Future Generation Computer Systems 166 (2025) 107630

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Machine Learning-Based Attack Detection for the Internet of Things

Dawit Dejene Bikila *, Jan Čapek

Institute of System Engineering and Informatics, Faculty of Economics and Administration, University of Pardubice, Studentska 95, Pardubice, 53009, Czech Republic

ARTICLE INFO

Dataset link: https://github.com/Dawitdjn/ML-based-Attack-Detection

Keywords:
Intrusion detection
Internet of Things
Machine learning
Elastic deep autoencoder
Deep learning
Grey wolf optimizer

ABSTRACT

The number of Internet of Things (IoT) device connections is increasing rapidly as IoT applications are vital in any operation. IoT must maintain safe internet access that withstands various malicious attacks for instance Recon, Mirai, Distributed Denial of Service (DDoS), and Spoofing which has gained much attention. Intelligently changing and zero-day attacks are emerging every day. This highlights the need for intelligent security solutions tailored specifically to this technology. Various Machine Learning (ML) based approaches have been utilized for intrusion detection to tackle IoT attacks. However, the flaws of current attack detection and feature extraction techniques result in low detection accuracy. Thus, it hindered their real-world applications and highlighted the need for a lightweight and computationally robust model trained and assessed on a recent datasets. Therefore, this work proposed an attack detection model trained and validated using the CICIoT2023 and CICIDS2017 datasets. Initially, data preprocessing is done then features are extracted by using an unsupervised Elastic Deep Autoencoder (EDA) with optimum hyperparameters. Further, the Extreme Gradient Boosting (XGBoost) binary classifier is tuned by the Grey Wolf Optimizer (GWO) and fed extracted feature sets to classify attacks. The results of the experiments show the effectiveness of our model with a higher detection accuracy in both datasets. Finally, the performance comparison confirmed that the results of the proposed work is competitive with other state-of-the-art method in securing IoT infrastructures.

1. Introduction

The Internet of Things (IoT) is a connection of numerous physical devices to the Internet and shares massive data in cyberspace. It is the leading technology of the modern industrial revolution and is involved in production, business, and management systems. IoT is a vital part of operations as it allows geographically distributed intelligence for monitoring assets. Due to their increasing pervasiveness, it is estimated that 55.7 billion IoT devices will be devised by 2025. These devices possess economic and physical constraints that lead to limitations in both software and hardware resources. With the heterogeneous nature and principal part in the current Cyber–Physical Systems (CPS) evolution, IoT devices have become a prime target for cyberattacks. Thus, the need for intelligent security solutions tailored specifically for this technology is unquestionable [1,2].

IoT has transformed and restructured socioeconomic life since its introduction. Initially, it was employed in local networks, today it is a commonly used smart solution for homes and offices [3]. Many organizations rely greatly on these smart solutions, and shortly most will devise these solutions. However, industries face intelligent security breaches due to the lack of the strangest security mechanisms for IoT, making it a priority target for cybercriminals [4]. As a result, researchers are investigating and proposing new Deep Learning (DL) and/or Machine Learning (ML) techniques to secure IoT. Therefore, it is essential to develop and improve security methods to keep IoT infrastructures secure [5].

ML-enabled intrusion detection models encounter problems due to high-dimensional and imbalanced IoT data which leads to poor detection capabilities. The limitations of ML techniques in feature extraction and interpreting complicated datasets can be solved using DL techniques [6]. Robust and effective feature representation leads to an unambiguous and generalizable model. However, irrelevant, and redundant features may also be categorized as the most relevant features. Moreover, extracting features using manual techniques from high-dimension data is time-consuming, costly, and leads to poor generalization [1]. Automatic feature extraction and dimensionality reduction can be done effectively using DL methods called autoencoders [7].

Autoencoders, prominent unsupervised feature selection, and learning methods are commonly utilized for automatic feature extraction and representation learning. They are well known for their effectiveness in capturing concise and nonredundant sets of pertinent features from

* Corresponding author.
E-mail addresses: dawitdejene.bikila@student.upce.cz (D.D. Bikila), capek@upce.cz (J. Čapek).

https://doi.org/10.1016/j.future.2024.107630
Received 29 March 2024; Received in revised form 9 November 2024; Accepted 22 November 2024
Available online 30 November 2024

high-dimensional data. Hence, combining autoencoders with ML methods such as Support Vector Machine (SVM) to enhance the detection rate has been explored by researchers [8]. Autoencoders can distinguish both nonlinear and linear transformations and are suitable for classifying data as well. They can effectively address the limitation of traditional Principal Component Analysis (PCA), during linear dimensionality reduction and problem associated with intrusion behavior and network data [9]. Further, in real-world scenarios, the availability of large, clean real datasets without outliers is a challenge. Conversely, the L1,2-norm is recognized for its robustness against outliers, and noises such as irregular, sparse and, laplacian patterns [10].

Attack traffic features target a wide range of devices, imposing tough challenges and a broad scope of potential attack vectors [11]. The primary challenges of the current attack detection models include acquiring high-quality labeled datasets. It is a resource-intensive process that demands significant time and expert manual analysis. Consequently, employing intrusion detection techniques that utilize supervised learning can be very costly. Moreover, probability density estimation and cluster analysis are often utilized by unsupervised detection techniques for dimensionality reduction that may not be suitable for high-dimensional data [12].

Recent methods, such as feature selection, ensemble, and robust optimization methods have demonstrated good results in detecting IoT attacks. However, many models rely on simulated data to construct detection models. This approach does not always reflect actual traffic conditions. The detection capabilities of the suggested models are limited as they are trained on simulated and older datasets. Moreover, using such data to develop reliable models is not suitable for mitigating current attacks targeting IoT.

Despite advancements in ML-based intrusion detection for IoT networks, significant and complex gaps remain to be addressed in developing models that are robust, lightweight, and resource-efficient. Specific research gaps that this paper aims to fill are described as follows. Diverse intelligent attack mitigation is still a major challenge as the resource-constrained nature of IoT makes the deployment of a resource-intensive model impossible. Supervised dimensionality reduction techniques devised by current methods are costly [12]. Utilizing a high dimensional, imbalanced and noisy dataset for intrusion detection model development results in poor detection rate, generalizability and higher training time. In addition, large, clean datasets without outliers are rare, which makes robust dimensionality reduction techniques essential for handling nonlinear and linear transformations and high-dimensional data. Moreover, existing detection systems have relatively low attack diversity [13]. Consequently, the need for optimization, scalability, simplification and dimensionality reduction, to develop robust and dependable models is evident.

This paper addresses these gaps by proposing an optimized lightweight model, trained on recent, larger diversity and real attack instances, with improved unsupervised dimensionality reduction and with optimal resource utilization.

1.1. Motivations and contributions

The increasing number of zero-day and diversity of attacks in the IoT environment is a major challenge for its applications. Complex and resource-intensive models are proposed for IoT security. Therefore, this research is motivated to develop a more robust, lightweight, and resource-efficient model. The major contributions of our work are:

- An optimized model has been proposed that uses an optimized Elastic Deep Autoencoder (EDA) for automatic feature representation with reduced overfitting and noise. It contributes to improved dimensionality reduction with reduced reconstructional and least resource utilization.
- GWO is used for tuning hyperparameters of the XGBoost classifier to achieve a higher performance.
- A lightweight model, trained on recent, larger diversity and real attack instances, with higher detection accuracy and less computational cost is realized. Our model is evaluated on CICIoT2023 and CICIDS2017 datasets, and achieved a higher detection accuracy.

The rest of the paper is structured as follows. Section 2 presents a literature survey on intrusion detection methods. Section 3 provides a detailed explanation of the methods of our work. In Section 4 the evaluation metrics and experimental design setup are discussed. Section 5 describes the experimental results and a comparison. The conclusion and future work are stated in the last section.

2. Related work

A study [8] proposed a feature representation sharing between an autoencoder and a classifier by applying the Weight-Embedding technique. The model allows the extraction of relevant information from the dataset by embedding the trained autoencoder layer before passing it to the classifier. The approach then employs the relevant features on Convolutional Neural Network (CNN) and Deep Neural Network (DNN) for classification. The results show the DNN improved the accuracy by 0.4% and CNN by 0.5% on the NSL-KDD dataset. Moreover, using the UNSW-NB15 dataset, CNN improves the accuracy by 0.5%, whereas DNN shows improvement by 2.8%. The results indicate that the Weight-Embedding technique is more effective with DNN. However, the weight is dependent on the trained encoder part which can result in a propagation error.

Other work [14] developed an Embedded Stacked Group Sparse Autoencoder (ESGSAE) for more efficient feature learning. They designed an SVM and weighted local discriminant preservation projection-based ensemble to enhance the feature extraction quality. The proposed approach was validated using publicly available representative datasets. The experimental results show utilizing the manifold reduction and L1 regularization in the proposed model has improved the classification accuracy. In [15] a stacking ensemble of deep learning models, termed Deep Integrated Stacking for the IoT (DIS-IoT) was proposed. DIS-IoT combines four different deep learning models—a shallow Multilayer Perceptron (MLP), a Deep Neural Network (DNN), and models based on Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) into a fully connected layer to enhance detection accuracy. The method was evaluated on three open-source datasets: ToN-IoT, CICIDS2017, and SWaT, demonstrating high accuracy and low false positive rates in both binary and multi-class classification tasks. However, the authors recommended using an edge-based network instead of running the model directly on constrained IoT devices. This is because DIS-IoT is a resource-intensive ensemble of four distinct models.

An intrusion detection model that employed a stacked sparse autoencoder with an improved Gaussian mixture [12] was developed . In this approach, linear dimensionality reduction was done using Pearson Correlation Coefficient (PCC). The non-linear dimensionality reduction was done using a stacked sparse autoencoder. The model achieved an accuracy of 96.7% on the UNSW-NB15 dataset indicating the advantage of the proposed model over the conventional unsupervised intrusion detection models. However, sparse autoencoders have a limited capacity to capture very complex patterns in high-dimensional data [16]. Moreover, evaluating model on a single dataset may not indicate better performance in broader setup.

The authors [17] a fusion-based approach that fuses Principal Component Analysis (PCA) with Autoencoder (AE) techniques to reduce dimensionality while maintaining linear and non-linear data relationships. This dimensionality reduction is integrated with a Long Short-Term Memory (LSTM) model for intrusion detection. The approach was evaluated using four benchmark datasets: The proposed approach was tested on four benchmark datasets: NSL-KDD, UNSW-NB15, CIC-IDS-2017, and MSCAD. The results demonstrated that the

fusion-based method yielded enhanced accuracy compared to the traditional AE+LSTM approach. On NSL-KDD, the fusion method exhibited a 3% improvement, while on UNSW-NB15 and CIC-IDS-2017, the gains were approximately 1%. Notably, the fusion method achieved comparable results on MSCAD. A Wilcoxon signed-rank test confirmed the statistical significance of these improvements. However, the paper does not sufficiently address the computational overhead of the fusion method, which could be a limitation in resource-constrained environments.

In another study [18] an intrusion detection system that uses Feed Forward Neural Network (FFNN) for attack classification was developed. They applied an information gain to evaluate features used for the parameters of the GWO. The Extreme Learning Machine (ELM) model used in this approach achieved 98% detection accuracy. In [19] a lightweight and fog computing-based intrusion detection model using a combination of Variational Autoencoder (VAE) and Multi-layer Neural Networks was introduced. The model applied a two-layered architecture for anomaly detection. They evaluated the approach using two datasets and reported a that the model achieved 99.98% detection accuracy and 0.01% false alarm rate. However, VAEs are prone to a fault in identifying outliers when the distribution of anomalies in the test and training data are the same [20].

A deep learning-based intrusion detection framework was developed [21] using a Denoising Autoencoder (DAE) for training DNN. The model is used for robust feature extraction from heterogeneous, unlabeled IoT datasets. A comparison was done with the IoT-based anomaly detection models sparse DAE, RBM, and SAE. The results of the experiments show their model achieves an accuracy of 93.5%. The authors indicates the inherent need of detection models trained on recent, heterogeneous and mixed data. In [22] authors developed an intelligent intrusion detection with IoT capabilities for the detection of trajectories for naval transportation systems. The model applied the Variational Autoencoders (VAE) to extract patterns between each dimension of trajectories. The transfer learning utilized in this model resulted in reduced training time. A Multi Step CNN Stacked LSTM architecture (MSCSL) attack detection model was proposed [23] for IoT-enabled medical applications. They optimized the hyperparameters of the proposed MSCSL using a Light Spectrum Optimizer (LSO). They evaluated the model using TON-IoT, IoHT, and SCADA IEC 60870-5-104 datasets. These datasets include attacks like DoS, brute force, and port scans. The evaluations done using common metrics show that the proposed method achieves 98.85%, 98.74%, 98.5%, 96.3% of accuracy, F1-score, precision, and recall respectively.

Anomaly classification and detection of malware-related records model in IoT infrastructures [24] was proposed. This model combines Particle Swarm Optimization (PSO) and the AdaBoost algorithm. They used the NSL-KDD dataset for training and validation of the proposed model. PSO was used for the feature selection process to identify 12 relevant features. The proposed PSO-AdaBoost model achieves an accuracy value of 98.7% and a recall value of 96.67%. The potential of ML-IDS security solutions for securing IoT infrastructures was indicated in the work. However, the limited capability of PSO in exploring the search space, mentioned by the authors, may lead to poor feature representation. Moreover, the reliance on a single and an older dataset may limit the generalizability of the results to contemporary IoMT security challenges [25] . A DNN-based bagging classifier [26] was developed in another work. In this work the DNN as a base estimator to balance class weight used for DNN training. Four distinct intrusion detection datasets were utilized to evaluate the model, namely, UNSW-NB-15, NSL-KDD, BoT-IoT, and CIC-IDS-2017. The result shows an accuracy of 98.99% when applied to the BoT-IoT dataset. Using larger and more diverse datasets for evaluation could have improved this work [15].

Most studies have reported good accuracy scores but the datasets utilized lack representation of recent attacks. In addition, some models with more reasonable feasibility solved the challenges. However, they are resource-demanding, and have low accuracy rates, high training time, and complex structures. Thus, hinders their real-world applications, and indicates the need for the development of a lightweight and computationally robust model. Moreover, models should be trained and validated on practical and real datasets so that their deployment can be realized immediately.

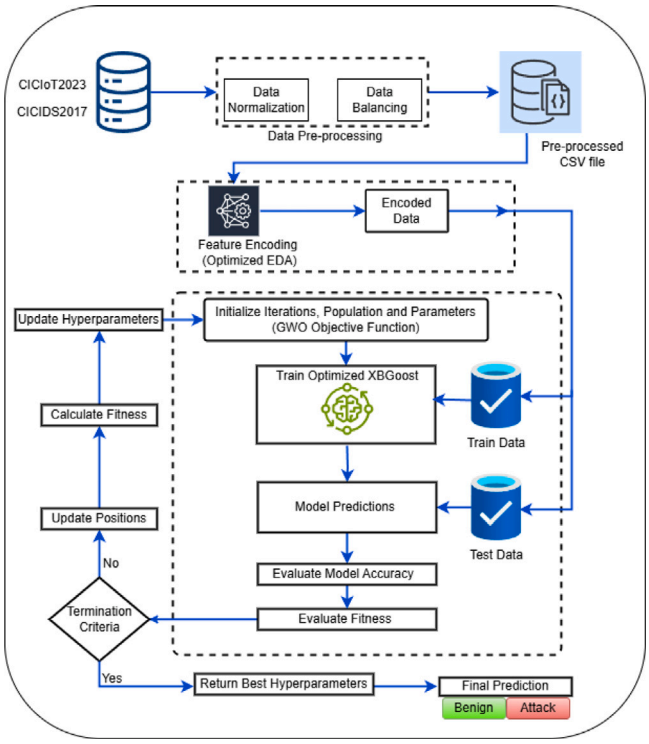


Fig. 1. Basic workflow of our model.

Based on the above findings, the need for efficient intrusion detection methods for IoT infrastructures is evident (see Table 1). Therefore, this work proposes an optimized attack classification model. It employs the EDA for feature selection and GWO-optimized XGBoost for classification.

3. Methodology

In this section, a detailed discussion is presented on the workflow of our model, the dataset chosen and preprocessing, data balancing, feature normalization and selection, and algorithms used. The data preprocessing on the dataset is conducted, which includes scaling using the Min–Max scaler for normalization. We used the Synthetic Minority Oversampling Technique (SMOTE) technique for data balancing before we extracted the most informative feature using EDA. This approach is suited for IoT datasets due to IoT datasets often have high dimensionality and imbalance. EDA facilitates the extraction of key features from high-dimensional data. They identify complex patterns and balance sparsity and feature representation. Finally, these selected features are then used by the GWO-optimized XGBoost binary classifier.

Generally, the use of the sparsity constraint enhances interpretability and feature selection [27]. Hence, EDA is trained using a loss function that combines the sparsity-inducing term with reconstruction error to diminish data loss caused by noise and encourage feature sparsity. In addition, regularization techniques such as L1 and L2 regularization are utilized to advance model generalization and diminish overfitting.

Fig. 1, depicts a data preprocessing followed by a deep learning EDA that compresses the feature set. A random search algorithm was used

Table 1
Comparison of previous works.

Work	Year	Technique	Pros	Cons
[15]	2024	DIS-IoT	High detection accuracy, Low FPR	The structure is complex, Resource intensive
[17]	2024	PCA+AE+LSTM	Reduced overfitting	High false positive rates
[12]	2023	SIGMOD	Evaluated with different models	Only one dataset is used to evaluate the model
[8]	2023	WE-AE+DNN/CNN	Handles high dimensional data, improved performance	Weight is dependent on the trained encoder part
[24]	2024	PSO-AdaBoost	Reduced features, Better classification	Slower, single and older dataset is used, data imbalance
[18]	2022	MGWO-ELM	Improved convergence and efficiency	No noise handling, tuned to a specific dataset

for tuning the hyperparameters of the EDA that has two encoder and decoder layers. Relevant features were extracted using this EDA and then used by the XGBoost classifier. This classifier is widely recognized for producing high levels of accuracy when dealing with datasets that have varied column types. Furthermore, it undergoes optimization by utilizing the GWO algorithm. This algorithm, which implements a meta-heuristics, was developed by [28]. It is used to select optimal hyperparameters, hence the classification task is further accelerated.

In this work, we proposed a feature reduction using EDA and GWO optimized XGBoost attack detection model to overcome the challenges in securing IoT infrastructures. To the best of our knowledge, our approach is novel. An optimized EDA, with a new loss function, is also integrated to minimize the reconstruction loss and overfitting problems associated with high dimensionality and imbalanced data. GWO is used for hyperparameter tuning of a classifier, trained with optimal parameters, which resulted in a better performance.

We utilized an optimized EDA model for feature selection and dimensionality reduction. A random search, which gives optimal parameters with few interactions, was used to obtain the best parameters for EDA. A simple network that has 3 dense layers with 64, 32, and a latent representation layer 16 were employed. The activation function Rectified Linear Unit (ReLU) was devised in each hidden layer and the EDA was compiled using the Adam optimizer. Then, the EDA is trained using the obtained parameters to extract the features. The Mean Absolute Error (MAE) and Mean Squared Error (MSE) loss are returned after the best parameters are generated. These losses are used to determine the reconstruction error.

An effective feature extraction and encoding technique (EDA) was realized to extract complex relationships and informative features from the high-dimensional data. As a result, the classifier benefited from the relevant features extracted by the optimized EDA which leads to a better generalization. In the case of high variation and real-time input data, XGBoost is advantageous. In addition, XGBoost is preferred for classification tasks because of its extreme speed of execution and effective real-world performance. Moreover, it has also major advantages, such as handling data imbalance internally, improved performance, and countering overfitting [29].

The initialization for the hyperparameters of the XGBoost classifier explained in Section 3.3.2 is set. Then the objective function of the GWO is used, which takes all the initialized hyperparameters, the best parameters. This function returns the best hyperparameters with the maximum accuracy attained. The classifier is trained using encoded features (X_train) labels by taking the best hyperparameter. After training the model on the train data, it is used to generate predictions using the encoded_features(X_test) labels. The details are given in Section 3.3.3.

The combination of EDA and GWO significantly advances the state-of-the-art by leveraging the strengths of both methods to address critical challenges in intrusion detection. The exploration and exploitation balance of GWO is beneficial to handle problems with large, unknown search domains and complexity. The optimized EDA excels in extracting rich feature representations, with a negligible reconstructional error, from high-dimensional data, enhancing anomaly detection accuracy. Moreover, the spatial complexity of the XGBoost classifier is addressed by utilizing the EDA for dimensionality reduction. When coupled with GWO for hyperparameter tuning, the model resulted in optimal configurations efficiently with reduced computational complexity and improved performance. Thus, the XGBoost's computational cost of the extensive grid search, to determine the best hyperparameters is diminished resulting in less training and inference time. As a result, a robust, adaptive, and resource-efficient intrusion detection model that effectively handles diverse and evolving attack vectors, offers superior real-world applicability and reduces false positives is realized.

XGBoost stack trees sequentially and allow trees to learn from the errors generated by previous trees. The computationally expensive extensive grid search is used for determining best parameters [30]. Therefore, we utilized GWO, which searches the hyperparameter space by balancing exploration and exploitation, to determine the best hyperparameters with reduced search iteration and time using the objective function. As a result, the XGBoost computational cost of the extensive grid search is diminished resulting in less training and test time.

3.1. Dataset and pre-processing

It is the preliminary stage of data preparation to render the dataset so that it fits for machine learning model training. This stage removes outliers and unwanted data, missing values from the dataset. A uniform scale of the whole features was generated initially using min–max scalar normalization techniques. Then, the EDA is applied to get the most relevant features from both CICIoT2023 and CICIDS2017 datasets.

3.1.1. Dataset

This work used a recent publicly available comprehensive realistic IoT dataset with benign and malicious data. The CICIoT2023 is a benchmark and real-time IoT dataset that is used for broad attack detection model development for IoT environments. We obtained it from the Canadian Institute of Cyberattack website. It is widely used for anomaly detection system use cases [31]. The dataset is generated from a real IoT environment which constitutes the real up-to-date broad types of novel and rare attacks, and complex and normal network traffic features. Moreover, it consists of a wide collection of IoT attacks that are real. It incorporates 33 distinct attacks classified into seven classes these are Recon, DDoS, Mirai, Spoofing, DoS, Brute Force, and Web-based. These attacks are executed on an IoT setup that involves 105 devices.

Further, we have used the real-time CICIDS2017 evaluation and benchmark intrusion detection dataset in this work. The main objective is to assess the robustness and performance of the proposed model through varied datasets and attacks. This dataset is also obtained from the Canadian Institute of Cyberattacks website. This dataset consists of 2,830,743 records, each with 79 attributes, and 19.70% of these records are network flows and the rest 80.30% are benign. The dataset contains a wide collection of attacks that are real. It incorporates six common attack types (DoS, Botnet, Infiltration, Brute Force, Portscan, Web Attack) along with 14 sub-attack types [32]. Due to the detailed and high-quality features, the dataset is mostly used for evaluating the performance of intrusion detection systems.

In this study, we exploited the SMOTE [33] to address class imbalance and enhance the predictive performance of our machine learning model. SMOTE creates synthetic samples for the minority class by selecting random observations and precisely modifying them to generate new data points. This up-sampling approach is employed to tackle the challenges posed by imbalanced datasets. This verifies that the machine learning model is effective in making accurate predictions for both

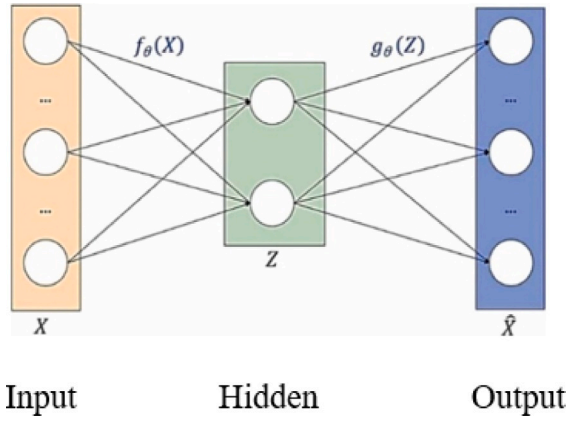


Fig. 2. A three-layer autoencoder architecture [8].

majority and minority classes. By introducing additional data from the under-represented class, our model becomes more robust and capable of handling class imbalances effectively, leading to improved overall performance. Data normalization is done using the Min–max scaler technique.

3.2. Autoencoder for feature encoding

The structure of an autoencoder is quite similar to that of general neural networks, comprising an input, hidden, and output layer. The output layer is a representation of the reconstructed features using the input and hidden layers that have an equal number of neurons. As a result, models that employ autoencoder achieve higher accuracy and reduced training time [34].

$$Z = f_\theta(x) \tag{1}$$

$$\hat{X} = g_\theta(x) \tag{2}$$

Fig. 2 illustrates a basic autoencoder with input, hidden, and output layers, as specified in the above Eqs. (1) and (2).

An encoder and decoder functions are the two main components of an autoencoder. The encoder functions to map the input X to Z, which is the hidden layer, representing X with the function f_θ. The decoder functions inversely to generate $\hat{X}$ using the hidden layer Z and the g_θ function. The reconstruction of the similarities between X and $\hat{X}$ is done using simultaneous learning of θ parameters. A minimized error between X and $\hat{X}$ is an indication of successful learning. The encoder and decoder functions of an autoencoder can be represented with the following notation.

3.2.1. Elastic Deep Autoencoders(EDA)

In this work, we have used EDA which is a variant of deep autoencoders, insensitive to noise and outliers, preferred due to a loss function they comprise [10]. These autoencoders utilize a resilient loss function called elastic loss, which draws its inspiration from the elastic net. The new reconstruction loss function is defined and denoted Lel(.), as follows:

$$L_{\text{el}}(x_i, \hat{x}_i) = \sum_i \frac{\delta \|x_i - \hat{x}_i\|^2}{\delta + \|x_i - \hat{x}_i\|^2} + \sum_i \frac{\|x_i - \hat{x}_i\|^2}{\|x_i - \hat{x}_i\|^2 + \delta} \tag{3}$$

L_2 pseudo and L_1 pseudo losses are included in this loss function and the value depends on δ scale parameters L_1 and L_2 norms are regularization techniques used for the training phase to prevent overfitting and improve the generalization of the model. Regularization norms are used in the loss function to penalize the model for having large weights. The sparsity generalization, elastic net regularization, L_1(lasso), and L_2(ridge) regularization terms are also used to encourage sparsity in the encoder weights (We) that can be denoted as follows:

$$L_{\text{er}} = \alpha \sum_{ij} |w_e^{(ij)}| + \frac{1}{2}\beta \sum_{ij} (w_e^{(ij)})^2 \tag{4}$$

where α controls the strength of L_1 regularization, β controls the strength of L_2 regularization. We have used the MSE, a common cost function used for autoencoders, which is optimal against Gaussian-distributed additive noise. We have also used MAE, less sensitive to outliers than MSE because it does not square the differences, for larger errors [35]. The total loss function combines elastic net regularizations and elastic loss:

$$L_t = L_{\text{er}} + L_{\text{el}} \tag{5}$$

3.2.2. Tuning hyperparameters of EDA using random search algorithm

A proper tuning of a model's hyperparameters leads to better performance, [36]. The EDA hyperparameters selected for random search tuning are batch size, L1 and L2 regularization coefficients, activation functions, dropout rate, number of epochs, learning rate, number of random trials (n_trials) and number of hidden layers. To enhance EDA efficiency, a random search was employed to fine-tune the hyperparameters mentioned above. Random search, randomly selects values from a bounded domain of search space, offers the advantage of easy implementation and computational efficiency. It identifies good hyperparameter combinations with only a few trials. Hence, it is ideal for situations with limited computational resources. The algorithm is described below.

Algorithm 1 Training an Optimized EDA for Feature Encoding

```
Initialize best_params and best_score
for trial in range(num_trials) do
    Sample params = RandomSampleParams()
    Build: AE = BuildAutoencoder(params)
    AE.fit(X_train, X_train, params)
    score = EvaluateModel(AE, X_train)
    if score < best_score then
        best_score = score
        best_params = params
    end if
end for
best_AE = BuildAutoencoder(best_params)
best_AE.fit(X_train, X_train, best_params)
encoded_features=EncodeFeatures(best_Encoder, X_train)
Output: encoded_features
```

3.3. Extreme Gradient-Boosting(xgboost) classifier

Chen and Guestrin developed XGBoost algorithm which is an ensemble learning algorithm. It is commonly used for classification tasks [37]. There are numerous reasons to select this classifier for malware classification. Firstly, it identifies the optimal tree model and copes with data values that are missing. Additionally, it enhances computational performance using parallelism and it stores statistical information using internal buffers. It solves the problem complexity using lasso and ridge inbuilt regression regularizations techniques. It create various tree nodes concurrently. Finally, tree pruning is carried out using a depth-first search approach.

A built-in weighted quantile sketch technique is used by XGBoost algorithm which helps to identify precise split points. Moreover, it incorporates a validation technique using cross-validation. This algorithm employs an ensemble learning approach that sequentially creates decision trees. Each tree assigns weights to all independent variables, which are then utilized in prediction. Further, the weights of inaccurately predicted variables are increased and fed into the next decision

tree. These trees are combined to create a model that is not only more accurate but also more robust.

The XGBoost shows better performance in many practical applications. For a given data set D = {(x_i, y_i) i = 1, 2, …, n, y_i = {0, 1}} the prediction result is expressed as follows:

$$\hat{y}_i = \varphi(x_i) = \sum_{s=1}^{\text{n\_estimator}} f_s(x_i) \tag{6}$$

where "fs" refers to the sth regression tree and "n_estimator" represents the number of trees, the objective function of XGBoost is denoted as follows:

$$L(\varphi) = \sum_{i=1}^{n} l(\hat{y}_i, y_i) + \sum_{s=1}^{\text{n\_estimator}} \Omega(f_s) \tag{7}$$

where l($\hat{y}_i$, y_i) denotes convex loss function between the real value of y_i and the predicted value $\hat{y}_i$, $\Omega(fs)$ stands for complexity penalty term of the model. The ultimate goal is to establish a trained model capable of precisely predicting the category for each instance, the loss function is given by:

$$l(\hat{y}_i, y_i) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\left(\frac{1}{1+e^{-\varphi(x_i)}}\right) + (1-y_i)\log\left(1-\frac{1}{1+e^{\varphi(x_i)}}\right)\right] \tag{8}$$

The complexity factor of the tree is determined by the number and score of a leaf nodes. Eq. (9) below denotes the penalty function:

$$\Omega(f_s) = \gamma T + \frac{1}{2}\lambda\|\omega\|^2 \tag{9}$$

where T represents the count of tree leaves, while ω stands for the weight vector assigned to each leaf node, which is output score of each respective node. The parameters γ and λ represent the associated penalty coefficients, that needs precise tuning to achieve a well-balanced tree complexity. To train the model, the objective function must be optimized using a forward distribution algorithm. The prediction for the sample x_i in the sth tree determined by both the current prediction and preceding t - 1 prediction values. The evaluation of the new tree can be conducted using the following formula:

$$L^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\left(\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda}\right) + \gamma T \tag{10}$$

where g_i is the first-order derivative, and h_i is the second-order derivative of the loss functions, expressed as $g_i = \delta\hat{y}(t-1)|(y_i, \hat{y}(t-1))$ and $h_i = \delta 2\hat{y}(t-1)|(y_i, \hat{y}(t-1))$. The XGBoost model's learning capability is enhanced through incremental training of samples. The model's effectiveness is highly dependent on the chosen parameters. Hence, instead of random selection, it is evident to employ a heuristic optimization algorithm for parameter tuning. This approach can result in a model with better performance trained with optimal parameters.

3.3.1. XGBoost hyperparameters

We utilized the EDA for dimensionality reduction that helps to solve the spatial complexity of XGBoost classifier preordering. We also implement parameter tuning, such as the *learning_rate* parameter, to control the contribution of each tree to the final prediction. Careful tuning is necessary to converge to an optimal solution when the model has multiple hyperparameters. The important hyperparameters of the classifier, such as *learning_rate*, bootstrap, *max_depth*, *max_leaf_nodes*, *min_weight_fraction_leaf*, and *n_estimators*, have been prioritized. We have utilized these six parameters to develop faster and more accurate model.

The well-tuned hyperparameters listed above results in a more robust and accurate version of the classifier. It also reduces overfitting using a termination criterion triggered when the model reaches the specified tree limit. The higher values of tree depth of the classifier may consume significant memory, hence it must be set to optimal value to develop a more efficient model. Therefore, the learning rate is set to an optimal value.

3.3.2. GWO algorithm for hyperparameter optimization

The performance of machine learning models can be improved using optimization algorithms. A Grey Wolf Optimization (GWO) algorithm has several advantages including the exploration and exploitation balance. It handles problems with large, unknown search domains and complexity. It can perform multi-objective optimization in real applications [28]. Due to computational complexity, choosing the optimal hyperparameters is not easy. The primary benefit of using hyperparameter optimization is developing an objective function and evaluation metrics that result in a better model performance.

Algorithm 2 Hyperparameter optimization of XGBoost Classifier using GWO Algorithm

function GreyWolfAlgorithm(iterations, search_space)
  $pos_{\alpha,\beta,\delta} \leftarrow$ rand_init(search_space)
  for iteration in range(iterations) **do**
    $fitness_{\alpha,\beta,\delta} \leftarrow$ objective_function($_\alpha$)
    $\alpha, \beta, \delta \leftarrow$ update_hierarchy($pos_{\alpha,\beta,\delta}$,$fitness_{\alpha,\beta,\delta}$)
    $a = 2 -$ iteration $\times$ (2/iterations)
    $A1, A2, A3 \leftarrow 2 \times a \times$ rand_val[0-1](())$-a$
    $C1, C2, C3 \leftarrow 2 \times$ rand_valval[0-1](())
    $X_\alpha \leftarrow \alpha - A1 \times |C1 \times \alpha - \text{pos}_\alpha|$
    $X_\beta \leftarrow \beta - A2 \times |C2 \times \beta - \text{pos}_\beta|$
    $X_\delta \leftarrow \delta - A3 \times |C3 \times \delta - \text{pos}_\delta|$
    $\text{pos}_\alpha, \text{pos}_\beta, \text{pos}_\delta \leftarrow X_\alpha, X_\beta, X_\delta$
  end for
  best_hyperparameters $\leftarrow \alpha \mid \beta \mid \delta$
  return best_hyperparameters
end function
function random_initialize(search_space)
  $\text{pos}_{\alpha,\beta,\delta} \leftarrow$ random_uniform(search_space[0 − 1])
  return $\text{pos}_{\alpha,\beta,\delta}$
end function
function objective_function(hyperparameters)
  $Xgb_model \leftarrow$XGBoostClassifier(** hyperparameters)
  $scores\_val \leftarrow$ cross_val_scores($Xgb\_model, X\_tr, y\_tr, cv = 5, scoring =' acc'$)
  return mean($scores\_val$)
end function
function update_hierarchy(pos_α, β, δ, $fitness_{\alpha,\beta,\delta}$)
  $hierarchy \leftarrow$ argsort([$fitness_{\alpha,\beta,\delta}$])
  $\alpha, \beta, \delta \leftarrow$ hierarchy
  return pos[α], [β], [δ]
end function
function random_values
  return random_uniform(0, 1)
end function

The algorithm begins by initializing a population of wolves, alpha, beta, and delta, randomly using the search domain. Then, it sets the maximum iteration, and calculates the fitness based on the objective function for each wolf. Fitness is recalculated for each wolf and the best fitness values are identified to update the positions of all wolves. Fitness is re-evaluated for every wolf based on their new positions, and the wolves are updated according to their fitness. The termination check determines if the algorithm should continue iterating, based on a criterion. If the criterion is not met, the process repeats; otherwise, the algorithm terminates, providing the best solution found. We have the optimized XGBoost classier once the execution of algorithm 2 is completed. Thus, it can now be applied to the encoded data to perform binary classification.

3.3.3. An optimized XGBoost for binary classification

The optimized XGBoost Classifier receives EDA-encoded data features as input for predicting whether the feature is normal or malware. These steps are described in the algorithm below:

Table 2
Hyperparameter tuning example.

Trial	Bat_Size	L1	L2	Enc_Dim	Num epochs	LR	Acc (%)	Best_Score	Best_Params
1	128	0.01	0.001	16	100	0.001	98.5	98.5	Iteration 1 Params
2	64	0.001	0.0001	32	150	0.0001	99.0	99.0	Iteration 2 Params
3	256	0.1	0.01	32	200	0.02	98.8	99.0	Iteration 2 Params
4	1024	0.0001	0.0001	16	300	0.03	99.2	99.2	Iteration 4 Params
5	32	0.01	0.1	16	50	0.001	98.7	99.2	Iteration 4 Params

Algorithm 3 Optimized XGBoost Classifier for Binary Classification

Initialization:
$XGBoost \leftarrow$ XGBoost(** hyperparameters)
$XGBoost \leftarrow$ XGBoost(EDA_enc_feature (X_train, y_train))
Train XGBoost Classifier:
initial predictions $\leftarrow F_0(x_i)$ for each instance in X_train
$F\_previous$ = np.zeros(len(X_train))
for $t = 1$ to T **do**
  for t in interval $(1, T+1)$ **do**
    $r_{it} = y_i - F_{t-1}(x_i)$
    for i in X_train **do**
      e $\leftarrow y\_train - F\_previous$
      $h_t \leftarrow$ DTRegressor(max_depth = Max_depth)
      h_t.fit(X_train, e)
      Update: $F_t(x) = F_{t-1}(x) + \eta \times h_t(x)$
      $F_{current} = F_{previous} + lr\_rate \times h_t.predict(X_{train})$
      $r_i(t+1) = r_{it} - \eta \times h_t(x_i)$
      $e \leftarrow e -$ lr_rate $\times h_t$.predict(X_{train})
      Set F_{current} **next iteration:**
      $F\_previous = F_{\text{current}}$
    end for
  end for
end for
$XGBoost$ Classifier $\leftarrow L1/L2$
for x_i in X_test, calculate final prediction $F_T(x_i)$ **do**
  $y\_pred = F_{\text{previous}}$
  Evaluate (Model, accuracy, ROC-AUC)
end for
Output: Final XGBoost Classification Results

The basic principles of the classifier and its regularization are used in this work. To minimize the loss function for the encoded features it builds trees iteratively. The sum of the trees is calculated for each sample to acquire the final predictions.

3.4. Random search for optimizing the EDA(example)

Consider we have the following search spaces given below: hyperparameter_ranges = 'batch_size': [32, 64, 128, 256, 1024], 'l1_reg': [0.1, 0.01, 0.001, 0.0001], 'l2_reg': [0.1, 0.01, 0.001, 0.0001], 'encoding _dim': [16, 32] 'num_epochs': [50, 75, 100, 150.200.300], 'learning_ rate': [0.0001,0.001,0.01, 0.02.0.03]. This example will iterate 5 times, taking samples from the hyperparameter space to determine the best hyperparameters. Hence, iteration 4 parameters are considered as the best parameters in this example (see Table 2).

3.5. Simplified grey wolf optimization over 5 iterations(example)

For this simple example, 5 iterations and a search space [0, 1, 2, 4, 5] are considered. The initial positions of α, β, and δ are randomly selected. The fitness values of each wolf are calculated using the objective function and updated in each iteration. For each iteration, the variable a decreases linearly from 2 to 0. A1, A2, and A3 are calculated based on random values and (a). C1, C2, and C3 are also generated randomly. New positions for α, β and delta δ are calculated using the formulas given in the algorithm. The hierarchy of alpha α, β and delta δ is updated based on their fitness. The final iteration indicates the position of α, β, δ are 3.61, 1.7, and 4.85 respectively and fitness of α, β and delta δ: 0.92, 0.63, and 0.97 respectively. Hence the delta δ is the best wolf with the highest fitness value of 0.97 in this example (see Table 4).

Table 3
GWO hyperparameters ranges.

Parameter	Range
learning_rate	(0.01, 0.3)
max_depth	(5, 20)
subsample	(0.6, 1.0)
min_child_weight	(0.5, 2.0)
n_estimators	(100, 300)
max_leaf_nodes	(1, 9)

3.6. Hyperparameter optimization of XGBoost(example)

Example of hyperparameter tuning of XGBoost using the Grey Wolf Optimization (GWO) algorithm. The example assumes a population of three wolves (α, β, δ) and shows their evolution over the iterations. The hyperparameter ranges are indicated in Table 3.

After 4 iterations (see Tables 5–8), the alpha (α) wolf has found the best hyperparameters with a fitness score of 99.2. The selected hyperparameters with the best accuracy are indicated in bold in Table 8 and are also given below:

Parameter	**Value**
learning_rate	0.15
max_depth	12
subsample	0.82
min_child_weight	1.1
n_estimators	220
max_leaf_nodes	6

4. Experimental setup

This section presents the experimental setup and the evaluation metrics that we have used in this work. We have trained and experimentally validated the performance of our model on the chosen recent CICIoT2023 and CICIDS2017 datasets. Experiments were conducted on both the CICIoT2023 and CICIDS2017 datasets after We divided the datasets into features (X) and labels (y). Except the 'label' column, which is the target label, the rest are all features. Specifically, we have used the *train_test_split* function of sci-kit-learn, and divided the dataset into training, testing, and validation sets for CICIoT2023, with a proportion of 60%, 30%, and 10% respectively. Besides, we have used the same function to split CICIDS2017 dataset into train and test sets. we had a *train_test_split* with a proportion of 80%, 20% respectively. A comparison of the experimental results is made with the state-of-the-art approach which we have presented in the next section.

Table 4
Grey Wolf algorithm iterations.

Iteration	Position of α, β, δ	Fitness of α, β, δ	a	A1, A2, A3	C1, C2, C3	Updated positions of α, β, δ
1	4, 1, 5	0.8, 0.5, 0.9	2.0	−0.4, 0.8, −1.2	0.2, 0.7, 1.3	4.08, 0.9, 5.4
2	4.08, 0.9, 5.4	0.85, 0.4, 0.92	1.6	0.5, −0.6, 1.1	1.1, 0.4, 0.9	3.79, 1.15, 4.59
3	3.79, 1.15, 4.59	0.88, 0.52, 0.94	1.2	0.3, −1.0, 0.4	1.2, 1.3, 0.7	3.95, 1.58, 4.33
4	3.95, 1.58, 4.33	0.9, 0.6, 0.95	0.8	−0.2, 0.4, −0.9	0.5, 0.9, 0.8	3.61, 1.7, 4.85
5	3.61, 1.7, 4.85	0.92, 0.63, 0.97	0.4	0.1, −0.3, 0.2	0.7, 0.8, 0.5	3.74, 1.85, 4.71

Table 5
Grey Wolf optimization results - Iteration 1.

Wolf	learning_rate	max_depth	subsample	min_child_weight	n_estimators	max_leaf_nodes
α	0.25	15	0.9	1.5	250	8
β	0.1	10	0.8	1.0	200	6
δ	0.05	8	0.7	0.8	150	4

Fitness Accuracy after iteration-1: $\alpha = 99.0$, $\beta = 98.7$, $\delta = 98.5$

Table 6
Grey Wolf optimization results - Iteration 2.

Wolf	learning_rate	max_depth	subsample	min_child_weight	n_estimators	max_leaf_nodes
α	0.22	14	0.88	1.3	240	7
β	0.12	11	0.82	1.1	210	5
δ	0.07	9	0.75	0.9	160	4.5

Fitness Accuracy after iteration-2: $\alpha = 99.1$, $\beta = 98.8$, $\delta = 98.6$

Table 7
Grey Wolf optimization results - Iteration 3.

Wolf	learning_rate	max_depth	subsample	min_child_weight	n_estimators	max_leaf_nodes
α	0.18	13	0.85	1.2	230	6.5
β	0.14	12	0.8	1.0	220	5.5
δ	0.09	10	0.78	0.95	170	5

Fitness Accuracy after iteration-3: $\alpha = 99.15$, $\beta = 98.9$, $\delta = 98.7$

Table 8
Grey Wolf optimization results - Iteration 4.

Wolf	learning_rate	max_depth	subsample	min_child_weight	n_estimators	max_leaf_nodes
α	**0.15**	**12**	**0.82**	**1.1**	**220**	**6**
β	0.13	12	0.79	1.05	210	5.5
δ	0.11	11	0.76	1.0	190	5.2

Fitness Accuracy after iteration-4: $\alpha = 99.2$, $\beta = 99.0$, $\delta = 98.8$

4.1. Environment

We used the Jupyter Notebook for Python programming on version Python 3.11.5, pandas, NumPy, Keras, and TensorFlow deep neural network frameworks. We used a 64-bit Microsoft Windows 11th Gen. platform for training and experiments. The platform is equipped with Intel(R) Core (TM) i5, 2.40 GHz CPU with 16 GB of memory, and 1TB SSD.

4.2. Result evaluation metrics

A machine learning model performance can be evaluated with several metrics, for this work we have used well-known metrics namely f1-score, accuracy, recall, and precision. A confusion matrix is usually used to calculate these metrics. Given that True Positive (TP) stands for instances that are predicted correctly as attacks whereas the True Negative (TN) are correctly predicted as benign. The False Negative (FN) stands for instances that are predicted incorrectly as benign whereas the False Positive (FP) are incorrectly predicted attacks. The accuracy, precision, recall, and f1-score formulas are derived [8] as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{12}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{13}$$

$$\text{F1-score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}} \tag{14}$$

The Receiver Operating Characteristic Curve (ROC) as well as the area under the ROC curve (AUC) were also used to evaluate the model. Moreover, the False Positive Rate (FPR) and True Positive Rate (TPR) are used to assess the effectiveness of our approaches. The FPR and TPR can be mapped using the ROC curve. The equations for FPR and TPR are denoted as follows:

$$\text{FPR} = \frac{FP}{FP + TN} \tag{15}$$

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

5. Result and discussion

This section is dedicated to show the experimental results of the proposed model and their implications for the overall attack classification task.

Table 9
Optimal hyperparameters of the EDA using random search algorithm.

Parameters	CICIoT2023	CICIDS2017
Batch-size	32	128
L1-Reg	0.001	0.0001
L2-Reg	0.01	0.001
Encoding dim	16	16
Num epochs	100	100
Learning rate	0.0001	0.0001

Table 10
GWO optimized hyperparameter values of XGBoost classifier.

Parameters	CICIoT2023	CICIDS2017
Sub-sample	0.64806	0.75138
Learning rate	0.01597	0.04475
Max-Depth	8	8
Min-Weight-Frac. Leaf	1.35096	0.89540
Max_Leaf nodes	3	5
n-estimators	114	150

5.1. Elastic autoencoder architecture

The EDA used in this work has a fully connected layers. The initial layer consists of 64 units with an activation function ReLU. Then, the following layers decrease the number of units to 32 and 16 and then reconstructs to 32 and 64. The final layer employs the Sigmoid activation function. Adam optimizer is used to compile the final autoencoder with the MSE loss function. The best encoder obtained from the optimized EDA is used to get the most informative features.

5.2. Tuning EDA hyperparameters using a random search

The EDA hyperparameters, such as the batch size, learning rate, activation function, number of epochs, loss function, regularizations, and optimizer can be considered for better encoding. To achieve this goal suitable hyperparameters should be selected and configured. Random search which explores search space with feasible number iterations, for higher encoding performance. In this work, we have used 22 search space values for tuning the EDA. Then the EDA was trained on the training set using the encoders that learn lower-dimensional representation. The decoders are then used to reconstruct the original data using the input from the encoder layer. Optimal hyperparameters of EDA shown in Table 9 are chosen by random search.

5.2.1. Optimized EDA feature encoding

After training the EDA using the optimal hyperparameters chosen by the random search, we have the encoder layer, the last layer of the first three layers. The encoded train and test features are obtained from the encoder part of the trained EDA which is trained using scaled train and test features.

5.2.2. Impacts of reconstruction loss on feature encoding

The reconstruction loss is computed through a comparison between the reconstructed and the original input data. The MSE is employed to evaluate the efficacy of an Autoencoder. It is commonly used to measure the squared differences between the original and reconstructed data. Alternative loss functions, such as MAE, are also utilized in our work.

The reconstruction errors indicate the EDS's ability to accurately reconstruct each feature, which is performed for every feature in the datasets. Fig. 3 illustrates that features with higher reconstruction errors are considered less significant. This ensures EDA has accurately reproduces them. A low reconstruction loss implies the EDA's effectiveness in preserving vital information during encoding and reconstructing the data. On the contrary, a high reconstruction loss indicates that the encoded features do not include informative features. The main factor to evaluate the efficiency of the autoencoder is a reconstructional error. The proposed model has a minimal reconstruction loss of 0.000175 on the CICIoT2023 and 0.0008 on the CICIDS2017 datasets across feature indices.

In Fig. 3(a) and (b) show the histogram distribution of errors across encoded data. The scatter plot in Fig. 4(a) and (b) is generated using the EDA encoded features and class labels. This visual representation effectively shows the extent to which the autoencoder has succeeded in capturing the relevant feature of the data.

5.2.3. Feature encoding performance of EDA

The performance of the EDA can be evaluated by using the reconstructional error. Which is calculated by comparing the encoded and the original input data. Hence, we have used the MSE and MAE to evaluate the EDA performance. A lower value of these metrics indicates a better performance. The results of the metric for the EDA that we have used in our model have achieved an MSE of 0.000175 and MAE 0.006 on CICIoT2023 dataset. On the CICIDS2017 dataset the MSE and MAE values are 0.0008 and 0.00786 respectively. These values indicate that the EDA has a very small reconstruction error.

5.3. The XGBoost classifier

This classifier is a gradient-boosting-based framework that constructs decision trees which are an ensemble of weak prediction models. These trees are merged to build a more powerful prediction model. The objective function f(h), which can be adjusted based on a model's specification, determines the loss that must be minimized during the training phase.

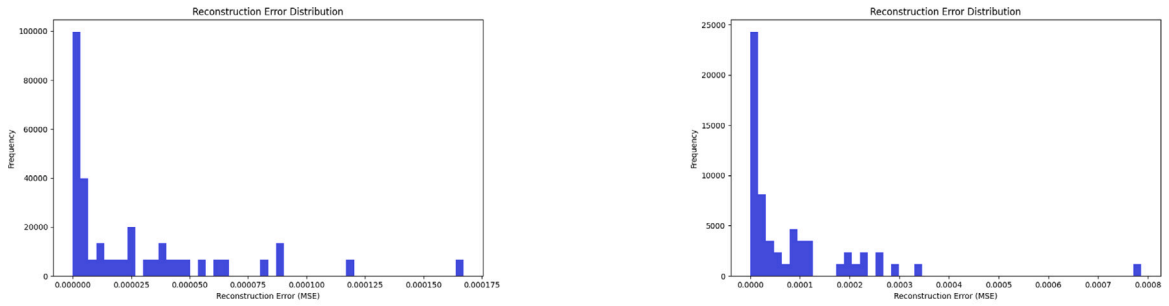
5.3.1. Tuning hyperparameter of XGBoost classifier using GWO

In this optimization framework, the primary objective of the GWO is to maximize the objective function F(H) values by identify optimal hyperparameters (H). GWO explores various hyperparameter combinations iteratively. Then it evaluates their performance utilizing a classification performance function. The algorithm identifies the position and best fitness of the wolves to convergence. This process results in the optimal hyperparameter combination that maximizes the objective function. To obtain the most suitable hyperparameters for optimizing the performance of the classifier we employed GWO.

We have used 50 iterations and a convergence threshold set to 1e−6 in our experiment, with a maximum consecutive iteration of 5 used to determine unchanged fitness. The combination of possible hyperparameters generated and the classifier is used to evaluate the fitness of these hyperparameters using a metric. The optimal or best hyperparameters, the output of the final iteration, were taken when the termination criteria were satisfied. These hyperparameters are used to train and evaluate the final optimized XGBoost classifier. Table 10 show the summary of optimized hyperparameters values. This values are chosen by the optimizer after running 50 iterations using a population size of 20 wolves.

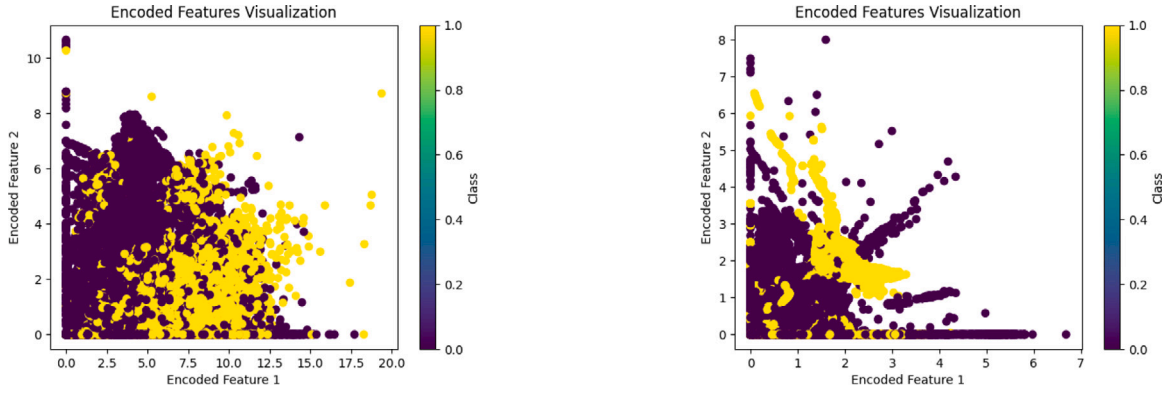
5.3.2. The performance of proposed GWO-optimized XGBoost model

The machine learning models' performance is evaluated using various metrics, including, recall, F1-score, precision, accuracy, and support. These metrics offer a comprehensive evaluation of a model's effectiveness in classification tasks. The output of total correct predictions divided by the total predictions made on the dataset used is the accuracy. ROC curve graphically represents the trade-off between TPR and FPR. These are used to show sensitivity and specificity respectively, across different classification thresholds. AUC represents the integral of the ROC, between 0 and 1, a curve that summarizes the model's performance across all possible thresholds. Ranging from 0 to 1, a higher classification performance is indicated by a ROC-AUC value approximately close to 1. Log loss, on the other hand, computes the



(a) CICIoT2023 (b) CICIDS2017

Fig. 3. EDA reconstruction error rates visualization in MSE.



(a) Encoded Features of CICIoT2023 (b) Encoded Features of CICIDS2017

Fig. 4. Encoded features using optimized EDA.

Table 11
The performance of proposed GWO-Optimized model.

Metrics	CICIoT2023	CICIDS2017
Accuracy	0.9938	0.9987
Recall	0.9999	0.9999
F1_score	0.9938	0.9987
Precision	0.9878	0.9975
ROC-AUC	0.9938	0.9987
Log-Loss	0.00125	0.00116

dissimilarity between actual and predicted labels. Table 11 shows the performance of our model in terms of the above discussed metrics. The accuracy of the GWO-optimized XGBoost classifier is 99.38% and 99.87% accuracy with a higher recall and a negligible log loss value on CICIoT2023 and CICIDS2017 datasets respectively.

5.3.3. Evaluation of the CICIoT2023 dataset

Table 12 depicts that our model achieved very high TP and TN values, and a negligible value of misclassified samples with a rate of 0.006% on the CICIoT2023 dataset. This indicates the GWO-optimized XGBoost classifier performance is high. In the attack analysis based on the CICIoT2023 dataset, 98.76% of the occurrences were correctly predicted as true positives on the first row. 1.24% of occurrences were predicted as false positives erroneously. However, from the second row, we can observe that 99.99% of attacks were accurately classified as true negatives or attacks. Very small incidents of 0.01% were misclassified as true negatives.

5.3.4. Evaluation of the CICIDS2017 dataset

Further, we have conducted experiments utilizing the CICIDS2017 evaluation and benchmark dataset. The main objective is to assess the robustness and performance of the proposed model through varied

Table 12
Confusion matrix for GWO-Optimized XGBoost on CICIoT2023.

		Predicted	
		Benign	Malicious
Actual	Benign	69,061 (98.76%)	865 (1.24%)
	Malicious	6 (0.01%)	69,921 (99.99%)

Table 13
Confusion matrix for GWO-Optimized XGBoost on CICIDS2017.

		Predicted	
		Benign	Malicious
Actual	Benign	423,353 (99.751%)	1,057 (0.249%)
	Malicious	17 (0.004%)	424,392 (99.996%)

datasets. The total number of instances for test set used for this evaluation are indicated as 848 819. Table 13 shows the confusion matrix of CICIDS2017 datasets. The proposed model achieved 99.87% detection accuracy, loss of 0.0013, recall 99.99%, F1-score 99.87%, precision of 99.75% and FPR 0.0025%. The evaluation result confirmed our model's robustness and the applicability of a model in real-world environments for attack mitigation.

5.4. Comparison with other state-of-the-art works

We have conducted a comparative analysis, with the state-of-the-art, to further evaluate the performance of the proposed model. We have used accuracy, recall, precision, F1-score, FPR and test time metrics for the comparison. All models used in the comparison employed the same CICIoT2023 or CICIDS2017 datasets. This guarantees to demonstrate a fair and significant comparison of related works. Moreover, it ensures fairness and avoids a biased comparison which is expected

Table 14
A comparison of the proposed model to the state-of-the-art on CICIoT2023 datasets for binary classification.

Work	Year	Model	Dataset	Accuracy (%)	Recall (%)	Precision (%)	F1-score (%)	FPR (%)	Test time (s)
[38]	2023	FL-DNN	CICIoT2023	99.00	99.00	99.00	99.00	0.01	–
[39]	2024	CNN-LSTM	CICIoT2023	98.43	97.46	98.43	97.15	9.17	2.08
[40]	2023	AE	CICIoT2023	98.76	99.00	99.00	99.00	–	1.15
[41]	2024	SELDP4-FS	CICIoT2023	99.32	98.59	**99.77**	99.23	0.49	–
Ours	2024	**Proposed**	CICIoT2023	**99.38**	**99.99**	98.78	**99.38**	**0.006**	**0.89**

Table 15
A comparison of the proposed model to the state-of-the-art on CICIDS2017 dataset for binary classification.

Work	Year	Model	Dataset	Accuracy (%)	Recall (%)	Precision (%)	F1-score (%)	FPR (%)	Test time (s)
[38]	2023	CNN-LSTM	CICIDS2017	97.46	97.15	98.85	97.17	2.08	2.08
[15]	2023	DIS-IoT	CICIDS2017	98.70	97.60	95.90	96.70	0.010	–
[17]	2024	AE+PCA+LSTM	CICIDS2017	93.78	95.65	96.57	96.11	13.88	2.15
[42]	2023	DNN	CICIDS2017	99.80	99.94	99.85	99.89	1.2	–
Ours	2024	**Proposed**	CICIDS2017	**99.87**	**99.99**	**99.75**	**99.87**	**0.0025**	**1.02**

when different datasets are used to develop a model. This comparison with state-of-the-art techniques includes [15,17] explained in Section 2. The result of the comparison show that the propose model outperforms the other approaches in terms of matrices used for this comparison. Tables 14 and 15, presents the comparative analysis.

5.5. Discussion

The experiments were conducted on the pre-processed CICIoT2023 and CIC-IDS-2017 datasets. To ensure the reliability of our method, we conducted a series of comprehensive experiments and evaluated the performance our model in multiple runs. The results, presented in Table 11, demonstrate that our method has the potential to deliver improved or comparable performance compared to state-of-the-art approaches. The employed optimization, dimensionality reduction, data balancing and scaling enables our model to quickly identify the most informative feature from the datasets. As a result, significantly reduced size of encoded features are extracted and utilized for attack detection. Thus, the model resulted in higher performance in a reduce training and test time.

It can also be deduced, that the optimized XGBoost capability and the regularization (L1 and L2) used resulted in an improved handling of overfitting and noise, resulted in an improved performance. Additionally, the experimental results conducted on both high dimensional datasets shows that our model is robust.

It can be deduced that the model's training, test and computational complexity are reduced, due to the much smaller training and test sets encoded by the EDA, fast classification capability and diminished spatial and grid search complexity of GWO tuned XGBoost. Hence, reduces the test time as we compared to the existing techniques in Tables 14 and 15. The model achieves higher detection accuracy with less computational cost. Thus, a lightweight model that utilize less memory to execute is realized.

The proposed model is capable of efficiently handling both linear and non-linear data changes while being scalable and adaptive. The optimization techniques reduce the grid search and spatial complexity of XGBoost, thereby enhancing its performance. The optimized feature selection allows the model to adapt to changes in IoT data in selecting informative features. This indicates its ability to handle evolving attack vectors with minimal false positives, thus making it a robust intrusion detection system. Moreover, the use of real-world datasets further improves its relevance and applicability, enabling seamless integration into existing IoT security infrastructures.

6. Conclusion

This study presented a novel autoencoder-based GWO-optimized attack detection model for intrusion detection in IoT infrastructures. With the increasing number of intelligent attack, it is a necessity to develop a tool to mitigate such attacks. Data preparation and balancing techniques, feature normalization using a scalar function, and feature selection using optimized EDA are employed. Our model was trained using encoded features obtained from optimized EDA. Our model shows a promising performance in detecting attack. We also exploited GWO for hyperparameter tuning of the XGBoost binary classifier, which resulted in classifying attacks with a higher accuracy. The proposed approach was evaluated on CICIoT2023 and CICIDS2017 datasets, and achieved a higher detection accuracy.

The proposed model achieved promising and competitive performance in comparison with state-of-the-art detection techniques with a negligible misclassification and less computational resources. Additionally, the EDA can adopt dynamic data changes and select relevant features for on-demand IoT data changes, adapting to new attack features. The optimization, dimensionality reduction, the sparsity regularizations used results in a lightweight and robust model. We plan on integrating a more dynamic feature selection technique using DL, creating a more robust real-time detection model and deploy it to a real-world IoT network as a future contribution.

CRediT authorship contribution statement

Dawit Dejene Bikila: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Jan Čapek:** Writing – review & editing, Supervision, Resources, Project administration, Formal analysis, Conceptualization.

Declaration of competing interest

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

Acknowledgement

This paper was supported by Grant No. *SGS*_2024_017, Institute of System Engineering and Informatics, Faculty of Economics and Administration, University of Pardubice.

Data availability

Data sharing does not apply to this article as no datasets were generated during the current study. The link for the code is: https://github.com/Dawitdjn/ML-based-Attack-Detection.

References

[1] G. Bovenzi, G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, A. Pescapé, Network anomaly detection methods in IoT environments via deep learning: A fair comparison of performance and robustness, Comput. Secur. 128 (2023) 103167.

[2] M.N. Bhuiyan, M.M. Rahman, M.M. Billah, D. Saha, Internet of things (IoT): A review of its enabling technologies in healthcare applications, standards protocols, security, and market opportunities, IEEE Internet Things J. 8 (13) (2021) 10474–10498.

[3] M. Zeeshan, Q. Riaz, M.A. Bilal, M.K. Shahzad, H. Jabeen, S.A. Haider, A. Rahim, Protocol-based deep intrusion detection for dos and ddos attacks using unsw-nb15 and bot-iot data-sets, IEEE Access 10 (2021) 2269–2283.

[4] M.R. Babu, K. Veena, A survey on attack detection methods for iot using machine learning and deep learning, in: 2021 3rd International Conference on Signal Processing and Communication, ICPSC, IEEE, 2021, pp. 625–630.

[5] V. Mothukuri, P. Khare, R.M. Parizi, S. Pouriyeh, A. Dehghantanha, G. Srivastava, Federated-learning-based anomaly detection for IoT security attacks, IEEE Internet Things J. 9 (4) (2021) 2545–2554.

[6] O. Habibi, M. Chemmakha, M. Lazaar, Imbalanced tabular data modelization using CTGAN and machine learning to improve IoT botnet attacks detection, Eng. Appl. Artif. Intell. 118 (2023) 105669.

[7] C. Zhang, G. Wang, S. Wang, D. Zhan, M. Yin, Cross-domain network attack detection enabled by heterogeneous transfer learning, Comput. Netw. 227 (2023) 109692.

[8] M. Mulyanto, J.-S. Leu, M. Faisal, W. Yunanto, Weight embedding autoencoder as feature representation learning in an intrusion detection systems, Comput. Electr. Eng. 111 (2023) 108949.

[9] T. Yi, X. Chen, Y. Zhu, W. Ge, Z. Han, Review on the application of deep learning in network attack detection, J. Netw. Comput. Appl. 212 (2023) 103580.

[10] F. Daneshfar, S. Soleymanbaigi, A. Nafisi, P. Yamini, Elastic deep autoencoder for text embedding clustering by an improved graph regularization, Expert Syst. Appl. 238 (2024) 121780.

[11] W. Ma, L. Ma, K. Li, J. Guo, Few-shot IoT attack detection based on SSDSAE and adaptive loss weighted meta residual network, Inf. Fusion 98 (2023) 101853.

[12] T. Zhang, W. Chen, Y. Liu, L. Wu, An intrusion detection method based on stacked sparse autoencoder and improved gaussian mixture model, Comput. Secur. 128 (2023) 103144.

[13] F.K. Örs, A. Levi, Data driven intrusion detection for 6LoWPAN based IoT systems, Ad Hoc Netw. 143 (2023) 103120.

[14] Y. Li, Y. Lei, P. Wang, M. Jiang, Y. Liu, Embedded stacked group sparse autoencoder ensemble with L1 regularization and manifold reduction, Appl. Soft Comput. 101 (2021) 107003.

[15] R. Lazzarini, H. Tianfield, V. Charissis, A stacking ensemble of deep learning models for IoT intrusion detection, Knowl.-Based Syst. 279 (2023) 110941.

[16] S. Aktar, A.Y. Nur, Towards DDoS attack detection using deep learning approach, Comput. Secur. 129 (2023) 103251.

[17] A. Thakkar, N. Kikani, R. Geddam, Fusion of linear and non-linear dimensionality reduction techniques for feature reduction in LSTM-based intrusion detection system, Appl. Soft Comput. 154 (2024) 111378.

[18] A. Alzaqebah, I. Aljarah, O. Al-Kadi, R. Damaševičius, A modified grey wolf optimization algorithm for an intrusion detection system, Math. 10 (6) (2022) 999.

[19] Y. Labiod, A. Amara Korba, N. Ghoualmi, Fog computing-based intrusion detection architecture to protect iot networks, Wirel. Pers. Commun. 125 (1) (2022) 231–259.

[20] H. Akrami, A.A. Joshi, J. Li, S. Aydöre, R.M. Leahy, A robust variational autoencoder using beta divergence, Knowl.-Based Syst. 238 (2022) 107886.

[21] A. Abusitta, G.H. de Carvalho, O.A. Wahab, T. Halabi, B.C. Fung, S. Al Mamoori, Deep learning-enabled anomaly detection for IoT systems, Internet Things 21 (2023) 100656.

[22] J. Hu, K. Kaur, H. Lin, X. Wang, M.M. Hassan, I. Razzak, M. Hammoudeh, Intelligent anomaly detection of trajectories for IoT empowered maritime transportation systems, IEEE Trans. Intell. Transp. Syst. 24 (2) (2022) 2382–2391.

[23] T. Thulasi, K. Sivamohan, LSO-CSL: Light spectrum optimizer-based convolutional stacked long short term memory for attack detection in IoT-based healthcare applications, Expert Syst. Appl. 232 (2023) 120772.

[24] Z. Sun, G. An, Y. Yang, Y. Liu, Optimized machine learning enabled intrusion detection 2 system for internet of medical things, Frankl. Open 6 (2024) 100056.

[25] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, A. Anwar, TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems, Ieee Access 8 (2020) 165130–165150.

[26] A. Thakkar, R. Lohiya, Attack classification of imbalanced intrusion data for IoT network using ensemble learning-based deep neural network, IEEE Internet Things J. (2023).

[27] G. Mohiuddin, Z. Lin, J. Zheng, J. Wu, W. Li, Y. Fang, S. Wang, J. Chen, X. Zeng, Intrusion detection using hybridized meta-heuristic techniques with Weighted XGBoost classifier, Expert Syst. Appl. 232 (2023) 120596.

[28] S. Mirjalili, S.M. Mirjalili, A. Lewis, Grey wolf optimizer, Adv. Eng. Softw. 69 (2014) 46–61.

[29] A. Lev, XGBoost versus random forest performance and benefits, 2022, URL https://www.qwak.com/post/xgboost-versus-random-forest. (Accessed 21 September 2023).

[30] S. Fraihat, S. Makhadmeh, M. Awad, M.A. Al-Betar, A. Al-Redhaei, Intrusion detection system for large-scale IoT NetFlow networks using machine learning with modified Arithmetic Optimization Algorithm, Internet Things 22 (2023) 100819.

[31] E.C.P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, A.A. Ghorbani, CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment, Sensors 23 (13) (2023) 5941.

[32] I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, et al., Toward generating a new intrusion detection dataset and intrusion traffic characterization, ICISSp 1 (2018) 108–116.

[33] A. Fernández, S. Garcia, F. Herrera, N.V. Chawla, SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary, J. Artif. Intell. Res. 61 (2018) 863–905.

[34] H. Bandyopadhyay, Autoencoders in deep learning: Tutorial & use cases, 2023, URL https://www.v7labs.com/blog/autoencoders-guide. (Accessed 14 November 2023).

[35] Y. Huang, Y. Wang, P. Wang, Y. Lai, An XGBOOST predictive model of void ratio in sandy soils with shear-wave velocity as major input, Transp. Geotech. 42 (2023) 101100.

[36] Sayantini, Autoencoders tutorial: A beginner's guide to autoencoders, 2023, URL https://www.edureka.co/blog/autoencoders-tutorial/.

[37] X. Kan, Y. Fan, J. Zheng, C.-h. Chi, W. Song, A. Kudreyko, Data adjusting strategy and optimized XGBoost algorithm for novel insider threat detection model, J. Franklin Inst. 360 (16) (2023) 11414–11443.

[38] S. Abbas, A. Al Hejaili, G.A. Sampedro, M. Abisado, A. Almadhor, T. Shahzad, K. Ouahada, A novel federated edge learning approach for detecting cyberattacks in IoT infrastructures, IEEE Access (2023).

[39] K.R. Narayan, S. Mookherji, V. Odelu, R. Prasath, A.C. Turlapaty, A.K. Das, Iids: Design of intelligent intrusion detection system for internet-of-things applications, in: 2023 IEEE 7th Conference on Information and Communication Technology, CICT, IEEE, 2023, pp. 1–6.

[40] H.Q. Gheni, W.L. Al-Yaseen, Two-step data clustering for improved intrusion detection system using CICIoT2023 dataset, e-Prime-Adv. Electr. Eng. Electron. Energy 9 (2024) 100673.

[41] A. El-Sayed, W. Said, A. Tolba, Y. Alginahi, A.A. Toony, Mp-guard: A novel multi-pronged intrusion detection and mitigation framework for scalable SD-IoT networks using cooperative monitoring, ensemble learning, and new P4-extracted feature set, Comput. Electr. Eng. 118 (2024) 109484.

[42] A. Thakkar, R. Lohiya, Fusion of statistical importance for feature selection in deep neural network-based intrusion detection system, Inf. Fusion 90 (2023) 353–363.

Dawit Dejene Bikila received his BSc. and MSc. in computer science from the Institute of Technology, Hawassa University, Ethiopia in 2010 and 2018 respectively. From 2018 to 2022 he was a lecturer in the Department of Computer Science at the Institute of Technology, Hawassa University, Ethiopia. He is pursuing a doctoral degree at the University of Pardubice, Czech Republic. His research interests include network security, IoT, data-driven cybersecurity, machine learning, and Artificial Intelligence Algorithms and Applications.

Jan Čapek served as a professor of Managerial Informatics at the Institute of System Engineering and Informatics of the Faculty of Economics and Administration (FEA), University of Pardubice (UPa), Czech Republic. He was born in 1946, graduated from VŠDS Žilina (Transport and Communication University) in Slovakia year 1970, PGS ČVUT Praha 1976, candidate of science (Ph.D.) ČVUT (Czech Technical University) Prague 1981, habilitate assoc. Prof. VŠDS Žilina 1990, professor of managerial informatics VŠB TU Ostrava 2002. From 1990–1993 Head of the Department of Reliability and Diagnostics VŠDS Žilina. From 1993–1995 Head of the Department of Information Systems (University of Pardubice) UPa, 1995–1997 vice-dean FEA - UPa, 1997–2000. Vice-rector UPa, in 2002–2007, dean FEA - UPa in 2008–2011, and vice-dean for research from 2016–2020. He is currently a professor at the Institute of System Engineering and Informatics at the University of Pardubice. His research interests are cybersecurity and information protection.