International Journal of Electrical and Computer Engineering (IJECE)
Vol. 15, No. 3, June 2025, pp. 3149~3161
ISSN: 2088-8708, DOI: 10.11591/ijece.v15i3.pp3149-3161

Exploring the effectiveness of hybrid artificial bee PyCaret classifier in delay tolerant network against intrusions

Rajashri Chaudhari, Manoj Deshpande
Department of Computer Engineering, A. C. Patil College of Engineering, Kharghar, Maharashtra, India

Article Info

Article history:

Received Jun 26, 2024
Revised Dec 20, 2024
Accepted Jan 16, 2025

Keywords:

Artificial bee colony
Delay tolerant network
Distributed denial-of-service attack
Flooding attack
Optimization
Security

ABSTRACT

In challenging environments with intermittent connectivity and the absence of end-to-end paths, delay tolerant networks (DTNs) require robust security measures to safeguard against potential threats. This study addresses these issues by implementing an intrusion detection system (IDS) enhanced with machine learning techniques. Common threats such as distributed denial-of-service (DDoS) and flood attacks are tackled using datasets like network intrusion detection (NID) and flood attack datasets. Multiple machines learning methods, including k-nearest neighbors (K-NN), decision trees (DT), logistic regression (LR), and others, are utilized to improve detection accuracy. A PyCaret-based approach is developed to increase efficiency while preserving attack detection accuracy in DTNs. Comparative research demonstrates that PyCaret outperforms Scikit-learn models, and the artificial bee PyCaret classifier (ABPC) optimizes hyperparameters to improve model performance. NS2 simulation shows the system's ability to thwart attacks, offering useful insights into DTN security and improving communication reliability in various situations.

Corresponding Author:

Rajashri Chaudhari
Department of Computer Engineering, A.C. Patil College of Engineering
Kharghar, Maharashtra, India
Email: c.rajashri2021@gmail.com

1. INTRODUCTION

Delay tolerant networking (DTN) contrasts with conventional network architectures by accommodating intermittent connectivity, a characteristic often absent in modern architectures [1]. DTN is designed to operate effectively under challenging or sporadic network conditions, categorized by higher latency, bandwidth constraints, increased error prospect, path instability, or variable node longevity [2]. Initially proposed as an abstraction of message switching, DTN architecture revolves around the concept of bundles, wherein messages are aggregated and transmitted. These bundles are managed by bundle routers or DTN gateways, which play a pivotal role in handling the instability and frequent disconnections inherent in DTN environments. DTNs address the challenges posed by intermittent connections, long or variable delays, asymmetric data rates, and high error rates through the utilization of store-and-carry message switching. DTN follows the store and forward technique in DTN [3]. Figure 1 shows characteristics of delay tolerant network. DTNs transfer messages between nodes without requiring constant connectivity, making them suited for demanding environments such as emergency services and disaster management [4]. However, their decentralized structure raises security concerns, such as packet dropping and flooding assaults. Improving DTN security is critical for providing reliable communication in such settings [5].

DTNs, despite their decentralized form, pose security risks from compromised nodes. Attacks such as packet dropping, flooding, and others damage routing systems, compromising message integrity and

confidentiality. Addressing these vulnerabilities before deployment is critical, as it necessitates adequate security methods to construct a reliable infrastructure. Challenges such as inconsistent connectivity, limited bandwidth, and node mobility compound security concerns, necessitating novel solutions. An intrusion detection system (IDS) augmented with machine learning is presented to identify flood and distributed denial-of-service (DDoS) attacks [6]. Various routing algorithms including the store-and-forward technique improve resilience in uncertain circumstances. Implementations such as interplanetary overlay network-delay-tolerant networking (ION-DTN) highlight research efforts to address security in DTNs, emphasizing the necessity of integrative techniques and machine learning for dependable communication [2].

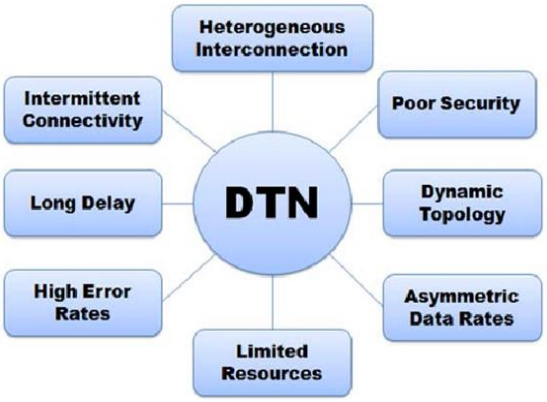

Figure 1. Characteristics of delay tolerant network [2]

The objectives include creating a system to distinguish between legitimate and malicious packets in network traffic, devising an algorithm to thwart DOS attacks in DTN, suggesting a technique for enhancing network performance, and assessing the model's efficacy using the NS2 simulator [7]. Numerous studies in DTNs have explored using machine learning (ML) techniques to improve network security, with a focus on intrusion detection [8]. Initially, network intrusion detection systems (NIDS) relied on known attack signatures, limiting their ability to detect new or modified attacks and leaving networks vulnerable [9]. ML-based IDS methods have since emerged, analyzing overall behavioral patterns rather than specific attack signatures, and offering increased robustness in intrusion detection [10]. These ML approaches have been validated in various studies, demonstrating their potential to enhance security and resilience within DTNs. DTN studies have looked at a variety of security concerns, such as flood attacks, Black Hole attacks, fake packet attacks, packet drops, colluding attacks, faulty nodes, and DDOS attacks [11], [12]. Chatterjee *et al.* [1] explored the detection of various routing attacks in DTNs, revealing hurdles to enhancing performance while maintaining reliability and security. Their research focuses on the impact of these assaults on network performance, emphasizing nodes' vulnerability to malicious actions. Flood attacks, for example, include hostile nodes flooding the network with packets, reducing DTN resources and resulting in lower packet delivery ratios (PDR) and higher packet loss ratios (PLR) [13].

As demonstrated by numerous studies across various network applications [14], [15], they address security concerns such as congestion traffic management and intrusion detection, enabling proactive approaches to classify and mitigate security threats [16], [17]. ML integration in network security extends to meeting the evolving security needs of modern network environments, playing a vital role in fortifying network defenses and ensuring robust security measures against potential threats [18], [19]. Recent research has looked into machine learning methods to improve network security [20]. One study used a machine learning-based strategy to classify hypertext transfer protocol (HTTP) traffic as benign or dangerous to detect malware, obtaining an amazing 90% accuracy with the random forest (RF) algorithm [21]. Another study proposed the buffer management for RPRTD (BMRTD) algorithm to optimize buffer management in DTNs, improving message delivery efficiency by employing strategies such as drop oldest (DOA), drop least recently received (DLR), and drop largest (DLA) to effectively manage message queues. Studies have explored various methods to enhance attack detection in network security [22]. One study focuses on detecting behavioral botnet attacks, employing a smart system that combines the random forest classifier with principal component analysis (PCA) to achieve robust botnet detection [23]. Another study introduces Shield recurrent neural network (RNN), a framework designed to detect DDoS attacks within IoT networks, incorporating 12 different classifiers to enhance accuracy and reliability.

The study explores machine learning-driven self-healing mechanisms in cyber-physical systems, highlighting their potential to improve security and prevent system failures [24]. It identifies three critical components for self-healing: anomaly detection, fault warning, and fault auto-remediation, emphasizing the importance of integrating these elements for practical use. The study introduces the efficacy artificial bee colony optimization-based Gaussian AOMDV (EABCO-GAOMDV) routing protocol, which addresses routing problems in stochastic vehicular ad hoc networks (SVANETs) [25]. The protocol attempts to improve route discovery and rerouting efficiency by integrating artificial bee colony optimization (EABCO) and Gaussian AOMDV algorithms. Extensive simulations assess EABCO-GAOMDV's performance, revealing better route stability, packet delivery ratio, and end-to-end delay. The protocol is adaptable to unanticipated environments, improving traffic rerouting efficiency and network resilience.

Cloud computing and the internet of things (IoT) are driving future technologies, notably smart city development [26]. Cloud services successfully handle remote access, resulting in increased usage by organizations for resource optimization. Cloud service selection involves optimization based on customer objectives and service quality standards. Combining genetic algorithms (GAs) with ant colony optimization (ACO) improves cloud computing performance. The ACO+GA approach outperforms existing optimization methods, including energy- and reliability-aware multiobjective optimization and hybrid cuckoo particle swarm with artificial bee colony optimization.

2. METHOD

The paper describes a machine learning model for detecting various assaults in DTNs. It compares the performance of several ML algorithms to improve attack detection, training the model on a variety of datasets, including DDoS and flood attacks. Initially, standard ML models built with Scikit-learn use methods such as random forest (RF), XGBoost, CatBoost, logistic regression (LR), decision tree (DT), extra trees, and light GBM. Their performance is evaluated using many factors. The model is then enhanced using PyCaret to boost performance and reduce execution time. Accuracy, F1-Score, AUC, precision, recall, Matthews correlation coefficient (MCC), and Kappa are some of the evaluation metrics, as is training time with different classifiers. By comparing Scikit-learn and PyCaret models, the study seeks to discover the best-performing model for intrusion detection in DTNs. Figure 2 displays the ML model architecture, which includes data collection, preprocessing, and model training on specific attributes. The performance of the ML algorithm in classifying attacks is measured using the accuracy, precision, and recall metrics.

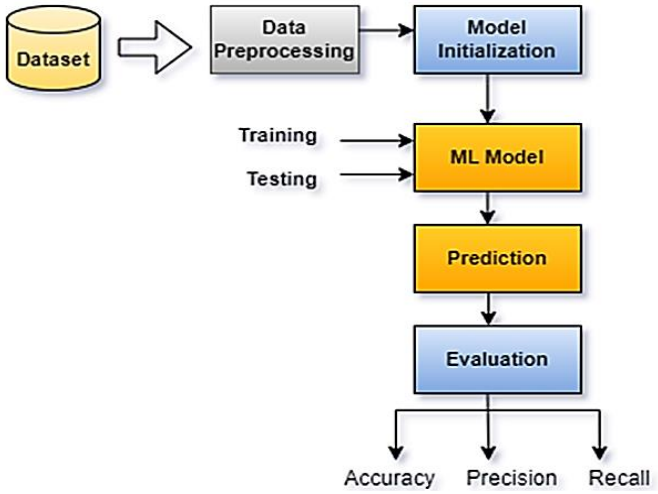

Figure 2. Training and deployment of ML model using scikit-learn

2.1. Data collection

The DTN-specific ML-based attack detection model makes use of flood attack datasets obtained from publicly available DDoS and flood attack records. These datasets are essential for training and testing the model since they offer information on network traffic patterns, communication habits, and flood attack indicators. With a dataset of 25,192 samples, the model is effectively trained. To enable strict evaluation, the dataset is divided into two subsets: 17,634 for training and 7,558 for testing. This phase evaluates the model's performance on unseen data, providing insights into its generalization capabilities.

2.2. Data pre-processing

After collecting and partitioning the dataset, pre-processing procedures are used to improve data quality. Data cleaning detects and eliminates duplicate records, preserving integrity and decreasing redundancy. Missing data is handled by removal or attribution, and outliers are managed to avoid undue influence on results. PCA decreases dimensionality by preserving useful features and removing irrelevant ones, so overcoming the curse of complexity and improving generalization. Normalization and scaling normalize feature values, preventing larger features from dominating the learning process while also enhancing algorithm stability during training. This technique improves ML model efficiency, stability, and accuracy. Model initialization configures algorithms with specific hyperparameters that are fine-tuned for maximum performance. The model is iteratively trained and tested until it achieves sufficient efficiency, ensuring that attacks are detected and reduced in DTN.

2.3. Training

During the training phase, each data point from the given dataset is pre-processed and features are extracted. This stage involves training features for each data point in the dataset and assigning classes to the trained features. This algorithm has two classes: normal and DDoS or flood attack.

2.4. Testing

During testing, the test samples are passed into a machine learning classifier, which uses training features to classify the test instance into the provided class j. If the classifier accurately classifies a particular class, the process will yield a superior outcome. If the machine learning classifier's decision isn't final, the choice is decided using a cost minimization procedure.

2.5. Dataset

The dataset is an important part of the ML model that should include a variety of intrusion data. here used the NID [19] dataset available on Kaggle source to train and test models. The dataset details are shown in Table 1. The NID dataset counterfeit in a military network environment includes extensive diversity of data including intrusion samples. The US Air Force LAN was blasted with multiple attacks using raw TCP/IP dump data. For each TCP/IP connection, data flows from a source IP address to a target IP address for some time duration. The dataset contains normal and attack data with 41 features including 3 qualitative and 38 quantitative features.

Table 1. Dataset details

Dataset	NID
Size	5.29 MB
Features extracted	41
Features selected	39
Class	– Normal – Anomaly
Test dataset	2.42 MB (40%)
Train dataset	2.87 MB (60%)

2.6. ML model and classification

The machine learning-based attack detection model is built on Python, which includes libraries such as Scikit-learn and PyCaret. Scikit-learn handles carefully data before treatment and method selection, allowing for detailed comparisons of machine learning algorithms. PyCaret improves productivity by automating operations like model training, hyperparameter tuning, and performance evaluation. While both libraries serve machine learning goals, Scikit-learn provides greater flexibility and control over individual components of the ML pipeline, whereas PyCaret accelerates workflows by automating common activities like as data preprocessing and model selection. The proposed model's flow utilizing PyCaret is depicted in the Figure 3. The process of building a PyCaret model for DTN attack detection on a flood attack dataset involves several key steps as mentioned below. The process begins with setting up the PyCaret environment and a simple interface for initializing the environment and specifying the target variable, streamlining the setup process.

- Data preprocessing: PyCaret automates data preprocessing tasks like handling missing values and feature scaling for flood attack dataset in DTNs.
- Compare models: Multiple machine learning models, like LightGBM, XGBoost, CatBoost, and ExtraTrees, are cross-validated to detect DTN attacks with default hyperparameters. Performance measures such as precision, accuracy, recall, F1-Score, AUC, Kappa, MCC, and evaluation time are calculated for each model.

- Best performing model: A top model is chosen after comparing results, trained on full dataset to test capabilities.
- Artificial bee PyCaret classifier (ABPC): Analyze solution space for optimization issues, find hyperparameters, optimize PyCaret model, protects DTNs with IDS.
- Optimal solution and Exploitation search: ABPC sees food sources as problem-solving with nectar indicating value.
- Results: The results from the validation phase are analyzed to determine the best-performing model based on predefined evaluation metrics. The best-performing model is selected for further refinement and deployment.
- Prediction: The best-performing model is kept for later usage, making it easily available for predictions on new data. This persistence enables rapid detection of flood attacks in real-time DTN scenarios, allowing for rapid reaction and mitigation.

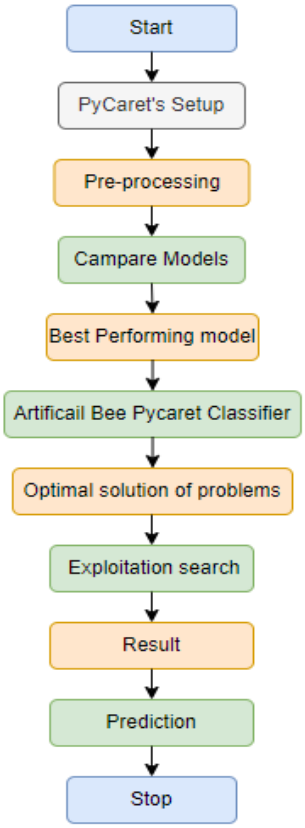

Figure 3. Process of training PyCaret model

Overall, the PyCaret model-building process streamlines the workflow for DTN attack detection, automating many tedious tasks and providing a user-friendly interface for model training, evaluation, and deployment. The ABCO method is a nature-inspired optimization strategy based on honeybee behavior. It is used in swarm intelligence algorithms to find the best solutions to complex problems. ABCO is important in machine learning models for setting hyperparameters and improving security in DTNs through IDSs. IDSs monitor network traffic for suspicious activity in DTNs, which face connectivity challenges. ABCO adjusts IDS hyperparameters to enhance detection accuracy and reduce false positives, improving DTNs' security. The algorithm uses food sources as solutions, with nectar content representing the desired outcome. Employed and observer bees focus on exploitation while scout bees explore new solutions. The algorithm begins with a population of food sources.

The goal of the employed bee mechanism is to exploit the existing food source. To actualize this idea, they create a new solution near the current food supply. For each solution i, one element j (real number) is randomly chosen and coupled with the equivalent element from another randomly selected food source k. Take note that they have $k \neq i$. This is accomplished using the following formula.

$$v_{ij=x_{ij}} + \theta\left(x_{ij-}x_{kj}\right) \quad (1)$$

where x_{ij} is *j*th real number of food source i, $\theta$is a randomly generated number between [-1,1], and v_{ij} is the new real number of *j*th element of food source i. The hired bee accepts a new food source if its nectar amount exceeds that of the present one. Otherwise, it continues with the previous one. After looking for food sources, hired bees to share their discoveries with observer bees. Onlooker bees then assess the nectar information from all employed bees before selecting a food source based on its nectar content. Probability functions are used to calculate the possibility of each food source being selected by onlooker bees.

$$\mathcal{P}_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \quad (2)$$

where fit_i is the nectar amount (objective value) of the food source (solution *i*). Using each onlooker select a new nearby food source and its nectar amount. If a new food source provides more nectar than the previous one, the bee will store the new position and discard the old one. When employed and observer bees run out of food, they become scout beesScout bees replace depleted food sources randomly. The procedure entails preserving a specific number of recent food sources produced by employed and onlooker bees. When replacing an exhausted food source, one of the conserved food sources is chosen using a roulette wheel based on their probability.

$$\frac{(f_k - f_{cs})^{-1}}{\sum_{i=1}^{back}\left(f_i - f_{cs_i}\right)^{-1}} \quad (3)$$

The amount of nectar in the saved and current food sources is represented by f_i and f_{cs}, respectively. The roulette wheel divides the interval [0, 1] into back subintervals.

$$[0, (f_1 - f_{cs})^{-1}], \ldots, \left[\sum_{i=1}^{k-1}(f_i - f_{cs_i})^{-1}, \sum_{i=1}^{k}(f_i - f_{cs_i})^{-1}\right], \ldots, \left[\sum_{i=1}^{back-1}(f_i - f_{cs_i})^{-1}, 1\right] \quad (4)$$

The subintervals correspond to saved food sources, with the *i*-th subinterval denoting the most recent saved food source. The next step is to generate a random real number between 0 and 1. The food source is then chosen depending on the subinterval to which this number belongs.

Pseudo code for PyCaret model:

a. Load the flood attack datasets, which contain records of DDoS and flood attacks.
b. Preprocess the data to remove missing values, scale features, and encode categorical variables, resulting in well-formatted datasets appropriate for model training.
c. To analyze the model, many machine learning techniques are used, including LightGBM, XGBoost, CatBoost, ExtraTrees, and others.
d. Prepare the data for training validation.
e. Use metrics like accuracy, AUC, recall, precision, F1-Score, kappa, and MCC to evaluate the model's performance.
f. The validation results are examined to determine the best-performing model based on established evaluation criteria.
g. Optimizing results using the ABC algorithm.
h. Compare the results to existing approaches and demonstrate the effectiveness of the new approach.
i. Use the trained model to make predictions.

Algorithm 1. For hybrid artificial bee PyCaret classifier

```
import numpy as np
def measom(variables_values=[0, 0]):
x1, x2 = variables_values
func_value = -np.cos(x1) * np.cos(x2) * np.exp(-((x1 - np.pi) * 2 + (x2 - np.pi) * 2))
return func_value
def artificial_bee_colony_optimization(food_sources=20, iterations=100,
    min_values=[5,5], max_values=[5, 5], employed_bees=5, outlookers_bees=5,
    limit=10,target_function=measom, verbose=True, start_init=None, target_value=None):
sources  = initial_variables(food_sources, min_values, max_values, target_function,
    start_init)
fitness  = fitness_function(sources, fitness_calc)
best_bee = sources[np.argmin(sources[:, -1]), :]
return best_bee
```

```
11. def initial_variables(size, min_values, max_values, target_function, start_init):
        a.  dim = len(min_values)
12. if start_init is not None:
        a.  population = start_init
13. else:
14. population = np.random.uniform(min_values, max_values, (size, dim))
15. if hasattr(target_function, 'vectorized'):
        a.  fitness_values = target_function(population)
16. else:
17. fitness_values = np.apply_along_axis(target_function, 1, population)
18. population = np.hstack((population, fitness_values[:, np.newaxis]))
19. return population
20. def fitness_function(sources, fitness_calc):
21. return sources[:, -1]
22. def fitness_calc():
23. pass
24. custom_grid = {
        a.  'max_depth': [None, 10, 20, 30, 40, 50],
        b.  'min_samples_split': [2, 5, 10],
        c.  'min_samples_leaf': [1, 2, 4],
        d.  }
25. best_params = None
26. best_min_value = float('inf')
27. for max_depth in custom_grid['max_depth']:
28.     for min_samples_split in custom_grid['min_samples_split']:
29.         for min_samples_leaf in custom_grid['min_samples_leaf']:
              i.  print(f"Running ABCO with max_depth: {max_depth}, min_samples_split:
                  {min_samples_split}, min_samples_leaf: {min_samples_leaf}")
             ii.  abco_result = artificial_bee_colony_optimization()
            iii.  variables = abco_result[:-1]
             iv.  minimum = abco_result[-1]
              v.  if minimum < best_min_value:
             vi.  best_min_value = minimum
            vii.  best_params = {'max_depth': max_depth, 'min_samples_split':
                  min_samples_split, 'min_samples_leaf': min_samples_leaf}
           viii.  print("Results:")
             ix.  print(f"Optimal Variables: {variables}")
              x.  print(f"Minimum Value: {minimum}")
             xi.  print("="*50)
30. print("Best Parameters:")
31. print(best_params)
print(f"Corresponding Minimum Value: {best_min_value}")
```

2.7. Simulation and experimental setup

NS2, or network simulator 2, is an open-source tool that is useful for investigating DTNs and their vulnerability to assaults. It simulates genuine DTN situations, including incompatible connectivity, delays, and disconnections. NS2 enables the injection of malicious attacks such as floods and DDoS attacks to evaluate security mechanisms. Simulation parameters and their values used during experimental setup is shown in Table 2. To simulate an IDS within NS2, the simulation parameters, including network topology and node characteristics, are defined. A routing protocol is established for a network of 20 nodes, as depicted in the figure showcasing nodes within the NS2 environment. Each node is assigned unique attributes, such as colors and shapes, specifying their locations and connectivity parameters. Simulation of a network in NS2 environment is shown in Figure 4.

Table 2. Experimental setup simulation parameters and their values

Parameter	Values
Transmission range	200 m
Number of nodes	20
Routing protocol	Maxprop, Epidemic, ProPhet
Simulation area	1000 m by 1000 m
Travel speed	2 ms
Antenna type	Omni Antenna
Propagation	Two Ray Ground (Radio)
MAC	IEEE 802.11
Traffic type	CBR Constant Bit Rate-UDP traffic
Interface Queue	Priority Queue
Queue length	50 (max packets in queue)
Network interface	WirelessPhy
Message generation rate	100 Kbps
Packet size	1000 bytes

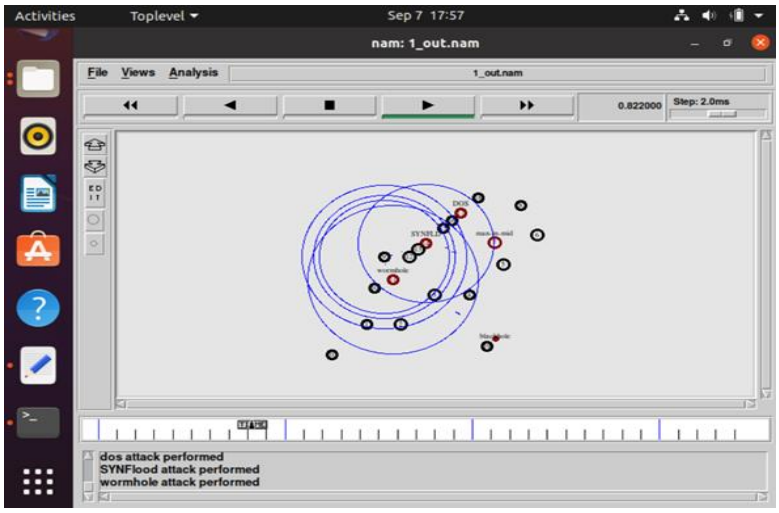

Figure 4. Simulation of nodes in NS2

3. RESULTS AND DISCUSSION

The proposed approach tests different models to identify the most effective technique for attack detection in DTNs. Scikit-learn and PyCaret models are trained and compared for performance. This analysis is crucial for developing an efficient attack detection system in the DTN environment. The model is evaluated using the NS2 simulator in a simulated environment.

3.1. Performance comparison while using Scikit-learn models

The classically trained model is tested with RF, XGBoost, CatBoost, LR, DT, Extra Trees, and Light GBM classifiers. LightGBM leads the competition in terms of accuracy, AUC, recall, and F1-Score. DT distinguishes out for its short time investment. CatBoost performs similarly to other high-performance models, although it takes significantly longer to execute. LR ranks last in total model performance while taking less time. RF, XGBoost, DT, and extra trees all perform well, albeit their execution times vary. RF has a longer execution time than LR, Light GBM, and extra trees. Figure 5 shows comparison of ML algorithm using Scikit-learn.

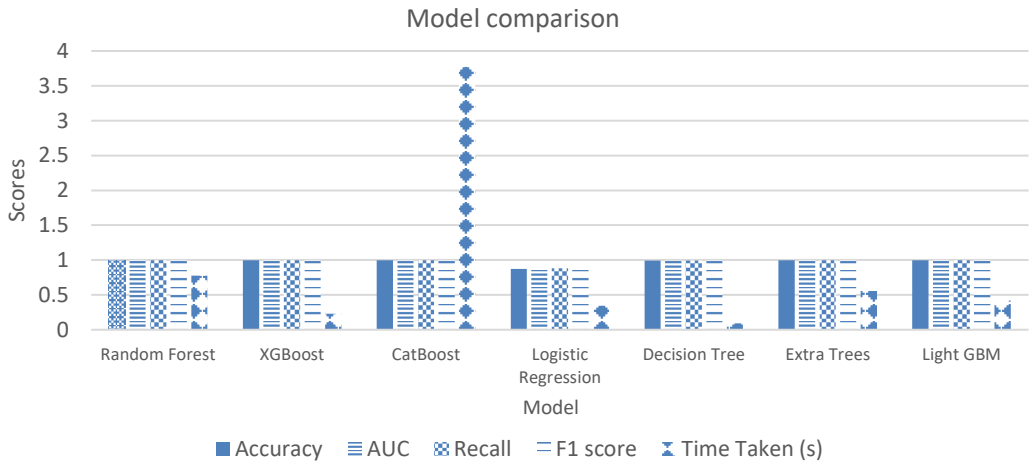

Figure 5. Model comparison in ML model using Scikit-learn

3.2. Performance comparison while using PyCaret

After applying the PyCaret model to the flood attack dataset, the model is tested with a variety of methods, including GB, k-nearest neighbors (K-NN), AdaBoost, support vector machines (SVM), linear

discriminant analysis (LDA), Ridge, quadratic discriminant analysis (QDA), Naïve Bayes (NB), and Dummy classifiers. LightGBM performs well across multiple metrics, with good Accuracy (0.9974), AUC (0.9999), Recall (0.9974), Precision (0.9975), F1-Score (0.9974), and Kappa (0.9949). Additionally, XGBoost, extra trees, RF, and DT work similarly to LightGBM. However, CatBoost has the longest execution time as shown in Table 3.

Table 3. Comparing performance of ML models in PyCaret

Model	Accuracy	AUC	Recall	Precision	F1-Score	Kappa	MCC	TT (S)
LightGBM	0.9974	0.9999	0.9974	0.9975	0.9974	0.9949	0.9949	0.4250
XGBoost	0.9972	0.9999	0.9972	0.9972	0.9972	0.9943	0.9943	0.1150
CatBoost	0.9969	0.9998	0.9969	0.9969	0.9969	0.9938	0.9939	1.7430
ExtraTrees	0.9968	0.9999	0.9968	0.9968	0.9968	0.9935	0.9935	0.1640
RF	0.9960	0.9999	0.9960	0.9960	0.9960	0.9920	0.9920	0.1870
DT	0.9947	0.9946	0.9947	0.9947	0.9947	0.9893	0.9893	0.0700
GB	0.9921	0.9996	0.9921	0.9921	0.9921	0.9842	0.9842	0.5590
K-NN	0.9916	0.9975	0.9916	0.9916	0.9916	0.9831	0.9832	0.1300
AdaBoost	0.9830	0.9987	0.9830	0.9831	0.9830	0.9659	0.9660	0.2000
SVM	0.9715	0.0000	0.9715	0.9717	0.9715	0.9426	0.9429	0.0620
LR	0.9704	0.9947	0.9704	0.9706	0.9704	0.9404	0.9407	0.1020
LDA	0.9639	0.9936	0.9639	0.9641	0.9638	0.9273	0.9276	0.0740
Ridge	0.9634	0.0000	0.9634	0.9637	0.9634	0.9264	0.9267	0.0570
QDA	0.9531	0.9702	0.9531	0.9549	0.9530	0.9055	0.9075	0.0670
NB	0.9248	0.9697	0.9248	0.9252	0.9248	0.8490	0.8494	0.0640
Dummy	0.5339	0.5000	0.5339	0.2850	0.3716	0.0000	0.0000	0.0730

LightGBM outperforms XGBoost slightly with superior performance in accuracy, AUC, recall, precision, F1, Kappa, and MCC. DT has faster execution but worse overall performance compared to LightGBM, XGBoost, CatBoost, Extra Trees, and RF models. Ridge, SVM, LR, LDA, QDA, NB, and Dummy classifier perform poorly with shorter execution time. Figure 6 depicts comparison of ML algorithms using PyCaret.

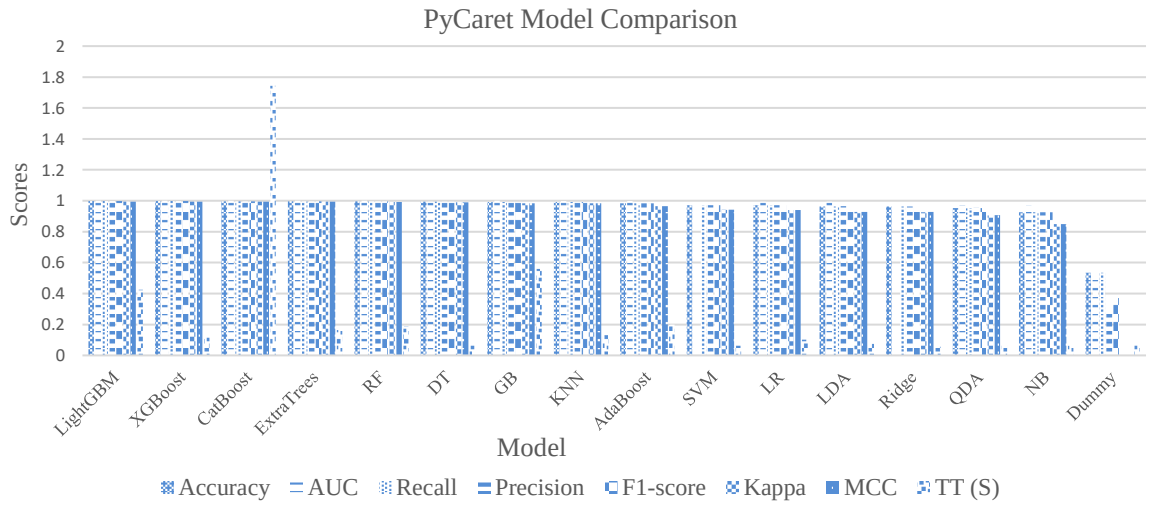

Figure 6. Model Comparison in ML model using PyCaret

3.3. Artificial bee colony optimizer

These scores are the outcome of a 10-fold cross-validation. Table 4 shows performance metrics for each of the ten folds. Particularly, fold 0 has the best performance, with higher metrics across all categories, including accuracy (0.9977), AUC (0.9977), recall (0.9977), precision (0.9977), F1-Score (0.9977), Kappa coefficient (0.9954), and MCC (0.9954). As a result, the other folds perform significantly worse than fold 0. Fold 1 has the lowest accuracy compared to the other folds. The average performance of all folds is accuracy (0.9957), AUC (0.9957), recall (0.9957), precision (0.9957), F1 (0.9957), kappa (0.9913), and MCC (0.9913). The standard deviation of all folds is accuracy, AUC, recall, precision, F1 is 0.0019 and Kappa, MCC is 0.0039.

Table 4. Fold cross-validation score

Fold	Accuracy	AUC	Recall	Precision	F1-Score	Kappa	MCC
0	0.9977	0.9977	0.9977	0.9977	0.9977	0.9954	0.9954
1	0.9909	0.9911	0.9909	0.9910	0.9909	0.9818	0.9818
2	0.9960	0.9961	0.9960	0.9960	0.9960	0.9920	0.9920
3	0.9943	0.9942	0.9943	0.9943	0.9943	0.9886	0.9886
4	0.9960	0.9962	0.9960	0.9960	0.9960	0.9920	0.9920
5	0.9943	0.9942	0.9943	0.9943	0.9943	0.9886	0.9886
6	0.9972	0.9971	0.9972	0.9972	0.9972	0.9943	0.9943
7	0.9972	0.9970	0.9972	0.9972	0.9972	0.9943	0.9943
8	0.9960	0.9960	0.9960	0.9960	0.9960	0.9920	0.9920
9	0.9972	0.9972	0.9972	0.9972	0.9972	0,9943	0.9943
Mean	0.9957	0.9957	0.9957	0.9957	0.9957	0.9913	0.9913
Std	0.0019	0.0019	0.0019	0.0019	0.0019	0.0039	0.0039

3.4 Comparing the mean value before and after optimization

Figure 7 shows the comparison of mean values before and after optimization on various performance metrics. A comparison is done between the model's performance before and after optimization. Figure 7 depicts how the accuracy of the proposed model improves after optimization. Additionally, multiple performance indicators such as AUC, recall, precision, F1-Score, Kappa, and Matthews correlation coefficient (MCC) shows improvement after adjusting values. Before optimization, the model's TT(S) was 0.2558, indicating the computational effort involved. However, after optimization, the proposed approach improves performance across various criteria, indicating a more efficient and effective model.

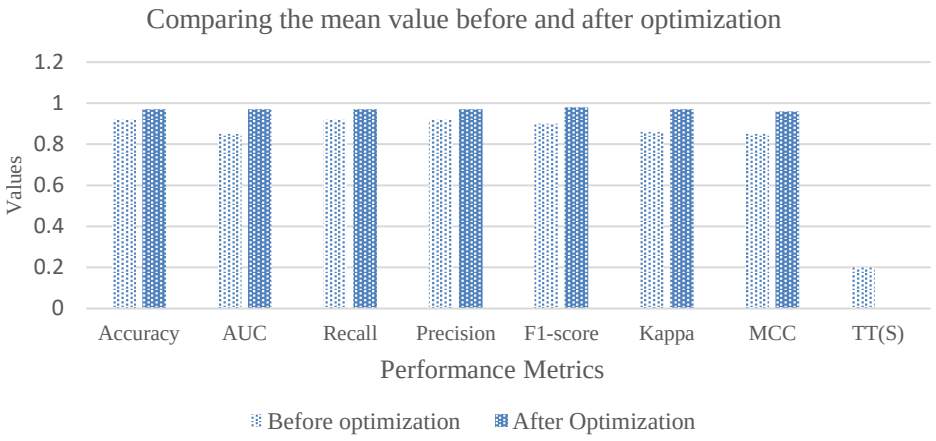

Figure 7. Comparing the mean value before and after optimization

3.5. Performance comparison of the proposed model with existing models

Table 5 shows comparison of artificial bee PyCaret Classifier model performance with other existing models. The suggested artificial bee PyCaret classifier outperforms various current models, including PyCaret using the random forest (RF) algorithm, PyCaret using extreme gradient boosting, the NP technique, and AutoML using the PyCaret model. This demonstrates the efficacy of the technique in enhancing prediction performance. The artificial Bee PyCaret classifier takes advantage of the capabilities of both PyCaret and ABC algorithms, resulting in more accurate and robust model predictions. Figure 8 depicts comparison of proposed model and existing algorithms as above.

Table 5. Comparison of the proposed model and other existing algorithms

Reference	Model	Algorithm	Accuracy	TT (S)
The proposed model	Hybrid ABPC	LightGBM	0.9957	0.4250
	PyCaret	RF	0.9960	0.1870
Bajpai and Sharma [27]	PyCaret	ExGBoost	0.8371	-
	PyCaret	RF	0.8355	-
Baroumand *et al.* [28]	NP approach	RF	0.757	-
Hassan and Alshareef [29]	AutoML using PyCaret	RF	0.9916	-

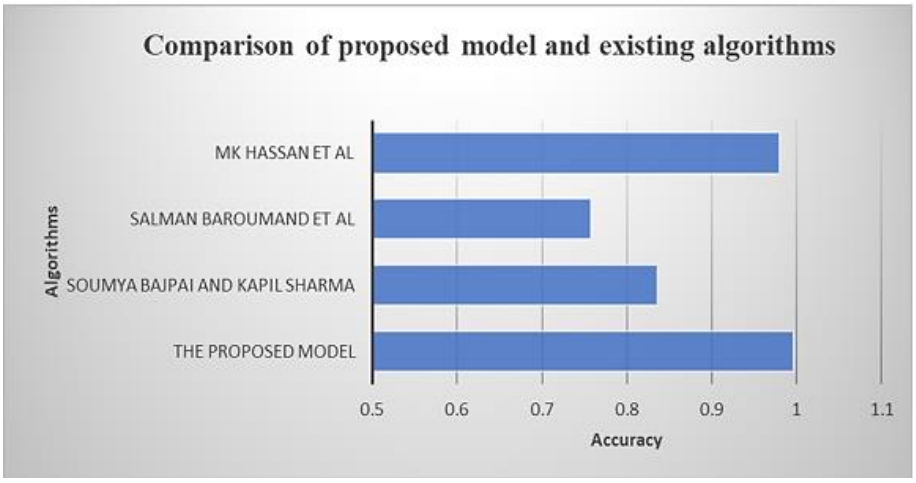

Figure 8. Comparison of proposed model and existing algorithms

3.6. Simulation results

Data transmission is initiated by a subset of nodes known as packet senders. The simulator monitors packet receipt at nodes and identifies any that display malicious features. Simulation results provide information about packet transmission dynamics and the detection of hazardous packets in the network. Figure 9 depicts simulation results for sender node 5 to receiver node 9. During the DTN simulation, attacks are launched on certain nodes, resulting in the creation of malicious traffic or unusual behavior. The IDS examines network traffic, comparing packets and nodes to predetermined criteria to detect malicious behavior. As the simulation runs, packets move over the network, and the results are utilized to evaluate metrics such as packet loss and received packet correctness. This aids in determining the IDSs efficacy in minimizing attacks within the DTN.

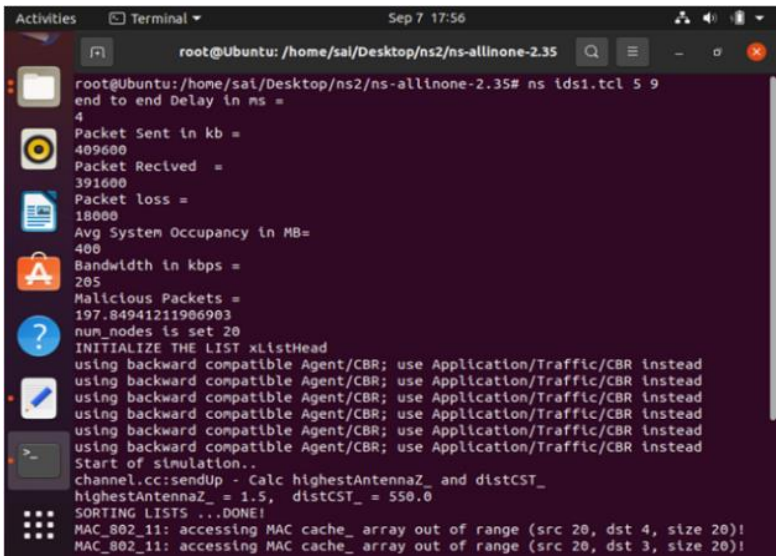

Figure 9. Simulation results for 5 to 9 nodes

4. CONCLUSION

DTNs are at risk from increasing attacks such as DDoS, flooding, and attacks due to their inconsistent connectivity. ML-based IDSs use algorithms to detect malicious activity within DTNs, which improves network security against developing threats. The study demonstrates the effectiveness of ML approaches, especially the Scikit-learn and PyCaret models, in DTNs. LightGBM outperforms both models on criteria such as accuracy, AUC, recall, and F1-Score. While DT provides the best time efficiency, its overall performance

is lower. PyCaret simplifies data preprocessing, and LightGBM consistently outperforms across metrics while drastically lowering processing time. CatBoost outperforms other algorithms despite its greater execution time. PyCaret is recommended by the study for its highly efficient attack detection capabilities. ABCO improves feature selection by replicating honeybee foraging behavior, which effectively explores the feature space.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Rajashri Chaudhari	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
Manoj Deshpande		✓		✓		✓				✓	✓	✓	✓	

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The authors confirm that the data supporting the findings of this study are available within the article [and/or its supplementary materials].

REFERENCES

[1] S. Chatterjee, M. Nandan, A. Ghosh, and S. Banik, "DTNMA: Identifying routing attacks in delay-tolerant network," in *Cyber Intelligence and Information Retrieval*, 2022, pp. 3–15. doi: 10.1007/978-981-16-4284-5_1.
[2] S. P., S. U., C. Iwendi, S. Mohan, and G. Srivastava, "Field-programmable gate arrays in a low power vision system," *Computers & Electrical Engineering*, vol. 90, Mar. 2021, doi: 10.1016/j.compeleceng.2021.106996.
[3] A. V. Vasilakos, Y. Zhang, and T. Spyropoulos, Eds., *Delay tolerant networks*. CRC Press, 2016. doi: 10.1201/b11309.
[4] D. I. Elewaily, H. A. Ali, A. I. Saleh, and M. M. Abdelsalam, "Delay/disruption-tolerant networking-based the integrated deep-space relay network: state-of-the-art," *Ad Hoc Networks*, vol. 152, Jan. 2024, doi: 10.1016/j.adhoc.2023.103307.
[5] P. Tiwari, "Secure group communication in delay tolerant mobile ad-hoc network," *International Journal of Advanced Computer Technology*, vol. 12, no. 5, pp. 1–10, 2023.
[6] S. Shafi, S. Mounika, and S. Velliangiri, "Machine learning and trust based AODV routing protocol to mitigate flooding and blackhole attacks in MANET," *Procedia Computer Science*, vol. 218, pp. 2309–2318, 2023, doi: 10.1016/j.procs.2023.01.206.
[7] S. Perumal, V. Raman, G. N. Samy, B. Shanmugam, K. Kisenasamy, and S. Ponnan, "Comprehensive literature review on delay tolerant network (DTN) framework for improving the efficiency of internet connection in rural regions of Malaysia," *International Journal of System Assurance Engineering and Management*, vol. 13, no. S1, pp. 764–777, Mar. 2022, doi: 10.1007/s13198-022-01632-2.
[8] S. Fraihat, S. Makhadmeh, M. Awad, M. A. Al-Betar, and A. Al-Redhaei, "Intrusion detection system for large-scale IoT NetFlow networks using machine learning with modified Arithmetic optimization algorithm," *Internet of Things*, vol. 22, Jul. 2023, doi: 10.1016/j.iot.2023.100819.
[9] S. V. N. S. Kumar, M. Selvi, and A. Kannan, "A comprehensive survey on machine learning-based intrusion detection systems for secure communication in internet of things," *Computational Intelligence and Neuroscience*, vol. 2023, no. 1, Jan. 2023, doi: 10.1155/2023/8981988.
[10] A. Halimaa A. and K. Sundarakantham, "Machine learning based intrusion detection system," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, Apr. 2019, pp. 916–920. doi: 10.1109/ICOEI.2019.8862784.
[11] K. V. Krishna and K. G. Reddy, "Classification of distributed denial of service attacks in VANET: A survey," *Wireless Personal Communications*, vol. 132, no. 2, pp. 933–964, Sep. 2023, doi: 10.1007/s11277-023-10643-6.
[12] W. Khalid *et al.*, "A taxonomy on misbehaving nodes in delay tolerant networks," *Computers & Security*, vol. 77, pp. 442–471, Aug. 2018, doi: 10.1016/j.cose.2018.04.015.
[13] W. Khalid, N. Ahmed, M. Khalid, A. Ud Din, A. Khan, and M. Arshad, "FRID: Flood attack mitigation using resources efficient intrusion detection techniques in delay tolerant networks," *IEEE Access*, vol. 7, pp. 83740–83760, 2019, doi: 10.1109/ACCESS.2019.2924587.

[14] H. Sharma, A. Haque, and F. Blaabjerg, "Machine learning algorithms for wireless sensor networks: A survey," *Electronics*, vol. 10, no. 9, Apr. 2021, doi: 10.3390/electronics10091012.
[15] D. P. Kumar, T. Amgoth, and C. S. R. Annavarapu, "Machine learning algorithms for wireless sensor networks: A survey," *Information Fusion*, vol. 49, pp. 1–25, Sep. 2019, doi: 10.1016/j.inffus.2018.09.013.
[16] J.-Y. Yu, E. Lee, S.-R. Oh, Y.-D. Seo, and Y.-G. Kim, "A survey on security requirements for WSNs: Focusing on the characteristics related to security," *IEEE Access*, vol. 8, pp. 45304–45324, 2020, doi: 10.1109/ACCESS.2020.2977778.
[17] S. B. Balasubramanian *et al.*, "Machine learning based IoT system for secure traffic management and accident detection in smart cities," *PeerJ Computer Science*, vol. 9, Mar. 2023, doi: 10.7717/peerj-cs.1259.
[18] K. He, D. D. Kim, and M. R. Asghar, "Adversarial machine learning for network intrusion detection systems: a comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 538–566, 2023, doi: 10.1109/COMST.2022.3233793.
[19] R. Ahmad, R. Wazirali, and T. Abu-Ain, "Machine learning for wireless sensor networks security: An overview of challenges and issues," *Sensors*, vol. 22, no. 13, Jun. 2022, doi: 10.3390/s22134730.
[20] A. Singh, "Classification of malware in HTTPs traffic using machine learning approach," *El-Cezeri Fen ve Mühendislik Dergisi*, Jan. 2022, doi: 10.31202/ecjse.990318.
[21] A. Karami and N. Derakhshanfard, "BMRTD: Buffer Management policy based on Remaining Time to encounter nodes with the Destination node in delay tolerant networks." Jan. 31, 2023. doi: 10.21203/rs.3.rs-2522936/v1.
[22] O. E. Taylor and P. S. Ezekiel, "A smart system for detecting behavioural botnet attacks using random forest classifier with principal component analysis," *European Journal of Artificial Intelligence and Machine Learning*, vol. 1, no. 2, pp. 11–16, Mar. 2022, doi: 10.24018/ejai.2022.1.2.4.
[23] F. Alasmary, S. Alraddadi, S. Al-Ahmadi, and J. Al-Muhtadi, "ShieldRNN: A distributed flow-based DDoS detection solution for IoT using sequence majority voting," *IEEE Access*, vol. 10, pp. 88263–88275, 2022, doi: 10.1109/ACCESS.2022.3200477.
[24] O. Johnphill *et al.*, "Self-healing in cyber–physical systems using machine learning: A critical analysis of theories and tools," *Future Internet*, vol. 15, no. 7, Jul. 2023, doi: 10.3390/fi15070244.
[25] M. Kayalvizhi and S. Geetha, "Efficacy artificial bee colony optimization-based gaussian AOMDV (EABCO-GAOMDV) routing protocol for seamless traffic rerouting in stochastic vehicular ad hoc network," *International Journal of Computer Networks and Applications*, vol. 10, no. 6, pp. 993–1014, Dec. 2023, doi: 10.22247/ijcna/2023/223694.
[26] J. Jayaudhaya, R. Jayaraj, and K. K. Ramash, "A new integrated approach for cloud service composition and sharing using a hybrid algorithm," *Mathematical Problems in Engineering*, vol. 2024, pp. 1–11, Feb. 2024, doi: 10.1155/2024/3136546.
[27] S. Bajpai and K. Sharma, "A framework for intrusion detection models for IoT networks using deep learning." Sep. 01, 2022. doi: 10.21203/rs.3.rs-2010844/v1.
[28] S. Baroumand, A. Zaman, and L. Mihaylova, "Attack detection and fault-tolerant control of interconnected cyber-physical systems against simultaneous replayed time-delay and false-data injection attacks," *IET Control Theory & Applications*, vol. 17, no. 5, pp. 527–541, Mar. 2023, doi: 10.1049/cth2.12393.
[29] M. K. Hassan and I. Y. Alshareef, "Drift detection and model update using unsupervised AutoML in IoT," *WSEAS Transactions on Computers*, vol. 22, pp. 332–337, Dec. 2023, doi: 10.37394/23205.2023.22.38.

BIOGRAPHIES OF AUTHORS

Rajashri Chaudhari is a research scholar and currently pursuing a Ph.D. in Computer Engineering from ACPCE, Kharghar, Navi Mumbai, Maharashtra, India. She completed her M. E. in Computer Science and Engineering in 2017 and completed B. E. degree in Computer Engineering in 2014. Her research interest includes network security, wireless network, delay tolerant network, network routing, machine learning, PyCaret optimization. She can be contacted at email: c.rajashri2021@gmail.com.

Manoj Deshpande has obtained his M.Tech and PhD from Indian Institute of Technology Bombay, Mumbai in 2002 and 2009 respectively from IDP in Systems and Control Engineering. Currently he is working as professor and dean at A. C. Patil College of Engineering Navi Mumbai. He can be contacted at email: mmdeshpande@acpce.ac.in.