for
18: for  $f_i$  in ( $A_i, A_{ii}, A_{iii}$ ) do
19:   if  $f_i \in (A_i \cap A_{ii} \cap A_{iii})$  then  $\triangleleft$  finding com-
       mon features
20:     select  $f_i$ 
21:     save  $Y \leftarrow f_i$ 
22:   end if
23: end for
24: Return( $Y$ )

```

In summary, three feature selection techniques were employed to rank the features of the datasets based on their importance. The cumulative variance computation was utilized to aid in selecting feature subsets for each technique. Then, a subset of features common to the initial subsets was chosen and used as input for the next phase of the work.

4.1.5 LSTM-Autoencoder architecture

As highlighted in Sect. 3.2, the common features, which were the output of the feature selection stage, were used as input in the input layer of the LSTM-AE model. While the redundant features have been removed, it was imperative for the structure and density of the data to be sustained. Therefore, we followed a specific procedure to compress the common feature data using an LSTM-AE model. First, we generated a sequence with a time-step of 5 and a sliding window of 1. The number of samples was determined on the basis of the number of observations. The LSTM-AE model was then fed the sequence arising from the common feature data as input, where each element represented a time step. As the sequence progressed, the LSTM cells processed

the data and produced a compressed representation of the input, as shown in equations 6 and 7. In the implementation, we maintained four layers in the LSTM-AE model and used the *tanh* activation function to achieve compression. We employed the *sigmoid* function for recurrent activation with a *glorot_uniform_kernel* initializer. In addition, we applied a *regularizer* with a value of 0.001 to effectively fit the training data and avoid overfitting. The resulting compressed representation was obtained from the encoder at the bottleneck layer, where we compressed the data to 5 nodes. In making the decision as to the size of the nodes at the bottleneck layer of the encoder, we were mindful of the fact that over-compression of data might lead to the elimination of discrimination of information between categories of data, reduced model generalization, decreased model robustness, and limited adaptability [30, 68].

4.2 Imbalanced class oversampling

In both datasets, there was a class imbalance of approximately 30,000 instances between the malicious and benign classes. This imbalance can potentially cause bias and skew the output of the learning algorithm. To address this, a minority oversampling technique based on the Sort-Augment and Combine (SAC) strategy with data perturbation augmentation [41] was employed. Initially, the data was sorted into distinct classes, and a function called *combine_samples* was developed to blend features from two samples by averaging their corresponding elements. Synthetic samples were then generated by randomly selecting pairs of samples from the minority class array, combining their features, and perturbing them with Gaussian noise controlled by a magnitude parameter. This noise introduces variability without significantly increasing variance, aiding the learning algorithm in generalization without compromising predictive accuracy. The augmented data was subsequently used to enrich the original dataset.

4.3 Model training

The dense model, a fully connected neural network, is a deep learning model characterized by multiple layers of interconnected neurons. Each neuron in a layer is connected to every neuron in the previous layer, and all neurons contribute to the model output. In this study, we incorporated an input layer and a hidden layer, which performed the bulk of the computations in the model. The output layer represents the final layer responsible for generating the model's predictions or outputs. The output of the bottleneck layer of the LSTM-AE encoder was used as the input to train the dense model. Here, each neuron in the dense layer underwent a linear transformation of its input, followed by a non-linear activation function. The linear transformation involved calculating a weighted

sum of the inputs from the previous layer, where each input is multiplied by an adaptable weight parameter. The result of this linear transformation was then passed through the activation function, which introduced non-linearity into the model. We selected the *ReLU* (Rectified Linear Unit) activation function for both the input and hidden layers. ReLU sets negative inputs to zero and leaves positive inputs untouched. We employed the *sigmoid* activation function for the output layer, which maps inputs to a range of 0-1 for binary classification.

The dense model can be mathematically represented as a function that takes an input vector x and produces an output vector y . The function encompasses L layers, including an input layer, one or more hidden layers, and an output layer. Each layer l in the model consists of N_l neurons and is connected to the preceding layer $l - 1$ and the subsequent layer $l + 1$. The processes of the transformation are indicated in equations 8 - 10.

The output of the neurons in layer l is given by:

$$z_l = W_l \times a_{l-1} + b_l \quad (8)$$

where W_l is a matrix of weights with dimensions $(N_l \times N_{l-1})$, a_{l-1} is the output of the previous layer, and b_l is a vector of biases with dimensions $(N_l \times 1)$. The operation \times denotes matrix multiplication.

The output of the neurons in layer l is then passed through an activation function g_l to introduce non-linearity into the model. The output of layer l is given by:

$$a_l = g_l(z_l) \quad (9)$$

The output of the final layer L is the output of the training model y , which is a function of the output of the previous layer a_{L-1} :

$$y = f(a_{L-1}) \quad (10)$$

where f is the activation function of the output layer.

However, during the data training process using the dense model, the model goes through weight and bias updates using an optimization algorithm and a loss function to minimize the error. Therefore, the Adam optimizer and the binary cross-entropy were utilized as the loss function to create stability. The loss function basically quantifies the difference between the model predictions and the true outputs, while the optimization algorithm adjusts the weights and biases to decrease this difference. To achieve a fair amount of training to enhance the effective learning process, the model was trained and evaluated using an epoch of 300 and a batch size of 64. After the training stage, we performed pruning to reduce the model size and unnecessary weights.

4.4 Pruning and deparameterisation

The success of model training is often accompanied by a significant increase in computation cost and parameter storage [25]. Ignoring the increase in computation cost occasioned by the introduction of parameters would certainly not aid our efforts to achieve a LIDS for the IoT. Therefore, pruning was adopted to reduce the overheads through weight compression of the various layers without affecting the original accuracy. By doing this, less important connections are removed which then leads to a more efficient model without significantly losing efficiency and performance. To achieve this, we used the *TensorFlow-keras-cloned-model* library to clone the model and then pruned 50% of the weights. Mathematically, weight pruning can be represented as follows:

Let W be the weight matrix of a neural network with dimensions (m, n) , where m represents the number of neurons in the dense layer and n indicates the number of neurons in the current layer.

Let θ be the pruning threshold, a value between 0 and 1 that determines the percentage of weights to be pruned. For example, if $\theta = 0.5$, then 50% of the weights will be pruned. To perform weight pruning, the weights in W_{ij} are ranked based on their magnitude from smallest to largest. The lowest $w_{ij}\%$ of weights are then removed and set to zero, and in this case, 50% of it was set to zero. The weight pruning operation for a specific weight w_{ij} in the weight matrix W can be defined as:

$$w_{ij} = \begin{cases} 0, & \text{if } |w_{ij}| < \theta \\ w_{ij}, & \text{otherwise} \end{cases} \quad (11)$$

where w_{ij} represents the weight connecting neuron i to neuron j in the dense layer. θ is the pruning threshold.

From Eq. 11:

- If the absolute value of the weight w_{ij} is less than the pruning threshold θ , the weight is set to zero, effectively pruning the connection between neurons.
- If the absolute value of the weight w_{ij} is greater than or equal to the pruning threshold θ , the weight remains unchanged.

Equation 11 can be applied iteratively across the weight matrix W , resulting in a pruned weight matrix where many connections below the pruning threshold (θ) have been set to zero (0).

Pruning reduces memory requirements, accelerates inference time, and makes the network more efficient; however, the sparsity (0) values introduced by pruning are also unnecessary weights. This is because, upon the validation of the pruned model, it was discovered that the model size was still large (refer to Tables 2 and 3). Therefore, stripping the

structured sparsity patterns introduced during the pruning process was necessary to further reduce the memory footprint. This is achieved through the deparameterization of the model. Deparameterization is a technique applied after pruning to remove additional weights, leading to a more efficient model. It reduces the number of parameters to compute, leading to faster inference times. This is particularly crucial in real-time applications where the response time is critical. We accomplished deparameterization by employing the TensorFlow Keras pruning method. This process enhanced portability and simplified the model by removing the sparsity introduced through pruning. This reduces memory requirements, simplifies subsequent optimization steps, and makes the deparameterized model easier to work with. By combining pruning and deparameterisation, the issue of high model size was effectively addressed while maintaining or improving performance. Furthermore, we used the Bayesian optimization technique to search and select the best parameters for optimum model performance of our model training process, as indicated in Sect. 3.3. The steps to the data compression technique and model training are explained in Algorithm 2.

With pruning and deparameterization accomplished, the model was retrained using the output of the deparameterized model. However, before the training, we performed data splitting to adopt semi-supervised learning approaches. First, we divided the dataset into training and testing subsets with a 70:30 ratio to ensure that there was no data leakage. Subsequently, we further partitioned the training data into three subsets: X_labeled, y_labeled, and X_unlabeled. Next, we trained a model, denoted as mod_L , on the labeled data (X_labeled and y_labeled) for 200 epochs, employing a batch_size of 64. We then used mod_L to predict the labels for the X_unlabeled data, generating pseudo-labels known as y_pseudo. To create a combined dataset for training, both X_labeled and X_unlabeled data were concatenated, and similarly, y_labeled and y_pseudo were concatenated. After shuffling to ensure a balanced mixture of labeled and pseudo-labeled samples, the model was trained again using the same number of epochs and batch_size. This approach expanded the training data and aided the model in refining its predictions and adapting to the pseudo-labeled data.

4.5 Adversarial attack

Adversarial attacks manifest in various forms, including poison attacks, where attackers manipulate training data to mislead the learning algorithm into producing erroneous classifications [42]. This form of attack significantly degrades the performance of the system, and in this study, we introduced a labeled poisoned attack by randomly flipping three (3) label samples in the training dataset. This is a form of adversarial attack that was done to undermine the

Table 2 MQTT-IoT-IDS2020 dataset metrics

	Table Column Head					
	Accuracy	Precision	Recall	F1	Time	Model_size
Model on all 68 Features	99.96	100	100	100	1.7s	31.5 MB
Model + Pruned (5 nodes)	99	99	100	100	1.05	25 MB
Model + Pruned + Deparame	99	99	100	100	0.932 s	85 Kb

Table 3 CIC-DS2017 dataset metrics

	Table Column Head					
	Accuracy	Precision	Recall	F1	Time	Model_size
Model on all 68 Features	97.8	96	100	97	1.7 s	31.5 MB
Model + Pruned (5 nodes)	98	96	100	97	1.2 s	25 MB
Model + Pruned + Depar (5 nodes)	98	96	100	97	1.03 s	85 Kb

Algorithm 2 for data compression and model fitting

```

1: Input: Common — features,  $C_f$ , Sequence, pruning,
   Deparameterization
2: Output:  $Encode_{data}$ 
3: for each  $X_j$  in  $X_{Train}$  : do
4:   while  $x_i \leq time - step$  : do           input layer of
   LSTM-AE
5:      $AE \leftarrow AutoEncoder(X_{Train})$ 
6:      $C_t = f_t x C_{t-1} + i_t x g_t$            calculate  $C_t$  (Eqn.6)
7:      $h_t = o_t x tanh(C_t)$            update state  $h_t$  (Eqn.7)
8:      $Encode_{data} \leftarrow compress AE to 5 nodes$ 
9:   end while
10: end for
11:  $train Encode_{data}$ 
12: while  $Mod_{size} = high$  : do
13:    $Pruned_{mod} \leftarrow Prune(Mod)$ 
14:    $Depar_{Mod} \leftarrow Deparameterize(Pruned_{mod})$ 
15: end while
16: for  $y$  in parameters : do
17:    $perform(BayesianOptimisationSearch)$ 
18: end for
19:  $Train(training_{data})$ 

```

performance of the performance model and thus cause it to make incorrect predictions on new and unseen data.

4.6 Default quantisation

Quantization is a technique employed in deep learning to reduce neural network models' memory and computation requirements without compromising their accuracy. This method entails mapping continuous value ranges to discrete values, effectively diminishing the number of bits essential to represent each parameter or activation in the network. In our study, after model training, we performed quantization and used the TensorFlow Lite model to preserve the weight of the model. Quantization can be performed either after model training or during training, which is known as Quantisation Aware Training (QAT). We chose QAT over post-training quantization because the latter may affect the

overall accuracy. Quantization-Aware Training integrates the quantization process seamlessly into the training itself, thus optimizing the model for quantization right from the start. The following are the steps we followed in implementing QAT:

1. The deparameterized model was trained and optimised using float DEFAULT Byte quantisation, resulting in a q_model . This approach explored different optimisation options using float 8 Byte, float 16 Byte, and float 32 Byte, selecting the most effective.
2. The output (q_model) from the previous step was fine-tuned and trained for 30 epochs, followed by validation, resulting in a fine-tuned- q_model .
3. Finally, a TensorFlow Lite model was applied to the fine-tuned model to obtain a $TFLite_model$.

QAT, or Quantisation Aware Training, enables us to train the model using a combination of both full-precision and quantised versions of the parameters. During the forward and backward passes, we utilize the quantised version, while during weight updates, we employ the full-precision version to mitigate any information loss resulting from quantisation errors. Training with an awareness of quantisation yields several advantages, including reduced memory and computation requirements, minimal loss of accuracy, flexibility in choosing quantisation methods, and seamless integration into the deep learning workflow. Once quantisation is complete, the model is converted into the TensorFlow Lite (TFLite) format. With that, the model is ready for implementation and inference on the test data using the TensorFlow Lite interpreter, allowing us to evaluate the results effectively.

4.7 Model deployment and inferencing

To validate the efficacy of our proposed model, we deployed it on two distinct devices for evaluation. The first device

was a Windows 11 Desktop machine, where we leveraged Google Colab to run the Python code. The Windows system employed a single-threaded CPU with an Intel Core i7-1065G7 processor running at a frequency of 1.30–1.50 GHz. With 16 GB of RAM and a 500 GB hard disk, it provided the necessary hardware resources. The second device used was a Raspberry Pi 4 with an 8 GB model, operating on the Bullseye Debian-based OS with aarch64 architecture. While the Python Notebook was the IDE used on the Windows system, the Thorny IDE was the platform for the experiment on the Raspberry Pi. TensorFlow and other essential Python libraries were installed and used on the Windows Desktop machine, while the Raspberry Pi used TFLite, a lightweight version of TensorFlow, for inference operations. Subsequently, we loaded the TFLite model using the *tf.lite.Interpreter* class and allocated memory for the input and output tensors through the *allocate_tensors* method. Using the *get_input_details* and *get_output_details* methods, we obtained information regarding the input and output tensors of the lightweight model. The input data shape was defined, and a random numpy array of the test data was generated to mimic the input data. The numpy array was then assigned as the input data via the *set_tensor* method. After configuring the input tensor, we invoked the interpreter using the *invoke* method to perform inference and acquire the output tensor. A threshold value of 0.5 was applied to the output tensor to convert the probability scores into binary predictions of 0 and 1. Finally, we generated a classification report by leveraging the numpy form of the data alongside the predicted labels. In addition, we computed the model size and time required for the computations. The steps to achieve quantisation, deployment and inferencing are highlighted in Algorithm 3.

Algorithm 3 for quantization, deployment and inferencing

```

1: Mod ← Model
2: if Mod = deparameterised then
3:   Mod.h5                                save model
4: end if
5: GetModelSize()
6: Convert to tflite file
7: if Mod = tflite then
8:   quantize_model(model)
9: end if
10: get input & output tensors
11: while Tensor ≠ 0 do
12:   Resize tensor
13:   interpreter.invoke()
14: end while
15: Classification
16: classification Report()
17: getValidationTime()
18: getModelSize()
  
```

5 Results and discussion

This section provides an overview of the results obtained while evaluating the proposed LIDS model on the two datasets. It also compares the results of other studies with our study and explains how the proposed model answers the research questions.

5.1 Result comparison

Following the methodology encompassing feature selection and data compression, the subsequent stages of this experiment were evaluated to determine the reduction in computation cost, which is the mainstay of this study. The evaluation results of the two datasets (MQTT-IoT-IDS202 and CIC-IDS2017 datasets) are displayed in Tables 5 and 6. To ensure consistency and unbiased, we limited the experiment to semi-supervised learning and then computed the classification report.

1. We evaluated the approach at different stages and recorded the overall accuracy, precision, recall, F1-score, computation time and model size.
2. The result of the model evaluation on all the 68 features (benchmark data), the pruned data, the deparameterized model and the inferencing for comparison.
3. The ROC curve showing the Area Under the Curve (AUC) was plotted, and the training and validation loss and accuracy plots were also plotted to check for overfitting or underfitting.

Tables 5 and 6 present the comparative assessment of overall accuracy between the LSTM model on the benchmark dataset and our proposed model, demonstrating parity at 99% and 97.8%, respectively. This equivalence is corroborated by the convergence patterns depicted in Fig. 6b (right plot), affirming the absence of overfitting. The observed convergence indicates a minimal disparity between the training and loss metrics. In addition, the ROC curves in Figs. 4 and 5 visually encapsulate the proposed model's discriminative prowess, elucidating its ability to distinguish between the two classes.

To further scrutinize model performance, we plotted training and validation losses (Figs. 6a and 7) to gauge potential overfitting or underfitting concerns. This analysis assumes paramount importance in evaluating the behavior and efficacy of the model throughout the training and validation phases. The trajectory of training loss offers insights into the learning progression of a model on the training data. Concurrently, the validation loss plot indicates a model proficiency in generalizing to novel and unobserved data. A consistent downward trend in both the training and validation loss plots,

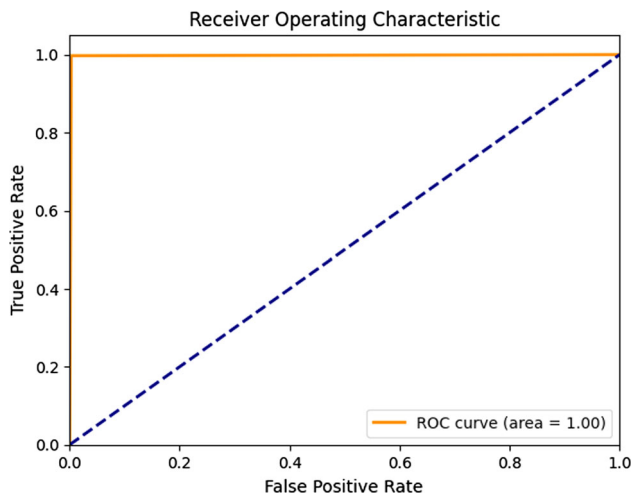


Fig. 4 ROC curve showing the Area Under the Curve (AUC) for the MQTT dataset

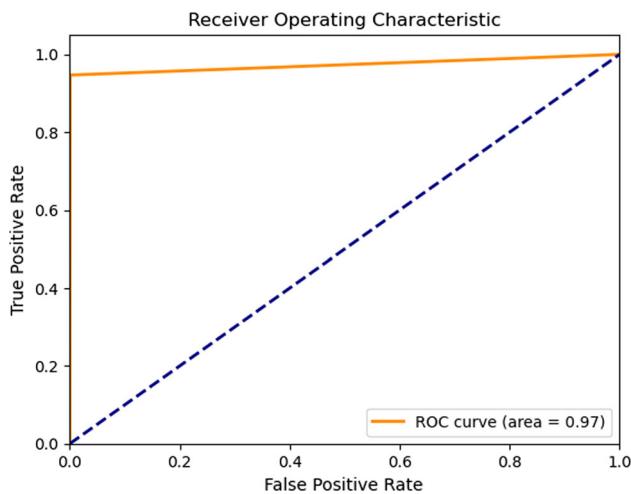


Fig. 5 ROC curve showing the Area Under the Curve (AUC) for CIC-IDS2017 dataset

as illustrated in Figs. 6 and 7, signifies that the model is adept at refining predictions on hitherto unseen data. Conversely, an ascending trend could indicate overfitting. Similar to the accuracies and the ROC curve are the precision score, recall, and F1 score, as shown in Tables 3, 4, 5, 6, 7 and 8. These metrics are instrumental in classification (refer to Sect. 3.4). The score value in both tables further attests to the high performance of the proposed model on the two datasets.

Considering the importance of computation cost in a resource-constrained device such as the IoT, the size of the model, which indicates the amount of memory used, was drastically reduced from 315 and 265KB to 2KB in 0.3 and 0.12s, respectively. The import is that while the benchmark model achieved its classification in 1.7s with a model size of 315 and 256KB, the proposed model accomplishes classification in considerably less time and effectively reduces

the model size to 2KB. In addition, the recall values in both tables indicate the percentage classification of the positive class, which our proposed model effectively achieved.

5.2 Model resilience against label Poison attack

Overcoming adversarial attacks and ensuring the resilience of a model requires effective training and learning. For this reason, [43] proposed a deep k-NN approach to counter label poisoning attacks. Similarly, [3] proposed a scalable and transferable clean-label poisoning attack on label poisoning attack. In the same vein, [66] proposed a non-perturbation approach that relies on the effectiveness of the resource-efficient model to withstand label adversarial attacks. In this case, to ensure that the model is resilient against label poisoning attacks, we implemented two approaches: (a) Robust Learning Approach: This involves integrating outlier detection into the training process to identify and eliminate poisoned examples. To execute this, we utilized the *EllipticEnvelope* class from the scikit-learn library [9] in Python for detecting and removing outliers from the training set. (b) Robust Training by Generating Pseudo-Labels: In this approach, pseudo-labels were generated by leveraging the model trained on labeled data to predict unlabeled data, and the output was then concatenated with the original dataset. By adopting these two strategies, we effectively cleansed the data of any potential poisoning while also enhancing generalization through data augmentation. Subsequently, the model underwent evaluation on the testing dataset. The model output after label poisoning is displayed in Tables 6, 7, 8 and 9.

The result of the benchmark model and the degradation resulting from the label poisoning attack are detailed and compared in Tables 6, 7, 8, and 9. The tables showcase the performance of the models both pre and post-attack. In Table 6, a 20.9% decline in overall accuracy was noted for the benchmark dataset, while Table 7 demonstrated a 16.16% decrease for the proposed model. Similarly, the CIC-IDS2017 dataset (Table 8) exhibited a 22.9% accuracy drop for the benchmark model and a 19.38% reduction for the proposed model (Table 9). In both cases, the recall, which is the proportion of the true positive cases correctly classified, indicates that the proposed model performed very well. Therefore, we can infer that the proposed model is more resilient than the benchmark dataset.

5.2.1 Receiver operating characteristic (ROC) curve

The Receiver Operating Characteristic (ROC) curve is an important tool for the graphical representation of a binary classification model performance. It provides valuable insight into the Area Under the Curve (AUC), quantifying the model classification accuracy. For example, when the



Fig. 6 Plot showing training and validation loss (left) and the training and validation accuracies (right) for MQTT-IoT-IDS2020 dataset

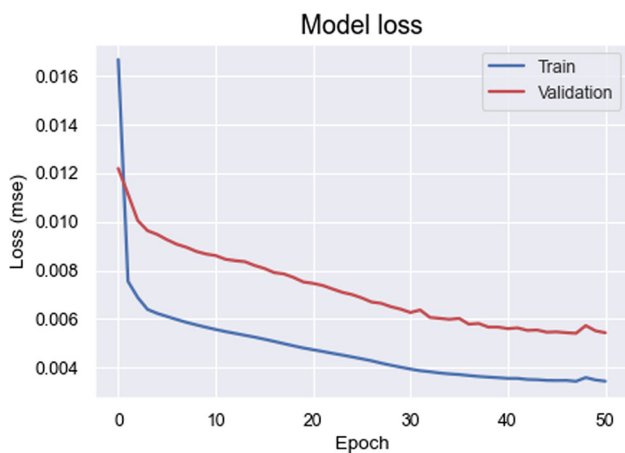


Fig. 7 Plot showing training and validation loss for CIC-IDS2017 dataset

AUC reaches 1, as illustrated in Fig. 4, it signifies a perfect classification. This indicates that the model has effectively differentiated between benign and malicious classes. In other words, the model has successfully distinguished between positive and negative instances in the dataset, resulting in a classification with 100% sensitivity (true positive rate) and 100% specificity (true negative rate).

5.3 Comparison with other works

We compared our findings with those of other relevant studies on Lightweight Intrusion Detection in the context of the IoT. To maintain objectivity, we limited our comparison to results from studies that utilized the same two datasets we employed. Table 10 displays the detailed comparison.

Table 4 MQTT-IoT-IDS2020 dataset Classification report

	Metrics					
	Accuracy	Precision	Recall	F1	Time	Model_size
Model on all 68 Features	99.96	100	100	100	1.7s	315 KB
model + Pruned (5 nodes)	99	99	100	100	1.05	255 KB
Model + Pruned + Depar (5 nodes)	99	99	100	100	0.932s	85 KB
OCFSDA (proposed model)	99	100	98	99	0.30s	2 KB

Table 5 CIC-IDS2017 dataset Classification report

	Metrics					
	Accuracy	Precision	Recall	F1	Time	Model_size
Model on all 68 Features	97.8	96	100	97	1.7 s	265 KB
Model + Pruned (5 nodes)	98	96	100	97	1.2 s	223 KB
Model + Pruned + Depar (5 nodes)	98	96	100	97	1.03 s	85 KB
OCFSDA (proposed model)	97	95	100	97	0.12 s	2 KB

Table 6 Table showing benchmark model results before and after label poisoning attack on MQTT-IoT-IDS2020 dataset

	Acc	Presc	Recall	F1	Time	Model_size
Benchmark model before poison Attack	99.96	100	100	100	1.7	315 KB
Benchmark model after poison Attack	79	43	94	59.01	1.73s	308 KB

Table 7 Table showing proposed model results before and after label poisoning attack on MQTT-IoT-IDS2020 dataset

	Acc	Presc	Recall	F1	Time	Model_size
Proposed model before Poison Attack	99	100	98	98.99	0.30s	2KB
Proposed model after poison Attack	83	68	100	80.95	0.14s	2KB

Table 8 Table showing benchmark model results before and after label poisoning attack on CIC-IDS2017 dataset

	Acc	Presc	Recall	F1	Time	Model_size
Benchmark model before poison Attack	97.8	96	100	97	1.7	265KB
Benchmark model after poison Attack	75.4	53	84	64.8	1.63s	255KB

Table 9 Table showing proposed model results before and after label poisoning attack on CIC-IDS2017 dataset

	Acc	Presc	Recall	F1	Time	Model_size
Proposed model before Poison Attack	97	95	100	97	0.12 s	2KB
Proposed model after poison Attack	78.2	63	100	76	0.13s	2KB

Table 10 Model result comparison with other related approaches of other authors

LID Model	Metrics						
	Ref	Acc	Precision	Recall	F1-score	Cmp_Time	Mem_size
OCSVM	[11]	99	?	?	?	?	?
Isolation F	[11]	84	?	?	?	?	?
SENMQTT	[56]	100	?	?	100	0.04s	?
NL_SVM-IoT	[18]	99.34	?	?	?	17.57s	?
DL-HIDS	[16]	96.69	?	?	?	2e-6	2.704KB
GAN-AE	[7]	97.3	97.4	97.3	97.3	?	?
1D-DCNN	[46]	99.7	?	?	?	?	?
SS-DEEP-ID	[1]	99.6	99.48	99.23	99.35	1.1s	?
ELETL-IDS	[39]	100	?	?	?	?	?
Self-Attention	[26]	98.9	98	98.6	98.3	?	?
KD-TCNN	[63]	99.44	99.48	99.47	99.46	?	18.1KB
HFS-KODE	[20]	99.99	99.2	99.75	99.3	208 s	?
NewData-IDS	[53]	100	?	?100	0.04s	?	?
IBGJO	[13]	98.21	98.48	98.92	97.25	?	?
HDFEF	[28]	99.7	99.73	99.96	99.84	138.098s	?
OCFSDA	Our model	99	100	98	99	0.30s	2KB

Table 10 shows that significant progress has been made in the domain of lightweight intrusion detection for resource-constrained devices such as the IoT. Nevertheless, while noteworthy advancements have been demonstrated in terms

of overall accuracy, it remains challenging to determine the computational time and model size as they are essential factors in computation cost. Notably, the approach presented by [16] achieved an accuracy of 96.69% in 0.000002s, using

a model size of 2.7KB. However, this study did not disclose the precision, recall, and F1-score values nor provided information on the ROC necessary for effective comparison. Similarly, [53] achieved an average accuracy classification rate and F1-score of 100% in 0.04s. However, this study failed to provide scores for other critical metrics and the authors did not also provide the model size. To this end, because of the absence of values for some metrics, a question mark (?) was entered in the table. In line with the assessment, our OCFSDA approach to the two datasets yielded overall accuracy rates of 99% and 97%. In addition, our proposed model successfully generated precision, recall, and F1-score values within 0.30 and 0.12 s, respectively, accompanied by a model size of 2KB.

6 Conclusion

The evolution of IoT technology has ushered in a transformative era, revolutionizing critical infrastructure, smart home environments, and smart cities with unprecedented connectivity and convenience. However, the inherent security vulnerabilities within IoT systems expose them to potential cyber threats. Conventional intrusion detection methods, constrained by protocol limitations and memory constraints, struggle to protect these devices effectively. Thus, our study focused on developing a lightweight intrusion detection solution for resource-constrained IoT devices.

Central to our approach was the utilization of efficient dimensionality reduction techniques, including feature selection and feature extraction. By applying three feature selection methods, we identified pertinent features correlated with the target variable. These features underwent further extraction using an LSTM-autoencoder, compressing the encoder output to just five nodes. Subsequent optimization involved pruning and deparameterization to eliminate redun-

dant weights and enhancing model efficiency. In addition to resilience against adversarial attacks like poison attacks, we leveraged quantization to optimize inferencing efficiency, resulting in a lightweight model with reduced memory footprint and accelerated inference times. However, prior to inferencing, the input data underwent further processing using the TFLite interpreter, ensuring compatibility with the model's input requirements. The interpreter facilitated the seamless execution of inferences, processing both input and output data to generate accurate predictions.

Our innovative Lightweight Intrusion Detection (OCFSDA) model, utilizing semi-supervised learning and deployed on a Raspberry Pi4, achieved overall accuracies of 99% and 97% from two datasets, along with improved performances across various evaluation metrics. Notably, the model demonstrated an improved classification run-time within 0.30 and 0.12 s while utilizing 2KB of memory, highlighting its efficiency and effectiveness in real-world IoT deployment scenarios.

Future work: Based on this study and our comprehensive work, we strongly believe that it is crucial to prioritize future work to enhance the model's resilience and broaden its capabilities to detect various types of data poisoning attacks. These include poison frog, bullseye polytope, and convex attacks.

Appendix A

See Table 11.

Table 11 MQTT-IoT-IDS2020 dataset feature ranking

Feature Ranking				
	Dataset Features	Info-Gain Ranking	Chi-Squared Ranking	Gini-Index Ranking
1	Destination.Port	Total.Length.of.Fwd.Packets	URG.Flag.Count	Total.Length.of.Fwd.Packets
2	Flow.Duration	Subflow.Fwd.Bytes	min_seg_size_forward	Subflow.Fwd.Bytes
3	Total.Fwd.Packets	Average.Packet.Size	Destination.Port	Average.Packet.Size
4	Total.Backward.Packets	Bwd.Packet.Length.Mean	Min.Packet.Length	Destination.Port
5	Total.Length.of.Fwd.Packets	Destination.Port	Bwd.IAT.Total	Bwd.Packet.Length.Mean
6	Total.Length.of.Bwd.Packets	Avg.Bwd.Segment.Size	Fwd.Packet.Length.Std	Avg.Bwd.Segment.Size
7	Fwd.Packet.Length.Max	Total.Length.of.Bwd.Packets	Fwd.Packet.Length.Mean	Subflow.Bwd.Bytes
8	Fwd.Packet.Length.Min	Subflow.Bwd.Bytes	Avg.Fwd.Segment.Size	Total.Length.of.Bwd.Packets
9	Fwd.Packet.Length.Mean	Fwd.Header.Length.l	Bwd.IAT.Std	Fwd.Header.Length.l
10	Fwd.Packet.Length.Std	Fwd.Header.Length	Fwd.Packet.Length.Max	Fwd.Header.Length
11	Bwd.Packet.Length.Max	Bwd.Packet.Length.Max	Bwd.IAT.Max	Bwd.Packet.Length.Max
12	Bwd.Packet.Length.Min	Init_Win_bytes_forward	Total.Length.of.Fwd.Packets	Init_Win_bytes_forward
13	Bwd.Packet.Length.Mean	Fwd.Packet.Length.Mean	Subflow.Fwd.Bytes	Avg.Fwd.Segment.Size
14	Bwd.Packet.Length.Std	Avg.Fwd.Segment.Size	Fwd.PSH.Flags	Fwd.Packet.Length.Mean
15	Flow.Bytes.s	Fwd.Packet.Length.Max	SYN.Flag.Count	Fwd.Packet.Length.Max
16	Flow.Packets.s	Bwd.Header.Length	Bwd.Packet.Length.Min	Bwd.Header.Length
17	Flow.IAT.Mean	Fwd.IAT.Max	Bwd.IAT.Mean	Fwd.IAT.Max
18	Flow.IAT.Std	Fwd.IAT.Total	Init_Win_bytes_backward	Fwd.IAT.Total
19	Flow.IAT.Max	Fwd.IAT.Mean	Total.Fwd.Packets	Fwd.IAT.Mean
20	Flow.IAT.Min	Total.Fwd.Packets	Subflow.Fwd.Packets	Subflow.Fwd.Packets
21	Fwd.IAT.Total	Fwd.IAT.Std	Fwd.Header.Length	Total.Fwd.Packets
22	Fwd.IAT.Mean	Subflow.Fwd.Packets	Fwd.Header.Length.l	Init_Win_bytes_backward
23	Fwd.IAT.Std	Init_Win_bytes_backward	Fwd.Packet.Length.Min	Fwd.IAT.Std
24	Fwd.IAT.Max	Packet.Length.Mean	act_data_pkt_fwd	Packet.Length.Mean
25	Fwd.IAT.Min	act_data_pkt_fwd	Flow.Packets.s	act_data_pkt_fwd
26	Bwd.IAT.Total	Packet.Length.Std	Fwd.Packets.s	Packet.Length.Variance
27	Bwd.IAT.Mean	Packet.Length.Variance	Init_Win_bytes_forward	Packet.Length.Std
28	Bwd.IAT.Std	Bwd.IAT.Max	Bwd.Packet.Length.Mean	Bwd.IAT.Total
29	Bwd.IAT.Max	Bwd.IAT.Total	Avg.Bwd.Segment.Size	Fwd.Packet.Length.Std
30	Bwd.IAT.Min	Fwd.Packet.Length.Std	Bwd.Packet.Length.Max	Bwd.Packet.Length.Std
31	Fwd.PSH.Flags	Bwd.Packet.Length.Std	Active.Std	Bwd.Packets.s
32	Fwd.Header.Length	Bwd.Packets.s	Bwd.Header.Length	Bwd.IAT.Max
33	Bwd.Header.Length	Bwd.IAT.Mean	Bwd.Packet.Length.Std	Total.Backward.Packets
34	Fwd.Packets.s	Total.Backward.Packets	Bwd.Packets.s	Subflow.Bwd.Packets
35	Bwd.Packets.s	Subflow.Bwd.Packets	Bwd.IAT.Min	Fwd.Packets.s
36	Min.Packet.Length	Fwd.Packets.s	Fwd.IAT.Min	Max.Packet.Length
37	Max.Packet.Length	Max.Packet.Length	Total.Backward.Packets	Bwd.IAT.Mean
38	Packet.Length.Mean	Flow.Duration	Subflow.Bwd.Packets	Flow.Duration
39	Packet.Length.Std	Bwd.Packet.Length.Min	FIN.Flag.Count	Bwd.Packet.Length.Min
40	Packet.Length.Variance	Flow.Bytes.s	Down.Up.Ratio	Flow.Bytes.s
41	FIN.Flag.Count	Flow.IAT.Max	Average.Packet.Size	Flow.IAT.Max
42	SYN.Flag.Count	Flow.IAT.Std	Packet.Length.Mean	Flow.IAT.Std
43	RST.Flag.Count	Flow.IAT.Mean	Flow.IAT.Min	Flow.IAT.Mean
44	PSH.Flag.Count	Bwd.IAT.Std	Flow.Bytes.s	Flow.Packets.s
45	ACK.Flag.Count	Flow.Packets.s	Packet.Length.Std	Bwd.IAT.Std
46	URG.Flag.Count	Active.Min	Packet.Length.Variance	Active.Min

Table 11 continued

	Feature Ranking			
	Dataset Features	Info-Gain Ranking	Chi-Squared Ranking	Gini-Index Ranking
47	ECE.Flag.Count	Active.Mean	Max.Packet.Length	Bwd.IAT.Min
48	Down.Up.Ratio	Bwd.IAT.Min	Idle.Std	Active.Mean
49	Average.Packet.Size	Active.Max	Total.Length.of.Bwd.Packets	Active.Max
50	Avg.Fwd.Segment.Size	Fwd.IAT.Min	Subflow.Bwd.Bytes	Fwd.IAT.Min
51	Avg.Bwd.Segment.Size	Fwd.Packet.Length.Min	Fwd.IAT.Mean	Fwd.Packet.Length.Min
52	Fwd.Header.Length.1	Min.Packet.Length	RST.Flag.Count	URG.Flag.Count
53	Subflow.Fwd.Packets	URG.Flag.Count	ECE.Flag.Count	Min.Packet.Length
54	Subflow.Fwd.Bytes	Down.Up.Ratio	PSH.Flag.Count	Down.Up.Ratio
55	Subflow.Bwd.Packets	min_seg_size_forward	Idle.Max	min_seg_size_forward
56	Subflow.Bwd.Bytes	Flow.IAT.Min	Active.Min	Flow.IAT.Min
57	Init_Win_bytes_forward	Idle.Min	Flow.IAT.Max	Idle.Mean
58	Init_Win_bytes_backward	Idle.Max	Idle.Mean	Idle.Max
59	ac_data_pkt_fwd	Idle.Mean	Flow.IAT.Std	Idle.Min
60	min_seg_size_forward	PSH.Flag.Count	Active.Mean	PSH.Flag.Count
61	Active.Mean	Idle.Std	Flow.IAT.Mean	Idle.Std
62	Active.Std	Fwd.PSH.Flags	Fwd.IAT.Max	Fwd.PSH.Flags
63	Active.Max	SYN.Flag.Count	Idle.Min	SYN.Flag.Count
64	Active.Min	Active.Std	Fwd.IAT.Total	Active.Std
65	Idle.Mean	ACK.Flag.Count	Flow.Duration	ACK.Flag.Count
66	Idle.Std	FIN.Flag.Count	Active.Max	FIN.Flag.Count
67	Idle.Max	RST.Flag.Count	Fwd.IAT.Std	ECE.Flag.Count
68	Idle.Min	ECE.Flag.Count	ACK.Flag.Count	RST.Flag.Count

Appendix B

See Table 12.

Table 12 CICIDS2017 dataset feature ranking

Feature Ranking				
	Dataset Features	Information Gain	Chi Squared	Gini Index
1	Destination Port	Packet Length Mean	PSH Flag Count	Avg Fwd Segment Size
2	Flow Duration	Packet Length Std	ACK Flag Count	Packet Length Std
3	Total Fwd Packets	Packet Length Variance	URG Flag Count	Flow Bytes/s
4	Total Backward Packets	Destination Port	Flow Duration	Destination Port
5	Total Length of Fwd Packets	min_seg_size_forward	Fwd IAT Total	min_seg_size_forward
6	Total Length of Bwd Packets	Average Packet Size	Bwd IAT Total	Average Packet Size
7	Fwd Packet Length Max	Init_Win_bytes_backward	Min Packet Length	Init_Win_bytes_backward
8	Fwd Packet Length Min	Init_Win_bytes_forward	Packet Length Mean	Init_Win_bytes_forward
9	Fwd Packet Length Mean	Flow Bytes/s	Avg Bwd Segment Size	Packet Length Variance
10	Fwd Packet Length Std	Subflow Fwd Bytes	Bwd Packet Length Mean	Subflow Fwd Bytes
11	Bwd Packet Length Max	Total Length of Fwd Packets	Init_Win_bytes_forward	Total Length of Fwd Packets
12	Bwd Packet Length Min	Fwd Packet Length Mean	Average Packet Size	Packet Length Mean
13	Bwd Packet Length Mean	Avg Fwd Segment Size	Packet Length Std	Fwd Packet Length Mean
14	Bwd Packet Length Std	Fwd Packet Length Max	Fwd PSH Flags	Fwd Packet Length Max
15	Flow Bytes/s	Subflow Bwd Bytes	SYN Flag Count	Subflow Bwd Bytes
16	Flow Packets/s	Total Length of Bwd Packets	Bwd Packet Length Std	Total Length of Bwd Packets
17	Flow IAT Mean	Bwd Packet Length Mean	Bwd Packet Length Max	Avg Bwd Segment Size
18	Flow IAT Std	Avg Bwd Segment Size	Flow IAT Max	Bwd Packet Length Mean
19	Flow IAT Max	Bwd Packet Length Max	Max Packet Length	Bwd Packet Length Max
20	Flow IAT Min	Bwd Packet Length Min	Fwd IAT Max	URG Flag Count
21	Fwd IAT Total	Bwd Packets/s	Init_Win_bytes_backward	Bwd Packets/s
22	Fwd IAT Mean	Flow Duration	Idle Max	Flow Duration
23	Fwd IAT Std	Flow IAT Max	Idle Mean	Flow IAT Max
24	Fwd IAT Max	Fwd Packets/s	Bwd IAT Max	Fwd Packets/s
25	Fwd IAT Min	Flow Packets/s	Bwd Packet Length Min	Flow Packets/s
26	Bwd IAT Total	Bwd Header Length	Idle Min	Bwd Header Length
27	Bwd IAT Mean	Flow IAT Mean	FIN Flag Count	Flow IAT Mean
28	Bwd IAT Std	Fwd Header Length	Fwd Packet Length Std	Total Fwd Packets
29	Bwd IAT Max	Fwd Header Length.1	Flow IAT Std	Fwd Header Length
30	Bwd IAT Min	Max Packet Length	Fwd Packet Length Mean	Max Packet Length
31	Fwd PSH Flags	PSH Flag Count	Avg Fwd Segment Size	PSH Flag Count
32	Fwd Header Length	Fwd Packet Length Min	Fwd IAT Std	Fwd IAT Mean
33	Bwd Header Length	Flow IAT Min	Fwd Packet Length Min	Flow IAT Min
34	Fwd Packets/s	Min Packet Length	Fwd Packet Length Max	Min Packet Length
35	Bwd Packets/s	Fwd IAT Max	Bwd IAT Mean	Fwd IAT Max
36	Min Packet Length	Fwd IAT Mean	Fwd IAT Mean	Subflow Bwd Packets
37	Max Packet Length	Fwd IAT Total	Packet Length Variance	Fwd IAT Total
38	Packet Length Mean	Fwd IAT Min	min_seg_size_forward	Fwd IAT Min
39	Packet Length Std	Total Fwd Packets	Bwd IAT Std	Fwd Header Length.1
40	Packet Length Variance	Subflow Fwd Packets	Bwd IAT Min	Subflow Fwd Packets
41	FIN Flag Count	Subflow Bwd Packets	Flow IAT Mean	Fwd Packet Length Min
42	SYN Flag Count	Total Backward Packets	Fwd IAT Min	Total Backward Packets
43	RST Flag Count	act_data_pkt_fwd	Bwd Packets/s	act_data_pkt_fwd
44	PSH Flag Count	Flow IAT Std	Total Length of Fwd Packets	Flow IAT Std
45	ACK Flag Count	Bwd IAT Max	Subflow Fwd Bytes	Bwd IAT Total
46	URG Flag Count	Bwd IAT Mean	Idle Std	Bwd IAT Max

Table 12 continued

	Feature Ranking			
	Dataset Features	Information Gain	Chi Squared	Gini Index
47	ECE Flag Count	Bwd IAT Total	act_data_pkt_fwd	Bwd IAT Mean
48	Down/Up Ratio	Bwd IAT Min	Fwd Packets/s	Bwd IAT Min
49	Average Packet Size	Down/Up Ratio	Total Fwd Packets	Down/Up Ratio
50	Avg Fwd Segment Size	Fwd IAT Std	Subflow Fwd Packets	Fwd IAT Std
51	Avg Bwd Segment Size	Fwd Packet Length Std	Active Max	Fwd Packet Length Std
52	Fwd Header Length.l	ACK Flag Count	Destination Port	ACK Flag Count
53	Fwd Avg Bytes/Bulk	Bwd IAT Std	Bwd Header Length	Bwd IAT Std
54	Fwd Avg Packets/Bulk	Bwd Packet Length Std	Total Backward Packets	Bwd Packet Length Std
55	Fwd Avg Bulk Rate	Active Mean	Subflow Bwd Packets	Active Mean
56	Bwd Avg Bytes/Bulk	Idle Max	Fwd Header Length	Idle Mean
57	Bwd Avg Packets/Bulk	Active Max	Fwd Header Length.l	Active Max
58	Bwd Avg Bulk Rate	Idle Min	Total Length of Bwd Packets	Active Min
59	Subflow Fwd Packets	Idle Mean	Subflow Bwd Bytes	Idle Min
60	Subflow Fwd Bytes	Active Min	Active Std	Idle Max
61	Subflow Bwd Packets	Idle Std	Active Mean	Idle Std
62	Subflow Bwd Bytes	Active Std	Active Min	Active Std
63	Init_Win_bytes_forward	URG Flag Count	Down/Up Ratio	Bwd Packet Length Min
64	Init_Win_bytes_backward	SYN Flag Count	Flow IAT Min	Fwd PSH Flags
65	act_data_pkt_fwd	Fwd PSH Flags	RST Flag Count	SYN Flag Count
66	min_seg_size_forward	FIN Flag Count	ECE Flag Count	FIN Flag Count
67	Active Mean	RST Flag Count	Flow Bytes/s	ECE Flag Count
68	Active Std	ECE Flag Count	Flow Packets/s	RST Flag Count

Research data policy and data availability statements MQTT-IoT-IDS2020 dataset can be found at <https://ieee-dataport.org/open-access/mqtt-iot-ids2020-mqtt-internet-things-intrusion-detection-dataset> OR <https://doi.org/10.21227/bhxy-ep04> CIC-IDS2017 dataset can be found at: <https://doi.org/10.5220/0006639801080116>.

Declarations

Conflict of interest I declare that the authors have no conflict of interest.

Human and animals rights The study did not involve the use of humans or animals.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Abdel-Basset, M., Hawash, H., Chakraborty, R.K., Ryan, M.J.: Semi-supervised spatiotemporal deep learning for intrusions detection in IoT networks. *IEEE Internet Things J.* **8**(15), 12251–12265 (2021)
2. Abdullah, M., Alshannaq, A., Balamash, A., Almabdy, S.: Enhanced intrusion detection system using feature selection method and ensemble learning algorithms. *Int. J. Comput. Sci. Inf. Secur. (IJCSIS)* **16**(2), 48–55 (2018)
3. Aghakhani, H., Meng, D., Wang, Y.X., Kruegel, C., Vigna, G.: Bullseye polytope: a scalable clean-label poisoning attack with improved transferability. In: 2021 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, pp. 159–178 (2021)
4. Amrita, K.K.R.: A hybrid intrusion detection system: integrating hybrid feature selection approach with heterogeneous ensemble of intelligent classifiers. *Int. J. Netw. Secur.* **20**(1), 41–55 (2018)
5. Azhagusundari, B., Thanamani, A.S., et al.: Feature selection based on information gain. *Int. J. Innov. Technol. Explor. Eng. (IJITEE)* **2**(2), 18–21 (2013)
6. Baldi, P.: Autoencoders, unsupervised learning, and deep architectures. In: Proceedings of ICML Workshop on Unsupervised and Transfer Learning. JMLR Workshop and Conference Proceedings, pp. 37–49 (2012)
7. Boppana, T.K., Bagade, P.: GAN-AE: an unsupervised intrusion detection system for MQTT networks. *Eng. Appl. Artif. Intell.* **119**, 105805 (2023). <https://doi.org/10.1016/j.engappai.2022.105805>

8. Borgohain, T., Kumar, U., Sanyal, S.: Survey of security and privacy issues of internet of things. arXiv preprint [arXiv:1501.02211](https://arxiv.org/abs/1501.02211) (2015)
9. Chen, Y., Wang, S., Zhao, Q., Sun, G.: Detection of multivariate geochemical anomalies using the bat-optimized isolation forest and bat-optimized elliptic envelope models. *J. Earth Sci.* **32**(2), 415–426 (2021)
10. Choi, S.K., Yang, C.H., Kwak, J.: System hardening and security monitoring for IoT devices to mitigate IoT security vulnerabilities and threats. *KSII Trans. Internet Inf. Syst.* **12**(2) (2018)
11. Ciklabakkal, E., Donmez, A., Erdemir, M., Suren, E., Yilmaz, M.K., Angin, P.: ARTEMIS: An intrusion detection system for MQTT attacks in internet of things. In: 2019 38th Symposium on Reliable Distributed Systems (SRDS). IEEE (2019). <https://doi.org/10.1109/srds47363.2019.00053>
12. Halim, Z., Yousaf, M.N., Waqas, M., Sulaiman, M., Abbas, G., Hussain, M., Ahmad, I., Hanif, M.: An effective genetic algorithm-based feature selection method for intrusion detection systems. *Comput. Secur.* **110**, 102448 (2021)
13. Hanafi, A.V., Ghaffari, A., Rezaei, H., Valipour, A., arasteh, B.: Intrusion detection in internet of things using improved binary golden jackal optimization algorithm and LSTM. *Cluster Comput.* 1–18 (2023)
14. Hindy, H., Tachtatzis, C., Atkinson, R., Bayne, E., Bellekens, X.: Mqtt-ids2020: Mqtt internet of things intrusion detection dataset. IEEE Dataport (2020)
15. Hoque, N., Singh, M., Bhattacharyya, D.K.: EFS-MI: an ensemble feature selection method for classification. *Complex Intell. Syst.* **4**(2), 105–118 (2018)
16. Idrissi, I., Moussaoui, O., Azizi, M.: A lightweight optimized deep learning-based host-intrusion detection system deployed on the edge for IoT. *Int. J. Comput. Digital Syst.* **11**(1), 209–216 (2022). <https://doi.org/10.12785/ijcds/110117>
17. Ito, Y.: Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. *Neural Netw.* **4**(3), 385–394 (1991)
18. Jaafar, F., Malik, Y., Serre, J., Wang, H., Wang, T.: Lightweight intrusion detection in MQTT based sensor network. In: 2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME). IEEE (2022). <https://doi.org/10.1109/iceccme55909.2022.9988354>
19. Jan, S.U., Ahmed, S., Shakhov, V., Koo, I.: Toward a lightweight intrusion detection system for the internet of things. *IEEE Access* **7**, 42450–42471 (2019)
20. Jaw, E., Wang, X.: Feature selection and ensemble-based intrusion detection system: an efficient and comprehensive approach. *Symmetry* **13**(10), 1764 (2021)
21. Kim, S., Hwang, C., Lee, T.: Anomaly based unknown intrusion detection in endpoint environments. *Electronics* **9**(6), 1022 (2020)
22. Lahasan, B., Samma, H.: Optimized deep autoencoder model for internet of things intruder detection. *IEEE Access* **10**, 8434–8448 (2022)
23. Le, Q.V., et al.: A tutorial on deep learning part 2: Autoencoders, convolutional neural networks and recurrent neural networks. *Google Brain* **20**, 1–20 (2015)
24. Lelewer, D.A., Hirschberg, D.S.: Data compression. *ACM Comput. Surv. (CSUR)* **19**(3), 261–296 (1987)
25. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. arXiv preprint [arXiv:1608.08710](https://arxiv.org/abs/1608.08710) (2016)
26. Li, J.: Research on intrusion detect system of internet of things based on deep learning. In: 2022 International Conference on Machine Learning and Knowledge Engineering (MLKE), pp. 55–58. IEEE (2022)
27. Li, X., Chen, W., Zhang, Q., Wu, L.: Building auto-encoder intrusion detection system based on random forest feature selection. *Comput. Secur.* **95**, 101851 (2020)
28. Li, Y., Qin, T., Huang, Y., Lan, J., Liang, Z., Geng, T.: HDFEF: a hierarchical and dynamic feature extraction framework for intrusion detection systems. *Comput. Secur.* **121**, 102842 (2022)
29. Li, Y., Wang, J.L., Tian, Z.H., Lu, T.B., Young, C.: Building lightweight intrusion detection system using wrapper-based feature selection mechanisms. *Comput. Secur.* **28**(6), 466–475 (2009)
30. Liang, Y.: Efficient temporal compression in wireless sensor networks. In: 2011 IEEE 36th Conference on Local Computer Networks, pp. 466–474. IEEE (2011)
31. Manek, A.S., Shenoy, P.D., Mohan, M.C.: Aspect term extraction for sentiment analysis in large movie reviews using Gini Index feature selection method and SVM classifier. *World Wide Web* **20**, 135–154 (2017)
32. Mendonca, R.V., Silva, J.C., Rosa, R.L., Saadi, M., Rodriguez, D.Z., Farouk, A.: A lightweight intelligent intrusion detection system for industrial internet of things using deep learning algorithms. *Expert. Syst.* **39**(5), e12917 (2022)
33. Moukhafi, M., El Yassini, K., Bri, S.: A novel hybrid GA and SVM with PSO feature selection for intrusion detection system. *Int. J. Adv. Sci. Res. Eng.* **4**(5), 129–134 (2018)
34. Mushtaq, E., Zameer, A., Umer, M., Abbasi, A.A.: A two-stage intrusion detection system with auto-encoder and LSTMs. *Appl. Soft Comput.* **121**, 108768 (2022)
35. Neisse, R., Baldini, G., Steri, G., Ahmad, A., Fournieret, E., Leg-eard, B.: Improving internet of things device certification with policy-based management. In: 2017 Global Internet of Things Summit (GIoTS), pp. 1–6. IEEE (2017)
36. Neumann, U., Riemenschneider, M., Sowa, J.P., Baars, T., Kälsch, J., Canbay, A., Heider, D.: Compensation of feature selection biases accompanied with improved predictive performance for binary classification by using a novel ensemble feature selection approach. *BioData Mining* **9**(1), 1–14 (2016)
37. Nguyen, B.H., Xue, B., Zhang, M.: A survey on swarm intelligence approaches to feature selection in data mining. *Swarm Evol. Comput.* **54**, 100663 (2020)
38. Nguyen, X.H., Nguyen, X.D., Huynh, H.H., Le, K.H.: Realguard: a lightweight network intrusion detection system for IoT gateways. *Sensors* **22**(2), 432 (2022)
39. Okey, O.D., Melgarejo, D.C., Saadi, M., Rosa, R.L., Kleinschmidt, J.H., Rodriguez, D.Z.: Transfer learning approach to ids on cloud IoT devices using optimized CNN. *IEEE Access* **11**, 1023–1038 (2023)
40. Osanaiye, O., Ogundile, O., Aina, F., Periola, A.: Feature selection for intrusion detection system in a cluster-based heterogeneous wireless sensor network. *Facta Universitatis Ser. Electron. Energet.* **32**(2), 315–330 (2019)
41. Otokwala, U.J., Petrovski, A., Kotenko, I.V.: Enhancing intrusion detection through data perturbation augmentation strategy, Unpublished (2024)
42. Paudice, A., Muñoz-González, L., Lupu, E.C.: Label sanitization against label flipping poisoning attacks. In: ECML PKDD 2018 Workshops: Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10–14, 2018, Proceedings 18, pp. 5–15. Springer, Berlin (2019)
43. Peri, N., Gupta, N., Huang, W.R., Fowl, L., Zhu, C., Feizi, S., Goldstein, T., Dickerson, J.P.: Deep k-nn defense against clean-label data poisoning attacks. In: Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16, pp. 55–70. Springer, Berlin (2020)
44. Perumal, G., Subburayalu, G., Abbas, Q., Naqi, S.M., Qureshi, I.: VBQ-Net: a novel vectorization-based boost quantized network model for maximizing the security level of IoT system to prevent intrusions. *Systems* **11**(8), 436 (2023)

45. Rachburee, N., Punlumjeak, W.: A comparison of feature selection approach between greedy, IG-ratio, chi-square, and MRMR in educational mining. In: 2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE), pp. 420–424. IEEE (2015)
46. Rizvi, S., Scanlon, M., McGibney, J., Sheppard, J.: Deep learning based network intrusion detection system for resource-constrained environments. In: International Conference on Digital Forensics and Cyber Crime, pp. 355–367. Springer, Berlin (2022)
47. Rodríguez, D., Ruiz, R., Cuadrado-Gallego, J., Aguilar-Ruiz, J.: Detecting fault modules applying feature selection to classifiers. In: 2007 IEEE International Conference on Information Reuse and Integration, pp. 667–672. IEEE (2007)
48. Roesch, M., et al.: Snort: Lightweight intrusion detection for networks. In: Lisa, vol. 99, pp. 229–238 (1999)
49. Roy, S., Li, J., Choi, B.J., Bai, Y.: A lightweight supervised intrusion detection mechanism for IoT networks. *Futur. Gener. Comput. Syst.* **127**, 276–285 (2022)
50. Sandri, M., Zuccolotto, P.: A bias correction algorithm for the gini variable importance measure in classification trees. *J. Comput. Graph. Stat.* **17**(3), 611–628 (2008)
51. Sayood, K.: Introduction to Data Compression. Morgan Kaufmann, Burlington (2017)
52. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **1**, 108–116 (2018)
53. Sharmila, B., Nagapadma, R.: QAE-IDS: DDoS anomaly detection in IoT devices using post-quantization training. *Smart Sci.* 1–16 (2023)
54. Sharmila, B., Nagapadma, R.: Quantized autoencoder (QAE) intrusion detection system for anomaly detection in resource-constrained IoT devices using rt-iot2022 dataset. *Cybersecurity* **6**(1), 41 (2023)
55. Shone, N., Ngoc, T.N., Phai, V.D., Shi, Q.: A deep learning approach to network intrusion detection. *IEEE Trans. Emerging Top. Comput. Intell.* **2**(1), 41–50 (2018)
56. Siddharthan, H., Deepa, T., Chandhar, P.: SENMQTT-set: an intelligent intrusion detection in IoT-MQTT networks using ensemble multi cascade features. *IEEE Access* **10**, 33095–33110 (2022)
57. Soe, Y.N., Feng, Y., Santosa, P.I., Hartanto, R., Sakurai, K.: Implementing lightweight IoT-IDS on raspberry PI using correlation-based feature selection and its performance evaluation. In: Advanced Information Networking and Applications: Proceedings of the 33rd International Conference on Advanced Information Networking and Applications (AINA-2019), vol 33, pp 458–469. Springer, Berlin (2020)
58. Subbiah, S., Anbananthen, K.S.M., Thangaraj, S., Kannan, S., Chelliah, D.: Intrusion detection technique in wireless sensor network using grid search random forest with boruta feature selection algorithm. *J. Commun. Netw.* **24**(2), 264–273 (2022)
59. Tao, P., Sun, Z., Sun, Z.: An improved intrusion detection algorithm based on GA and SVM. *IEEE Access* **6**, 13624–13631 (2018)
60. Van Der Maaten, L., Postma, E., Van den Herik, J., et al.: Dimensionality reduction: a comparative. *J. Mach. Learn. Res.* **10**(66–71), 13 (2009)
61. Wang, J., Xu, J., Zhao, C., Peng, Y., Wang, H.: An ensemble feature selection method for high-dimensional data based on sort aggregation. *Syst. Sci. Control Eng.* **7**(2), 32–39 (2019)
62. Wang, Z., Chen, H., Yang, S., Luo, X., Li, D., Wang, J.: A lightweight intrusion detection method for IoT based on deep learning and dynamic quantization. *PeerJ Comput. Sci.* **9**, e1569 (2023)
63. Wang, Z., Li, Z., He, D., Chan, S.: A lightweight approach for network intrusion detection in industrial cyber-physical systems based on knowledge distillation and deep metric learning. *Expert Syst. Appl.* **206**, 117671 (2022)
64. Xiao, F., Honma, Y., Kono, T.: A simple algebraic interface capturing scheme using hyperbolic tangent function. *Int. J. Numer. Methods Fluids* **48**(9), 1023–1040 (2005)
65. Xu, Y., Tang, Y., Yang, Q.: Deep learning for IoT intrusion detection based on LSTMs-AE. In: Proceedings of the 2nd International Conference on Artificial Intelligence and Advanced Manufacture, pp 64–68 (2020)
66. Zakariyya, I., Kalutarage, H., Al-Kadri, M.O.: Towards a robust, effective and resource efficient machine learning technique for IoT security monitoring. *Comput. Secur.* **133**, 103388 (2023)
67. Zebari, R., Abdulazeez, A., Zeebaree, D., Zebari, D., Saeed, J.: A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction. *J. Appl. Sci. Technol. Trends* **1**(2), 56–70 (2020)
68. Zeng, D., Wu, Z., Ding, C., Ren, Z., Yang, Q., Xie, S.: Labeled-robust regression: simultaneous data recovery and classification. *IEEE Trans. Cybernet.* **52**(6), 5026–5039 (2020)
69. Zhao, R., Gui, G., Xue, Z., Yin, J., Ohtsuki, T., Adebisi, B., Gacanin, H.: A novel intrusion detection method based on lightweight neural network for internet of things. *IEEE Internet Things J.* **9**(12), 9960–9972 (2021)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.