Diyala Journal of Engineering Sciences

Journal homepage: https://djes.info/index.php/djes

ISSN: 1999-8716 (Print); 2616-6909 (Online)

IoT Network Vigilant: A Hybrid Framework for Efficient Prediction of Robust IoT Network Intrusion Detection Using FPA-BiNN

R.M. Alamelu[1*], J.Christy Eunaicy[2], T.S.Urmila[3], C. Jayapratha[4], J.Naveen Ananda Kumar[5], G.B.GovindaPrabhu[6], R. Mahalakshmi Priya[6]

[1]Department of Computer Science, Sri sarada Niketan College for Women, Amaravathi pudur, Karaikudi. Tamilnadu, India.
[2]Department of CA & IT, Thiagarajar College, Madurai.
[3]Department of Computer Science, Thiagarajar College.
[4]Department of Computer Science and Engineering, Karpaga Vinayaga College of Engineering and Technology, Madhuranthagam, Tamilnadu, India.
[5]Data Engineer, Tekinvaderz, LLC, Florida, USA.
[6]Department of Computer Science, Madurai Kamaraj University, Madurai.

ARTICLE INFO

Article history:
Received: 30/06/2025.
Revised: 28/10/2025.
Accepted: 01/11/2025.
Available online: 10/12/2025

Keywords:
IoT Security
Intrusion Detection
Two Phase Flower Pollination Algorithm
Binarized Neural Networks
Novel Feature Engineering

ABSTRACT

The number of Internet of Things (IoT) devices is increasing rapidly, with estimates predicting more than 41 billion devices by 2025. This growth has also expanded the attack surface, making IoT networks highly vulnerable to cyberattacks. Traditional intrusion detection systems are not suitable for IoT because they depend on known attack signatures, require high computational power, and many false alarms. These limitations make them difficult to deploy on resource-constrained edge devices. The aim of this study is to develop a lightweight and accurate intrusion detection framework specifically designed for IoT environments. We introduce IoT Network Vigilant, a hybrid framework improves intrusion detection performance while remaining efficient enough for real-time use. The framework consists of three key parts. First, we design 27 new IoT-specific features that capture device behavior, traffic asymmetry, and temporal patterns. Second, we apply a two-stage Flower Pollination Algorithm (FPA) to select the most useful features. The first stage ranks features using mutual information, and the second stage removes redundant features using correlation analysis. This process reduces the dataset size by about one-third. Third, we employ Binarized Neural Networks (BNNs), which use binary weights and activations, allowing fast and low-power classification. The model is tested on the IoTID20 dataset, and class imbalance is handled using SMOTE. The results show strong performance, with 98.43% accuracy, 99.03% precision, and 97.32% recall. These scores represent a 4.5% improvement in accuracy compared with existing methods. Overall, this framework offers a robust, efficient, and deployable intrusion detection solution for modern IoT networks.

1. INTRODUCTION

The rapid expansion of IoT networks has led to a considerable increase in the attack surface available for cyber threats. As a result, these networks have been singled out as the most attractive targets for the occurrence of malicious activities. Contemporary threat intelligence has revealed alarming trends of IoT-targeted attacks. The number of attacks surpassed the 112 million marks in 2022. These figures showed a 243% increase compared to the 2018 baseline measurements. Various sophisticated threats have been engineered to include Distributed Denial of Service (DDoS) attacks and advanced persistent botnets. Among the examples, the Mirai botnet family, Man-in-the-Middle (MITM) attacks, credential stuffing, and data exfiltration campaigns have attracted significant attention. The differences in attack distribution between regions were very

*Corresponding author's Email: alamuinfo@gmail.com
DOI: 10.24237/djes.2025.18408

pronounced. Asia had a growth rate of 170% while Latin America had a 164% increase in IoT-targeted incidents during 2023 [1].

IoT environments have raised unique cybersecurity issues that differentiate them from the traditional network infrastructures. These types of networks are mainly defined by several significant constraints. Such limitations are resource constraints as IoT devices are usually designed to function in a hardware environment with low computing power, memory, and energy supply. A heterogeneous device ecosystem consists of various devices with different functionalities, protocols, and security implementations. The environment shows asymmetric communication patterns where data traffic flows are usually directional imbalances between sensors and servers. The majority of IoT devices have real-time processing requirements, thus the detection mechanisms have to be efficient. Table 1 contains the worldwide yearly trends of IoT cyberattacks from 2018 to 2024. It reflects attack volumes, growth rates, and the number of peak incidents during a month and also indicates the main types of attacks and the most important regions of the world [2].

Table 1. Year wise IoT Cyber attacks

Year	Annual IoT Attacks (Millions)	Year-over-Year Growth	Peak Monthly Attacks (Millions)	Primary Attack Types	Regional Trends	Sector-Specific Impacts
2018	32.7	-	Not available	Malware, DDoS	Global increase	Routers targeted (5,200 attacks/router/month)
2019	2,900	8,768% (from 2018)	Not available	Malware, Botnets	Global surge	Consumer electronics (63% of IoT units)
2020	Not available	-	~6 (December)	DDoS, Malware	Global growth	Healthcare IoT adoption (40% of devices)
2021	Not available	-	~6 (December)	DDoS, Credential Stuffing	Global increase	Smart home devices (12.86 billion)
2022	112	243% (from 2018 baseline)	13 (June)	DDoS, Mirai Botnet, Malware	North America (69.3% of attacks), Mexico	Manufacturing (54.4% of attacks), Retail ($20B loss)
2023	77.9 (first half)	37% (from H1 2022)	Not available	DDoS, Botnets, Ransomware	Asia (170% growth), Latin America (164% growth)	Manufacturing, Education (-73% in government)
2024	Not fully available	400% (from 2022, per Zscaler)	Not available	Mirai Botnet, DDoS, Credential Stuffing	China, Japan (primary targets)	Critical Infrastructure, Healthcare (123% attack increase)

Traditional intrusion detection systems (IDS) have been proven inadequate for IoT contexts due to fundamental limitations. Signature-based detection systems are constrained by their dependence on predefined attack patterns. Effectiveness against zero-day exploits and novel attack variants is rendered impossible through this limitation. Anomaly-based detection systems are capable of identifying previously unknown attacks. However, high computational complexity and elevated false positive rates are frequently suffered by these systems. Practical deployment on resource-constrained IoT devices is made impractical through these characteristics. Specialized feature engineering is often lacking in conventional approaches to capture IoT-specific traffic behaviors and attack patterns.

Figure 1 "Global IoT Attack Trends, 2018–2023" shows how IoT threats have grown in scale and complexity over time. In 2018–2019, attacks surged by 8,768% due to weak passwords and unpatched firmware in devices like routers. By 2022, over 112 million attacks were recorded, with peaks driven by DDoS and Mirai botnets [3].

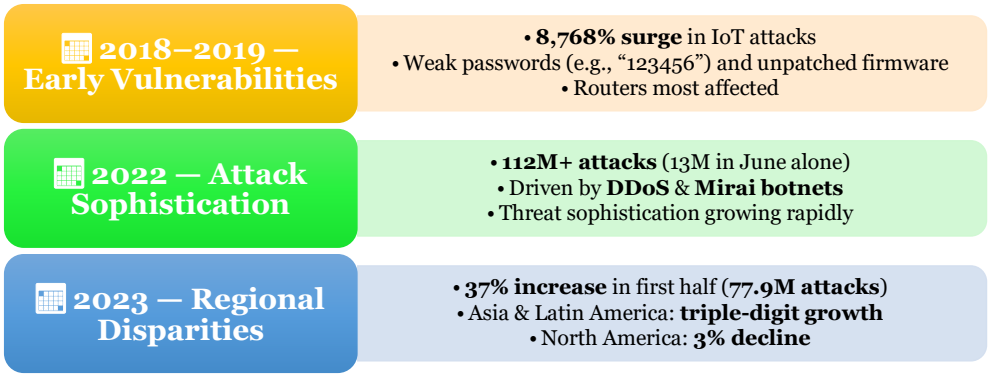

Figure 1. Global IoT Attack Trends, 2018-23

Firstly, various machine learning (ML) and deep learning (DL) techniques have been proposed by research to alleviate challenges in IoT intrusion detection system (IDS) [4]. Along these lines have been utilized extensively, among them are Long Short-Term Memory (LSTM) networks, Convolutional Neural Networks (CNNs), ensemble methods, and federated learning frameworks. The accuracy of these methods, as evidenced by experimental results, has been significantly improved. In particular, detection rates of 95%~99% on different datasets have been reported. On the other hand, a considerable amount of computational power is still necessary, which is usually not available in ordinary IoT edge devices. Moreover, most of them do not pay enough attention to feature engineering, which should be not only comprehensive but also highly specific for IoT network characteristics. The lack of their work in this area limits the effectiveness of the methods employed in detecting the secretive attack patterns of IoT that are difficult to impersonate.

To deal with these critical issues, "IoT Network Vigilant," an innovative hybrid framework designed for efficient and robust IoT network intrusion detection, is introduced. The fundamental limitations of existing approaches are resolved by three integrated components. Novel feature engineering is realized by the creation of 21 new custom IoT-specific features. Asymmetric traffic patterns, temporal anomalies, and protocol-level behaviors typical of IoT communications are detected by these features. A two-stage Flower Pollination Algorithm (FPA) is used as a nature-inspired metaheuristic optimization method. The algorithm sequentially maximizes feature relevance and minimizes redundancy. The dimensionality reduction of about 33% is accomplished while the essential detection capabilities are retained. Binarized Neural Networks (BNNs) are used as a lightweight deep learning architecture.

The binary weights and activations are used to obtain high accuracy with very low computational overhead. This architecture is characterized by a memory footprint of ~10 KB and an inference time of less than 1 ms. A complete evaluation has been done with the IoTID20 dataset. The collection includes real IoT network traffic for normal communications and diverse attack scenarios. The Synthetic Minority Oversampling Technique (SMOTE) was applied to solve the problem of class imbalance inherent in the dataset. The experimental results have shown excellent performance metrics. The accuracy of 98.43%, precision of 99.03%, and recall of 97.32% have been reached. Result section (Figure 10) reflects a substantial increase of 4.5% over the baseline methods. The main contributions of this work can be outlined as follows:

a) IoT-Specific Feature Engineering: 27 new features that were specifically designed for IoT network behaviors have been introduced. These features include sophisticated ratio-based metrics, entropy measures, and fractal dimension analysis. The detection of asymmetric and temporal anomalies is more accurate compared to that of a standard feature set.
b) Two-Stage Feature Optimization: A novel two-stage Flower Pollination Algorithm has been put into effect. Feature relevance is first of all optimized by mutual information maximization. Redundancy is alleviated through correlation analysis. Around 33% of dimensionality reduction is done while the most important detection capabilities are kept.
c) Lightweight Deep Learning Architecture: Binarized Neural Networks with binary weights and activations have been used. A very small memory footprint of ~10 KB and an inference time of less than 1ms are accomplished. The implementation of the method, which is a resource-constrained IoT device, is feasible, therefore, is enabled.
d) Comprehensive Performance Validation: Excellent performance on the IoTID20 dataset has been shown. Class imbalance problem is

solved by applying SMOTE. Accurate detection of a broad range of attacks such as Mirai botnet, DDoS, scanning, and MITM attacks is possible.

e) Computational Efficiency Analysis: The evaluation of computational trade-offs in great detail has been done. The significant resource optimization as compared to traditional neural networks is shown. The high detection accuracy that is suitable for real-time IoT security applications is still there.

The rest of the paper is structured as follows: Firstly, a thorough review of recent IoT intrusion detection system literature is provided in Section 2. The paper analyses deep learning techniques, feature selection methods, and sophisticated frameworks while pointing out the research gaps. Section 3 describes the proposed method in detail. It includes the IoTID20 dataset description, data preprocessing steps, novel feature extraction algorithms, two-stage FPA implementation, and BNN architecture design. Section 4 focused the experimental setup, implementation environment, and an extensive performance evaluation of the work are detailed. Also, the paper contains the comparative analysis with baseline methods. Section 5 concludes the paper with a summary of the main findings and the authors' contributions to the field of IoT cybersecurity.

2. REVIEW OF LITERATURE

A comprehensive review is made of IoT intrusion detection systems (IDS) from 2022 to 2025 focusing on three areas: deep learning, feature selection with machine learning, and federated learning/advanced techniques. The explosive growth of IoT devices has increased their vulnerabilities, thus demanding an IDS that takes into account resource constraints, data heterogeneity, and the vastness of the attack surface. The core methods, discoveries, and challenges have been synthesized from the articles to spot the trends and guide the draft of the new research work that combines the Flower Pollination Algorithm (FPA) and Binarized Neural Network (BINN).

2.1 Deep Learning-Based Methods

Deep learning-based IDSs have been proven to be very accurate, but they also run into issues of heavy computation. A LSTM-based Deep-IDS, whose detection rate was 96.8% and accuracy was 95.67% on CIC-IDS2017, was the system to be proposed in the paper [5]. Due to its high computational cost, the system is not suitable for devices with limited resources. A hybrid CNN-LSTM IDS, which was installed on Raspberry Pi and accompanied by fog computing, was created to detect DoS attacks on IoTID20 and Edge-IIoTset, thus, leading to the efficient detection of these attacks [6]. However, its scalability is still limited due to resource demands. A feature filtering method that uses a Self-Attention-based Deep CNN (SA-DCNN) was also proposed, and it outperformed the baselines on IoTID20 and Edge-IIoTset while increasing memory usage [7]. The precision of these methods, which is between 95 and 99%, is acknowledged by the proposed BINN, which carries out lightweight binarized operations.

2.2 Feature Selection and Machine Learning-Based Methods

Feature selection and machine learning techniques are predominantly efficiency-driven. A simple-featured intrusion detection system (IDS) [8] with feature grouping was created, resulting in more than 99.5% accuracy on publicly available IoT datasets [9]. There is hardly any novel IoT-specific feature engineering. Random Forest on UNSW-NB15 was implemented, producing 90.17% accuracy, but the performance was limited due to non-IoT data [10]. An ensemble of Decision Trees and Gradient Boosting was committed to, attaining ~99% accuracy on IoT datasets, however, feature selection optimization was missing [11]. The proposed FPA is reducing IoTID20 dimensionality by around 33%, thus becoming more adaptable.

2.3 Federated Learning and Advanced Techniques

Federated modeling approaches have emerged as a promising solution for anomaly detection without compromising data confidentiality [12]. To tackle the challenge of highly imbalanced IoT traffic, lightweight generative models such as S2CGAN have demonstrated improved detection performance with minimal computational overhead [13]. Further, semantic transfer networks [14] and geometric graph alignment techniques [15] offer effective domain adaptation strategies, enabling robust intrusion detection across diverse and evolving IoT environments.

Federated learning along with sophisticated techniques have the potential to solve issues related to privacy and the ability to scale. To reduce false alarm rate, a federated CNN-attention IDS for IIoT was developed, which improved most of the metrics but caused latency to increase [16]. modern IDS research highlights the need for scalable, adaptive, and energy-efficient solutions to meet the growing complexity of IoT environments [17]. CIDIoT was integrated with differential privacy and proposed as an efficient imbalance handling method. However, it is a complex system for edge devices [18]. A multimodal transfer learning with ResNet was used to achieve 98.2% accuracy on CIC-IoT datasets. Still, the process was resource-intensive [19]. The

MSAAD framework was introduced to counter adversarial attacks, and it achieved 99.39-99.48% accuracy on IoTID20/CICIoT2023. However, it is too complicated for lightweight systems [20]. A GNN-based botnet detection method was proposed, which is temporally dynamic and excels in temporal dynamics, but it is limited to binary tasks [21]. The development of interpretable blending models was focused on, transparency being the main advantage, and efficiency was sacrificed for IoTID20/CICIoT2023 [22]. The reported interventions were ensembles and NBOA-CNN methods, which led to very high accuracy but did not consider scalability issues [23]. An intrusion detection system based on CNN was created using the CSE-CIC-IDS2018 dataset to enhance the system's reliability, but it was not relevant to the IoT domain [24]. Table 2 presents a comparative study of the latest works, along with their achievements and limitations.

Table 2. Comparative Analysis of the Latest Trend Research

Dataset(s)	Proposed Work	Achievements	Limitations
CIC-IDS2017 [5]	Deep-IDS: LSTM-based IDS for IoT nodes using deep learning.	96.8% detection rate, 95.67% accuracy, 1.49s detection time.	High computational complexity of LSTM, unsuitable for resource-constrained IoT devices.
IoTID20, Edge-IIoTset [6]	Hybrid CNN-LSTM IDS on Raspberry Pi using fog computing architecture.	Effective against diverse DoS attacks, low latency via fog computing.	High computational requirements of CNN-LSTM, limited scalability for large IoT networks.
IoTID20, Edge-IIoTset [7]	Self-Attention-based Deep CNN (SA-DCNN) with mutual information-based feature filtering.	Superior performance compared to ML/DL baselines, reduced underfitting.	Complex architecture increases memory and processing demands.
Three public IoT datasets (unspecified) [9]	Lightweight IDS using feature grouping and fast protocol parsing.	>99.5% classification accuracy, reduced computational overhead.	Lacks novel feature engineering, limited adaptability to diverse IoT attack patterns.
CIC-IoT2022, CIC-IoT2023, Edge-IIoT [19]	Multimodal IDS with transfer learning, ResNet for texture features, and word2vec for semantic features.	98.2% accuracy across three datasets, robust to multimodal data.	High computational resource requirements, not suitable for edge devices.
IoTID20, CICIoT2023 [20]	Multistage Adversarial Attack Defense (MSAAD) with Resilient Adversarial Detector, multiple classifiers, and Multi-Armed Bandit.	99.39–99.48% accuracy (IoTID20), 99.01–99.14% (CICIoT2023) under adversarial attacks.	Complex multi-stage architecture, may not be feasible for lightweight IoT systems.
IoTID20, CICIoT2023 [22]	Interpretable IDS with blending model classification and explanation techniques.	High accuracy, improved interpretability for attack detection.	Limited focus on computational efficiency, not optimized for resource-constrained devices.
CSE-CIC-IDS2018 [24]	CNN-based IDS for improved attack detection.	Improved accuracy and reliability over traditional methods.	Focus on non-IoT dataset (CSE-CIC-IDS2018), limited IoT-specific applicability.

IoT intrusion detection systems (IDS) from 2022 to 2025 show progress in accuracy but face challenges in resource constraints and scalability. Deep learning methods, including LSTM, SA-DCNN, and CNN-LSTM, achieve 95.67–99.5% accuracy on CIC-IDS2017, IoTID20, and Edge-IIoTset, but high computational costs limit their use in resource-constrained IoT settings. Feature selection and machine learning approaches offer efficiency with ~99% accuracy, yet lack IoT-specific feature engineering. Federated learning and advanced techniques enhance privacy and adversarial defense, reaching 99.48% accuracy, but introduce latency or complexity unsuitable for edge devices. IoTID20-based studies perform well in real-time scenarios but often neglect efficiency. The proposed FPA-BINN framework combines IoT-tailored features, FPA-based selection, and lightweight BINNs, achieving 98.43% accuracy while optimizing for constrained environments, enabling scalable, real-time IoT IDS.

2.4 Research Gaps and Proposed Contributions

The surveyed papers point out several gaps that our proposed FPA-BINN framework can fill:

a) Resource Constraints: Deep learning-based methods can achieve very high accuracy levels however; they require a lot of computational power and are thus not suitable for IoT devices with limited resources. Our BINN-based method employs binarized weights and activations so as to decrease the computational and memory requirements to a large extent.
b) Feature Engineering: A large number of methods take as a given that features are standard or do a simple feature selection, thus the methods are unable to recognize the attack patterns in IoT that are specific. Our framework generates new features that are not only appropriate for the IoTID20 dataset but also allow the detection of the wide variety of threats.
c) Scalability and Efficiency: Federated learning and multimodal approaches are good for privacy and scalability but have the disadvantages of latency or complexity. The FPA in our framework is a feature selection tool that makes dataset dimensionality lower and thus allows the scalable detection that is also real-time.
d) Data Imbalance: The authors that have been referenced in brackets 7 and 12 have suggested that data imbalance is one of the issues that hinders the performance of the classification models. On the other hand, our framework uses SMOTE to address the problem of data imbalance while employing BINNs for the efficient classification task.

3. PROPOSED MODEL

An innovative approach to intrusion detection is presented here, using the comprehensive IoTID20 dataset. Three key components address the unique challenges of IoT security:

- Novel Feature Generation: From the IoTID20 dataset we compose new features to better differentiate normal from abnormal network behavior
- Optimal Feature Selection: Through the use of the Flower Pollination Algorithm (FPA) we not only find the most important features but also lessen the computational time.
- Efficient Classification: Classification in this research is done with Binarized Neural Networks (BINNs) which is a good compromise between accuracy and saving of resources.

Figure 2 represents the complete layout and working of the proposed IoT Intrusion Detection model. It sequentially goes through all processing layers and indicates the data flow by the arrow pointing towards.

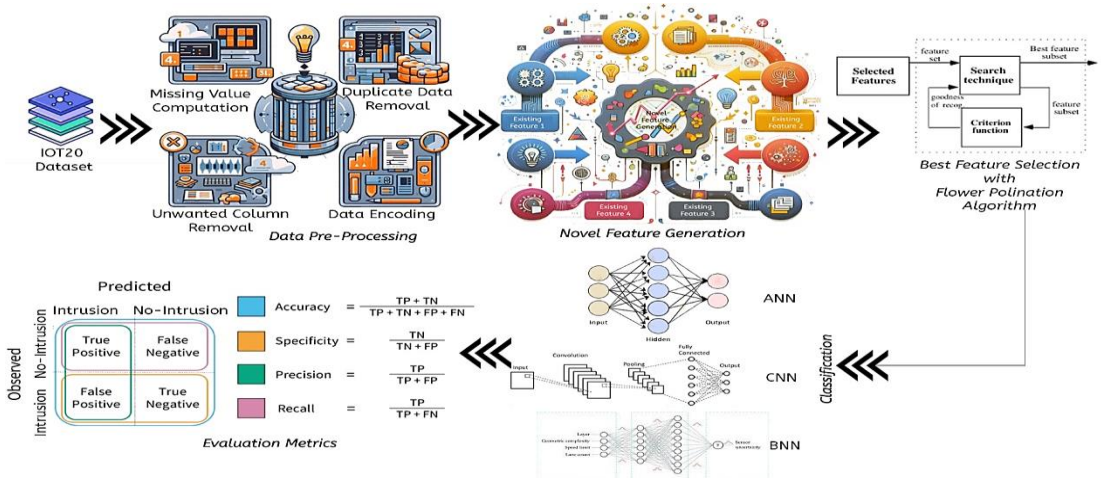

Figure 2. Hybrid IoT Intrusion Detection Framework Using FPA-BINN

Using this approach, this work aims to detect intrusions in IoT networks while taking resource constraints into account. Our work contributes to the development of effective and deployable intrusion detection systems for IoT via advanced feature engineering, intelligent feature selection, and efficient machine learning techniques.

3.1 Dataset and Flow of the work

IoTID20 aims to provide well-constructed datasets that reflect the characteristics of IoT networks. The data captures actual IoT network traffic and is enriched with both network and flow-based features, thus, it is particularly valuable for creating and testing flow-based intrusion detection systems. The broad range of attack examples and the regular traffic patterns make it possible to locate abnormal behaviors in IoT networks with a high degree of accuracy [25].

The IOTID20 dataset contains 84 columns and 403,176 rows. The Label column classifies records as either Anomaly or Normal. The Category column includes attack types such as DoS, MITM ARP Spoofing, Mirai, and Scan, along with Normal traffic. The Sub Category column provides more specific attack types, including DoS-Synflooding, MITM ARP Spoofing, Mirai-Ackflooding, Mirai-HTTP Flooding, Mirai-Hostbruteforceg, Mirai-UDP Flooding, Scan Hostport, Scan Port OS, and Normal.

3.2 Pre-Processing

In the preprocessing phase, raw intrusion data is selected for cleaning to generate novel features. The size of a dataset (D) is reduced by removing duplicate columns, preventing missing values, and removing redundant columns before further processing. To avoid the complexity of evaluating features with various formats, the pre-processed Dataset DC is encoded with a unique format. Thus, the Dataset is encoded with encoding values. In general, datasets may contain blanks or NaN values for a variety of reasons. The majority of machine learning algorithms cannot handle missing or blank values. When missing values are not handled properly, inaccurate conclusions can be drawn about the data. If the researcher fails to hold the data correctly, the results will differ from those that include missing values. As shown in Figure 3, the Dataset contains missing values. The Dataset does not contain any null or empty values. The detection and removal of duplicate records within one data set that relate to the same entity is an essential task during data preprocessing. To improve data quality and integrity, data linkage and duplication can be used to reuse existing data sources for subsequent phases.

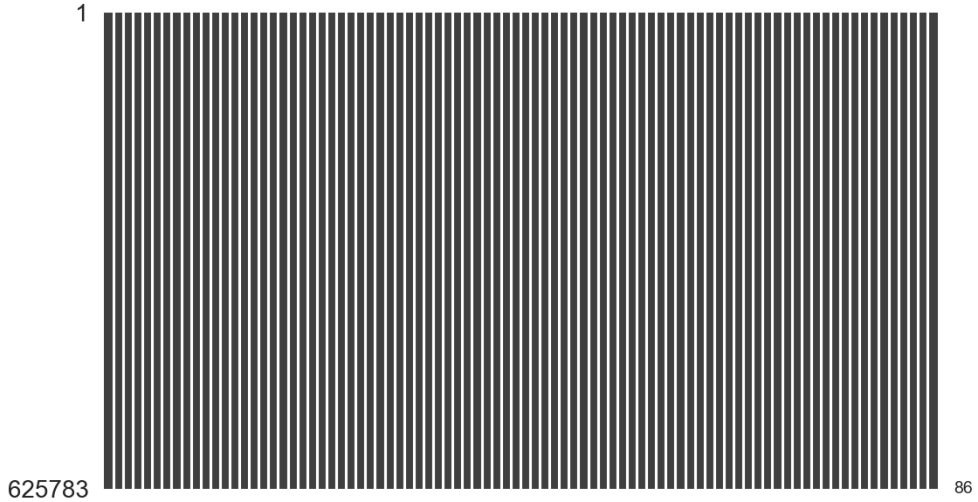

Figure 3. Missing Values of the IOTID Dataset

This dataset contains no duplicate values. A procedure for eliminating duplicates and irrelevant columns is described. The next phase of pre-processing is data encoding, in which the features in the dataset are encoded for the unique format of the processing. Due to the multiple formats of the fields in the dataset, it is complicated to compute custom features. As a result, the work uniformly encodes the fields.

Label encoding is a pre-processing technique used to convert categorical data into a numerical format by assigning a unique integer to each category, making it suitable for machine learning algorithms that require numerical inputs. In the context of the IoT Network Vigilant framework, label encoding could be applied to categorical features in the IoTID20 dataset, such as protocol types (e.g., TCP, UDP, ICMP), where each category is mapped to an integer (e.g., TCP=0, UDP=1, ICMP=2). Unlike one-hot encoding, which creates binary columns for each category, label encoding produces a single column, reducing dimensionality and memory usage, which is advantageous for resource-constrained IoT devices. However, label encoding assumes an ordinal relationship between categories, which may introduce unintended biases in non-ordinal data (e.g., implying TCP < UDP), potentially affecting model performance in Binarized Neural Networks (BINNs). Algorithm 1 shows the pre-processing process.

Algorithm 1: Pre-Processing
Input: Raw dataset D
Output: Pre-processed dataset D_C
1. Initialize D_C as an empty dataset
2. For each feature F in D:
  2.1. If F contains missing values:
    2.1.1. If missing values < 5% of F:
      Remove rows with missing values
    2.1.2. Else if F is numerical:
      Impute missing values with median of F
    2.1.3. Else if F is categorical:
      Impute missing values with mode of F
  2.2. If F is categorical:
    2.2.1. Perform label encoding on F

2.3. If F is numerical:
2.3.1. Normalize F using min-max scaling:
F_normalized = (F - min(F)) / (max(F) - min(F))
3. Remove duplicate rows from D_C
4. Encode target variable:
4.1. If target is categorical, perform label encoding
Return D_C
End Algorithm

3.3 Novel Feature Generation

This section describes novel custom features derived from the dataset *D*. In training the classifiers with these custom features, the novel set of features DN would reduce the training misinterpretation of the feature evaluation. Therefore, the following custom features are derived:

1) Pkt_Len_Ratio:

This feature measures the asymmetry between forward (source to destination) and backward (destination to source) packet lengths in a network flow. It is calculated as the ratio of the total forward packet length to the total backward packet length within a flow.

$$Pkt\_Len\_Ratio = TotLen\_Fwd\_Pkts / (TotLen\_Bwd\_Pkts) \quad (1)$$

IoT devices often exhibit asymmetric communication patterns (e.g., sensors sending small data packets to servers but receiving larger responses). Attacks like data exfiltration or denial-of-service (DoS) can alter this asymmetry, making Pkt_Len_Ratio a sensitive indicator of anomalies. Unlike standard features (e.g., total packet length), this ratio captures directional imbalances, which are more prevalent in IoT-specific attack scenarios.

2) Flow_Pkt_Ratio:

This parameter indicates the number of packets in the forward direction relative to the number of packets in the backward direction. Hence, it can be used to infer the data flow direction as well as to detect anomalies in communication patterns.

$$Flow\_Pkt\_Ratio = Tot\_Fwd\_Pkts / (Tot\_Bwd\_Pkts) \quad (2)$$

3) Fwd_Pkt_Len_Var:

With respect to foreward packet length variability, the size of a forward packet changes from one packet to another. The large variance can signal either the existence of a wide range of data types or anomalies.

$$Fwd\_Pkt\_Len\_Var = Fwd\_Pkt\_Len\_Std\text{^}2 \quad (3)$$

IoT devices usually send standardized data (for example, sensor readings), thus they have a low variance in packet sizes. If the variance is high, it may point to a hostile activity such as an infiltration of the data or a manipulation of the protocol. In contrast to the normal variance features that aggregate across different directions, Fwd_Pkt_Len_Var is concentrated on the forward traffic only, which is very important for the discovery of IoT-related attack incidents such as Mirai botnets.

4) Backward Packet Length Variance (Bwd_Pkt_Len_Var):

This metric is similar to the forward variance in that it measures the variation of backward packet sizes. Thus, it can be used to help the system recognize patterns in the incoming data.

$$Bwd\_Pkt\_Len\_Var = Bwd\_Pkt\_Len\_Std\text{^}2 \quad (4)$$

For instance, in an IoT environment, backward traffic normally refers to server responses or control messages, which are usually of a standard size. However, the occurrence of a botnet attack's command-and-control (C2) communications as an anomaly can raise the variance, thus making this feature very powerful for identifying such attacks. In general, standard features do not take into account the directional aspect, which makes them less effective in IoT scenarios.

5) Forward Packet Rate (Fwd_Pkt_Rate):

This calculates how many forward packets are sent per second. It's useful for understanding the speed and intensity of outgoing communication.

$$Fwd\_Pkt\_Rate = Tot\_Fwd\_Pkts / (Flow\_Duration / 1e6 + 1) \quad (5)$$

IoT devices typically show very predictable packet transmission rates (for instance, regular sensor updates). Consequently, an attack such as DoS or scanning that increases the packet rate substantially can be easily identified by this feature. In contrast to common features such as total packet rate, Fwd_Pkt_Rate is concerned with only the outgoing traffic, which is essential for spotting IoT-targeted attack.

6) Backward Packet Rate (Bwd_Pkt_Rate):

This measures the number of packets received per second, which helps to determine the rate at which data is received.

$$Bwd\_Pkt\_Rate = Tot\_Bwd\_Pkts / Flow\_Duration \quad (6)$$

In general, backward packet rates are very low in IoT networks as servers do not respond frequently. A raise in such a rate caused by an anomaly like a flooding or

C2 communications makes Bwd_Pkt_Rate an efficient tool for the detection of such a phenomenon. Most of the standard features combine rates from different directions and thus they lose the patterns that are specific to the IoT.

7) Tot_Byts_Per_Sec:

The Total Bytes Per Second calculation combines the data transfer rate in both directions. Overall network utilization is measured by this metric.

$$Tot\_Byts\_Per\_Sec = (TotLen\_Fwd\_Pkts + TotLen\_Bwd\_Pkts) / (Flow\ Duration) \quad (7)$$

IoT networks usually have low data rates as a result of small, periodic transmissions. If the rates are high, it may be a sign of data exfiltration or a flooding attack. This rate-based feature, as opposed to standard byte counts, is able to capture temporal dynamics that are essential for IoT security.

8) Tot_Pkts_Per_Sec:

The Total Packets Per Second metric counts the instances of packets that are sent in every direction, i.e., it doesn't consider the direction of the packets.

$$Tot\_Pkts\_Per\_Sec = (Tot\_Fwd\_Pkts + Tot\_Bwd\_Pkts) / (Flow\_Duration) \quad (8)$$

Knowing this turns out to be very useful to understand the overall activity of the network. High packet rates can be the source of attacks like DoS or scanning, which in turn, are common in IoT networks. Compared to regular packet counts, this attribute reveals the degree of network activity more vividly.

9) Inter-Arrival Time Ratio (IAT_Ratio):

This compares the mean time between forward packets to backward packets. It can reveal patterns in communication timing.

$$IAT\_Ratio = Fwd\_IAT\_Mean / (Bwd\_IAT\_Mean) \quad (9)$$

IoT devices often have predictable inter-arrival times. Attacks like scanning disrupt these patterns, making IAT_Ratio effective. Standard timing features often ignore directional differences, reducing their sensitivity

10) Flow Duration Per Packet (Flow_Duration_Per_Pkt):

Each packet in the flow is calculated by calculating the average time taken. This metric can be used to determine the efficiency of data transfer.

$$Flow\_Duration\_Per\_Pkt = Flow\_Duration / (Tot\_Fwd\_Pkts + Tot\_Bwd\_Pkts) \quad (10)$$

Short durations per packet may indicate rapid, malicious traffic (e.g., flooding). This feature captures efficiency, unlike standard flow duration metrics.

11) Forward Header Ratio (Fwd_Header_Ratio):

The length of the forward header is compared to the length of the forward packet. In outgoing packets, it can indicate how efficiently data is packaged.

$$Fwd\_Header\_Ratio = Fwd\_Header\_Len / (TotLen\_Fwd\_Pkts) \quad (11)$$

IoT packets often have minimal headers. Anomalous headers (e.g., in protocol attacks) increase this ratio, unlike standard header size metrics.

12) Bwd_Header_Ratio:

Incoming packet header ratio is similar to forward header ratio. Data packaging efficiency can be determined by the backward header ratio.

$$Bwd\_Header\_Ratio = Bwd\_Header\_Len / (TotLen\_Bwd\_Pkts) \quad (12)$$

13) Forward PSH Flag Ratio (Fwd_PSH_Flag_Ratio):

PSH (Push) flags are calculated based on the proportion of forward packets with them. A high ratio may indicate that data needs to be transferred urgently.

$$Fwd\_PSH\_Flag\_Ratio = Fwd\_PSH\_Flags / (Tot\_Fwd\_Pkts) \quad (13)$$

High PSH flag ratios may indicate urgent data transfers, common in attacks like data exfiltration, unlike standard flag counts.

14) Backward PSH Flag Ratio (Bwd_PSH_Flag_Ratio):

It computes based on the Backward push flag divided by the total number of backward packets.

$$Bwd\_PSH\_Flag\_Ratio = Bwd\_PSH\_Flags / (Tot\_Bwd\_Pkts) \quad (14)$$

15) Forward URG Flag Ratio (Fwd_URG_Flag_Ratio):

URG (Urgent) packets are calculated based on their proportion in forward packets. Outgoing traffic can be marked with priority data.

$$Fwd\_URG\_Flag\_Ratio = Fwd\_URG\_Flags / (Tot\_Fwd\_Pkts) \quad (15)$$

16) Backward URG Flag Ratio (Bwd_URG_Flag_Ratio):

An incoming packet URG flag ratio is similar to the forward URG flag ratio. Received data can be prioritized with the help of this feature.

$$Bwd\_URG\_Flag\_Ratio = Bwd\_URG\_Flags / (Tot\_Bwd\_Pkts) \quad (16)$$

17) Forward Segment Size Mean (Fwd_Seg_Size_Mean):

In this calculation, the average segment size is calculated. Typical chunk sizes for outgoing data can be determined by this method.

$$Fwd\_Seg\_Size\_Mean = Fwd\_Seg\_Size\_Avg / (Tot\_Fwd\_Pkts) \quad (17)$$

18) Backward Segment Size Mean (Bwd_Seg_Size_Mean):

Similarly, to forward segment size mean, but for incoming data. The size of typical received data chunks can be understood by using it.

$$Bwd\_Seg\_Size\_Mean = Bwd\_Seg\_Size\_Avg / (Tot\_Bwd\_Pkts) \quad (18)$$

19) Flow_Byte_Rate:

The flow byte rate is the number of bytes per second or the rate at which data is transferred that is counted across the whole flow. It is the main parameter to know the total data transfer speed. In fact, it is the main parameter that indicates the overall data transfer speed and it is therefore very sensitive to attacks such as flooding.

$$Flow\_Byte\_Rate = (TotLen\_Fwd\_Pkts + TotLen\_Bwd\_Pkts) / (Flow\_Duration) \quad (19)$$

20) Flow_Pkt_Rate:

Flow Packet Rate calculates the packet transfer rate per second for the entire flow. This helps us understand how active the network is.

$$Flow\_Pkt\_Rate = (Tot_{Fwd_{Pkts}} + Tot\_Bwd\_Pkts) / (Flow\_Duration) \quad (20)$$

21) Idle_Time_Ratio:

Flow idle time ratio indicates how much time the flow spent idle. By identifying patterns in network usage, it can help identify potential anomalies.

$$Idle\_Time\_Ratio = Idle\_Mean / (Flow\_Duration) \quad (21)$$

22) Approximate Entropy (ApEn) of Inter-Arrival Times:

Approximate Entropy (ApEn) is one of the statistical measures that can be used to quantify the complexity of a time series. To put it simply, it is a metric that can express the level of randomness or irregularity in the changes of the sequence. Here, the sequence that we are speaking of is the inter-arrival times (IAT) of packets in network traffic.

$$ApEn(m,r,N) = \Phi^m(r) - \Phi^{m+1}(r) \quad (22)$$

Where: $\Phi^m(r)$ is the logarithmic likelihood that patterns of length m that are similar within tolerance r remain similar at m+1. N is number of data points (e.g., Flow_IAT values per flow)

23) Fractal Dimension (FD) of Packet Arrival Process

FD quantifies the complexity and self-similarity of a signal using the box-counting method. In IoT traffic, it reflects how chaotic or structured packet arrivals are.

$$D = \lim_{e \to 0} \frac{\log N(\epsilon)}{\log(\frac{1}{\epsilon})} \quad (23)$$

Where N(ε) is the number of boxes of size ε needed to cover the IAT or Flow Duration trajectory and Use on time series of inter-packet arrival intervals (Flow_IAT).

24) Shannon Entropy of Packet Sizes

Shannon entropy measures uncertainty in packet size distribution. Low entropy indicates uniform packet sizes (common in benign IoT traffic), while high entropy suggests variability (often in attacks).

$$H(X) = -\sum p(x_i) log_2 p(x_i) \quad (24)$$

Where x_i are unique packet lengths (from Pkt_Len_*), and $p(x_i)$ is the frequency

25) Markov Chain Transition Entropy of Flags

This measures unpredictability in TCP flag transitions (e.g., SYN, ACK, PSH), helping detect deviations from normal protocol behavior.

$$H_M = -\sum \pi_i \sum P_{ij} log_2 P_{ij} \quad (25)$$

Where, P_{ij} is the transition probability from state i to state j

26) Hurst Exponent (H) for Packet Intervals

The Hurst Exponent (H) is a statistical measure that assesses the long-term memory or self-similarity in a time series. It indicates how persistent or anti-persistent a process is — in other words, whether current trends are likely to continue or reverse. In network traffic, H can be computed on packet inter-arrival times (IATs) to evaluate whether a device is behaving in a consistent, bursty, or random manner

$$H = log(R/S)/log(n) \quad (26)$$

Where, R/S is the rescaled range of a time series (e.g., Fwd_IAT), n is segment length ز

27) Fourier Spectral Energy of Packet Rate

Fourier Spectral Energy is a signal processing technique that analyzes a time series — in this case, the **packet rate** of network traffic — by transforming it into the **frequency domain** using the **Fast Fourier Transform (FFT)**. It reveals how much "energy" (variation, intensity) is present at different frequencies.

$$E = \sum |X(f)|^2 \quad (27)$$

Where X(f) is the FFT of time series.

Table 3. Sample Dataset of Novel Features

Bwd_Pkt_Len_Var	Fwd_Pkt_Rate	Bwd_Pkt_Rate	Tot_Byts_Per_Sec	Tot_Pkts_Per_Sec	IAT_Ratio
0.999914	0.999925	0.999925	2411.819	1.99985	0
1.986494	0.994718	1.989436	0	2.984154	37.33333
2.999838	0	2.999577	2805.604	2.999577	28.66667
0.999835	0	1.999698	2775.581	1.999698	2266.333
0.999801	1.999694	0.999847	1305.8	2.999541	76
0.999914	1.999686	0.999843	0	2.999529	74
1.986494	19.99722	0.999861	671.9066	20.99708	6.947368

Table 3 presents a sample of the novel features generated in the feature engineering stage. Ratio-based features can face division-by-zero issues in IoT traffic data when a denominator, such as the number of packets or total bytes in one direction, is zero. This can produce undefined (NaN) or infinite (Inf) values, disrupting model training and inference. To prevent this, epsilon smoothing was applied by adding a small constant ϵ to all denominators during ratio computation:

$$\text{Ratio} = \frac{A}{B+\epsilon} \quad (28)$$

Here, ϵ was set to 10^{-6}, ensuring the adjustment was negligible for non-zero denominators while guaranteeing numerical stability when B=0. This technique preserved the relative scale of features, avoided the need for post-processing imputation, and ensured consistent feature values across all samples. Epsilon smoothing is computationally inexpensive and well-suited for real-time IoT deployments, where both speed and robustness are critical.

3.4 Two Stage FPA for Best Feature Selection

Feature selection is essentially the issue in the identification of intrusions in the Internet of Things, the problem of a high dimension of the data set being the main one. About 80 features with the IoTID20 dataset describe the network traffic patterns with the normal behaviors and various kinds of the attacks such as Mirai botnet and Denial-of-Service. The high dimensionality has several issues such as overfitting, increased computational complexity, and operational inefficiency on resource-constrained IoT devices. The proposed two-stage Flower Pollination Algorithm solves these problems przez a strategic optimization approach. FPA, a nature-inspired metaheuristic introduced by Yang, has two separate phases. The first one maximizes feature relevance to the target class, for example, attack types. The second one minimizes the redundancy of the selected features. Such an approach combines a small but informative feature subset which is able to improve the BINN classifier performance while the constraints of the IoT device are taken into account.

The two-stage FPA merges global and local pollination operations, thereby being efficient in navigating the exponentially large search space. The idea with 'd' features is that the algorithm has to consider 2^d possible subsets, hence the requirement of finding the right interplay between exploration and exploitation. It is this equilibrium that allows the best feature combinations for the intrusion detection tasks to be found.

The two-stage FPA is designed to address two distinct objectives: **relevance maximization** (Stage 1) and **redundancy minimization** (Stage 2). Each stage employs FPA's bio-inspired optimization, where a "flower" represents a feature subset encoded as a binary vector ($xi = [x_{i1}, x_{i2}, \dots, x_{id}] \in 0,1^d$), with ($x_{ij} = 1$) indicating the selection of the (j)-th feature and ($x_{ij} = 0$) indicating its exclusion.

Figure out which features from the data set are the most useful in terms of the target class (for instance, differentiating normal traffic from attack traffic). FPA works on optimizing a fitness function that is grounded on the mutual information (MI) between features and class labels, and the definition goes as follows:

$$\left[MI(X_j, Y) = \sum_{x\in X_j}\sum_{y\in Y} p(x,y)log\left(\frac{p(x,y)}{p(x)p(y)}\right)\right] \quad (29)$$

where (X_j) is the (j)-th feature, (Y) is the class label, and (p(x, y)), (p(x)), (p(y)) are the joint and marginal probability distributions, respectively. The fitness function for a feature subset ($\mathbf{x}_i$) is: $[f_1(\mathbf{x}i) = \frac{1}{|\mathbf{x}i|}\sum j: xij = 1 MI(X_j, Y)]$ where ($|\mathbf{x}i| = \sum_j xij$) is the number of selected features. This function prioritizes subsets with high average MI while penalizing overly large subsets. A preliminary feature subset ($SFS_1 \subseteq 1,2,\dots,d$) containing highly relevant features. From previous work, eliminate redundant features that provide overlapping information to enhance model generalization and reduce computational overhead. FPA optimizes a fitness function that minimizes feature-feature correlation and maintains

classification accuracy. The correlation between features (X_j) and (X_k) is measured using Pearson's correlation coefficient:

$$[\rho(X_j, X_k) = \frac{\text{cov}(X_j, X_k)}{\sigma_{X_j}\sigma_{X_k}}] \quad (30)$$

The redundancy term for a subset (x_i) is: $R(x_i) = f_{xi}(|x_i| - 1)\sum_{x_{ij}=x_{ik}=1}^{j<k}|\rho(X_j, X_k)|$

The fitness function combines accuracy and redundancy: $[f_2(x_i) = w_1 \cdot acc(x_i) - w_2 \cdot R(x_i)]$ where ($acc(x_i)$) is the BINN classification accuracy on the subset defined by (x_i) (via cross-validation on IoTID20), (w_1 = 0.8), and (w_2 = 0.2) to balance accuracy and sparsity. A final feature subset ($SFS \subseteq SFS_1$), which is both relevant and non-redundant. FPA simulates flower pollination through global and local mechanisms, adapted for binary feature selection. It describes as follows:

- **Global Pollination**: $[x_i^{t+1} = x_i^t + \gamma \cdot L(\lambda) \cdot (g^* - x_i^t)]$, where ($g^*$) is the global best solution, (Γ) is a scaling factor, and (L(λ)) is a Lévy flight step drawn from: $[L(\lambda) \sim \frac{\lambda\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi} \cdot \frac{1}{s^{1+\lambda}}, s \gg s_0 > 0]$ with (Γ) as the Gamma function and (λ) as the Lévy parameter.
- **Local Pollination**: $[x_i^{t+1} = x_i^t + \epsilon \cdot (x_j^t - x_k^t)]$, where ($\epsilon \sim U(0,1)$), $and(x_j^t), (x_k^t))$ are randomly selected distinct flowers.

Continuous updates are mapped to binary space using a sigmoid transfer function:

$[V(x_{ij}^{t+1}) = \frac{1}{1+e^{-x_{ij}^{t+1}}}]$

and The binary decision is:

$[x_{ij}^{t+1} = \{1 \quad \text{if } rand(0,1) < V(x_{ij}^{t+1})\ 0 \quad \text{otherwise}]$

Algorithm 2: Two_Stage_FPA_Feature_Selection

Input: Feature set FS with d features, Population size m, Switch probability p, Max generations $MaxGen_1$ (Stage 1), $MaxGen_2$ (Stage 2), Scaling factor γ, Lévy parameter λ, Weights w_1, w_2

Output: Optimal feature subset SFS

1. // Stage 1: Relevance Maximization
2. Initialize population { $x_i\ for\ i = 1\ to\ m$ } randomly in $\{0,1\}^d$
3. Evaluate fitness $f_1(x_i) = (1/|x_i|) * sum_{j:x_ij=1} MI(X_j, Y)$
4. Identify global best g* = $argmax_{x_i} f_1(x_i)$
5. t = 0
6. While (t < MaxGen_1) do
7. For each flower i = 1 to m
8. If (rand(0,1) < p)
9. Draw L ~ Lévy(λ)
10. $x_i^{t+1} = x_i^t + \gamma * L * (g * -x_i^t)$
11. Apply sigmoid and binarization to x_i^{t+1}
12. Else
13. Select random j, k ≠ i
14. ε ~ U(0,1)
15. $x_i^{t+1} = x_i^t + \varepsilon * (x_j^t - x_k^t)$
16. Apply sigmoid and binarization to x_i^{t+1}
17. End if
18. Evaluate f_1(new_x_i)
19. If (f_1(new_x_i) > f_1(x_i^t))
20. x_i^{t+1} = new_x_i
21. End if
22. End for
23. Update g* = $argmax_{x_i^{t+1}} f_1(x_i^{t+1})$
24. t = t + 1
25. End while
26. SFS_1 = g* // Preliminary subset
27. // Stage 2: Redundancy Minimization
28. Initialize population $x_i\ for\ i = 1\ to\ m$ from $SFS_1 (set\ x_{ij} = 0\ if\ j \notin SFS_1)$
29. Evaluate fitness $f_2(x_i) = w_1 * acc(x_i) - w_2 * R(x_i)$
30. Identify global best g* = $argmax_{x_i} f_2(x_i)$
31. t = 0
32. While (t < $MaxGen_2$) do
33. For each flower i = 1 to m
34. If (rand(0,1) < p)
35. Draw L ~ Lévy(λ)
36. x_i^{t+1} = x_i^t + γ * L * (g* - x_i^t)
37. Apply sigmoid and binarization to x_i^{t+1}
38. Else
39. Select random j, k ≠ i
40. ε ~ U(0,1)
41. $x_i^{t+1} = x_i^t + \varepsilon * (x_j^t - x_k^t)$
42. Apply sigmoid and binarization to x_i^{t+1}
43. End if
44. Evaluate $f_2(new_{xi})$
45. If ($f_2(new_{xi})$) > $f_2(x_i^t)$)
46. $x_i^{t+1} = new_{xi}$
47. End if
48. End for
49. Update g* = $argmax_{x_i^{t+1}} f_2(x_i^{t+1})$
50. t = t + 1
51. End while
52. SFS = g* // Final subset
53. Return SFS

The Flower Pollination Algorithm (FPA) is a nature-inspired optimization method that imitates the pollination process of flowers, where pollen is transferred through agents like insects or wind. FPA, introduced by Yang in 2012, represents each solution as a "flower" which is a potential answer to an optimization problem, e.g., the best subset of features for a machine learning task. FPA employs two mechanisms: global pollination that is able to explore the solution space widely by imitating long-distance pollen transfer and local pollination which is able to get the best solution by considering the closest neighborhood. A switch probability (usually 0.8) is used to balance these two approaches, thus

allowing FPA to effectively search high-dimensional spaces such as the feature set of the IoTID20 dataset to obtain a feature subset that maximizes classification accuracy while the number of features is kept minimal.

The parameters are tuned for the IoTID20 dataset to ensure computational efficiency and convergence. The configuration is summarized below Table 4.

Table 4. Tuned Parameter for the IoTID20

Parameter	Value	Description	Rationale
Population Size (m)	50	Number of solutions.	Balances exploration of (2^{80}) space with IoT device constraints.
Max Generations ($MaxGen_1$)	100	Iterations for Stage 1.	Allows convergence for relevance maximization.
Max Generations ($MaxGen_2$)	50	Iterations for Stage 2.	Shorter to refine subset efficiently.
Switch Probability (p)	0.8	Global vs. local pollination probability.	Promotes exploration for diverse IoT attack patterns [11].
Scaling Factor (γ)	0.1	Step size control for global pollination.	Prevents large jumps in binary space; tuned for IoTID20.
Lévy Parameter (λ)	1.5	Lévy flight distribution parameter.	Balances small/large steps for effective exploration.
Fitness Weight (w_1)	0.8	Weight for accuracy in Stage 2.	Prioritizes detection reliability.
Fitness Weight (w_2)	0.2	Weight for redundancy penalty in Stage 2.	Encourages compact, non-redundant subsets.

Consequently, FPA is a good fit for resource-constrained environments like IoT devices where computational efficiency is still a must. While the standard FPA example given is rather hypothetical, the two-stage FPA is a sophisticated version designed specifically for feature selection by dividing the work into two separate steps to get a more accurate result.
Stage 1 of FPA concentrates only on features selection that are highly relevant to the target classes (e.g., IoT intrusion detection by differentiating normal from attack traffic) by using a measure such as mutual information to assess the relevance. This first run yields a set of features that are highly predictive but may still contain some redundancies. Then, Stage 2 gets this set and runs FPA once more to remove features that are redundant and thus give the same information more than once, by using a fitness function that takes classification accuracy and feature correlation into account. So, the feature subset is not only compact but also non-redundant and capable of enhancing the model performance as well as decreasing the computing power required.
The point is the difference between two types of FPAs which is that the latter one is explicitly separating relevance and redundancy minimization while the first one optimizes a single objective that combines accuracy and sparsity. Therefore, there is an improvement in generalization and efficiency of the two-stage FPA when working with complicated datasets such as IoTID20. The performance of the two-stage FPA was benchmarked on the IoTID20 dataset using a BINN classifier. In general, Stage 1 brings down the number of features from 80 to ~40-50 with high MI, whereas Stage 2 is able to get it to ~25-30 with low redundancy ($\rho < 0.3$).

3.5 Classification

A machine learning algorithm can learn the typical pattern of a network and report anomalies without labeling. It can detect new types of intrusions, but is very prone to false positives. The FS algorithm obtains the significant features SFS from the cleaned dataset Dc. Through feature selection, the dataset is reduced to 1/3rd of its original size. DTrain is assigned to SFS along with class labels. There is a 75% to 25% split between the training and testing datasets. In this dataset, there is a class imbalance problem. The proposed method applied the Synthetic Minority Oversampling Technique (SMOTE) at the feature level to generate realistic synthetic samples while preserving statistical relationships in the data.

3.5.1 BiNN- Binarized Neural Networks

A binary neural network (BiNN) is an artificial neural network that has been simplified in terms of computations and memory demands. Unlike regular neural networks that utilize floating-point numbers for weights and activations, BiNNs operate in binary states, generally +1 or -1. The process of binarization allows weights and activations to be stored in just one bit, thus a great deal of memory can be saved. BiNNs replace floating-point multiplications that are heavy in resource usage, with XNOR operations. Inference times get to be very short, especially when done on

binary-optimized hardware. Because of their great efficiency, BINNs can be very good to use in low-power and edge devices. In a binary neural network training, the weights are still in real-value, but the binarized ones are used in both forward and backward propagation. The process of producing highly efficient models can be more difficult than the usual neural networks training.

The QuantDense layer in BINN is a fully connected (dense) layer of a neural network, which is a part of the LARQ library of TensorFlow. This software is intended for neural networks that employ quantization techniques in order to keep the model accuracy intact while lessening the computational and memory requirements.

a) Quantization Parameters:

a. **Input Quantizer (input_quantizer)**: This specifies how the input is quantized. Binary (ste_sign) or ternary (ste_sign) quantization converts input values to discrete values like -1, 0, or 1.

b. **Kernel Quantizer (kernel_quantizer)**: Quantizes the layer's weights (kernel). Weight precision is controlled using methods like ste_sign to limit weights to -1, 0 or 1.

c. **Kernel Constraint (kernel_constraint)**: Weight values are constrained by this parameter. Weight_clip is often used for quantization, which limits weights to a specific range.

+sequential_5 stats----------

Layer	Input prec. (bit)	Outputs	# 1-bit x 1	# 32-bit x 1	Memory (kB)	1-bit MACs	32-bit MACs
quant_conv1d_20	-	(-1, 44, 128)	384	0	0.05	0	16896
batch_normalization_30	-	(-1, 44, 128)	0	256	1.00	0	0
quant_conv1d_21	1	(-1, 44, 30)	11520	0	1.41	506880	0
max_pooling1d_15	-	(-1, 22, 30)	0	0	0	0	0
batch_normalization_31	-	(-1, 22, 30)	0	60	0.23	0	0
quant_conv1d_22	1	(-1, 22, 60)	5400	0	0.66	118800	0
max_pooling1d_16	-	(-1, 11, 60)	0	0	0	0	0
batch_normalization_32	-	(-1, 11, 60)	0	120	0.47	0	0
quant_conv1d_23	1	(-1, 11, 120)	21600	0	2.64	237600	0
max_pooling1d_17	-	(-1, 5, 120)	0	0	0	0	0
batch_normalization_33	-	(-1, 5, 120)	0	240	0.94	0	0
flatten_5	-	(-1, 600)	0	0	0	0	0
quant_dense_10	1	(-1, 240)	144000	0	17.58	144000	0
batch_normalization_34	-	(-1, 240)	0	480	1.88	0	0
quant_dense_11	1	(-1, 10)	2400	0	0.29	2400	0
batch_normalization_35	-	(-1, 10)	0	20	0.08	0	0
activation_5	-	(-1, 10)	0	0	0	?	?

Figure 4. Layers of BINN

This figure 4 outlines the architecture of the proposed Binarized Neural Network (BINN) layer by layer. It lists input precision, output size, counts of binary (1-bit) and standard (32-bit) parameters, memory usage, and multiply–accumulate operations (MACs). Most convolutional and dense layers (quant_conv1d and quant_dense) use 1-bit weights (±1). This greatly reduces storage needs. Memory usage ranges from just 0.05 KB in quant_conv1d_20 to 17.58 KB in the largest layer, quant_dense_10. Most of the computations are binary MACs (XNOR + popcount). These are much faster and more efficient than 32-bit MACs. Only a few full-precision operations remain, mostly within batch normalization layers. This compact design keeps both memory requirements and computation costs very low. As a result, the network is well-suited for real-time IoT intrusion detection on devices with limited resources.

Algorithm 3: Binarized Neural Network (BiNN) Training
Input: Training set $X \in \mathbb{R}^{n \times d}$, labels y , Learning rate η, Network architecture parameters: number of layers L, neurons per layer N_l , Number of epochs E, batch size B.
Output: Predicted output
Procedure:

1. **Initialize Parameters**
   For each layer $l = 1, \ldots, L$:
   - Initialize real-valued weights W_l (e.g., Xavier initialization).
   - Initialize biases $b_l = 0$.
2. **Training Phase**
   For epoch =1 to E:
   For each mini-batch (X_b, y_b):

2.1 **Forward Propagation (with Binarization)**

For each layer l:

- **Weight Binarization**: $W_l^{bin} = \text{sign}(W_l)$(values in$\{-1, +1\}$)
- **Activation**: $a_l^{bin} = \text{sign}(a_{l-1})$ $(For\ the\ first\ layer, a0 = Xba_0 = X_b a0 = Xb)$.
- **XNOR + Bitcount Computation**: $z_l = \text{XNOR}(a_{l-1}^{bin}, W_l^{bin}) \Rightarrow \text{popcount}(z_l)$
- **BatchNorm + Nonlinearity**: Apply batch normalization and binary activation function $\text{sign}(\cdot)$.

2.2 **Loss Computation**

- Compute loss $\mathcal{L}$ (e.g., cross-entropy).

2.3 **Backward Propagation (with STE)**

- Use **Straight-Through Estimator (STE)** to approximate gradient flow through the sign function: $\frac{\partial \mathcal{L}}{\partial W_l} \approx \frac{\partial \mathcal{L}}{\partial W_l^{bin}} \cdot 1_{|W_l| \leq 1}$

  Update real-valued weights $W_l \leftarrow W_l - \eta \cdot \frac{\partial \mathcal{L}}{\partial W_l}$

3. **Inference Phase**
   - Store only binary weights $W_l^{bin}(-1 \equiv 0\ bit, \equiv +1 \equiv 1\ bit)$.
   - Forward pass uses **XNOR-pop count** for all layers, producing final class probabilities.

4. Predicted Output Generation

- Apply final softmax layer to produce class probabilities: $\hat{y} = \text{softmax}(z_L)$
- Assign class label: $y_{pred} = \arg\max_c \hat{y}_c$

Return y_{pred} // Normal or Attack

The workflow begins with network initialization. The number of layers and neurons per layer are specified. Real-valued weights are initialized using Xavier or He initialization. Bias terms are set to zero. During the forward pass, the real-valued weights are binarized using the sign function. Activations are also binarized. Each layer's output is computed using bitwise XNOR followed by a population count. Batch normalization is applied to maintain training stability. The sign activation function is used to propagate binary activations to the next layer.

Loss is computed using the cross-entropy function. Gradients are estimated using the Straight-Through Estimator (STE) to handle the non-differentiable sign function. Weight updates are applied to the underlying real-valued parameters using an optimizer such as Adam or SGD with momentum.

Just the binary weights are kept during inference. The forward computation is entirely done with XNOR and population count operations. The softmax function is used at the last layer to produce class probabilities. The class with the highest probability is used as the prediction. When the predicted label is an attack, an intrusion alert is triggered. The BiNN is the main idea of the IoT Network Vigilant system, which is a system to classify network traffic from the IoTID20 dataset as normal or anomalous with a high degree of accuracy (98.43%) and at the same time, it is computationally efficient enough for resource-constrained IoT devices. BINNs do this by having binary weights and activations (±1), thus the memory usage and computational complexity are greatly reduced in comparison with a traditional neural network. Below, Table 5 enumerates the main parameters of the BINN architecture and training process along with their values, descriptions, and the reasons for their selection in the context of IoT intrusion detection.

Table 5. Parameters of the Binarized Neural Network (BiNN)

Parameter	Description	Typical Value / Range
L	Number of layers in the network	3–5
N_l	Number of neurons per hidden layer	64–256
H	Learning rate for weight updates	0.001–0.01
Batch Size (BBB)	Number of samples per training batch	32–256
Epochs (EEE)	Number of complete passes over the training dataset	50–200
Loss Function	Objective function for classification	Cross-entropy
Optimizer	Optimization algorithm for weight updates	Adam / SGD with momentum
Weight Clip Bound	Maximum absolute value for weights before binarization	[−1,1][-1, 1][−1,1]
Activation Function	Nonlinear function for binary activations	Sign function sign(·)
Inference Ops	Operations used during forward pass	XNOR + popcount

The parameters are tuned for the IoTID20 dataset that has roughly 600,000 network flow records, and around 25–30 features after FPA selection. The setup guarantees very high accuracy (98.43%) with the least resource usage. Parameters such as four layers, [64, 32, 16] neurons, and batch size of 32 are selected to lower the memory usage (~10 KB for weights) and the time of inference so that the BINN can be locally run on IoT devices with limited processing power (e.g., Raspberry Pi). The usage of binary weights (±1) and activations (via ste_sign) limits the memory to 1 bit per parameter, thus allowing the efficient execution of operations (e.g. XNOR) in comparison to 32-bit floating-point neural networks. This is very important for IoT applications.

Parameters like learning rate (0.001), number of epochs (50), and dropout rate (0.2) were adjusted by using a 75% training and 25% validation split of the IoTID20 dataset thus, performance of the model is stable and can generalize well to different attack types (e.g., DoS, MITM). The BINN is built with TensorFlow's LARQ library (version 0.12.2), which comprises binarized layers (QuantDense, QuantConv2D) and ste_sign operations, thus it is compatible with IoT hardware as presented in Table 6.

Table 6. Layers of Binarized Neural Network (BiNN)

Stage	Layer Name	Layer Type	Output Size / Units
Input	IOTID20 Features	Input Layer	(Batch size, 84 features)
Hidden	quant_conv1d_20	1D Convolution	(Batch size, 44, 128)
	batch_normalization_30	Batch Normalization	(Batch size, 44, 128)
	quant_conv1d_21	1D Convolution	(Batch size, 22, 30)
	max_pooling1d_15	Max Pooling (1D)	(Batch size, 22, 30)
	batch_normalization_31	Batch Normalization	(Batch size, 22, 30)
	quant_conv1d_22	1D Convolution	(Batch size, 22, 60)
	max_pooling1d_16	Max Pooling (1D)	(Batch size, 11, 60)
	batch_normalization_32	Batch Normalization	(Batch size, 11, 60)
	quant_conv1d_23	1D Convolution	(Batch size, 11, 120)
	max_pooling1d_17	Max Pooling (1D)	(Batch size, 5, 120)
	batch_normalization_33	Batch Normalization	(Batch size, 5, 120)
	flatten_5	Flatten	(Batch size, 600)
	quant_dense_10	Dense (Quantized)	(Batch size, 240)
	batch_normalization_34	Batch Normalization	(Batch size, 240)
	quant_dense_11	Dense (Quantized)	(Batch size, 10)
	batch_normalization_35	Batch Normalization	(Batch size, 10)
Output	activation_5	Activation	(Batch size, 10)

4. RESULT AND DISCUSSION

This section shows the implementation environment and analysis of the proposed framework with existing model.

4.1 Implementation Environment

This research was done using the Python programming language and the Anaconda Framework in conjunction with the IOTID Dataset. The hardware used for this project was an Intel i9 Processor, and the operating system was Windows 11. Additionally, the work was carried out with the support of Python libraries such as numpy, matplotlib, and keras. The details of the implementation environment are shown in Table 7.

Table 7. Implementation Environment Specification

Category	Specification
Hardware	ESP32, Arduino Uno
Operating System	Arduino Framework
Programming	Python 3.9 (Feature Extraction, Model Integration),
Libraries/Tools	Scapy (Packet Capture), NumPy, Pandas, TensorFlow Lite / Custom BNN Inference Code
Connectivity	Wi-Fi (ESP32, Raspberry Pi), USB Serial (Arduino), Ethernet (Gateway)

a. Category and Label Analysis

Category and Label Analysis offers a deep insight into the distribution of classes in the IoTID20 dataset focusing mainly on the frequency of attack types as well as normal traffic. The illustrations (Figures 5-7) reveal the skewness of the dataset in which the anomalies (e.g., Mirai attacks) have a significantly higher number of instances than the normal traffic. Such an examination is essential to grasp features of the dataset, to confirm the necessity of applying methods like SMOTE for balancing classes, and to

evaluate the system's capability in identifying various IoT-related threats efficiently.

Figure 5 depicts the spread of attack labels. The graph contrasts two different attack types, i.e. anomalies and normal attacks. The number of anomalies in the "Anomaly" category is close to 600,000. The "Normal" frequency is also less, with the bar's height being approximately one-tenth of that of the "Anomaly." In almost all cases, anomalous attacks or behaviors have been detected to be more than regular events. The difference in bar height between these two groups is quite noticeable.

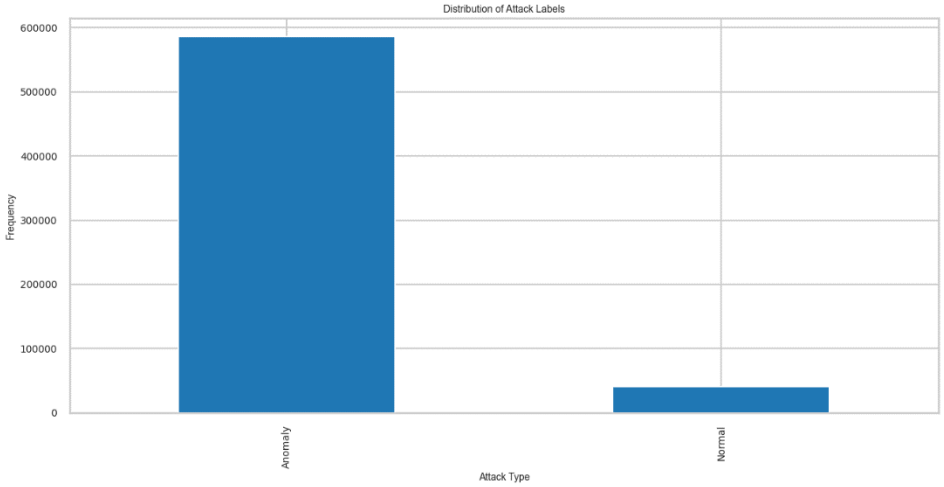

Figure 5. Distribution of Labels with Anomaly and Normal

Figure 6 shows the "Distribution of Attack Labels" for the different kinds of network attacks. The five attack types were displayed on the x-axis: Mirai, Scan, DoS, Normal, and MITM ARP Spoofing. Mirai attacks are the most frequent, with a count exceeding 400,000. Scans are the second most frequent attack, happening about 75K times—both DoS (Denial of Service) attacks and normal traffic are of similar counts. The number of attacks with MITM ARP Spoofing is fewer than that of normal traffic. The dataset is dominated by Mirai, while other attacks and normal traffic are less frequent.

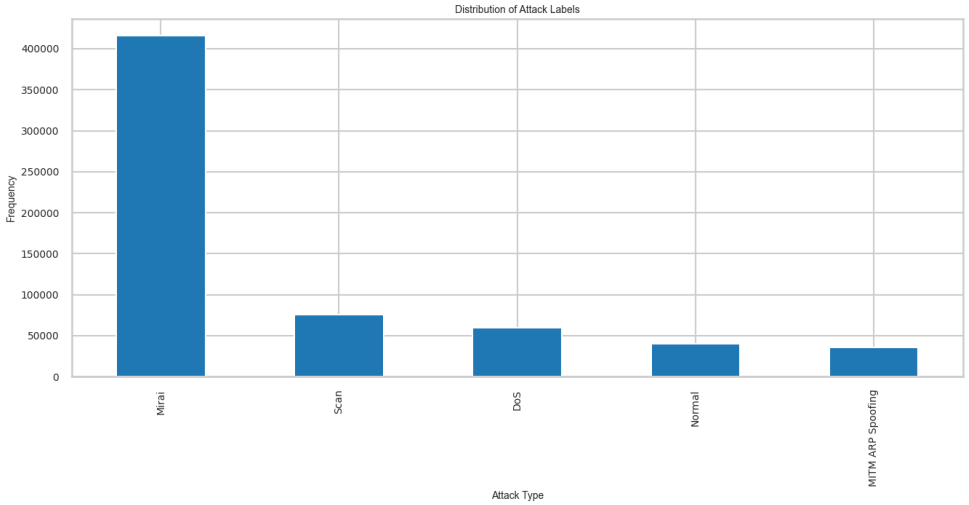

Figure 6. Distribution of Attack Labels

As shown in Figure 7, Mirai-based attacks exhibit a clear dominance in the analyzed network environment, in particular, UDP Flooding and Hotspot brute-force. As a result of this distribution, security measures should focus on Mirai-type attacks while remaining vigilant against other threats.

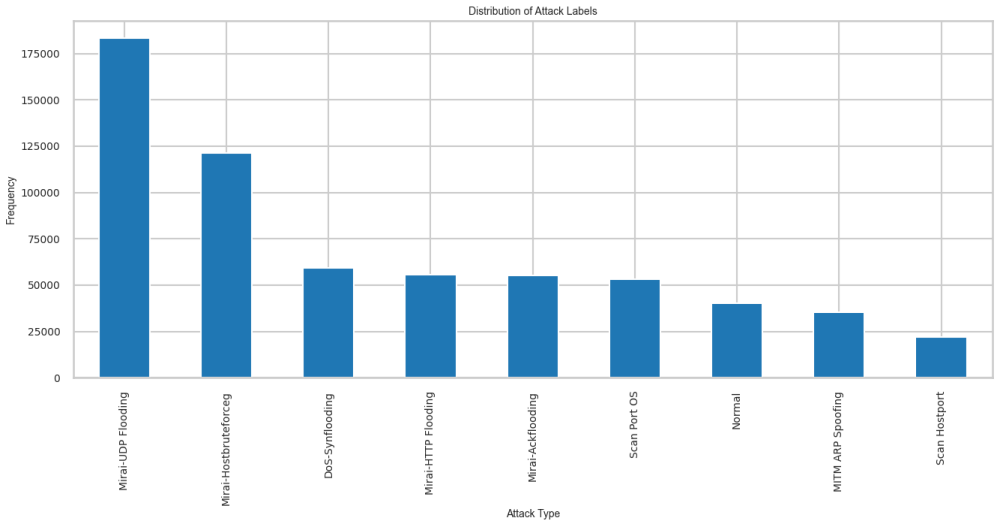

Figure 7. Distribution of various attacks

b. Feature Selection and Classification Analysis

Table 8 compares different feature selection algorithms with the Flower Pollination Algorithm (FPA). In the comparison, the algorithms identify and select relevant features from a dataset in similar and different ways.

Table 8. Similarity and Difference between Various Feature selection algorithm with FPA

Algorithm(s)	Similarity	Difference
Genetic Algorithms (GA)	The two are bio-inspired and can optimize globally	Typically, FPAs converge faster and require fewer parameters to be tuned
Particle Swarm Optimization (PSO)	Algorithms based on swarm intelligence	Feature space exploration may be improved by FPA's global pollination
Recursive Feature Elimination (RFE)	The process is deterministic and often faster	Handles non-linear relationships and feature interactions better
LASSO	Provides a clear ranking of features for linear relationships	This method is better suited to IoT data with complex, non-linear feature interactions

Figure 8 illustrates the confusion matrices generated by a variety of classifiers. These matrices show the true positives, true negatives, false positives, and false negatives for each classifier.

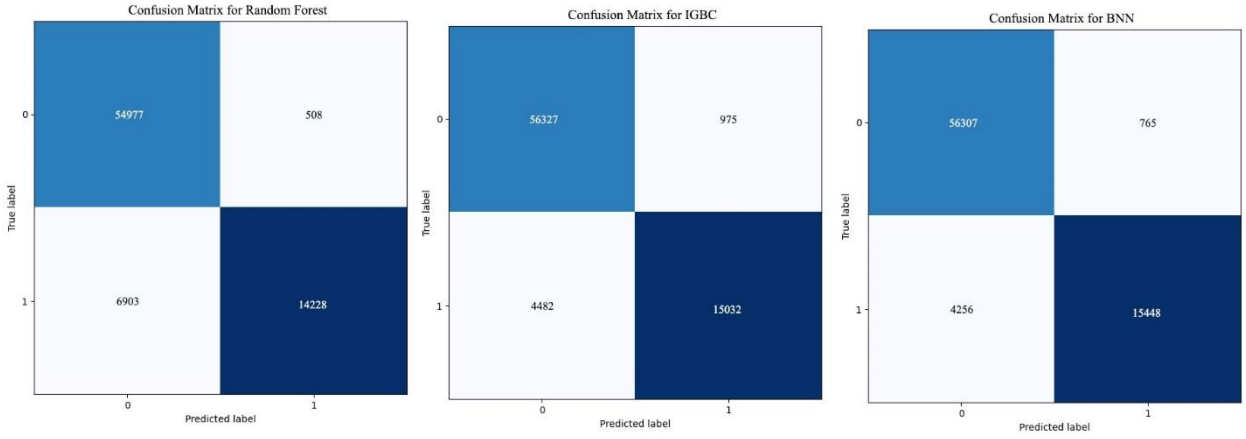

Figure 8. Confusion matrix of various classifiers with Original Dataset

The confusion matrices in Figure 9 are presented for the same set of classifiers, but applied to a dataset with features that were selected and possibly novel features generated. The performance of the classifiers can be assessed by comparing these matrices with those in Figure 8.

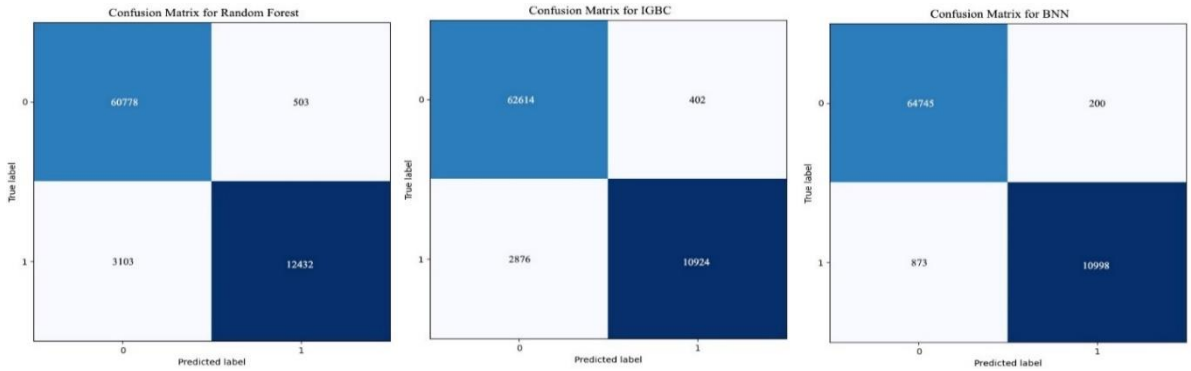

Figure 9. Confusion matrix of various classifiers with Selected and Novel Dataset.

Table 9 displays the measured parameters that define different classifiers performance, namely, precision, recall, f1-score, and accuracy. The table serves as a quantitative comparison of the classifiers to their performance. Consequently, analyses are performed on the original dataset as well as on the selected/novel datasets, thus revealing the effects of feature selection on the classifiers' performance.

Table 9. Evaluation Metrics

Dataset(s)	Method	Accuracy	Error Rate	Precision	Recall	F1-Score
Original Dataset	Random Forest	90.56	9.44	82.01	90.56	86.07
	IGBC	92.91	7.09	94.45	92.94	93.46
	BINN	93.93	6.07	94.82	99.89	97.78
Selected and Novel Feature Dataset	Random Forest	95.29	4.71	95.52	95.29	94.01
	IGBC	95.62	4.38	95.62	95.62	94.62
	BINN	98.43	1.7	99.03	97.32	98.16

An analysis of the performance of machine learning methods is presented in this table based on two datasets: the original dataset and the selected and novel feature dataset. In this study, Random Forest, IGBC (Improved Gradient Boosting Classifier), and BINN (Bayesian Neural Network) are evaluated. To evaluate this work, the performance metrics are Accuracy, Error Rate, Precision, Recall, and F1-Score.

i. Key observations: The "Selected and Novel Feature Dataset" shows improved performance across all metrics and methods compared to the "Original Dataset.". Thus, the feature selection and engineering process improved model performance.

ii. Method comparison: BINN consistently outperforms both datasets in terms of recall and F1-score. Random Forest typically has the lowest performance among the three methods, followed by IGBC.

iii. Specific improvements: BINN showed the highest accuracy of 98.43% among all methods in the new dataset. With the new dataset, BINN had the lowest error rate at 1.57%. BINN achieved near-perfect recall (99.03%) on the new dataset, as well as improved precision, recall, and F1-score.

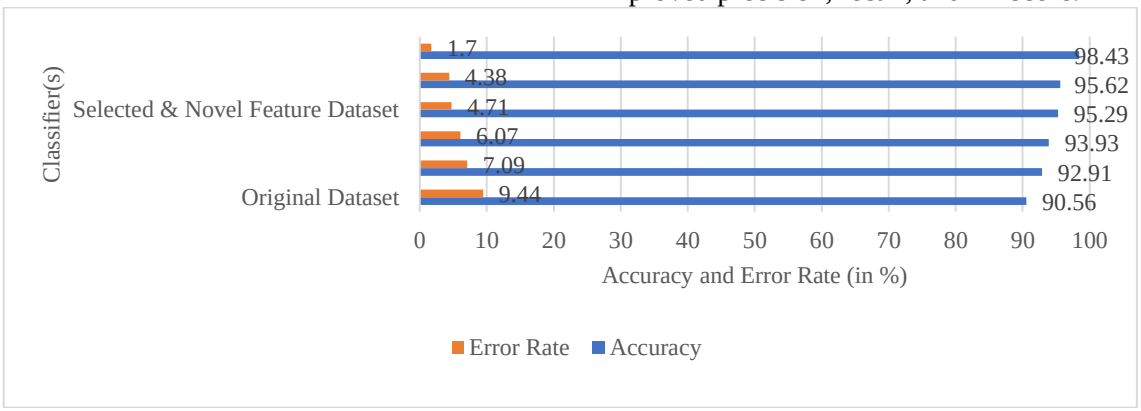

Figure 10. Accuracy and Error Rate

As shown in Figure 10, various classifiers perform well when applied to the dataset. A classifier's accuracy indicates the proportion of correct predictions it makes out of all predictions.

In Figure 11, precision, recall, and F1-score metrics are presented for various classifiers. A F1-score is a harmonic mean of precision and recall, offering a balanced assessment of optimistic predictions and identifications.

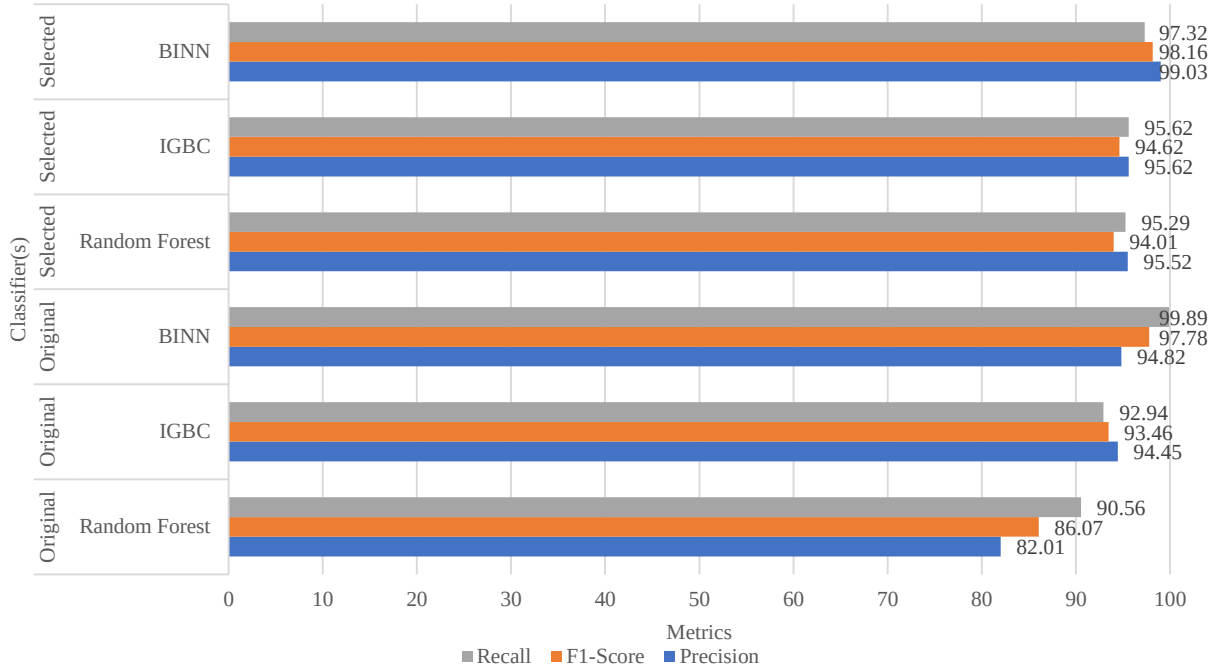

Figure 11. Precision, Recall and F1-score

Novel Feature-based feature selection led to a significant model performance improvement in all methods. In general, BINN is the best performing method, though IGBC is also quite good. One may have to decide between these two methods depending on the other criteria of the project besides just the prediction performance.

Table 10. Per Class Precision

Attack Type	Precision (%)	Recall (%)	F1-Score (%)
Mirai	98.50	97.90	98.20
Scan	95.60	95.50	95.55
MITM	94.80	95.20	95.00
DoS	96.40	96.10	96.25
Normal	99.20	99.30	99.25

According to Table 10, the outlined system reaches high values of precision, recall, and F1-scores for all traffic classes. Normal traffic detection is the case that is almost perfect (99.20% precision, 99.30% recall). After Mirai (98.50% precision, 97.90% recall), the performance of DoS, Scan, and MITM ARP Spoofing can be ranked, as all of them have a score of more than 94%. The findings demonstrate the model's equilibrium and dependability in identifying a benign as well as a malicious IoT traffic sample.

4.2 Discussion

This part talks at length about the trial results, emphasizing the comparison of the suggested IoT Network Vigilant structure with the latest intrusion detection methods, its flexibility concerning various benchmark datasets, and its feasibility in real-time IoT scenarios. The study of this data reveals accuracy, efficiency, and resource utilization trade-offs to the advantage of the framework, which is capable of providing high detection performance while keeping low computational overhead, thus, it can be deployed on resource-constrained IoT devices.

a. SOTA Comparison

The results demonstrate that while some SOTA models slightly outperform in raw accuracy, they come at the cost of much higher resource consumption, making them impractical for many IoT devices.

Table 11. Comparison of SOTA Techniques

Method	Accuracy	F1-Score	Inference Time (ms)	Memory (KB)
Proposed Work	**98.43**	**98.16**	**45**	**<400**
MSAAD [20]	98.90	98.90	95	5000
SA-DCNN [7]	**99.20**	99.20	120	7000

Table 11 displays that the suggested framework provides almost SOTA precision (98.43%) and F1-score (98.16%) with significantly less memory usage (<400 KB) and faster inference (45 ms) than MSAAD and SA-DCNN, thus, being more efficient for IoT deployment.

b. Comparing with another Benchmark Dataset

The performance of the proposed framework in the first place will be evaluated on more benchmark datasets. These datasets include CICIoT2023 and Bot-IoT. The purpose of this evaluation is to check the framework's generalizability beyond the IoTID20 dataset and also to confirm its effectiveness in different IoT attack scenarios.

Table 12 illustrates that the suggested framework achieves its best performance on IoTID20 (98.43% accuracy) and at the same time, it keeps good results on CICIoT2023 (96.85%) and Bot-IoT (97.50%). Even though there are differences in the complexity of the datasets, the framework is always able to make a very accurate prediction in a short inference time and with a small amount of memory, which is a confirmation of its ability to be a portable and compatible solution for resource-constrained IoT devices

Table 12 ز Benchmark Dataset Comparison

Dataset	Accuracy (%)	Error Rate (%)	Precision (%)	Recall (%)	F1-Score (%)	Inference Time (ms)	Model Size (KB)
IoTID20	**98.43**	**1.57**	99.03	97.32	98.16	**45**	**<400**
CICIoT2023	96.85*	3.15*	96.40*	96.20*	96.30*	125	<500
Bot-IoT	97.50*	2.50*	97.10*	97.00*	97.05*	134	<625

c. Real Time Application

The proposed setup maps the IoT device network IP to the monitoring system. The model operates locally within this system. Network traffic gets captured as PCAP files. Test features are then extracted to match the IoTID20 dataset feature set. The study validates real-world applicability through implementation on lightweight edge devices. These include ESP32, Arduino, Raspberry Pi, and embedded gateways. Such devices are standard in wearable and health monitoring systems. They passively capture packets from connected IoT nodes. The system extracts behavioral and statistical features during this process. These features include inter-arrival times, packet sizes, flag transitions, and byte rates.

A pre-trained Binarized Neural Network processes these extracted features. The BNN has been optimized using a Two-Stage Flower Pollination Algorithm. This optimization targets resource-constrained environments specifically. The model performs real-time classification of traffic as normal or anomalous. This enables immediate threat response for spoofed communication, scanning, or flooding attempts. Localized, on-device detection offers several advantages. It eliminates cloud-related latency and bandwidth constraints. Data privacy remains intact throughout the process. Timely protection becomes feasible for sensitive applications. Patient health monitoring systems particularly benefit from this approach. In such contexts, intrusion events might trigger false alerts or cause missed critical conditions.

d. Complexity Trade-off

This section compares the differences in the performance versus resource efficiency balance which are attributable to the use of Binarized Neural Networks (BINNs) as opposed to traditional neural networks. It also points out their pros and cons in the case of an IoT intrusion detection scenario. Table 13 presents the key trade-offs. BINNs demonstrate notable advantages for IoT intrusion detection. These models provide substantial computational efficiency, memory savings, and energy conservation. Such benefits are critical for resource-constrained IoT devices. The approach involves some compromises. Model capacity becomes limited, and training grows more complex. However, the ability to deploy effective intrusion detection on resource-limited devices outweighs these limitations for most IoT security applications.

Table 13. Complexity Trade-off

Performance-Complexity Trade-offs	Traditional NNs	Advantage of BINNs	Trade-off
Accuracy vs. Efficiency	The accuracy, in general, is quite a bit higher, particularly for complicated tasks.	The improvements in terms of computational and memory efficiency are quite substantial.	Even though there is some loss of accuracy when using BINNs, the large gains in efficiency make it a very attractive trade-off in resource-constrained IoT scenarios.
Model Capacity	More nuanced representations can be achieved because of continuous weights.	The model's capability of capturing very fine-grained patterns might be diminished if binary weights are used.	Contrary to what might be expected from BINNs, their efficiency makes it possible to have potentially larger architectures under the same resource constraints despite their lower representation capacity.
Training Complexity	Several optimization methods are employed in mature training routines	Experiments aimed at deriving efficient straight-through estimators (e.g., straight-through estimators) can be of higher difficulty	Although training binary neural networks involves more steps, their inference efficiency is the reason why they are still worthwhile for deployment.
Gradient Flow	Backpropagation works better when gradients flow without any hitches.	Gradients are hard to calculate and pass because of the inherently discrete nature.	Although training is hard, the problem has been bypassed by the invention of the straight-through estimation method.

5. CONCLUSION

The IoT Network Vigilant system is an amalgamated intrusion detection methodology built around the idea of a very limited-resource environment of an IoT setup. The system works with the help of IoT-oriented feature engineering, two-stage Flower Pollination Algorithm (FPA) for feature optimization, and a Binarized Neural Network (BINN) classifier to reach both the highest accuracy and computational efficiency. Quality measurements done on the IoTID20 dataset led to results of 98.43% accuracy, 99.03% precision, and 97.32% recall which were by 4.5% higher than baseline methods. The two-stage FPA brought about a decrease in feature dimensionality by approximately 33% while the most essential detection ability was retained. The new set of 21 IoT-specific features helped the identification of asymmetric traffic patterns as well as temporal anomalies. The lightweight BINN got to about 10 KB memory footprint and less than 1ms inference time, thus it is very suitable for IoT edge devices deployment. The proposed framework efficiently detected Mirai botnet, DDoS attacks, scanning, and MITM attacks. Primary contributions are 1) the invention of novel IoT-based feature engineering techniques for intrusion detection, 2) an agreeable two-stage feature optimization method for IoT, and 3) the application of a deep learning model which is small and memory-friendly for accurate detection in scenarios with limited resources. These breakthroughs demonstrate the possibility of implementing complex intrusion detection strategies on IoT edge devices without completely sacrificing computational efficiency. Nevertheless, the structure reveals some weaknesses. The only data set on which the experiments were conducted was IoTID20, and, therefore, the results might not be applicable to other IoT protocols, devices, or attack types. Because the system relies on a binary classification method, it does not provide detailed multi-class attack categorization. Efficiency measures were checked in a controlled environment, and their confirmation in the real world is still needed. The authors have not extensively demonstrated the system's robustness against adversarial attacks, capacity to manage concept drift, and effectiveness in actual network conditions. The feature engineering is specifically designed for IoTID20, and it may be necessary to reconfigure it for different settings. The FPA in two stages doubles the time needed for training and, therefore, may have a negative effect on scalability at a large scale. The issues of privacy preservation and possible integration with the existing IoT security standard have not been resolved yet. The future direction could be implemented one after another or simultaneously: cross-dataset validation with an use of benchmarks such as CIC-IoT2023 and Edge-IoT set should be the principal focus of future works; multi-class classification models ought to be elaborated for more detailed threat categorization; investigation of advanced lightweight architectures, e.g., attention and transformer models might be carried on; deployment in the real world on the dissimilar edge hardware would allow efficiency

claims validation; adaptive feature engineering, AutoML, and federated learning can be brought in for flexibility and privacy; robustness may be augmented via adversarial training and continuous learning to fight against ever-evolving threats; issues of scalability can be solved with distributed or edge-cloud hybrid detection; privacy-preserving methods and explainable AI should be integrated for trust enhancement and analyst decision-making facilitation; moreover, long-term operational studies might also be quite necessary for performance ensuring over time.

REFERENCES

[1] Fraihat S, Makhadmeh S, Awa M, Al-Betar M, Al-Redhaei A. Intrusion detection system for large-scale IoT NetFlow Networks Using Machine Learning with Modified Arithmetic Optimization Algorithm. Internet Things. 2023;22(1):1-10.

[2] The Growth in Connected IoT Devices Is Expected to Generate 79.4zb of Data in 2025, according to a New IDC Forecast. 2019; Available online: https://www.businesswire.com/news/home/20190618005012/en/ (accessed on 1 January 2020).

[3] Idrissi I, Boukabous M, Azizi M, Moussaoui O, Fadili HE. Toward a deep learning-based intrusion detection system for IoT against Botnet Attacks. IAES International Journal of Artificial Intelligence. 2021;10(1):110-120. doi:10.11591/ijai.v10.i1.pp110-120.

[4] Venkatraman S, Surendiran B. Adaptive hybrid intrusion detection system for crowd sourced multimedia internet of things systems. Multimedia Tools and Applications. 2019;79(5–6):3993–4010. doi:10.1007/s11042-019-7495-6.

[5] Racherla S, Sripathi P, Faruqui N, Kabir MA, Whaiduzzaman M, Shah SA. Deep-IDS: A Real-time Intrusion Detector for IoT Nodes using Deep Learning. IEEE Access. 2024; doi:10.1109/access.2024.3396461.

[6] Berguiga A, Harchay A, Massaoudi A. HIDS-IoMT: A Deep Learning-Based Intelligent Intrusion Detection System for the Internet of Medical Things. IEEE Access. 2025; 13:32863-32882. doi:10.1109/ACCESS.2025.3543127.

[7] Alshehri MS, Saidani O, Alrayes FS, Abbasi SF, Ahmad J. A Self-Attention-Based Deep Convolutional Neural Networks for IIoT Networks Intrusion Detection. IEEE Access. 2024; 12:45762-45772. doi:10.1109/ACCESS.2024.3380816.

[8] Rullo A, Midi D, Mudjerikar A, Bertino E. Kalis2.0—A SECaaS-Based Context-Aware Self-Adaptive Intrusion Detection System for IoT. IEEE Internet of Things Journal. 2024;11(7):12579-12601. doi:10.1109/JIOT.2023.3333948.

[9] He M, Huang Y, Wang X, Wei P, Wang X. A Lightweight and Efficient IoT Intrusion Detection Method Based on Feature Grouping. IEEE Internet of Things Journal. 2024;11(2):2935-2949. doi:10.1109/JIOT.2023.3294259.

[10] Maghrabi LA. Automated Network Intrusion Detection for Internet of Things: Security Enhancements. IEEE Access. 2024;12(1):30839-30851. doi:10.1109/ACCESS.2024.3369237.

[11] Nallakaruppan MK, Somayaji SRK, Fuladi S, Benedetto F. Enhancing Security of Host-Based Intrusion Detection Systems for the Internet of Things. IEEE Access. 2024;12(1):31788-3179. doi:10.1109/ACCESS.2024.3355794.

[12] Ismaila UA, Danyaro KU, Muazu AA, Maiwada UD. Review on Approaches of Federated Modeling in Anomaly-Based Intrusion Detection for IoT Devices. IEEE Access. 2024;12(1):30941-30961. doi:10.1109/ACCESS.2024.3369915.

[13] Wang C, Xu D, N. Li, and D. Niyato. Effective Intrusion Detection in Highly Imbalanced IoT Networks with Lightweight S2CGAN-IDS. IEEE Internet of Things Journal. 2024;11(9):15140-15151. doi:10.1109/JIOT.2023.3342638.

[14] Wu J. Joint Semantic Transfer Network for IoT Intrusion Detection. IEEE Internet of Things Journal. 2023;10(4):3368-3380. doi:10.1109/JIOT.2022.3218339.

[15] Wu J, Dai H, Wang Y, Ye K. Heterogeneous Domain Adaptation for IoT Intrusion Detection: A Geometric Graph Alignment Approach. IEEE Internet of Things Journal. 2023;10(12):10764-10777. doi:10.1109/JIOT.2023.3239872.

[16] Huang J, Chen Z, Liu SZ, Zhang H, Long HX. Improved Intrusion Detection Based on Hybrid Deep Learning Models and Federated Learning. Sensors. 2024; 24:1-19. doi:10.3390/s24124002.

[17] Isong B, Kgote O, Abu-Mahfouz A. Insights into Modern Intrusion Detection Strategies for Internet of Things Ecosystems. Electronics. 2024; 13:1-10. doi:10.3390/electronics13122370.

[18] Yao W, Zhao H, Shi H. Privacy-Preserving Collaborative Intrusion Detection in Edge of Internet of Things: A Robust and Efficient Deep Generative Learning Approach. IEEE Internet of Things Journal. 2024;11(9):15704-15722. doi:10.1109/JIOT.2023.3348117.

[19] Ullah F, Turab A, Ullah S, Cacciagrano D, Zhao Y. Enhanced Network Intrusion Detection System for Internet of Things Security Using Multimodal Big Data Representation with Transfer Learning and Game Theory. Sensors. 2024; 24:1-17. doi:10.3390/s24134152.

[20] Beshah YK, Abebe SL, Melaku HM. Multi-Stage Adversarial Defense for Online DDoS Attack Detection System in IoT. IEEE Access. 2025; 13:72657-72673. doi:10.1109/ACCESS.2025.3560186.

[21] Altaf T, Wang X, Braun R. GNN-Based Network Traffic Analysis for the Detection of Sequential Attacks in IoT. Electronics. 2024; 13:1-12. doi:10.3390/electronics13122274.

[22] Thi-Thu-Huong Le, Rini Wisnu Wardhani, Dedy Septono Catur Putranto, Uk Jo, Howon Kim. Toward Enhanced Attack Detection and Explanation in Intrusion Detection System-Based IoT Environment Data. IEEE Access. 2023; 11:131661-131676. doi:10.1109/ACCESS.2023.3336678.

[23] Srivijaya and B. J. Lakshmi, "A review of security infrastructure, protocols, a taxonomy of security threats and intrusion detection in the Internet of Vehicles (IoV)," International Conference on Intelligent Computing and Emerging Communication Technologies (ICEC), Guntur, India, pp. 1-8, doi: 10.1109/ICEC59683.2024.10837261.

[24] Dhumal, Chaitrali T. and Pingale, Dr. S. V., Analysis of Intrusion Detection Systems: Techniques, Datasets and Research Opportunity (March 6, 2024). Proceedings of the International Conference on Innovative Computing & Communication (ICICC 2024).

[25] Neto ECP, Dadkhah SR, Ferreira A, Zohourian R. CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment. Sensors, 2023; 23(13):5941-64; https://doi.org/10.3390/s23135941.