CAPITAL UNIVERSITY OF SCIENCE AND
TECHNOLOGY, ISLAMABAD

A Lightweight Hybrid Machine Learning Mechanism for Intrusion Detection in IoT Networks

by

Fareeha Ashraf

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2026

CERTIFICATE OF APPROVAL

A Lightweight Hybrid Machine Learning Mechanism for Intrusion Detection in IoT Networks

by

Fareeha Ashraf

(MCS241004)

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Syed Jawad Hussain	CASE, Islamabad
(b)	Internal Examiner	Dr. Rizwan Bin Faiz	CUST, Islamabad

Dr.M.Siraj Rathore
Thesis Supervisor
May, 2026

Dr. M. Masroor Ahmed
Head
Dept. of Computer Science
May, 2026

Dr. M. Abdul Qadir
Dean
Faculty of Computing
May, 2026

Author's Declaration

I, **Fareeha Ashraf** hereby state that my MS thesis titled "**A Lightweight hybrid Machine Learning Mechanism for Intrusion Detection in IOT networks**is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

(Fareeha Ashraf)

Registration No: MCS241001

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "**A Lightweight Hybrid Machine Learning Mechanism for Intrusion Detection in IoT Networks**" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

(Fareeha Ashraf)

Registration No: MCS241001

Acknowledgement

Above all, I am grateful to Almighty Allah for giving me the courage, strength, and direction I needed to successfully complete this thesis. The journey was made possible by his countless blessings.

I am deeply grateful to my supervisor, Dr.Siraj Rathore, for his unwavering support, kind encouragement, and valuable guidance throughout this work. Your trust in me meant more than words can express.

My heartfelt thanks to my family and friends for their constant prayers, love, and encouragement, which kept me going during challenging times.

(Fareeha Ashraf)

Abstract

With the rapid growth of IoT devices, cyber attacks also increase, creating security problems. While many current intrusion detection systems (IDS) offer high accuracy in detecting cyber attacks, most are resource-intensive and thus not suitable for lightweight IoT devices. Thus, the need for lightweight IDS that can achieve high detection accuracy while minimizing resource consumption. The use of large feature sets can increase the computational overhead. In this research, we propose a lightweight hybrid machine learning-based intrusion detection system. The system uses a hybrid feature reduction pipeline that combines logistic regression with L1 regularization to eliminate irrelevant and noisy features and Random Forest to rank and select the top features, which are then fed into the LightGBM for classification. The model is tested on the CICIoT2023 dataset, compared to baseline models, other feature selection methods, and using the full set of features to assess the accuracy and efficiency of the system. The results demonstrate that the proposed feature reduction pipeline achieves **22.46%** less CPU usage, **8.2** seconds shorter execution time, and **39.29%** less memory usage with a negligible accuracy loss of only **0.002%**, while the baseline model has a slightly better accuracy of 0.97% than the proposed model (0.968%), the difference is minimal. The findings show that the proposed approach achieves a good trade-off between accuracy and efficiency and can be used in the resource-limited IoT edge environments.

Contents

List of Figures

List of Tables

Abbreviations

ANOVA	Analysis of Variance
BRO	Battle Royale Optimization
CAT-S	Chaotic Atom Search Optimization
CFS	Correlation-based Feature Selection
CIC	Canadian Institute for Cybersecurity
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DL	Deep Learning
DoS	Denial of Service
DT	Decision Tree
ENN	Edited Nearest Neighbors
EO	Equilibrium Optimizer
GA	Genetic Algorithm
GBM	Gradient Boosting Machine
IAT	Inter-Arrival Time
IDS	Intrusion Detection System
IIoT	Industrial Internet of Things
IoT	Internet of Things
IR	Imbalance Ratio
KNN	K-Nearest Neighbors
L1	Lasso (L1) Regularization
LightGBM	Light Gradient Boosting Machine
LR	Logistic Regression

LSTM	Long Short-Term Memory
MITM	Man-in-the-Middle Attack
ML	Machine Learning
NR	Not reported
PCM	Pearson Correlation Metrics
PSO	Particle Swarm Optimization
RAM	Random Access Memory
RF	Random Forest
RFE	Recursive Feature Elimination
SMOTE-ENN	SMOTE with Edited Nearest Neighbors
TTL	Time-to-Live
XGBoost	Extreme Gradient Boosting

Chapter 1

Introduction

1.1 Background

Physical devices that are connected to the internet and have the ability to automatically sense, analyze, and transfer data are referred to as the Internet of Things (IoT).These devices include industrial machinery, smart meters, wearable technology, sensors, and cameras.

Reducing human involvement and enabling intelligent, autonomous, and automatic system operation are the main objectives of IoT. IoT's capacity to link physical equipment and enable real-time data exchange has made it an essential part of contemporary digital infrastructure. It is extensively used in smart cities for public safety and traffic monitoring, in smart homes for energy management and appliance automation, in healthcare for remote patient monitoring and medical data collection, and in industrial settings for automation and predictive maintenance. Cloud computing, wireless communication technologies, and inexpensive embedded hardware devices have all contributed to the explosive expansion of IoT installations.[1]. Additionally, through ongoing data collecting and analysis, IoT devices enhance operational efficiency, lessen human labor, and facilitate quicker decision-making. The development of intelligent systems that can monitor environments and react automatically to changing situations has also been made possible by the growing number of linked gadgets.

As IoT systems grow, the amount of data produced and the number of connected

devices rise dramatically. Efficiency and automation are increased by this expansion, but new technical and security issues are also brought about. Every linked device turns becomes a possible target for an attack, particularly when used with insufficient security measures.

1.2 Evolution and Growth of IoT

From simple machine-to-machine connectivity, the Internet of Things has developed into vast intelligent systems that can make decisions in real time. Modern IoT solutions incorporate analytics, automation, and artificial intelligence, whereas early IoT systems were primarily concerned with data collection. According to recent estimates, there are presently billions of IoT devices in use worldwide, and this figure is only projected to increase. While new services have been made possible by this quick growth, IoT network management and security have become more difficult [2].

1.3 Basic Architecture of IoT Systems

The architecture of a typical Internet of Things system is tiered. IoT devices sense information from the physical world at the perception layer.Gateways and routers use wired or wireless communication technologies to send data at the network layer. Figure 1.1 illustrates a basic layered architecture of an IoT system.

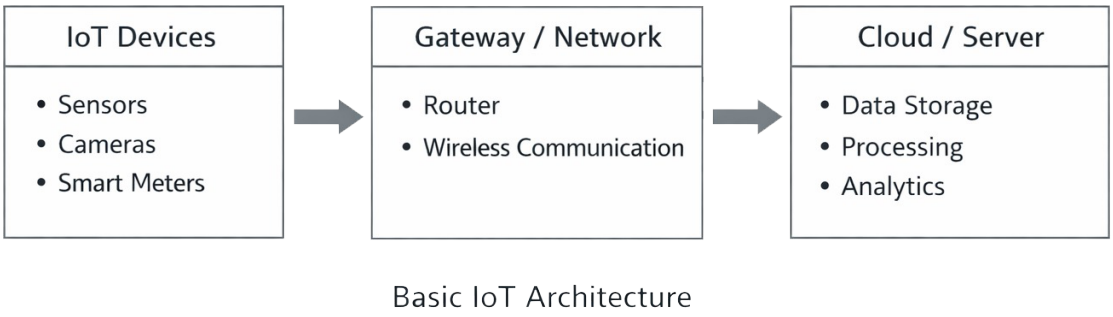

FIGURE 1.1: Basic layered architecture of an Internet of Things (IoT) system

In order to facilitate intelligent decision making, servers or cloud platforms store, process, and analyze the gathered data at the application layer [3]. The majority of Internet of Things devices are subject to significant resource limitations, such

as processor power, memory, and battery capacity limitations. It is challenging to directly implement computationally demanding processes and complex security measures on IoT devices because of these limitations.

1.4 Security Challenges in IoT Networks

One of the most important issues in IoT contexts is security. A large number of IoT devices are set up using outdated firmware, default credentials, or insufficient authentication. Furthermore, IoT communication frequently uses wireless channels, making it more vulnerable to spoofing and eavesdropping attacks [4].

Denial-of-service (DoS), distributed denial-of-service (DDoS), malware insertion, and botnet-based attacks are frequent threats directed at IoT networks. These attacks have the potential to seriously impair vital services, particularly in healthcare and industrial IoT systems.

1.5 Need for Intrusion Detection Systems in IoT

Conventional security techniques like firewalls, access control systems, and authentication methods are crucial to IoT security. However, these security measures are frequently ineffectual against novel, unidentified, or quickly changing assaults because their main purpose is to prevent unwanted entry. In dynamic IoT contexts, traditional security methods typically rely on predetermined rules and signatures, which restricts their capacity to identify complex or zero-day attacks. [5]. Intrusion Detection Systems (IDSs) are frequently utilized as an extra security layer in Internet of Things networks to overcome these constraints. In order to spot questionable or malicious activity, an intrusion detection system (IDS) continuously monitors network traffic, communication patterns, and device behavior. IDS concentrates on identifying persistent threats and unusual activity that might evade conventional defenses, in contrast to preventive security measures. IDS serves as a second line of defense by helping to detect threats that evade traditional security measures and by sending out early warning signals for possible cyber intrusions.

The IDS monitors mirrored gateway traffic to detect suspecious activities and raise security alerts as shown in the figure 1.2 [6].

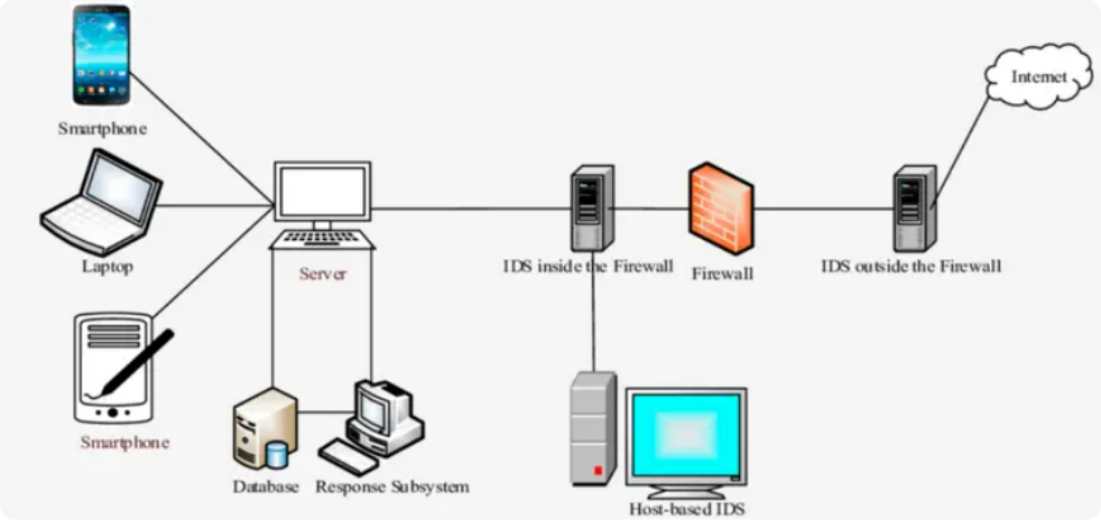

FIGURE 1.2: Intrusion Detection system [7]

1.6 Machine Learning for IoT Intrusion Detection

In IoT networks, machine learning (ML) techniques are frequently employed to enhance intrusion detection. ML-based intrusion detection systems identify patterns in past network traffic and categorize actions as either benign or malevolent. Decision trees, Random Forest, Support Vector Machines, and ensemble-based classifiers are examples of common machine learning methods [8].

Because ML-based IDS models can achieve strong detection performance with relatively little computational cost especially when paired with feature selection techniques they are appropriate for IoT environments.

1.7 Deep Learning for IOT Intrusion Detection

In intrusion detection tasks, Deep Learning (DL) techniques including Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks have shown excellent accuracy [9].

However, DL models demand large memory resources, high processing power, and large datasets. The deployment of DL-based IDS in IoT edge devices and real-time monitoring systems is challenging due to these constraints. Because of this, DL techniques are frequently inappropriate for IoT environments with limited resources [10].

1.8 Motivation for Lightweight and Hybrid IDS

Recent studies concentrate on lightweight and hybrid IDS techniques to maintain a balance between detection accuracy and processing efficiency. Hybrid IDS reduces complexity while maintaining strong detection capabilities by combining effective ML classifiers with feature selection approaches [11]. While existing machine learning and deep learning models achieve high detection accuracy, they often require substantial computational resources, making them unsuitable for deployment on resource-constrained IoT devices.

This creates a strong need for developing lightweight intrusion detection systems that can maintain high detection performance while minimizing computational overhead. Therefore, this research is motivated by the need to design an efficient and scalable IDS that balances accuracy and computational efficiency for real-world IoT environments.

Maintaining performance with reduced computational cost has been demonstrated by lightweight classifiers like LightGBM and ensemble learning models [12].

1.9 Problem Statement

Traditional intrusion detection systems primarily aim to improve the detection accuracy; but they are generally computationally intensive. This poses a problem when deploying such heavy IDS models in resource-limited IoT edge environments, where heavy learning algorithms cannot be run due to low CPU, memory, and power supply [8, 11]. Hence, there is a need for lightweight intrusion detection systems capable of effectively detecting cyber attacks with low resource

consumption. A good IDS should have a trade-off between processing time, memory consumption, CPU usage and detection accuracy to be more practical for IoT deployments [13, 14].

1.10 Research Objectives

The primary goals of this study are:

i. To provide a lightweight hybrid feature selection system for IoT intrusion detection.

ii. To create an effective machine learning-based intrusion detection system appropriate for IoT contexts with limited resources.

iii. To evaluate the proposed approach using efficiency and accuracy metrics.

1.11 Research Questions

RQ1: In IoT environments, which machine learning classifiers are most suited to maintain a balance between computing efficiency and accuracy?

RQ2: How can a lightweight intrusion detection framework based on hybrid feature selection increase detection accuracy and efficiency in IoT networks?

1.12 Organization for the Thesis

This is how the rest of the thesis is structured. The literature review is presented in Chapter 2. The proposed methodology of each experiment is explained in Chapter 3. Analysis and experimental results are covered in Chapter 4. The thesis is concluded and future research directions are outlined in Chapter 5.

Chapter 2

Literature Review

2.1 Introduction

The Internet is used to connect a wide range of devices, such as sensors, cameras, smart meters, and everyday objects, to the Internet of Things (IoT). The built-in security features of many IoT devices are usually insufficient because they are inexpensive and have few resources.

IoT security solutions are being researched continuously as a result of the fact that IoT networks are now a prime target for malware campaigns and attackers [3].

Intrusion Detection Systems (IDS) are commonly used to monitor network traffic or device behavior to identify malicious activities. On their own, traditional security mechanisms like firewalls and access control are insufficient since hackers can still overcome them, especially by using new and unknown attack methods.

IDS solutions are frequently used to give IoT networks an extra layer of security [15]. However, IoT data usually has a lot of components (traffic fields, timing statistics, and device specifics).

Numerous research highlight the significance of feature reduction prior to training machine learning models for intrusion detection [13].

By removing superfluous data and identifying the most informative qualities, feature reduction strategies increase model efficiency and lower resource consumption.

The IDS can maintain strong detection capabilities while achieving faster training

and testing performance by reducing the dataset's dimensionality. Thus, creating lightweight and effective intrusion detection systems for IoT networks depends heavily on feature selection.

2.2 Role of Feature Selection in IoT Intrusion Detection Systems

Feature selection is the process of choosing a smaller number of important features from the original dataset. Its main goal is to reduce dimensionality while keeping the most important information for classification. Feature selection in IoT IDS helps reduce overfitting, improve detection accuracy, and reduce training and inference time when the dataset contains repeated and noisy characteristics [14]. In the Internet of Things, feature selection is an effective strategy because many IDS solutions are anticipated to function near the network edge (gateways, routers, or fog nodes). The energy, memory, and CPU power of these devices are all limited. Therefore, by using fewer features, IDS models can become lighter and more suitable for near-real-time detection [16]. Another important factor to take into account is the possibility that features in IoT intrusion datasets are inconsistent between circumstances. For example, features that function well in one network arrangement may not work well in another. Selecting robust features may improve generalization and reduce dependence on environment-specific fields [17].

2.3 Feature Selection Method Types

Feature selection methods are often divided into three groups: filter-based, wrapper-based, and hybrid-based. This section explains each category and lists its benefits and drawbacks in relation to IoT IDS.

2.4 Filter-Based Approaches

Regardless of the classifier, filter techniques rank features using statistical or information-based metrics. Common filtering techniques include correlation-based

scoring, chi-square, and information gain. These methods are fast since they don't continuously train a model to assess feature subsets [18]. Because filter techniques can be quickly applied to large feature sets and reduce computation costs, they are attractive for IoT IDS. For example, one study used correlation-based feature selection, chi-square, and information gain to reduce features before applying clustering and random forest for intrusion detection [19]. Training and inference time can also be reduced by comparing filter selection to more complex feature reduction methods. When computational cost is a major consideration, feature selection often leads to shorter model training periods, as per a detailed comparison of feature selection and feature extraction [20]. Table 2.1 compiles recent filter-based feature selection methods used in IoT and network intrusion detection systems together with the datasets, classifiers, and significant limitations of each study. The performance trends and constraints of current filter-based methods in terms of feature relevance selection, computational efficiency, and detection accuracy.

TABLE 2.1: Filter-based feature selection approaches for IoT intrusion detection

Study	Year	Dataset	FS Technique	Classifier	Results
1	2021	N-BaIoT	Chi-square	K-means + Decision Tree	Error reduced from 0.39% to 0.01%; runtime discussed (NR) [18]
2	2021	AWID	IG, Chi-square, CFS	K-means + RF	High accuracy, high TPR, low FPR; efficiency NR [21]
3	2021	NSL-KDD, UNSW-NB15	Rule-based FS	Deep FFNN	NSL-KDD: Acc 99.0%, DR 99.0%, FPR 1.0%; UNSW: Acc 98.9%, DR 99.9%, FPR 1.1% [22]
4	2024	TON-IoT	FS vs FE comparison	Multiple ML models	FS reduces time; FE improves accuracy; runtime compared (NR) [23]

2.5 Wrapper-Based Approaches

Usually, wrapper techniques are computationally costly. They have to train and test the model repeatedly in order to evaluate different feature subsets. This becomes expensive when large datasets or near-real-time deployment are required [24].

Additionally, using wrapper methods increases the likelihood of overfitting. Because the feature subset is especially designed for a certain dataset and classifier, it may not transition well to unidentified IoT scenarios. This is a major issue in the Internet of Things since device behavior and traffic patterns are subject to vary over time [25]. Another disadvantage is that when the original feature set is big, wrapper search may become quite slow. Therefore, implementing wrapper-only selection at edge or fog nodes with minimal resources is challenging. Furthermore, wrapper techniques necessitate repeated model evaluation and training for various feature subsets, which greatly raises execution time and computational cost. [23]. Table 2.2 compares the wrapper-based feature selection methods examined in this study.

TABLE 2.2: Wrapper-based feature selection approaches for IoT intrusion detection

Study	Year	Dataset	FS Technique	Classifier	Results
1	2025	RF fingerprinting; CICIDS2017; CICIoMT2024; UNSW-NB15	GA wrapper	DNN + Bi-LSTM	Acc 99.84%, Prec 100%, Rec 99.69%, F1 99.84%; Model size 108.42 KB; XAI (LIME) [26]
2	2024	TON-IoT; UNSW-NB15	SFS + GA-optimized ELM	SVM	Acc 99% (TON-IoT), 86% (UNSW); Efficiency NR [24]
3	2025	Network traffic dataset	ACO-based wrapper	Deep learning IDS	Improved detection performance; Efficiency NR [27]

2.6 Hybrid-Based Approaches

Hybrid feature selection combines the ideas of a filter and a wrapper. First, characteristics that are clearly unnecessary are usually quickly removed using a filter process.

After that, a wrapper or optimization stage searches the reduced feature space for a robust subset for classification. This keeps computation cost and performance in check [28].

Because hybrid techniques can improve detection accuracy while reducing complexity, IoT IDS commonly adopt them. For example, a two-stage IDS achieved good accuracy on UNSW-NB15 and NSL-KDD by employing clustering and a nature-inspired search approach to pick effective features [29].

By integrating multiple feature reduction strategies and then employing an ensemble learner to enhance botnet detection, another hybrid feature selection study produced very high performance with good true positive rates [30].

Because hybrid selection requires fewer model evaluations overall, it is faster and more effective than pure wrapper selection. Nevertheless, it can provide better feature subsets than pure filters since it keeps classifier behavior in mind in later stages [31].

Hybrid techniques are also more flexible for IoT IDS. They can be encouraged to choose lightweight computation in the first step and more thorough optimization in the second. As a result, they work better in Internet of Things environments where speed and accuracy are essential [28].

Moreover, hybrid selection can better manage mixed IoT information (continuous, categorical, and derived traffic features).

For example, numerous feature categories used in intrusion detection datasets can be better captured by a hybrid technique that combines mutual information and additional selection procedures. The hybrid approach outperforms a single strategy in identifying both highly discriminative and statistically significant features by integrating many selection procedures. This raises the classifier's capacity to

identify various attack patterns and improves the overall quality of the chosen feature subset.

Hybrid feature selection techniques also aid in eliminating redundancy while maintaining crucial data needed for precise and effective intrusion detection.[30].

TABLE 2.3: Hybrid feature selection approaches for IoT intrusion detection

Study	Year	Dataset	FS Technique	Classifier	Results
1	2024	Real-world NIDS dataset	Correlation + Mutual Information + RFECV	Decision Tree; Random Forest	High accuracy; computational complexity reduced [20]
2	2022	IoT DDoS dataset	MI + ANOVA + Chi-square + L1	ML-based detector	Accuracy improved; training time reduced [32]
3	2025	TON-IoT; BoT-IoT	QIPSO + ANFIS	CapsNet + RNN	Accuracy up to 99%; efficiency Not Reported [33]
4	2024	Suspicious URL dataset	Filter + GA wrapper	Boosting models	Around 99% accuracy; computational cost reduced [34]
5	2024	IoT IDS dataset	Correlation + Harris Hawk Optimization	DT; SVM	Accuracy 96.46%; efficiency NR [35]
6	2023	UNSW-NB15; NSL-KDD; InSDN	Hybrid FS	CNN-BiLSTM	High accuracy; training time reduced [36]
7	2025	UNSW-NB15; NSL-KDD	K-means + Cuckoo Search	ML classifiers	Accuracy up to 99.78%; processing time improved [29]
8	2021	Benchmark datasets	CFS + BFS + DRSA	NN; SVM	Moderate accuracy; statistical evaluation reported [37]

TABLE 2.3: Continue from previous page

Study	Year	Dataset	FS Technique	Classifier	Results
9	2023	UNSW-NB15; BoT-IoT	Autoencoder + PSO	DNN	Accuracy up to 99.22%; efficiency Not Reported[38]
10	2023	CICIDS2017	RF importance + PSO	ML classifier	Accuracy ∼99.9%; efficiency Not Reported [39]
11	2020	AWID	GA + guo2023iot	SVM	High accuracy Reported ; computational cost reduced [40]
12	2025	CICIDS2017	PSO-GA	ELM + Bagging	High accuracy; statistical validation reported [41]
13	2024	CICIDS2017; NSL-KDD	Correlation + Bat Optimization	CNN	Accuracy 99.45%; efficiency Not reported [42]
14	2020	NSL-KDD	ABC + PSO	ML classifiers	Improved precision; cross-validation applied [43]
15	2021	AWID	Deep feature abstraction + wrapper	ANN	Accuracy up to 99.95%; efficiency NR [44]
16	2023	BoT-IoT	PCA + GA	KNN	Accuracy 99.99%; prediction time reduced [45]
17	2024	NSL-KDD; UNSW-NB15	Filter + fuzzy TOPSIS + GWO	ML models	Improved performance; efficiency NR [46]
18	2025	IoT datasets	Red Panda + Simulated Annealing	GBRT	Accuracy 99.6% reported ; efficiency Not Reported [47]

TABLE 2.3: Continue from previous page

Study	Year	Dataset	FS Technique	Classifier	Results
19	2021	NSL-KDD; BoT-IoT	Correlation + RF importance	RF; XGBoost	Accuracy above 99% reported ; efficiency Not Reported [48]
20	2024	CICIDS2017	Hybrid FS + GA tuning	ML models	Improved detection; detection time emphasized [31]
21	2025	CICIoT2023	Chi-square + RFE	RF; DT; XGB; KNN	Accuracy up to 99.95%; efficiency Not Reported[49]
22	2024	CICIoT2023	Spearman correlation + hierarchical clustering	CatBoost	High accuracy, precision, recall; very low prediction time reported [50]
23	2025	CICIoT2023; TON-IoT	CNN-LSTM + SMOTE-ENN	Deep learning model	Accuracy 99.12%; lightweight model with reduced complexity [51]
24	2023	CICIDS2017; CICIoT2023	SelectKBest + MI; CNN + SVM + GWO	RF; XGBoost	Accuracy up to 99.99%; prediction time reduced were reported [52]
25	2025	CICIoT2023	Correlation-based FS + feature importance + Bayesian optimization	Decision Tree; Random Forest	Accuracy up to 99.74%; computational time reduced significantly reported [53]
26	2025	CICIoT2023; CI-CIoMT2024	GWO feature selection + CNN-enhanced LightGBM	LightGBM	Accuracy 95.24%; low false alarm rate and reduced latency reported [54]

TABLE 2.3: Continue from previous page

Study	Year	Dataset	FS Technique	Classifier	Results
27	2025	CICIoT2023; UNSW-NB15	CNN + LSTM + GRU	Deep learning model	Accuracy up to 99.82%; efficiency NR [55]

2.7 Machine Learning-Based Intrusion Detection in IoT Networks

Machine learning (ML)-based intrusion detection systems (IDSs) classify traffic as malicious or genuine by seeing trends in labeled or unlabeled data. ML models commonly used for IDS include Random Forest, SVM, Decision Trees, and boosting models. These models are widely used because they often offer sufficient accuracy at a reasonable computational cost [29].

ML-based approaches in IoT IDS can potentially identify new threat variations if the model is trained with a range of patterns. A hybrid intrusion detection system study [25] found that combining different detection methods and machine learning models can reduce false positives and boost resilience against different attack patterns.

Another advantage is that ML models can continue to perform well even after feature reduction. Eliminating duplicate features often makes ML classifiers faster and more stable [21].

2.8 Deep Learning-Based Intrusion Detection in IoT Networks

Deep learning (DL) models such as CNN, LSTM, and hybrid CNN-LSTM architectures are also used for intrusion detection. They can automatically detect high-level patterns in partially or fully processed data. For example, one work

[48] presented a modified deep learning architecture for IDS with optimum feature selection. Deep learning often needs more memory, computing power, and training times than conventional machine learning models. This is a major challenge for low-resource IoT edge devices since they are unable to easily handle large models [56]. Furthermore, deep learning implementation may require the utilization of powerful CPUs or GPU support to meet real-time constraints, which is not always possible for conventional IoT gateways. For IDS designs that are lightweight and edge-friendly, ML models are therefore commonly selected over deep learning [23].Motivation for Choosing Machine Learning-Based Intrusion Detection Systems Because ML models can provide a useful trade-off between detection performance and computational economy, this study focuses on ML-based IDS. Compared to huge deep learning pipelines, IoT edge settings can train and deploy machine learning models with less resource consumption [25].

Another motivation is the effective combination of feature selection with ML models. By reducing the feature set through hybrid feature selection, ML-based IDS can be made faster while retaining good accuracy. This directly supports the deployment goals of low latency and low memory consumption [28]. Finally, ML-based IDS solutions can be adjusted through retraining or incremental updates when IoT threat patterns shift. When combined with careful feature selection design, their practicality makes them a great choice for long-term IoT security [31].

2.9 Critical Analysis and Research Gap

The studied research indicates that while filter techniques are fast, they may overlook important feature interactions and not optimize classifier performance. Wrapper approaches can improve accuracy, but they might occasionally be excessively slow and computationally expensive for IoT applications. Numerous papers claim that because hybrid techniques aim to combine both benefits, they are more suited for IoT intrusion detection [25]. Even though several hybrid IDS experiments have shown great accuracy, there are still gaps. First, it is still challenging to adapt several techniques to real-world IoT implementations because they are

mostly tested on benchmark datasets. Second, certain hybrid approaches continue to use costly optimization algorithms when frequent model updates are required or the dataset is large [29]. Another research gap is the need for better lightweight systems that minimize computation while maintaining multiclass attack detection accuracy. The tradeoff between robustness, runtime, and feature subset size is typically not thoroughly studied, despite the fact that certain works achieve exceptionally high accuracy [30].

Hence, the primary issue is to develop a machine learning-based IoT intrusion detection system that: (1) uses hybrid feature selection for strong performance; (2) maintain a balanace between accuracy and efficiency and (3) provides accurate detection in a range of IoT traffic situations.[28].

2.10 Chapter Summary

This chapter reviewed feature selection for IoT intrusion detection, organized the provided studies into filter-based, wrapper-based, and hybrid approaches, and highlighted why lightweight ML-based IDS is a strong direction for edge IoT deployments.

Chapter 3

Research Methodology

3.1 Chapter Overview

In this chapter, the research methodology for designing, developing, and testing the proposed multi-class intrusion detection system for Internet of Things networks is explained. Starting with problem identification and a review of relevant literature, it outlines the systematic approach used in this study. It then moves on to data preparation, model construction, and performance evaluation. Every stage is carefully organized to ensure that the research follows logically, uses appropriate techniques, and meets the research objective.

Additionally, the chapter describes the experimental design and the assessment methodology that are employed to evaluate the effectiveness of the proposed solution. By using this systematic approach, the study ensures accurate analysis, reproducible outcomes, and a solid foundation for the findings covered in the following chapter.

3.2 Proposed Research Methodology

The overall research methodology adopted can be seen in Figure 3.1, which highlights the systematic progress of the activities taken to meet the research's objectives.Every phase is structured to develop logically from the previous one, ensuring a systematic and valid method for addressing the issue.

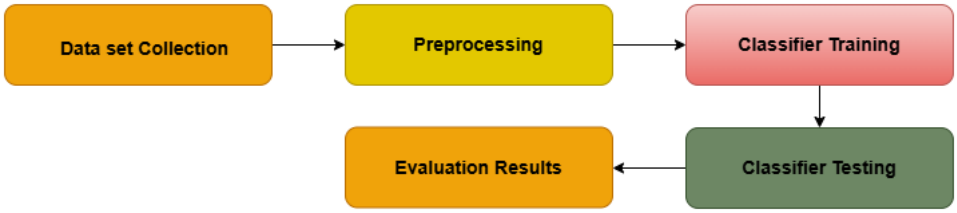

FIGURE 3.1: Research Methodology

But in our case ,a hybrid pipeline is introduced as shown in Figure 3.2

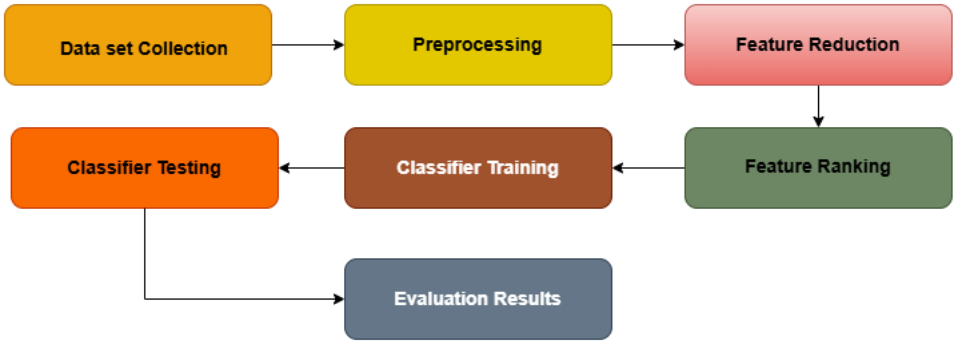

FIGURE 3.2: Proposed Research Methodology

The research starts with gap analysis and problem introduction where the limitations of traditional intrusion detection systems are examined. The majority of current systems either concentrate on binary classification (attack vs. normal) or struggle to generalize effectively across different attack types.

This highlights the necessity for a strong framework for classifying multi-class attacks that can differentiate between various attack types while maintaining high detection accuracy.

Detailed literature research is then carried out to examine machine learning and deep learning techniques for the detection of IoT network intrusion.

According to the review, deep learning models have a high accuracy rate but are less appropriate for IoT devices with limited resources since they involve a lot of processing, a long training time, and longer detection delays.

On the other hand, classic machine learning techniques are quicker and more effective, but they frequently have problems, including class imbalance, duplicated features, and poorer accuracy and detection of new attacks. The research goals and the hybrid technique suggested are affected by these findings, which show a

trade-off between accuracy and efficiency in current IDS systems.

In order to improve multi-attack classification performance, improve representation of features, and provide reliable evaluation, research questions and objectives are created based on the identified research need. These goals serve as a set of guidelines for throughout the experiment.

The dataset preparation and selection process is the next stage. A common dataset for intrusion detection that includes a variety of attack methods is used. Duplicate records, noise, and missing values are eliminated from the raw data by cleaning.

Normalization, encoding, and feature selection are examples of preprocessing methods used to prepare the data for training and to make it reflect actual network traffic.

The proposed framework is made to categorize network traffic into more than one attack class after preprocessing, rather than only normal or attack. Important patterns that help in differentiating between various attack types are learned by the model.

The model is implemented and examined. To ensure a fair evaluation, the dataset is divided into training and testing sets. Stable learning is tested after the model is trained with the best settings.

Lastly, the model is evaluated using multi-class metrics including detection time, total time, CPU consumption, and memory usage in efficiency measures and accuracy, precision, recall, and F1 score in accuracy measures.

3.3 Datasets

This research uses data from the Canadian Institute for Cybersecurity (CIC) IoT 2023 dataset, which is publically available at [57]. This dataset represents realistic network traffic behavior in both benign and attack scenarios, and it is specifically made to evaluate intrusion detection systems in Internet of Things environments. Several CSV files representing various traffic captures make up the original dataset repository. The `Merged01.csv` file, which offers a thorough and unified view of IoT network traffic, is used in this study. There are 712,311 network traffic records

and 33 attack classes in the chosen dataset, and each record represents a distinct network flow. Because each flow has 40 features, including one target label, it can be used for intrusion detection using supervised machine learning.

The feature set is made up of multiple categories that together describe the behavior of networks. These include protocol-level features like header length, protocol type, time-to-live (TTL), and packet rate; protocol and application-layer indicators like HTTP, HTTPS, DNS, TCP, UDP, and ICMP; statistical flow features that describe traffic behavior, such as variance, standard deviation, and total traffic volume; and TCP flag-based features that record the presence and frequency of control flags like SYN, ACK, FIN, RST, and PSH. Furthermore, the timing patterns between packets are described by inter-arrival time (IAT) features, which are used to capture temporal aspects. These characteristics work together to effectively distinguish between benign and malicious traffic patterns and enable a thorough depiction of network communication behavior.

Each network flow has a target variable, represented by *Label*, which indicates the sort of attack. Multiple-class intrusion detection is made possible by the dataset's inclusion of many attack categories. These attack types include spoofing-based attacks like DNS spoofing, Distributed Denial of Service (DDoS) attacks like DDoS-PSHACK Flood, DDoS-SYN Flood, and DDoS-UDP Flood, and other malicious traffic behaviors. To enable the learning models to capture unique traffic fingerprints corresponding to various attack types, each network flow is given a single attack label. Due to the high dimensionality of traffic variables and the variety of attack categories, the CICIoT2023 dataset is ideal to evaluate how well machine learning models perform in multi-class IoT intrusion detection scenarios.

3.4 Data Preprocessing

A reliable intrusion detection system must start with data preparation because raw IoT network traffic data frequently contains noise, typos, redundant information, and skewed class distributions. The dataset needs to be properly prepared before using machine learning techniques to ensure that the patterns that are recovered appropriately depict both benign and malicious network behavior. In addition

to improving model learning and stability, efficient preprocessing additionally improves generalization performance and detection accuracy.Figure 3.3 shows the complete preprocessing methodology used in this investigation.

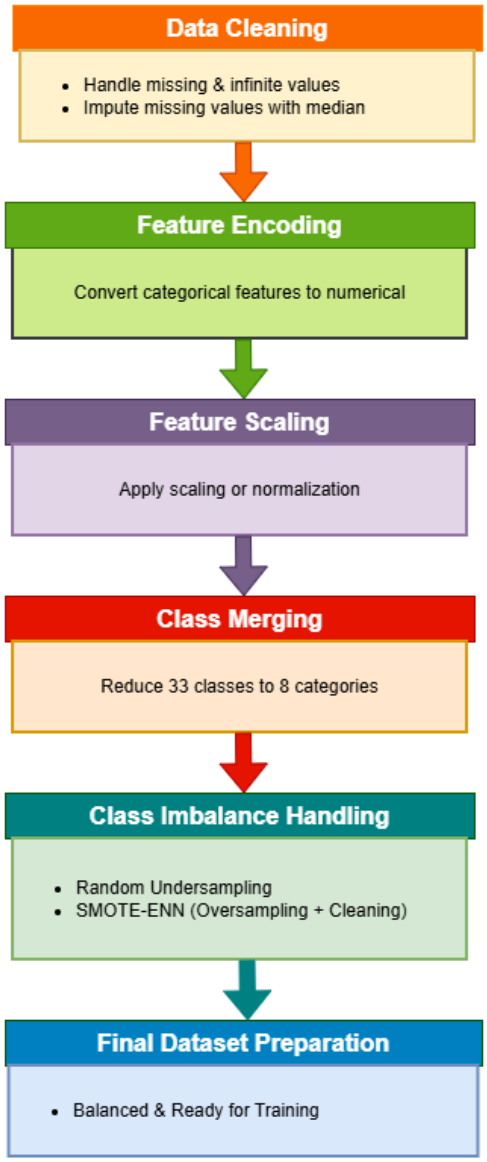

FIGURE 3.3: Data preprocessing pipeline applied to the CICIoT2023 dataset

The CICIoT2023 dataset is prepared for multi-class intrusion detection in this research using a structured data preprocessing pipeline.

Data cleaning, target label verification and encoding, feature type identification, feature scaling and normalization, handling class imbalance and dataset splitting into training and testing subsets are all examples of preprocessing steps. Every

stage is made to retain computational efficiency while addressing particular issues related to massive IoT traffic data.The following subsections offer thorough explanations of each preprocessing step.

3.4.1 Data Cleaning

Missing, infinite, and invalid values were present in the input dataset, which can have a negative impact on model performance. To maintain the statistical characteristics of the features, missing values were handled using median-based imputation, and infinite values were initially substituted with NaN. During model training, this phase ensured stability and consistency.

3.4.2 Feature Selection and Preparation

Relevant network traffic features were kept for analysis, while redundant and non-informative features were eliminated whereas needed. The numerical format of all features was verified to be compatible with machine learning methods.

3.4.3 Feature Encoding

Numerical representations were used to encode categorical features, such as protocol related indicators and flags. This change made it possible for machine learning models to understand categorical network behaviors with simplicity.

3.4.3.1 Feature Scaling

We used feature scaling to standardize the values of the numerical features, ensuring that all features have an equal impact during training. The features in the dataset have different values and units; hence, scaling will be applied to ensure that features with larger values do not overwhelm the others.

In this research, Min-Max scaling was used to normalize all features to the range $[0, 1]$. This helps with numerical stability and convergence. The Min–Max normalization is given by:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{3.1}$$

Here, x is the original value of the feature, $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of the feature, and x' is the scaled value.

In order to maintain reproducibility and prevent data leakage, the scaling parameters ($x_{\min}$ and $x_{\max}$) were calculated using only the training data, and then applied to both the training and testing data.

This data preparation step standardises the features, which enhances the model's stability, convergence and performance.

3.4.3.2 Class Merging Strategy

The CICIoT2023 dataset originally has 33 attack classes, which are inherently clustered into categories like DDoS, DoS, Recon, Web-based, Brute Force, Spoofing and Mirai [58].

Following this classification, the 33 classes were combined into 8 higher level classes (including benign) to simplify the problem and reduce class imbalance. This retains the attack characteristics while enhancing model performance and efficiency.After combining the original attack labels into main categories, the class-wise distribution is shown in Figure 3.4.

The distribution is still unbalanced across categories even though label merging reduces class fragmentation, underscoring the need for class imbalance control strategies in later preprocessing stages.

3.4.4 Handling Class Imbalance

The distribution of classes in the CICIoT2023 dataset is highly unbalanced, with some attack categories having a disproportionately high number of samples in comparison to others. By biasing predictions toward majority classes, this imbalance can have a negative impact on supervised learning models and is a frequent problem in intrusion detection datasets. Minority attack classes may thus have poor detection ability, especially in situations involving multi-class classification. The

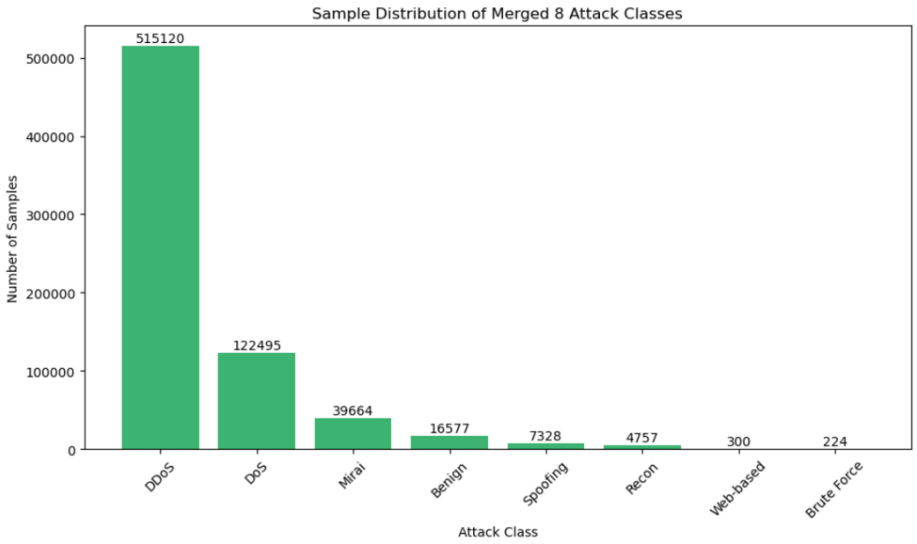

FIGURE 3.4: Class-wise distribution of network traffic samples after merging attack labels into major categories

class-wise distribution of network traffic samples before any balancing is shown in Figure 3.5 in order to analyze the scope of this problem. While a number of

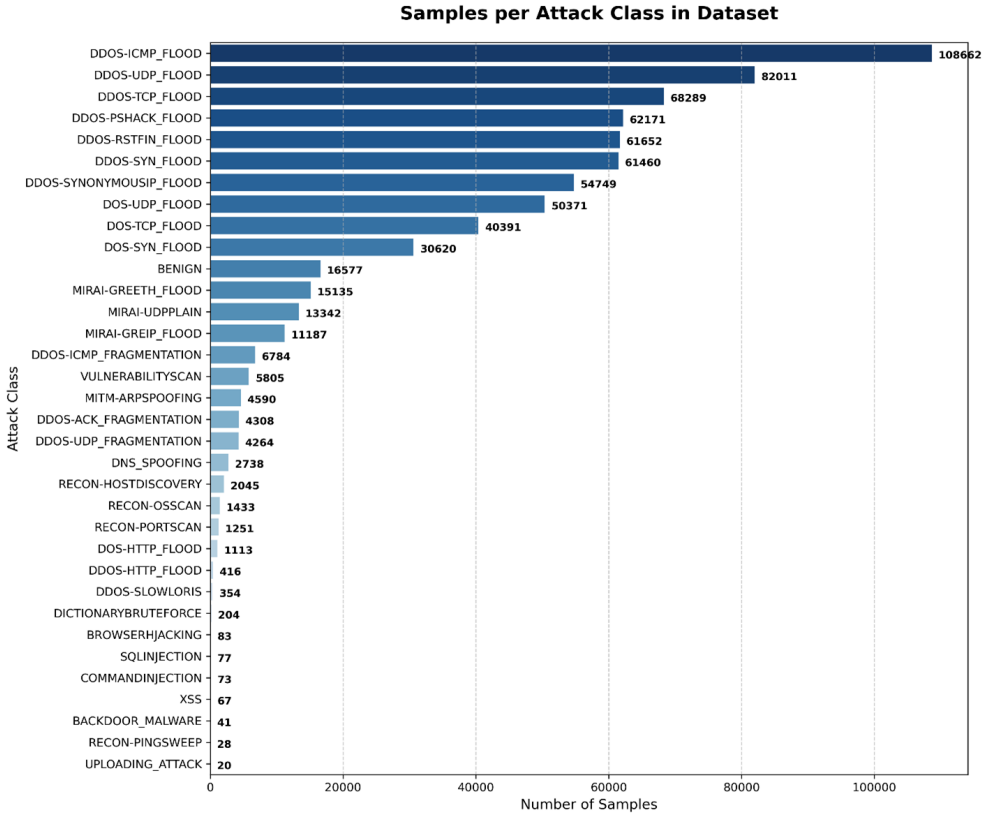

FIGURE 3.5: Class-wise distribution of network traffic samples before applying class balancing techniques

additional attack categories are represented by comparatively few instances, the

figure emphasizes the dominance of DoS- and DDoS-based attack classes. The requirement for suitable class imbalance handling strategies prior to model training is motivated by this skewed distribution.

We evaluated three approaches of class balancing technique i.e SMOTE ,Class Weights and SMOTE-ENN. Result of each method is described in chapter 4 . The best results evaluated by using SMOTE-ENN which uses a two-stage class balancing technique to address this issue. To keep majority classes from controlling the learning process, random undersampling is used in the initial step to lower their sample counts. Reducing computational overhead during training is another benefit of this procedure. A hybrid resampling method called SMOTEENN is used in the second step. Edited Nearest Neighbors (ENN) and the Synthetic Minority Over-sampling Technique (SMOTE) are combined in SMOTEENN.

3.4.4.1 Splitting Dataset and Preventing Data Leakage

The proposed intrusion detection system is evaluated in an unbiased and fair manner by first splitting the dataset into training and testing data, followed by data preprocessing and class imbalance handling. In particular, the dataset is divided using a stratified 80:20 ratio, with 80% of the data used for training and 20% for testing.

Stratified sampling is used to maintain the class distribution in the training and testing sets. This is crucial for multi-class intrusion detection, where some types of attacks may be less frequent.

Once the data is split, all the data preprocessing operations, such as missing value imputation, outlier detection, and normalization (if needed), are performed individually on the training and testing sets. Crucially, class balancing strategies like SMOTE, SMOTE-ENN and random undersampling are applied **only to the training set**.

This is important to avoid data leakage, since oversampling or undersampling before splitting the dataset would allow the model to learn from the test set (i.e. the model would have access to information from the test set when training), resulting in overly optimistic performance metrics.

The test set is kept strictly separate from the training set, and is only used for final performance evaluation.

In order to increase the representation of minority classes, SMOTE creates synthetic samples by interpolating between existing samples. Then, by removing examples that are different from most of their nearest neighbors, ENN eliminates ambiguous and noisy data. When SMOTEENN and random undersampling are applied together, the dataset becomes cleaner, more balanced, and has better class separability. Figure 3.6 displays the class distribution following the implementation of the balancing strategy. A more dependable basis for training machine learn-

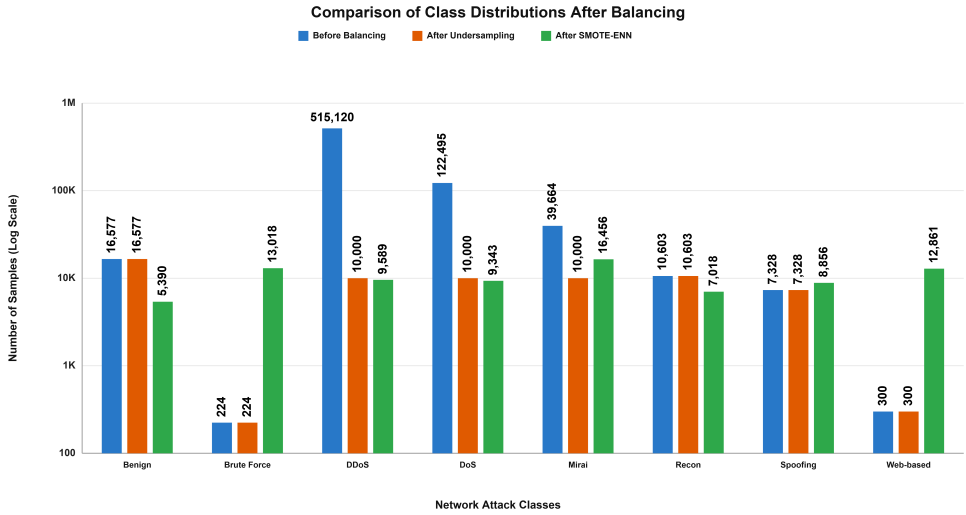

FIGURE 3.6: Class-wise distribution of network traffic samples after applying SMOTEENN

ing models is offered by this balanced dataset, which facilitates better learning of minority attack patterns and eventually improves recall and overall classification performance. After preprocessing, the merged attack categories were encoded into numeric labels to facilitate supervised multi-classification. Figure 3.7 illustrates the final class-wise distribution of the encoded dataset, where both numeric labels and their corresponding attack categories are shown.

3.4.5 Final Dataset Preparation

Each numeric label corresponds to a specific attack category, ensuring a compact and consistent representation for model training. This encoding strategy enables

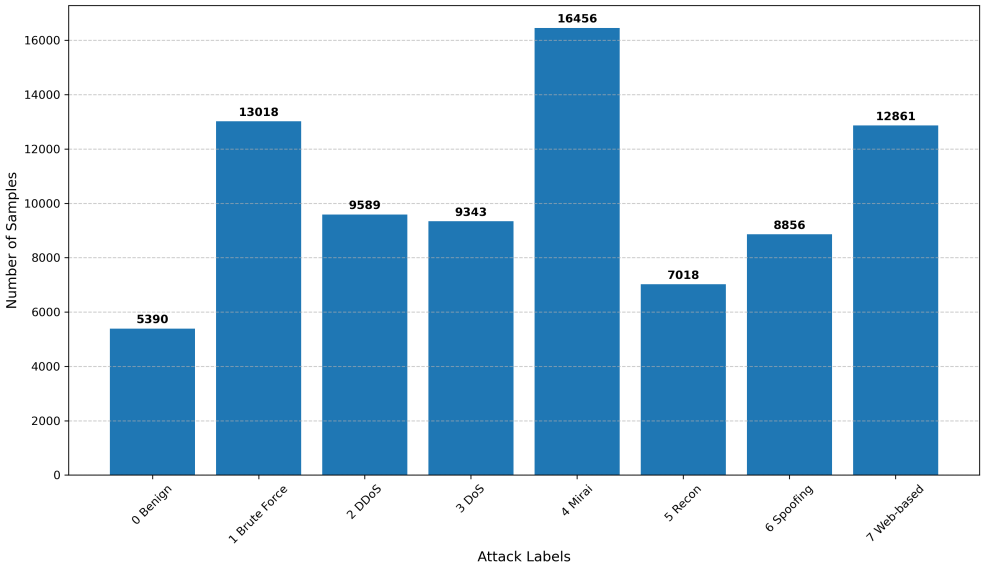

FIGURE 3.7: Final class-wise distribution of the encoded dataset after preprocessing

efficient learning while preserving the semantic meaning of each attack class. Eight attack categories—Mirai, Brute Force, Web-based attacks, DDoS, DoS, Spoofing, Reconnaissance, and Benign traffic—are included in the final dataset. The labels' numeric encoding maintains each attack category's semantic value while ensuring compliance with the learning methods. The final dataset shows consistent feature representations, less noise, and a more balanced class distribution.

The proposed framework attack class uses this finalized dataset as input for the feature selection and classification phases.

3.5 Proposed Framework Diagram

In order to make the feature selection process reliable, the L1-regularized Logistic Regression model uses stratified training data to ensure the presence of all classes.

The regularization strength was optimised to balance the sparsity of features and model performance.

Increasing the regularization strength results in the removal of more features, while decreasing it keeps more features. For our research, we empirically chose the parameter to prevent underfitting and overfitting.

Furthermore, Random Forest in the second stage is resistant to noise and interactions between features. Random Forest can capture complex interactions between features, which may not be linear, making it an ideal choice for network traffic. This prevents the loss of potentially valuable features.

In addition, it's worth noting that feature importance scores returned by Random Forest are normalized to sum to one. This enables comparative analysis of feature importance and allows the calculation of cumulative importance.

The ranking procedure guarantees that the most informative features are chosen.

Moreover, the cumulative importance value of 95% was chosen experimentally. It was observed that using more features than this threshold did not lead to noticeable gains in accuracy, but did increase the computational time. Thus, the top 23 features strike a balance between efficiency and performance.

Finally, the subset of features was fed into the LightGBM classifier. This process reduces the dimensionality of the data, making the model more efficient in terms of training time and memory usage, while achieving high detection rates.

This is especially important in real-time intrusion detection for IoT networks, which have limited computing resources. Figure 3.9 represented the overall architecture of the proposed multiclass intrusion detection system in IoT networks and the class labels. LightGBM as an attack classifier is used to detect these attacks that are mentioned in the diagram more accurately and in less time.

We then defined a cumulative feature importance threshold of 95%, in that features were selected until 95% of the importance was reached.

This led to the selection of 23 informative features as seen in Figure 3.8.A LightGBM classifier for multi-class classification is trained using the chosen features, and the model's performance is assessed using metrics for accuracy, precision, recall, and F1-score.After handling the class imbalancing issue ,we have 82513 samples, and then on the balanced dataset we evaluated the pipeline to ensure the better results in terms of accuracy and efficiency.The framework's architecture strikes a compromise between detection performance and computing efficiency for IoT scenarios. It maintains excellent classification accuracy while reducing

needless computing overhead through feature reduction. Because of this, the suggested framework is more suited for implementation in IoT scenarios with limited resources, where memory utilization, processor speed, and energy consumption are important considerations.

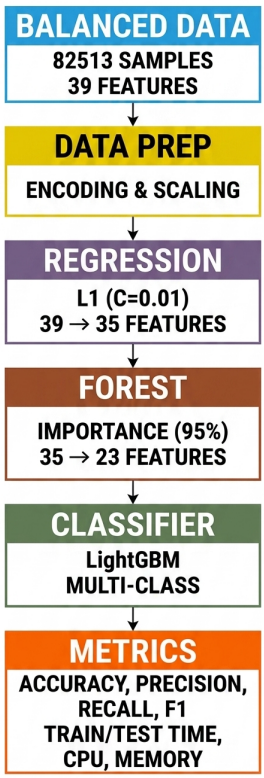

FIGURE 3.8: Machine learning pipeline with hybrid feature selection and LightGBM classification.

Figure 3.9 represented the overall architecture of the proposed multiclass intrusion detection system in IoT networks and the class labels. LightGBM as an attack classifier is used to detect these attacks that are mentioned in the diagram more accurately and in less time. We then defined a cumulative feature importance threshold of 95%, in that features were selected until 95% of the importance was reached. This led to the selection of 23 informative features as seen in Figure 3.8.

A LightGBM classifier for multi-class classification is trained using the chosen features, and the model's performance is assessed using metrics for accuracy, precision, recall, and F1-score.

After handling the class imbalancing issue ,we have 82513 samples, and then on the balanced dataset we evaluated the pipeline to ensure the better results in terms of accuracy and efficiency.

For IoT contexts, the framework's design maintains a balance between computational efficiency and detection performance.

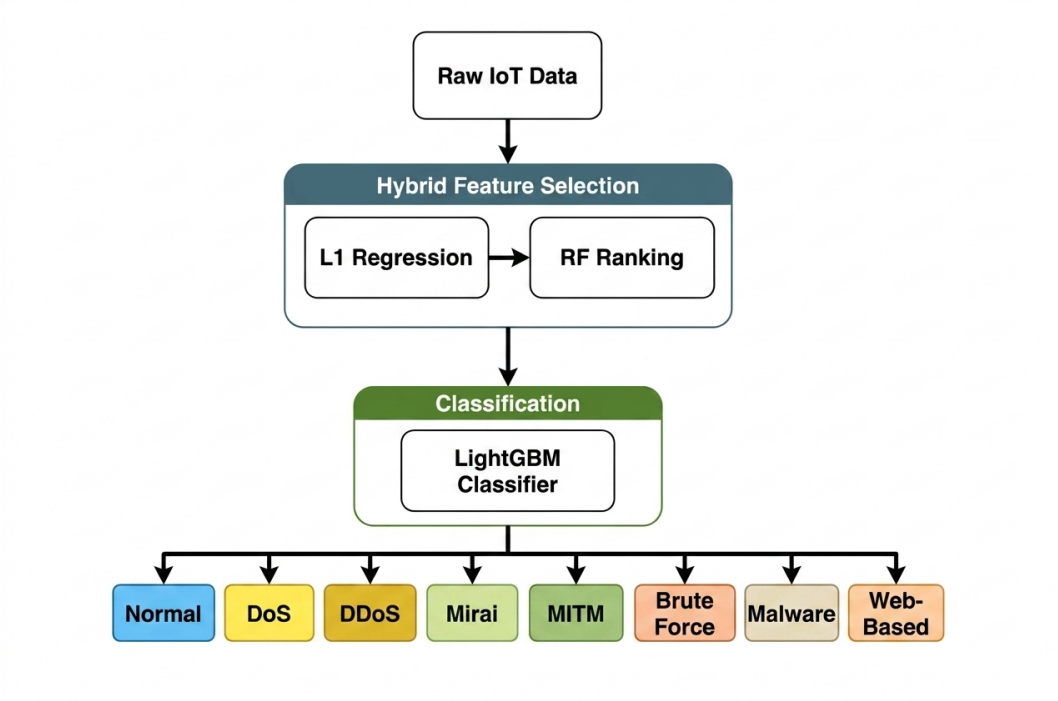

FIGURE 3.9: Proposed hybrid feature selection framework for IoT attack classification.

3.5.1 L1-Regularized Logistic Regression for Feature Selection

The first feature selection step in this study is Logistic Regression with L1 regularization.Before using the second-stage feature ranking (Random Forest), this step aims to eliminate features from the balanced IoT dataset that are redundant or less useful.

L1 regularization is helpful because it creates a sparse model, which means that some feature coefficients become absolutely zero, which is helpful because IoT datasets frequently contain correlated and noisy features. Features are considered irrelevant and eliminated when their coefficients drop to zero.

3.5.1.1 Input to the L1-Logistic Regression Stage

The L1-Logistic Regression process takes as input a balanced set of training data comprised of feature vectors and labels. This can be expressed as:

$$\mathbf{X} \in \mathbb{R}^{n \times d}, \quad \mathbf{y} \in \{0, 1, \ldots, K-1\}^n \tag{3.2}$$

In this case, $\mathbf{X}$ is a matrix of features, with n being the number of samples and d being the number of features.

The vector $\mathbf{y}$ corresponds to the labels for each sample, and K is the number of classes. In this study, $n = 82{,}531$, $d = 39$, and $K = 8$.

After preprocessing (encoding and scaling), $\mathbf{X}$ is used as input to the feature selection model.

3.5.1.2 Logistic Regression Model

For multi-class classification, Logistic Regression applies the softmax function. The probability of class k given the input sample $\mathbf{x}_i$ is given by:

$$P(y_i = k \mid \mathbf{x}_i) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x}_i + b_k)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x}_i + b_j)} \tag{3.3}$$

Here, $\mathbf{w}_k$ is the weight vector and b_k is the bias for class k.

The expression in the numerator represents the score for class k, and the denominator normalizes the scores to obtain probabilities for all classes.

3.5.1.3 Objective Function with L1 Regularization

The model parameters are learned by minimizing the negative log-likelihood loss with an L1 penalty:

$$\min_{\{\mathbf{w}_k, b_k\}_{k=1}^{K}} \left[-\frac{1}{n} \sum_{i=1}^{n} \log P(y_i \mid \mathbf{x}_i) + \lambda \sum_{k=1}^{K} \|\mathbf{w}_k\|_1 \right] \tag{3.4}$$

Sparsity is promoted by the L1 term $\|\mathbf{w}_k\|_1 = \sum_{m=1}^{d} |w_{k,m}|$, which shrinks less

significant coefficients to precisely zero. Only a subset of characteristics retain non-zero coefficients as a result, and these features are thought to be crucial for categorization.

The objective function has two main parts. The first term, $-\frac{1}{n}\sum_{i=1}^{n}\log P(y_i \mid x_i)$, measures how well the model predicts the correct class labels and helps improve classification accuracy. The second term, $\lambda\sum_{k=1}^{K}\|w_k\|_1$, uses L1 regularization, which reduces the weights of less useful features to zero. The parameter λ controls the trade-off between model simplicity and prediction accuracy, producing a lightweight and effective model..

In this study, parameter C is used to regulate the regularization strength, where:

$$\lambda \propto \frac{1}{C} \tag{3.5}$$

A smaller value of C means stronger regularization and more sparsity.

3.5.1.4 Feature Selection Rule

Following the L1-regularized Logistic Regression model's training, a feature m is chosen if it contains at least one non-zero coefficient among the class weight vectors:

$$\text{Select feature } m \text{ if } \max_{k\in\{1,\ldots,K\}} |w_{k,m}| > 0 \tag{3.6}$$

Features that satisfy Equation 3.6 are kept, while all other features are removed.

3.5.1.5 Output of Stage 1

Using L1-regularized Logistic Regression with $C = 0.01$, the feature set is reduced as follows:

- Input features: 39
- Selected features after L1 Logistic Regression: 35

This stage's primary advantage is the early removal of weak features, which lowers computing costs for subsequent stages and can enhance generalization by lowering

noise. Due to its ability to induce sparsity, logistic regression with L1 regularization is frequently utilized for feature selection.

TABLE 3.1: Stage 1 feature selection using L1-regularized Logistic Regression in this research.

Component	**Description**
Input data	Balanced dataset (n = 82,531, d = 39 features, $K = 8$ classes)
Preprocessing	Encoded and scaled feature matrix $\mathbf{X}$
Model	Multi-class Logistic Regression
Regularization	L1 penalty (sparse coefficients)
Hyperparameter	$C = 0.01$
Selection rule	Keep feature if $\max_k \lvert w_{k,m} \rvert > 0$ (Eq. 3.6)
Output	Reduced feature set: 35 features

3.5.2 Random Forest Feature Ranking and Final Feature Set

Random Forest is used for a second feature selection step following the first feature reduction using L1-regularized Logistic Regression. In order to further refine the chosen features, this stage aims to rank them in order of significance, keeping only the most informative features.

Because Random Forest can capture non-linear correlations and interactions between characteristics, which are prevalent in IoT network traffic data, it is selected for this step.

3.5.2.1 Input to the Random Forest Stage

This phase uses the smaller feature set of the L1-regularized Logistic Regression as the input. There are 35 features instead of 39 after Stage 1. These 35 features are used to train a Random Forest model for feature significance evaluation. Based on the decrease in impurity across decision trees, the Random Forest algorithm assesses each feature's contribution. Higher relevance features are preferred in the

final feature subset because they help to attack classification more successfully.

3.5.2.2 Random Forest Feature Importance

Based on the impurity reduction each feature achieves when used to divide decision tree nodes, Random Forest determines the relevance of each feature.Impurity is often quantified with the Gini index for classification tasks.

For a node t, the Gini impurity is defined as:

$$G(t) = 1 - \sum_{k=1}^{K} p_k^2 \tag{3.7}$$

where p_k is the proportion of samples of class k at node t.

When a node t is split using feature m into left child t_L and right child t_R, the decrease in impurity is:

$$\Delta G(t, m) = G(t) - \frac{N_{t_L}}{N_t} G(t_L) - \frac{N_{t_R}}{N_t} G(t_R) \tag{3.8}$$

where N_t is the number of samples at node t. The importance of feature m in a single decision tree is computed as the sum of impurity reductions over all nodes where the feature is used:

$$I_m^{(tree)} = \sum_{t \in T_m} \Delta G(t, m) \tag{3.9}$$

where T_m is the set of nodes split using feature m.

In a Random Forest with B trees, the final importance score of feature m is obtained by averaging over all trees:

$$I_m = \frac{1}{B} \sum_{b=1}^{B} I_m^{(b)} \tag{3.10}$$

The top 23 characteristics are displayed in Figure 3.10 in order of their Random Forest importance ratings.

Strong discriminative strength is demonstrated by the fact that features like *Tot sum*, *Rate*, and *AVG* contribute the most to impurity reduction.

Features with lower rankings contribute increasingly less, and their evaluation is thereafter based on cumulative relevance.

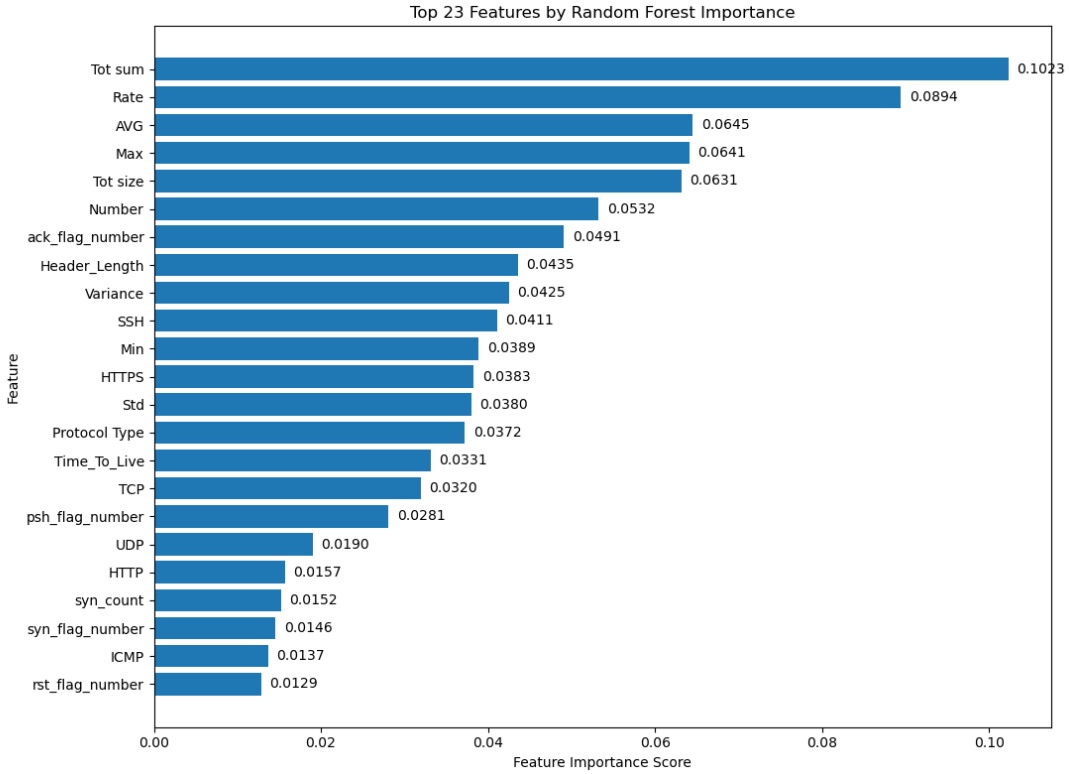

FIGURE 3.10: Top 23 features ranked by Random Forest importance scores.

3.5.2.3 Cumulative Feature Importance and Selection Rule

Every input feature in Random Forest is assigned a relevance score that is determined by the overall impurity reduction attained while using that feature for node splitting. Individual significance scores do not immediately address the issue of how many features should be kept for the finished model, even though they do show the relative contribution of each feature. Let the feature importance scores obtained from Random Forest be:

$$\{I_1, I_2, \ldots, I_d\} \tag{3.11}$$

where d is the number of input features to the Random Forest model. The features are first sorted in descending order of importance such that:

$$I_{(1)} \geq I_{(2)} \geq \cdots \geq I_{(d)} \tag{3.12}$$

The cumulative importance after selecting the top k features is defined as:

$$C_k = \frac{\sum_{i=1}^{k} I_{(i)}}{\sum_{j=1}^{d} I_{(j)}} \tag{3.13}$$

The 35 characteristics derived from L1-regularized Logistic Regression are given Random Forest in this study. After sorting the important scores in descending order, Equation 3.13 is used to determine the cumulative importance C_k progressively for $k = 1, 2, \ldots, 35$.

Cumulative importance plays a crucial role in feature selection for three main reasons:

- It offers a comprehensive perspective on the distribution of importance among features.
- It does not rely on manual tuning or intuition to choose an arbitrary number of features.
- It allows for the automatic selection of a small feature subset while maintaining the majority of the predictive data

This study uses a 95% cumulative importance level. This criterion indicates that at least 95% of the Random Forest-estimated overall relevance can be explained by the chosen features taken together. In feature selection research, the 95% value is frequently utilized due to its robust trade-off between information retention and dimensionality reduction.By letting the data decide how many features are adequate, the threshold-based approach eliminates the need to manually change the feature count.

Random Forest chooses the smallest feature subset that meets the 95% importance threshold, while L1-regularized Logistic Regression eliminates weak features through coefficient sparsity. Formally, the final feature subset is selected by choosing the smallest value of k such that:

$$C_k \geq 0.95 \tag{3.14}$$

This criterion gives $k = 23$ features in this study. The LightGBM classifier uses these 23 features as its final input. The Random Forest model's cumulative feature significance curve is displayed. The vertical dashed line shows the number of traits that were chosen, while the horizontal dashed line shows the 95% importance criterion.

The graphic illustrates how the cumulative relevance of the top-ranked traits rises quickly before saturating gradually.

At $k = 23$, the curve crosses the 95% barrier, meaning that the majority of the predictive value is captured by the top 23 traits taken combined. Beyond this threshold, features have a negligible contribution and are hence not included in the final feature set.

This selection process, which is based on cumulative relevance, ensures that feature reduction is not dictated by human adjustment.

Instead, the suggested hybrid feature selection approach is robust and repeatable since the threshold-based technique automatically adjusts to the underlying data distribution.

The total contribution of chosen features to the model's overall prediction ability is represented by the cumulative feature significance score. After using L1-regularized Logistic Regression, Random Forest feature significance values were initially computed for each of the characteristics that were chosen for this investigation. Then, based on their importance rankings, the features were sorted in descending order. Following sorting, the important values of each attribute were continually added to determine the cumulative importance.

As more features are added, this method aids in determining how much overall information is kept. In order to keep the most useful traits while eliminating less important ones, a threshold of 95% cumulative importance was chosen for this study. The top 23 features were chosen for the final classification stage based on this criterion.

The cumulative feature importance curve shows that a smaller collection of features can retain the majority of the original dataset's predictive power and offers a visual depiction of the feature selection procedure.

TABLE 3.2: Summary of cumulative importance based feature selection decision.

Criterion	**Description**
Initial feature count	35 (after L1-regularized Logistic Regression)
Importance estimation	Random Forest mean decrease in impurity
Ranking method	Descending order of feature importance
Selection strategy	Cumulative importance threshold-based
Threshold value	95% of total importance
Selected feature count	23
Output usage	Input to LightGBM classifier

3.5.3 LightGBM Classifier

The classification model is trained using the final reduced feature set, which has 23 features, following the completion of the two-stage feature selection process.

The last classifier used in this study for multi-class intrusion detection is the Light Gradient Boosting Machine (LightGBM).

LightGBM is a decision tree-based gradient boosting system that boasts low memory usage, quick training times, and great efficiency. Because of these characteristics, it is especially well-suited for Internet of Things settings, where processing power is frequently constrained.

LightGBM is selected for three main reasons:

- It uses histogram-based splitting to handle huge datasets efficiently.
- It has native capability for multi-class classification.
- It maintains a low computational cost while offering good prediction performance.

LightGBM sequentially constructs an ensemble of decision trees. By fitting the loss's negative gradient, a new tree is trained at each iteration to minimize the loss function.

LightGBM reduces the multi-class cross-entropy loss for multi-class classification with K classes:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\mathbb{I}(y_i = k)\log\left(p_{i,k}\right) \tag{3.15}$$

where $p_{i,k}$ is the predicted probability of class k for sample i, and $\mathbb{I}(\cdot)$ is the indicator function. Only the 23 features chosen during the L1-regularized

Logistic Regression and Random Forest cumulative significance stages are used to train LightGBM in the suggested hybrid IDS.

The classifier avoids learning from redundant or noisy features by limiting the input to the most informative features, hence concentrating on pertinent traffic patterns.

The LightGBM classifier creates a probability distribution across all attack categories for every network instance. The possibility that the given network traffic falls into a certain class is represented by each probability value.

The classifier compares the incoming traffic features with the learnt decision patterns that were assessed during the training phase.

The final predicted class is determined by selecting the class with the highest expected probability.The model can successfully differentiate between benign and malevolent network behavior in IoT contexts thanks to this prediction technique. Several attack types that represent both benign and malicious traffic classes make up the model's output layer. The output classes include benign traffic and several IoT attack types, such as Brute Force, DDoS, DoS, Mirai, Recon, Spoofing, and Web-based assaults.

Following the label consolidation procedure on the CICIoT2023 dataset, these attack categories were chosen.

Instead of just identifying whether traffic is malicious or regular, the suggested system may simultaneously recognize many attack patterns thanks to LightGBM's multi-class classification feature. This enhances the intrusion detection system's usefulness in real-world IoT settings where several attack kinds could happen simultaneously.

TABLE 3.3: Configuration of the LightGBM classifier used in the proposed hybrid IDS.

Component	**Description**
Input features	23 selected features
Learning framework	Gradient boosting decision trees
Loss function	Multi-class cross-entropy
Classification type	Multi-class
Output	Predicted attack class
Design objective	High accuracy with low computational cost

3.5.3.1 Model Hyperparameters

We fine-tuned the hyperparameters of the LightGBM model to enhance its performance. The chosen hyperparameters are:

- **Learning Rate:** 0.05
- **Number of Estimators:** 100
- **Max Depth:** -1
- **Number of Leaves:** 31
- **Feature Fraction:** 0.8
- **Bagging Fraction:** 0.8
- **Random State:** 42

These settings were chosen to ensure a good trade-off between accuracy and speed, and to prevent overfitting. In conclusion, the proposed hybrid intrusion detection system (IDS) combines a LightGBM classifier with two-stage feature selection to provide effective and precise multi-class intrusion detection for Internet of Things settings.

3.5.4 Feature Reduction Summary Table

Table 3.4 summarizes how the feature count is reduced in the proposed pipeline.

TABLE 3.4: Feature reduction across the proposed hybrid IDS pipeline.

Stage	Method	Number of Features
Initial	Final dataset features	39
Stage 1	Logistic Regression (L1), $C = 0.01$	35
Stage 2	Random Forest cumulative importance (95%)	23
Final	LightGBM input features	23

3.5.5 Evaluation Metrics

Two different types of metrics are employed to evaluate the suggested intrusion detection methodology: (i) Measurements related to detection performance (accuracy), and (ii) Metrics related to computational efficiency. The trade-off between resource usage and detection efficacy in IoT systems may be clearly analyzed thanks to this split.

3.5.5.1 Detection Performance Metrics

Detection performance metrics evaluate how accurately the intrusion detection system classifies normal and attack traffic in a multi-class setting. Let $y = \{y_1, y_2, \ldots, y_N\}$ denote the true labels and $\hat{y} = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N\}$ denote the predicted labels for N test samples. Accuracy measures the overall proportion of correctly classified samples and is defined as:

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i = \hat{y}_i) \tag{3.16}$$

where $\mathbb{I}(\cdot)$ is an indicator function that equals 1 when the prediction is correct and 0 otherwise. For each class c, precision is defined as:

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c} \tag{3.17}$$

The macro-averaged precision is computed as:

$$\text{Precision}_{macro} = \frac{1}{C} \sum_{c=1}^{C} \text{Precision}_c \tag{3.18}$$

For each class c, recall is defined as:

$$\text{Recall}_c = \frac{TP_c}{TP_c + FN_c} \tag{3.19}$$

The macro-averaged recall is computed as:

$$\text{Recall}_{macro} = \frac{1}{C} \sum_{c=1}^{C} \text{Recall}_c \tag{3.20}$$

The macro-averaged F1-score is calculated as:

$$\text{F1}_{macro} = \frac{2 \times \text{Precision}_{macro} \times \text{Recall}_{macro}}{\text{Precision}_{macro} + \text{Recall}_{macro}} \tag{3.21}$$

For multi-attack intrusion detection scenarios, it is crucial to apply macro-averaging to ensure that each attack class contributes equally to the evaluation.

3.5.5.2 Computational Efficiency

Efficiency, in the context of intrusion detection, means the "capability of a model to train and make predictions with minimum computational cost in terms of time, CPU, and memory usage." This is crucial in the IoT environment due to the resource constraints of the devices [59]

The following metrics were used to measure efficiency:

- **Training Time:** Time required to train the model.
- **Testing Time:** Time required to make predictions.

- **Total Time:** Time needed to train and test.

- **CPU Usage:** Average load on the processor.

- **Memory Usage:** RAM used while the model is running.

These measures offer a holistic assessment of the model's IoT readiness.

Let t_0 represent the training start time, t_1 the training end time, and t_2 the testing end time.

$$T_{train} = t_1 - t_0 \tag{3.22}$$

$$T_{test} = t_2 - t_1 \tag{3.23}$$

$$T_{overall} = t_2 - t_0 \tag{3.24}$$

Throughout training and testing, CPU usage is continuously tracked.

Let sampling CPU utilization values be represented by $\{u_1, u_2, \ldots, u_M\}$.

$$CPU_{avg} = \frac{1}{M} \sum_{i=1}^{M} u_i \tag{3.25}$$

$$CPU_{max} = \max(u_i) \tag{3.26}$$

Let RSS_{before} be the resident memory usage before training, and $RSS(t)$ be the sampled memory consumption during execution.

The highest rise in RSS during the training and testing of the model is used to calculate the peak memory use.

$$\Delta M_{peak} = \max(RSS(t)) - RSS_{before} \tag{3.27}$$

To estimate deployment cost, each trained pipeline is serialized to disk. The model size is computed as:

$$\text{Model Size (MB)} = \frac{\text{Serialized file size (bytes)}}{1024^2} \tag{3.28}$$

These efficiency measures make it possible to evaluate the proposed intrusion detection method's deployability on IoT and edge devices with limited resources, where low latency, a small memory footprint, and less CPU consumption are crucial.

3.6 Chapter Summary

This chapter described the entire research approach used for this investigation. The datasets used and the data preparation procedures such as feature encoding, scaling, class merging, data cleaning, and managing class imbalance were covered. Following that, a hybrid feature selection framework was presented, which combined random forest for ranking and choosing the most significant features with L1-regularized logistic regression for eliminating irrelevant features. A LightGBM classifier was trained for multi-class classification using the chosen features. Lastly, an outline was provided of the assessment measures that were utilized to evaluate computational efficiency and detection performance.

Chapter 4

Results and Discussion

4.1 Chapter Overview

This chapter demonstrates all experiments and results that have been implemented in this research.First the class imbalancing issue is resolved and then we have a balanced dataset .And on the balanced dataset ,six total experiments are conducted ,and after each experiment the results and findings are discussed there.

4.2 Experimental Setup

Each experiment was carried out locally on a CPU-based system without the use of GPU acceleration or cloud platforms. The machine was equipped with an Intel 64 Family 6 Model 142 Genuine Intel processor with four logical cores and 7.9 GB of RAM. Jupyter Notebook in the Anaconda distribution environment was used to carry out the Python experiments. All machine learning models, including Random Forest, LightGBM, and Logistic Regression, were run using multi-threaded CPU processing. During local execution, all performance-related parameters were measured, including CPU utilization, memory consumption, training time, and testing time. During the trial, neither distributed computing frameworks nor external servers were utilized. In order to replicate a realistic resource-constrained environment akin to real-world IoT edge systems, where advanced computational resources are frequently unavailable, this experimental design was chosen. It is

also possible to more precisely assess the computational efficiency and lightweight nature of the suggested intrusion detection framework by conducting experiments in a CPU-only setting.

4.3 Dataset Before Preprocessing

The complete dataset before any handling class imbalancing and merging classes is mentioned in Table 4.1. With **712,311 samples**, **39 features**, and **33 classes**, the dataset offers enough information for a trustworthy model evaluation. The **80%/20%** train–test split maintains an independent test set for equitable performance evaluation while guaranteeing sufficient data for training. Therefore, this dataset configuration is appropriate for comparing the efficiency and accuracy of detection across various models.

TABLE 4.1: Dataset summary.

Item	**Result**
Dataset name	CICIoT2023
Final total samples	712,311
Total features	40
Total Attack Classes	33
Train/Test split	80% / 20%
Training samples	569,850
Testing samples	142,462

4.3.1 Classifier Performance on Original Imbalanced Dataset

The performance of several classifiers on the original unbalanced dataset is reported in Table 4.2. Given that the majority classes dominate in the data, all models perform exceptionally well in binary classification.

However, 8-class and 33-class classifications result in far worse performance, particularly when it comes to F1-score. This demonstrates the negative impact of class imbalance on multi-class intrusion detection by showing that minority attack classes are poorly trained.

TABLE 4.2: Classifier performance metrics on the original (imbalanced) dataset.

Classifier	Class Type	Accuracy	Precision	Recall	F1-score
TabNet	2-class	0.99	0.99	0.99	0.99
TabNet	8-class	0.84	0.83	0.84	0.81
TabNet	33-class	0.76	0.78	0.76	0.74
LightGBM	2-class	0.99	0.99	0.99	0.99
LightGBM	8-class	0.85	0.84	0.85	0.83
LightGBM	33-class	0.77	0.77	0.77	0.75
Random Forest	2-class	0.99	0.99	0.99	0.99
Random Forest	8-class	0.85	0.83	0.85	0.83
Random Forest	33-class	0.78	0.78	0.78	0.77

Class imbalance handling is required to enhance minority class detection because of the mentioned performance decrease in multi-class environments. The methods used to deal with this problem are covered in the next section.

4.3.2 Merged Class Distribution Result

33 different attack labels and one benign label were initially present in the dataset. Similar attacks were combined into 8 categories to make the classification work easier.

Table 4.3 demonstrates how the **Benign** class is maintained apart, while the initial 33 attack labels are combined into **8 attack classes**. The learning process is made simpler and less redundant by this label condensation. After combining the original labels into eight classes, the final class distribution is shown in Table 4.4. It is clear that DDoS and DoS attacks dominate the dataset, but there are very few samples in classes like Web-based and Brute Force. Class imbalance treatment is necessary prior to model training because this extreme imbalance may have a detrimental effect on minority class detection.

As shown in Figure 4.1, the majority of samples belong to the DDoS attack class, followed by DoS and Mirai attacks.

4.3.3 Class Imbalancing

The dataset is extremely unbalanced, according to the merged class distribution in Table 4.4. As a result, class imbalance handling is used prior to model training.

TABLE 4.3: Mapping of original labels (33 attacks + benign) into final 8 classes.

Final Class	**Original Labels Merged**
Benign	BENIGN
DDoS	DDOS-ICMP_FLOOD, DDOS-UDP_FLOOD, DDOS-TCP_FLOOD, DDOS-PSHACK_FLOOD, DDOS-RSTFINFLOOD, DDOS-SYN_FLOOD, DDOS-SYNONYMOUSIP_FLOOD, DDOS-ICMP_FRAGMENTATION, DDOS-ACK_FRAGMENTATION, DDOS-UDP_FRAGMENTATION, DDOS-HTTP_FLOOD, DDOS-SLOWLORIS
DoS	DOS-UDP_FLOOD, DOS-TCP_FLOOD, DOS-SYN_FLOOD, DOS-HTTP_FLOOD
Mirai	MIRAI-GREETH_FLOOD, MIRAI-UDPPLAIN, MIRAI-GREIP_FLOOD
Recon	RECON-HOSTDISCOVERY, RECON-OSSCAN, RECON-PORTSCAN, RECON-PINGSWEEP
Spoofing	DNS_SPOOFING, MITM-ARPSPOOFING
Web-based	SQLINJECTION, XSS, COMMANDINJECTION, BROWSERHIJACKING
Brute Force	DICTIONARYBRUTEFORCE, UPLOADING_ATTACK

TABLE 4.4: Final class distribution after label consolidation (8 classes).

Class	**Samples**	**Percentage (%)**
DDoS	515,120	72.31
DoS	122,495	17.20
Mirai	39,664	5.57
Benign	16,577	2.33
Spoofing	7,328	1.03
Recon	4,757	0.67
Web-based	300	0.04
Brute Force	224	0.03
Total	**712,312**	**100.00**

The imbalance ratio (IR), which measures imbalance, is calculated as follows: With $N_{\max}$ representing the number of samples in the majority class and $N_{\min}$ representing the number of samples in the minority class.

$$IR = \frac{N_{\max}}{N_{\min}} \tag{4.1}$$

Our dataset's $IR \approx 2290.71$ indicates a significant imbalance, with DDoS ($N_{\max} =$ 515,120) as the majority class and Brute Force ($N_{\min} = 224$) as the minority class.

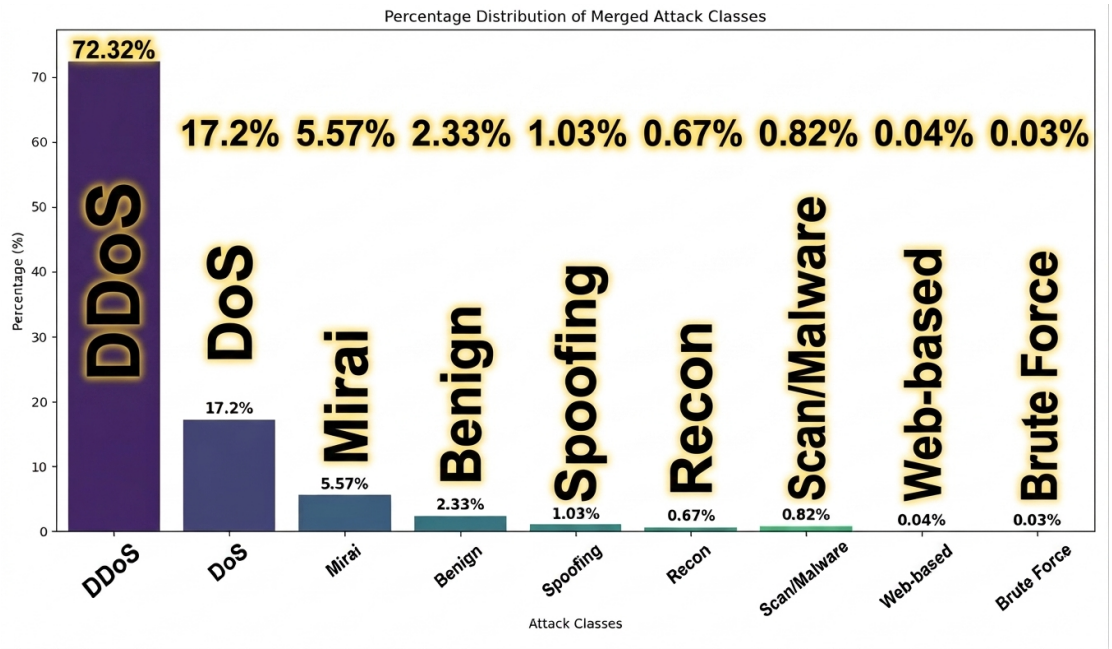

FIGURE 4.1: Percentage distribution of merged attack classes after label consolidation.

TABLE 4.5: Imbalance metrics after label consolidation (before balancing).

Metric	**Value**
Majority class ($N_{\max}$)	DDoS (515,120)
Minority class ($N_{\min}$)	Brute Force (224)
Imbalance Ratio (IR)	2290.71

Three balancing strategies—SMOTE, Class Weight balancing, and SMOTE-ENN were evaluated in order to reduce class imbalance.

4.3.4 Results of SMOTE Balancing

The impact of SMOTE on the dataset's class-wise distribution is displayed in Table 4.6.

By creating synthetic instances, SMOTE significantly increases the quantity of samples for minority classes while maintaining the majority classes.

This eliminates bias toward the majority class and enhances class balance.SMOTE, however, has the potential to add synthetic data, which can affect classifier performance.

TABLE 4.6: Class-wise sample distribution before and after applying SMOTE balancing

Class Label	Attack Type	Before Balancing	After SMOTE
0	Benign	16577	16577
1	Brute Force	500	19500
2	DDoS	515120	515120
3	DoS	122495	122495
4	Mirai	39664	39664
5	Reconnaissance	10603	50603
6	Spoofing	7328	10200
7	Web-based	500	19800

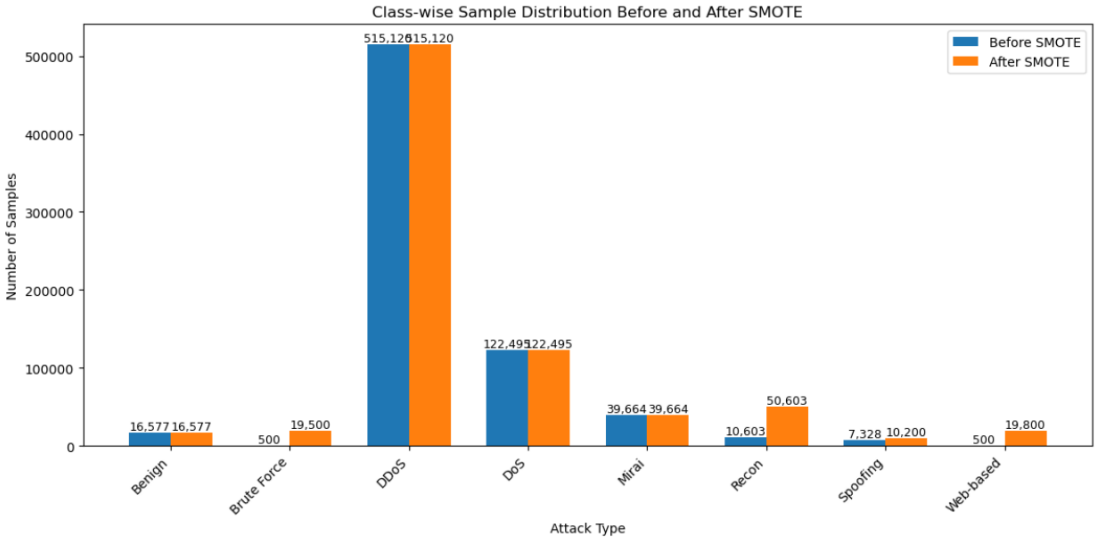

FIGURE 4.2: Class-wise distribution of samples after label merging

Figure 4.2 shows how common DDoS and DoS attacks are, and how rare minority attack classes are.

4.3.5 Results of Class Weight Balancing

Without changing the size of the dataset, class weight balancing gives minority classes more weight during training. Minority classes are given larger weights in order to enhance their effective contribution, as indicated in Table 4.7. Despite increasing sensitivity to infrequent attacks, this method may lead to irregular learning behavior and uneven performance on various evaluation requirements. The effective contribution of each attack class following the application of class weight balancing is shown in Figure 4.3. By giving minority classes more weights,

TABLE 4.7: Class weights and effective contribution for handling imbalance

Class Label	Attack Type	Samples	Class Weight	Effective Contribution
0	Benign	16577	0.21	3481
1	Brute Force	224	15.52	3476
2	DDoS	515120	0.01	5151
3	DoS	122495	0.04	4899
4	Mirai	39664	0.13	5156
5	Reconnaissance	10603	0.49	5195
6	Spoofing	7328	0.71	5200
7	Web-based	300	17.30	5190

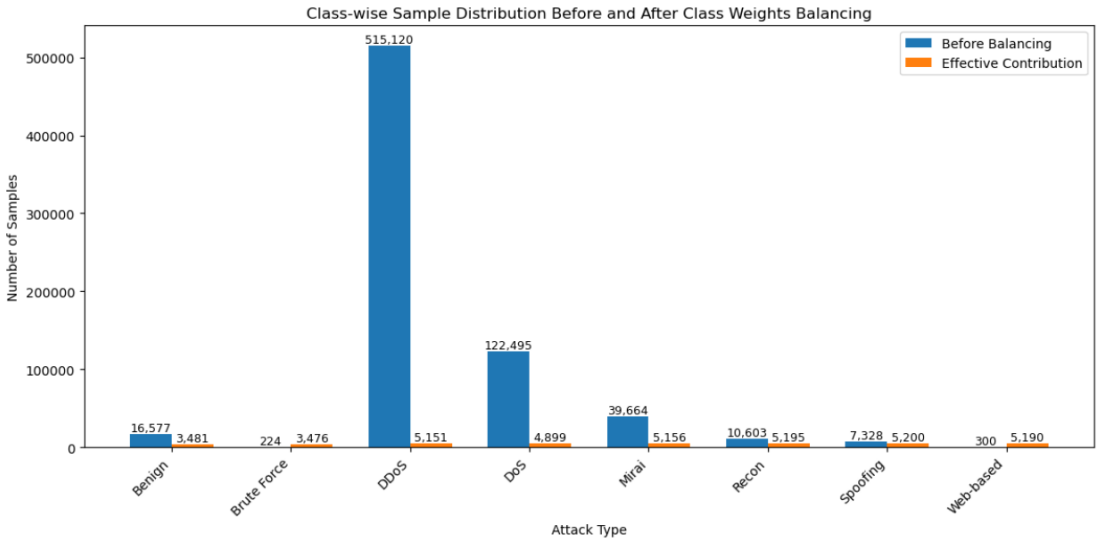

FIGURE 4.3: Effective contribution of each class after applying class weight balancing

the training process amplifies their impact without changing the initial sample size. Consequently, the effective contributions from various classes are comparatively equal.Even when balanced contributions are obtained, noisy or overlapping samples are not eliminated by class weight balancing. The evaluation metrics show that this restriction may result in uneven classification performance and weak decision boundaries.

4.3.6 Results of SMOTE-ENN

The class-wise sample distribution before and after applying the SMOTE-ENN balancing technique is illustrated in Fig. 4.4.

Prior to balancing, the dataset exhibits severe class imbalance, characterized by a small number of samples in minority attack classes and a dominance of majority classes such as DDoS and DoS.

Table 4.8 provides a summary of the quantitative changes in class distribution before and after balancing. The outcomes demonstrate that SMOTE-ENN successfully reduces class imbalance without going overboard with oversampling, offering a solid basis for later feature selection and classification phases. After applying SMOTE-ENN, the dataset becomes significantly more balanced. The Edited Nearest Neighbour (ENN) component removes noisy and ambiguous samples from majority classes, while SMOTE synthetically oversamples minority classes. This process produces a cleaner and more representative dataset, which improves class separability and enhances classifier generalization capability.

TABLE 4.8: Class-wise distribution before and after applying SMOTE-ENN balancing

Class Label	Attack Type	Before	After Undersampling	After SMOTE-ENN
0	Benign	16577	16577	5390
1	Brute Force	224	224	13018
2	DDoS	515120	10000	9589
3	DoS	122495	10000	9343
4	Mirai	39664	10000	16456
5	Reconnaissance	10603	10603	7018
6	Spoofing	7328	7328	8856
7	Web-based	300	300	12861

4.3.7 Class Distribution Before and After Data Balancing with SMOTE-ENN

Table 4.9 compares the performance of Random Forest and LightGBM classifiers under different imbalance control algorithms. While SMOTE and class weight balancing provide only modest or inconsistent improvements, SMOTE-ENN consistently achieves the highest accuracy, precision, recall, and F1-score for both classifiers. This demonstrates how noise reduction and oversampling are more effective when combined than when used independently. In addition to increasing the representation of minority classes, the combination of SMOTE and Edited

Nearest Neighbor (ENN) eliminates noisy and incorrectly classified samples from the dataset. Consequently, more discriminative and balanced decision limits can be learned by the classifiers. The experimental results further show that LightGBM's effective gradient boosting mechanism and optimized tree learning technique regularly outperform Random Forest on the majority of evaluation measures. The Random Forest classifier, on the other hand, performs significantly worse, especially in minority attack classes where class imbalance has a greater effect.

TABLE 4.9: Attack classifier performance with different load balancing techniques

Classifier	Balancing Method	Accuracy	Precision	Recall	F1-score
LightGBM	Without balancing	0.85	0.84	0.85	0.83
	SMOTE	0.79	0.77	0.79	0.78
	Class Weights	0.77	0.74	0.47	0.51
	SMOTE-ENN	0.97	0.97	0.97	0.97
Random Forest	Without balancing	0.85	0.83	0.85	0.83
	SMOTE	0.78	0.76	0.78	0.76
	Class Weights	0.75	0.48	0.48	0.84
	SMOTE-ENN	0.98	0.98	0.98	0.98

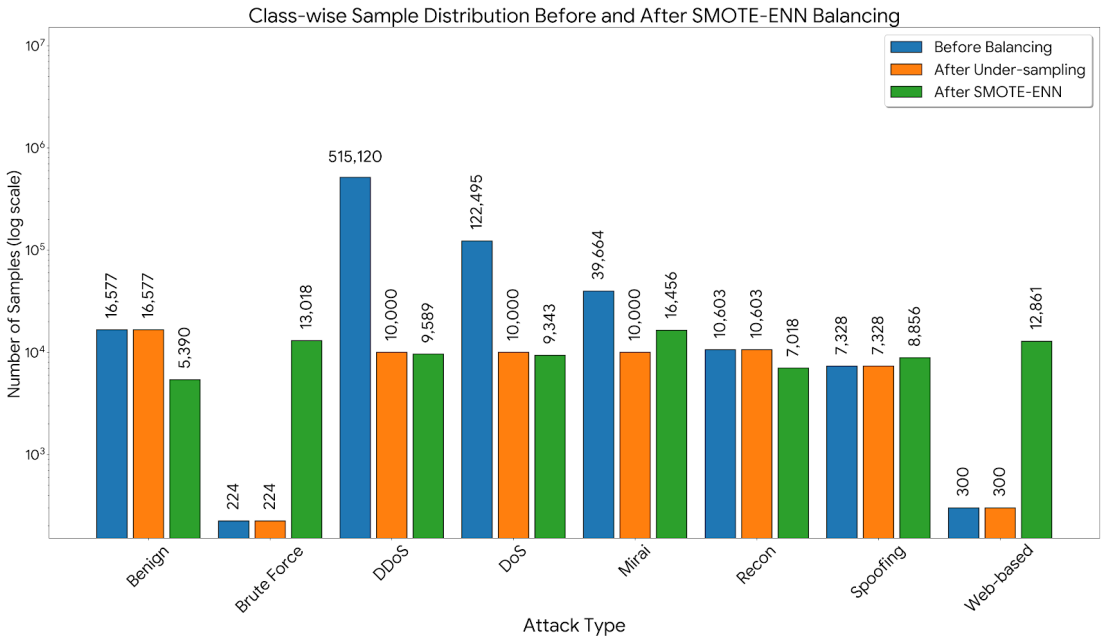

FIGURE 4.4: Class-wise sample distribution before and after applying SMOTE-ENN balancing with LightGBM.

4.3.8 Impact of Class Imbalance Techniques on Model Performance

According to the experimental findings as shown in Figure 4.5 , SMOTE-ENN performs better on each evaluation measures than both SMOTE and class weight balancing. Its capacity to balance the dataset while eliminating ambiguous and noisy samples is responsible for the better performance. Consequently, SMOTE-ENN is chosen as the best imbalance handling for the proposed intrusion detection system. Now the dataset is fully balanced .In the next section we will perform different experiments on the balanced dataset to check the accuracy and efficiency metrics.

4.4 Experimental Evaluation of Lightweight Hybrid Intrusion Detection system

The core contribution of this study is the development of a **lightweight hybrid intrusion detection system** that maintains a balance between computational efficiency and detection performance, an aspect that is often overlooked in current IDS solutions. Although a lot of research focuses on complex or resource-intensive models, the suggested framework combines Logistic Regression (L1),random forest feature selection, and LightGBM as an attack classifier to offer an efficient and compact solution appropriate for actual IoT scenarios.

In order to systematically evaluate the accuracy and efficiency, six experiments are conducted out:

i. Selecting which classifier is best for Lightweight on the balanced dataset

ii. Comparing the proposed hybrid pipeline with a baseline classifier to evaluate improvements.

iii. Testing the strength of the Lightweight pipeline with different classifiers.

iv. Performing an ablation study to identify the contribution of each component.

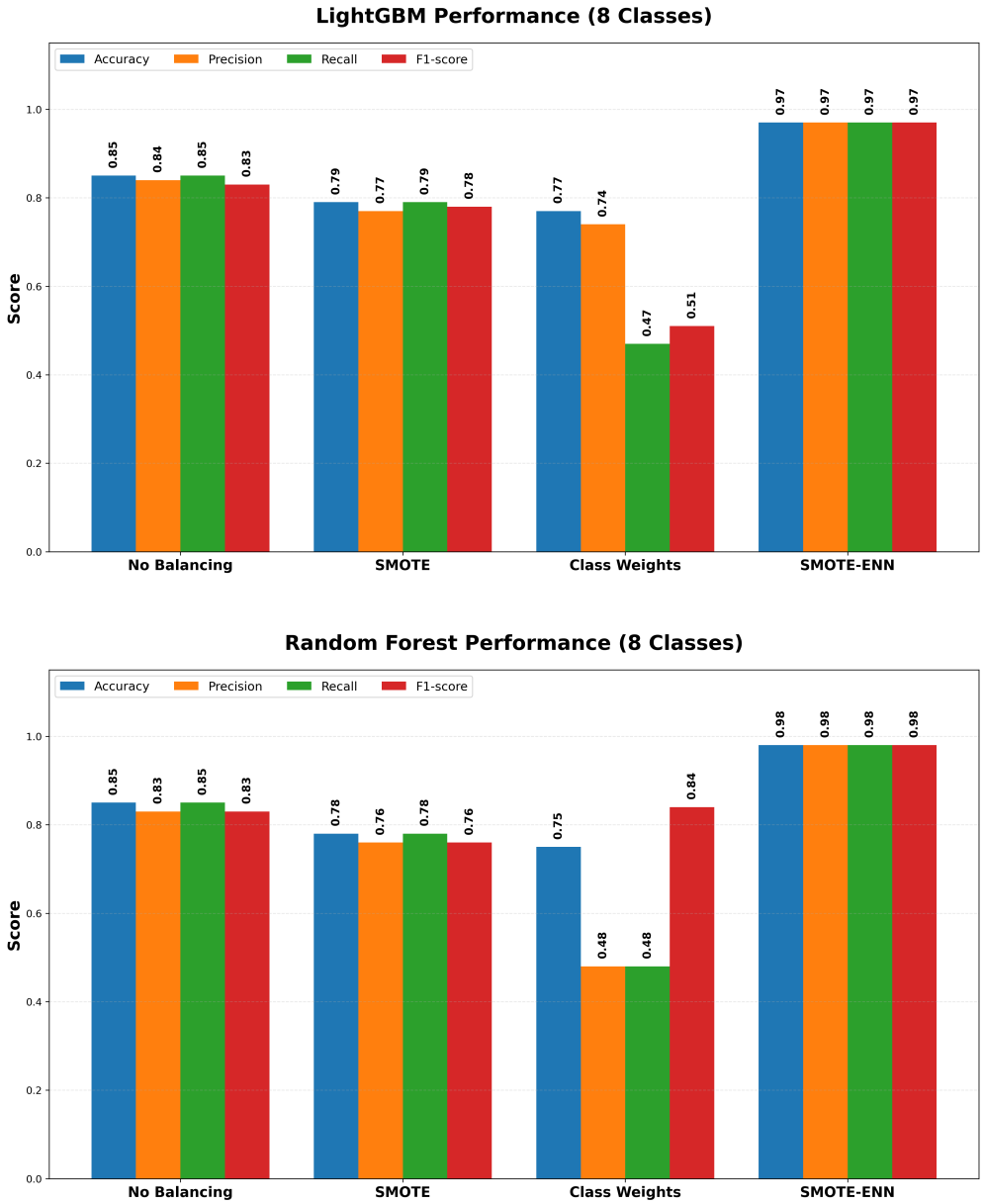

FIGURE 4.5: Performance comparison of LightGBM and Random Forest (8 classes) using different imbalance handling techniques.

v. Analyzing the impact of varying the number of top-ranked features from Random Forest.

vi. Evaluating the proposed Lightweight pipeline against alternative feature selection approaches.

The efficiency and feasibility of the suggested **lightweight hybrid framework** for IoT intrusion detection are demonstrated by this experimental setup.

4.5 Experiment 1: Classifier Selection for IoT Intrusion Detection

In Experiment 1,Multiple machine learning models are trained on the balanced data set and then we evaluated the accuracy and efficiency metrics as shown in Figure 4.6. Random Forest ,XGboost,Decision Tree ,Logistic Regression ,lightGBM and K-Nearest Neighbors (KNN) are the classifier that have been investigated.The same evaluation technique is used in each classifier, and the same efficiency and performance measures are calculated for every model.The goal of this experiment is to check that that which machine learning model is best for maintaining accuracy and efficiency.

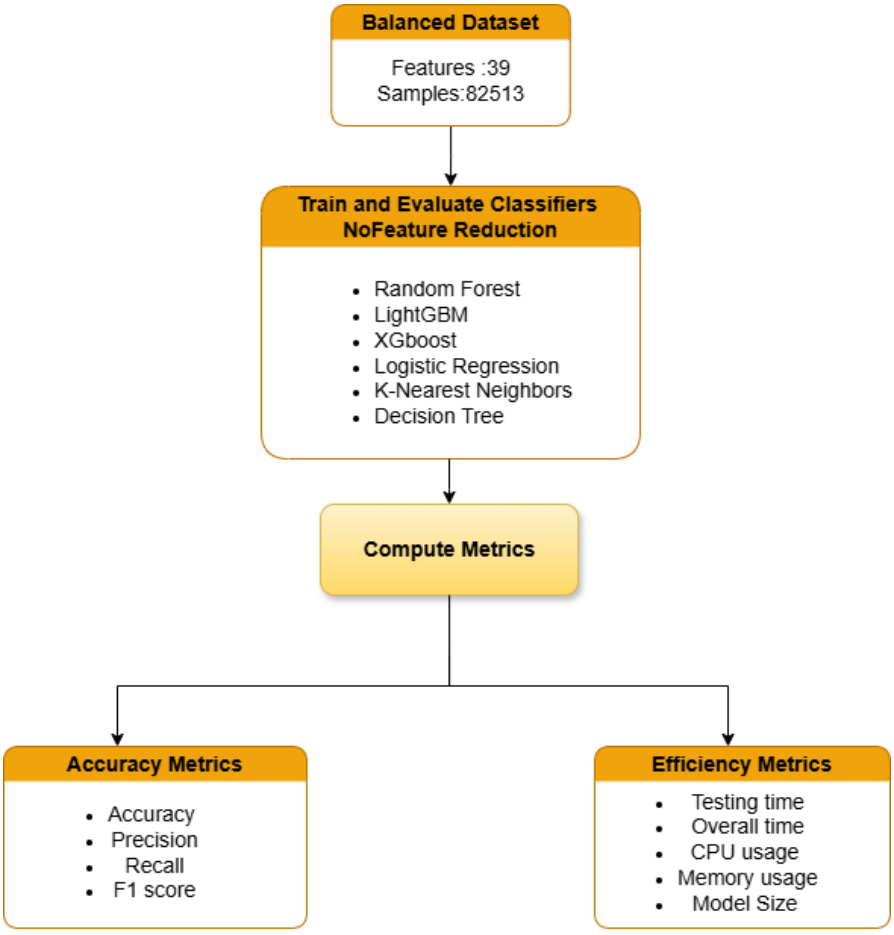

FIGURE 4.6: Workflow of Experiment 1 illustrating classifier comparison using the balanced dataset without feature reduction.

4.5.1 Result of Experiment 1

Several machine learning classifiers were evaluated using the same dataset split and the same performance and efficiency measures to provide a fair comparison.Both computational efficiency and predictive performance were used to evaluate each classifier. Accuracy and the macro-averaged Precision, Recall, and F1-score were used to evaluate accuracy performance. Training time, testing time, total execution time, peak RAM consumption, model size, and CPU utilization were all used to assess efficiency performance. Table 4.10 presents the comparative performance and efficiency metrics of all evaluated classifiers. The results indicate that while several models achieve competitive accuracy,their computational cost varies significantly across CPU usage, memory consumption, and execution time.

TABLE 4.10: Comparing the Accuracy and Efficiency on balanced dataset using all features.

Model	Acc	F1	Train(s)	Test(s)	CPU_{avg}%	RAM(MB)
LightGBM	0.9738	0.9685	22.2071	2.0173	86.41	65.36
Random Forest	0.9760	0.9703	34.7026	0.5975	95.29	327.71
XGBoost	0.9684	0.9626	42.4279	0.5976	95.50	24.32
Decision Tree	0.9570	0.9468	2.3936	0.0164	24.51	31.95
Logistic Regression	0.7872	0.7665	9.8508	0.0518	0.61	43.04
KNN	0.9291	0.9139	0.1049	6.1904	44.28	39.50
Naive Bayes	0.6521	0.5934	0.1442	0.0775	17.87	24.05

4.5.2 Accuracy and Efficiency Metrics

Figures 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, and Table 4.10. provide a baseline comparison of potential classifiers. The best accuracy (0.9760) is obtained by Random Forest, but at a significant memory and model size cost (max RAM delta 327.71 MB), which is not suitable for IoT-devices with limited computational resources.KNN testing time is higher than other machine learning models.Random Forest model size is higher which is 231.04 MB and XGboost consumes more time in training and testing which is 43.03 s.

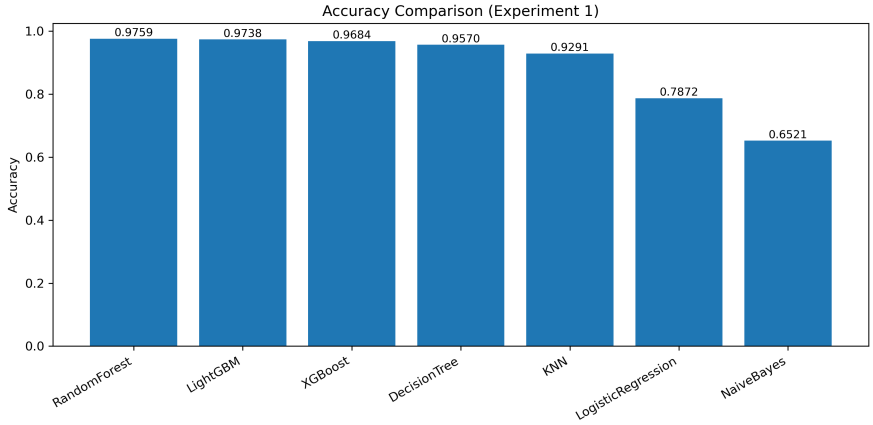

FIGURE 4.7: Comparing the accuracy of various alternative classifiers

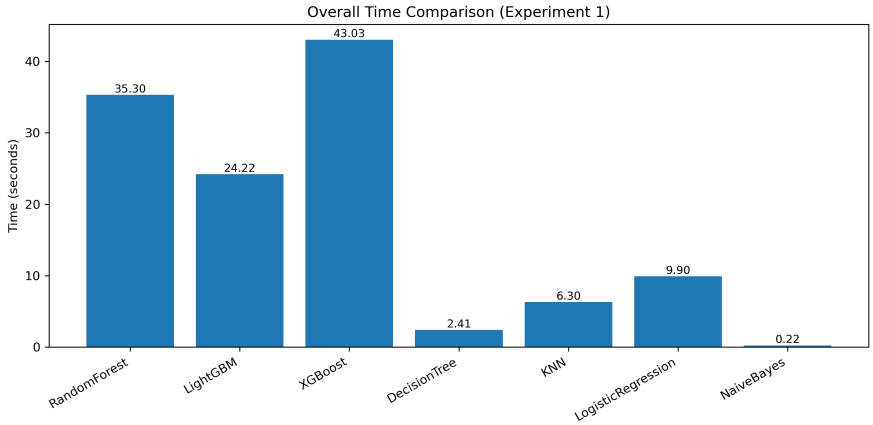

FIGURE 4.8: Overall execution time comparison (training + testing)

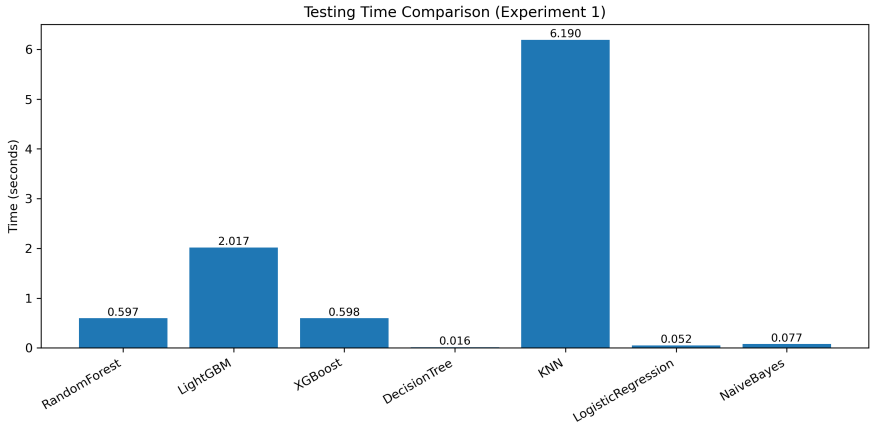

FIGURE 4.9: Testing time comparison across different classifiers

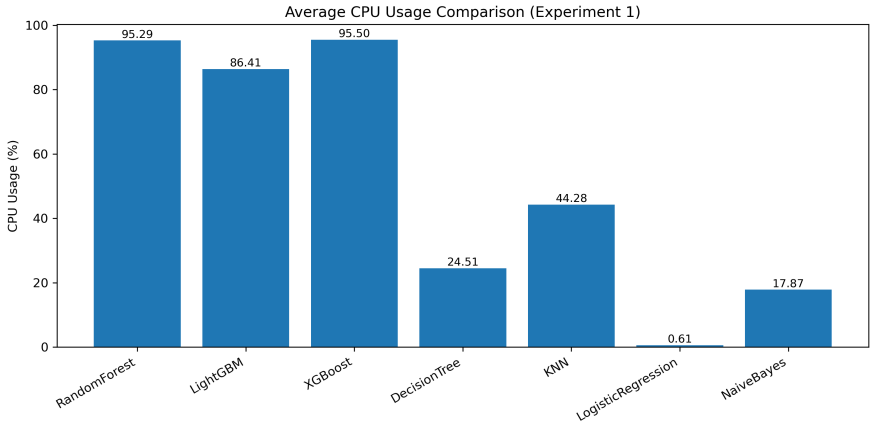

FIGURE 4.10: Average CPU utilization across different classifiers

LightGBM maintains a substantially lower memory consumption than Random Forest (peak RAM delta of 65.36 MB) while achieving near-best prediction performance (accuracy 0.9738 and macro F1-score 0.9685).

Although it has competitive performance as well, XGBoost is less appealing from an efficiency standpoint because to its high CPU utilization and the longest training time (42.43 s).

All things considered, LightGBM offers the best balance between computational efficiency and predictive performance.In order to further increase efficiency while maintaining detection performance, LightGBM is chosen as the baseline attack classifier for upcoming experiments that add feature reduction.

KNN's low inference efficiency (testing time of 6.19 s) has an adverse effect on intrusion detection in real time.

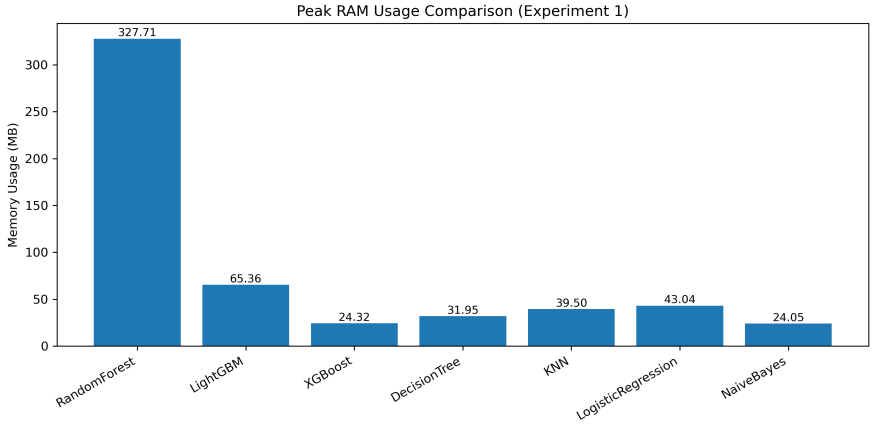

FIGURE 4.11: Overall CPU usage across classifiers

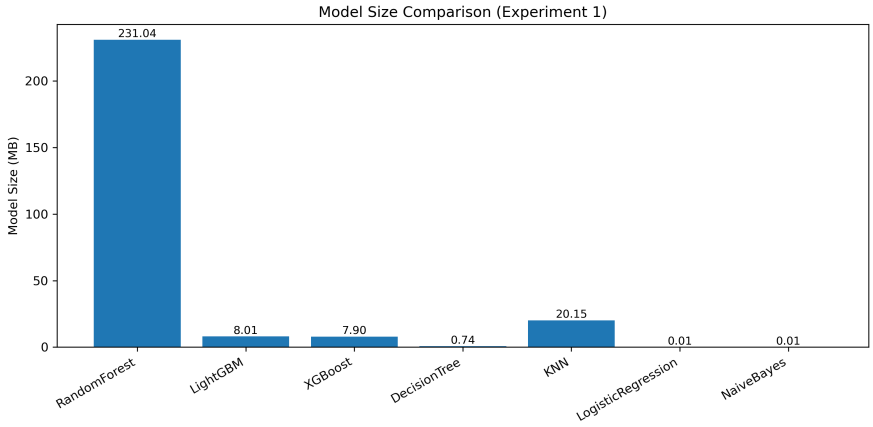

FIGURE 4.12: Model size comparison across classifiers

4.6 Experiment 2: Comparison of the Proposed Lightweight Hybrid Pipeline with Baseline LightGBM

Figure 4.13 evaluates the comparison between only LightGBM and our proposed hybrid pipeline on a balanced dataset to check whether our pipeline maintains the balance between accuracy and efficiency in terms of all evaluation metrics.

Two approaches are assessed:

- Baseline: Only LightGBM. This approach does not use any feature reduction technique.

- Proposed Lightweight Hybrid Pipeline: The suggested hybrid pipeline is as follows: L1-regularized Logistic Regression → Random Forest ranking and selecting the best features → LightGBM.

Two steps make up the suggested pipeline's feature reduction process: Random Forest ranks the remaining features and chooses the most informative subset after L1-regularized logistic regression automatically eliminates irrelevant or noisy features by setting some coefficients to zero. The smaller feature set is then used to train LightGBM.

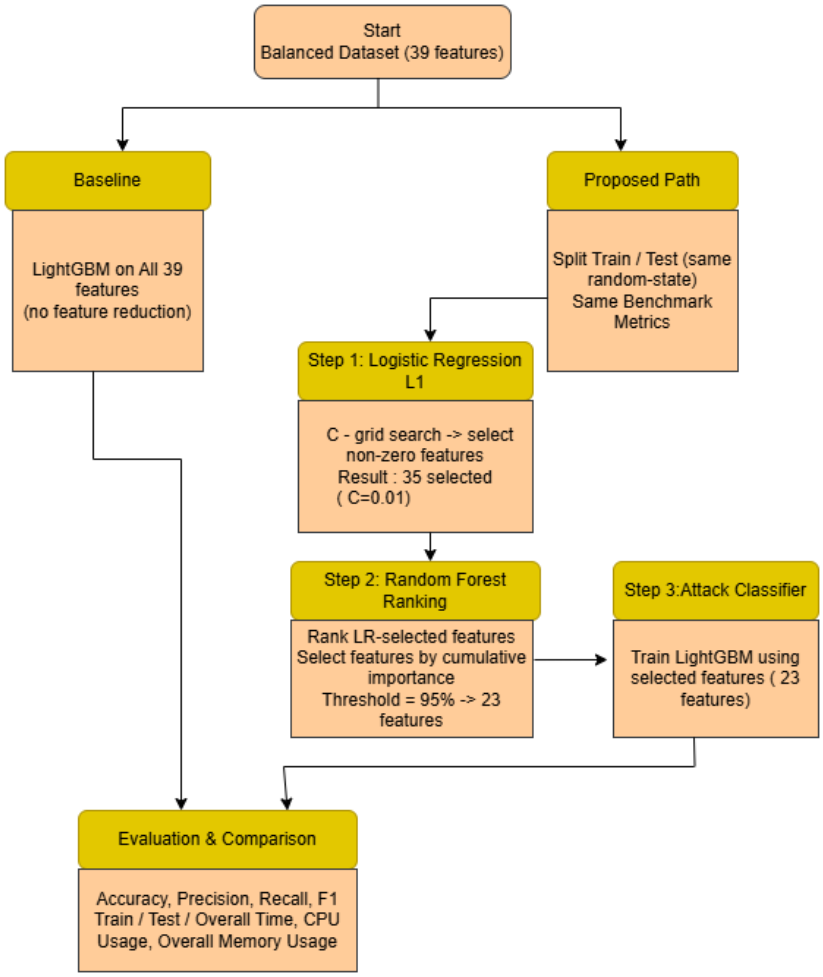

FIGURE 4.13: Comparing the baseline with our proposed hybrid pipeline

4.6.1 Results of Experiment 02

There are three phases to the proposed pipeline.Logistic regression with L1 regularization is used for removing noisy and irrelevant features, which enforces sparsity in the model coefficients. The remaining features are ranked and reduced according to their relevance scores in the second stage by using Random Forest. In order to train a LightGBM classifier for attack detection, a subset of the most crucial features is finally chosen. The progressive feature reduction achieved by the proposed hybrid pipeline is summarized in Table 4.11. While Random Forest further refines the feature set by choosing the most informative features, Logistic Regression with L1 regularization eliminates noisy features.

TABLE 4.11: Feature reduction achieved by the proposed pipeline

Method	Original Features	After LR (L1)	After RF
Baseline(only LightGBM)	39	39	39
LR+RF+LightGBM	39	35	23

4.6.2 Results

Table 4.12 contrasts the proposed hybrid feature selection process with the baseline LightGBM classifier. The findings show that significant efficiency gains can be attained with a minor reduction in classification accuracy.

TABLE 4.12: Comparison between baseline LightGBM and proposed feature selection pipeline

Method	Acc	$F1_{mac}$	Train(s)	Test(s)	$CPU_{avg}\%$	RAM
Baseline LightGBM	0.9738	0.9685	22.21	2.02	86.41	65.36
LR+RF+LightGBM	0.9683	0.9620	14.70	1.30	63.95	26.09

In your document:

4.6.3 Accuracy and Efficiency Comparison

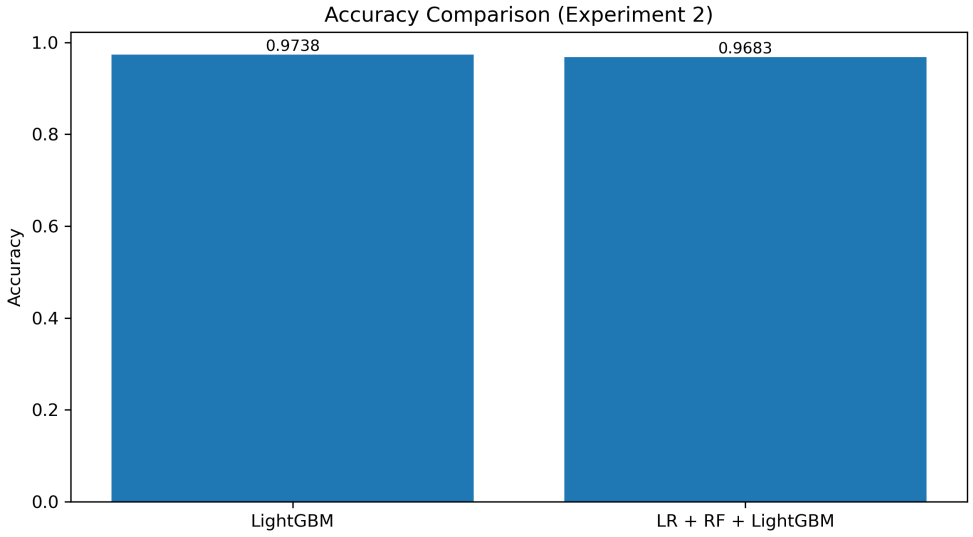

FIGURE 4.14: Accuracy comparison between baseline and proposed pipeline

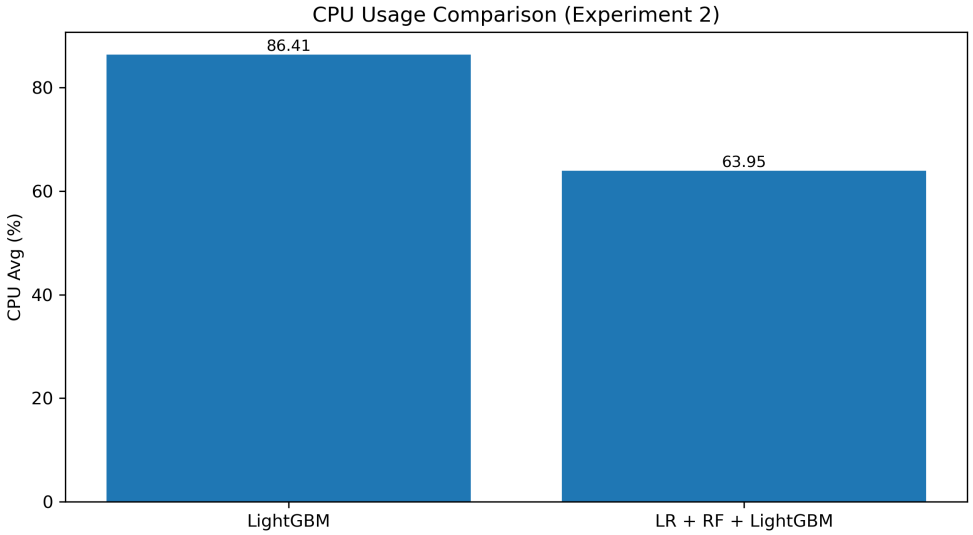

FIGURE 4.15: Average CPU usage comparison

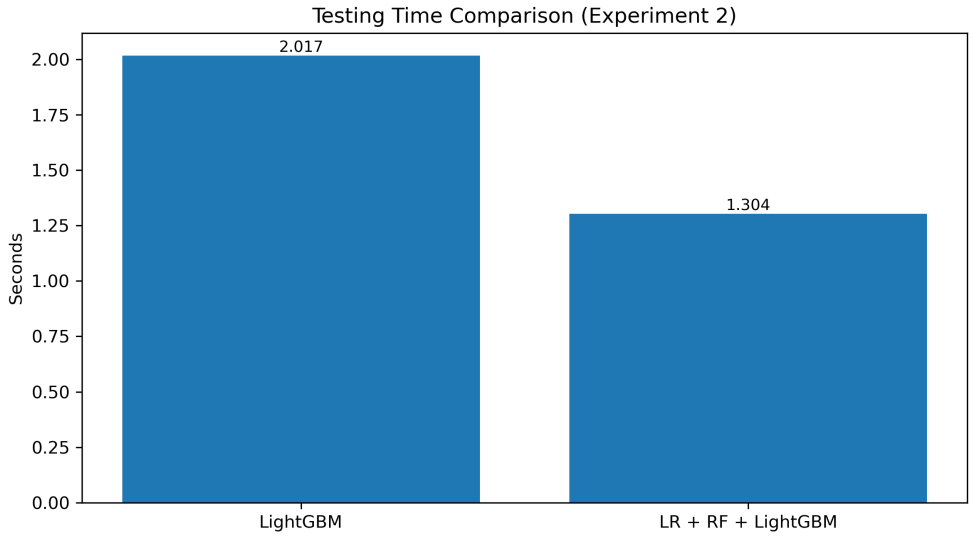

FIGURE 4.16: Test Time Comparison

As we can see in Figure 4.16, the testing time of our proposed pipeline is 1.304 which is less than the baseline. The Figure 4.14 and 4.15 shows that the suggested hybrid feature selection pipeline considerably increases computational efficiency while preserving similar detection performance.

Despite a slight drop in accuracy (from 0.9738 to 0.9683), the macro F1-score remains high, suggesting that detecting capability is mostly preserved. This shows that the suggested lightweight feature reduction strategy greatly increases computational efficiency and lowers resource consumption while maintaining the overall classification performance.

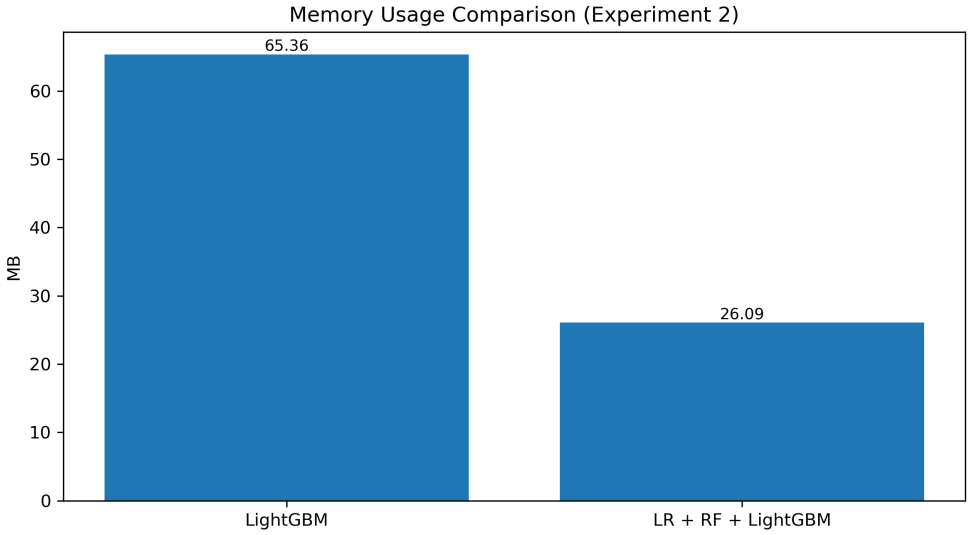

FIGURE 4.17: Overall memory usage comparison

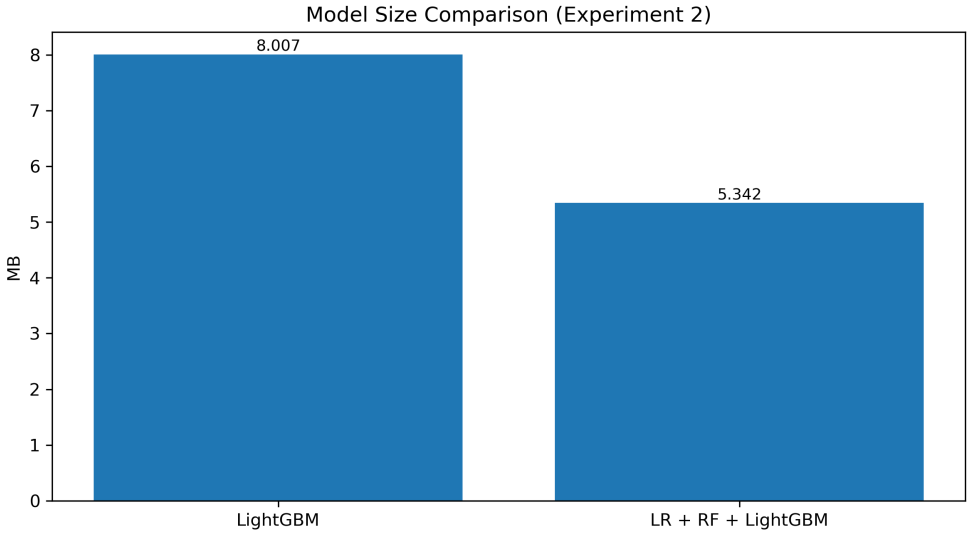

FIGURE 4.18: Model size comparison

The suggested pipeline 4.17 and 4.18 decreases model size by over 33%, peak memory use by over 60%, overall execution time by nearly 33%, and training time by about 34%.

These findings support the central claim of this thesis, which is that a significantly better efficiency–accuracy trade-off occurs from eliminating noisy and unnecessary variables.

As a result, the suggested hybrid pipeline works better in IoT intrusion detection systems with limited resources.

4.7 Experiment 3: Proposed Lightweight Hybrid Pipeline with Multiple Classifiers

The proposed hybrid feature selection strategy's generalizability across several classifiers is assessed in Figure 4.19.

This experiment's goal is to evaluate how well the proposed hybrid feature selection pipeline performs when paired with various machine learning classifiers.

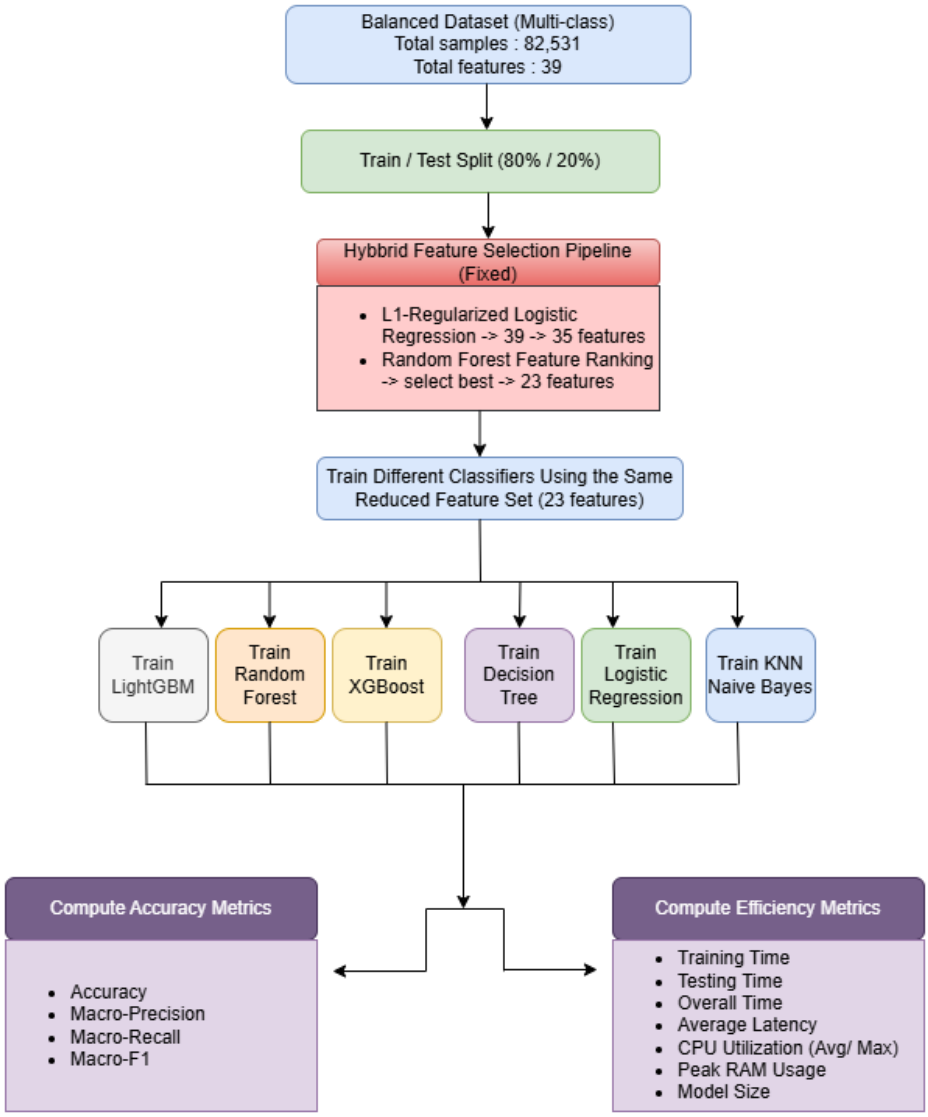

FIGURE 4.19: Workflow of the proposed hybrid feature selection with other machine learning classifier

4.7.1 Results of Experiment 03

TABLE 4.13: Experiment 3: The suggested lightweight hybrid pipeline's efficiency and performance using several classifiers

Model	Accuracy	$F1_{mac}$	Train (s)	Test (s)	Overall (s)	CPU_{avg} (%)	RAM (MB)
Proposed + LightGBM	0.9683	0.9620	14.70	1.30	16.00	63.95	26.09
Proposed + Random Forest	0.9768	0.9712	35.87	1.20	37.07	79.86	158.71
Proposed + XGBoost	0.9698	0.9637	39.29	0.69	39.98	57.85	38.89
Proposed + Decision Tree	0.9579	0.9481	2.26	0.01	2.28	24.38	17.40
Proposed + Logistic Regression	0.7795	0.7586	10.00	0.02	10.02	0.50	23.20
Proposed + KNN	0.9314	0.9169	0.10	8.68	8.77	38.08	18.07
Proposed + Naive Bayes	0.6849	0.6385	0.11	0.05	0.16	18.76	12.98

The performance and computational efficiency of the suggested hybrid feature selection pipeline when paired with various classifiers are shown in Table 4.13. The findings demonstrate how the efficiency-accuracy trade-offs vary significantly amongst models.

4.7.2 Accuracy and Efficiency Metrics

LightGBM continues to offer the best trade-off between accuracy and efficiency among all evaluated classifiers. Despite having a marginally higher accuracy, Random Forest uses a lot more memory and has a larger model.

These results confirm that the proposed pipeline is classifier-agnostic and support LightGBM as the optimal attack classifier for IoT intrusion detection scenarios with limited resources. The experimental findings further show that the suggested hybrid feature selection framework can function well with various machine learning classifiers without sacrificing detection performance.

This adaptability makes the suggested method more suitable for a range of IoT security applications and deployment situations. Additionally, LightGBM's decreased computational cost makes it better suited for real-time intrusion detection, where quick response times and low resource usage are essential.

The findings thus validate that in IoT-based intrusion detection systems, a feasible balance between detection effectiveness and computing efficiency may be achieved by combining effective feature reduction with a lightweight classifier.

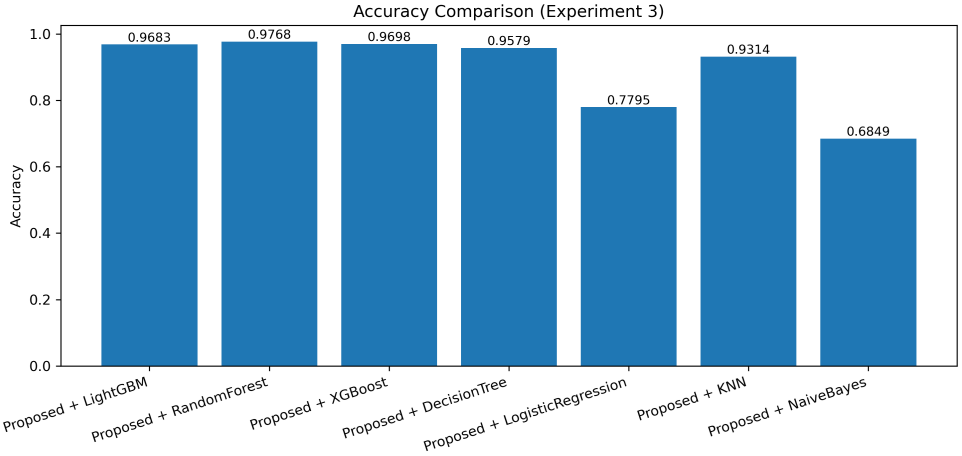

FIGURE 4.20: Accuracy comparison of classifiers using the proposed pipeline

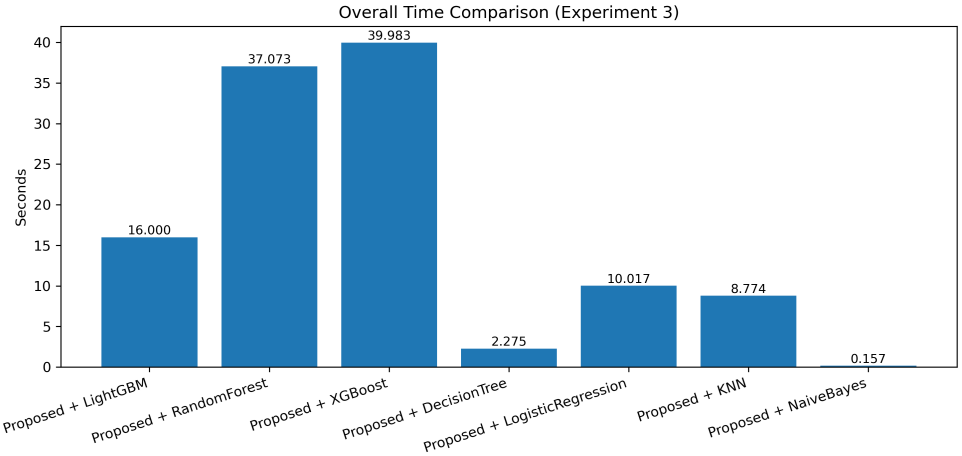

FIGURE 4.21: Overall execution time comparison using the proposed pipeline

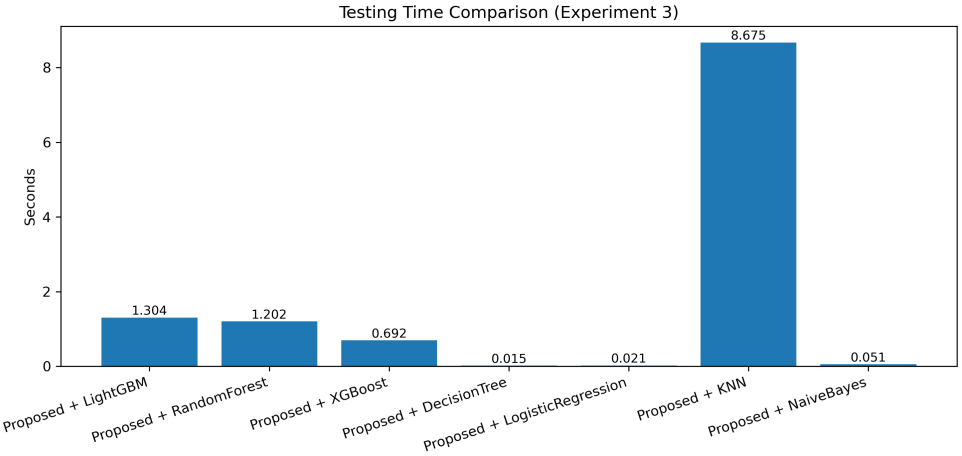

FIGURE 4.22: Testing time comparison across classifiers

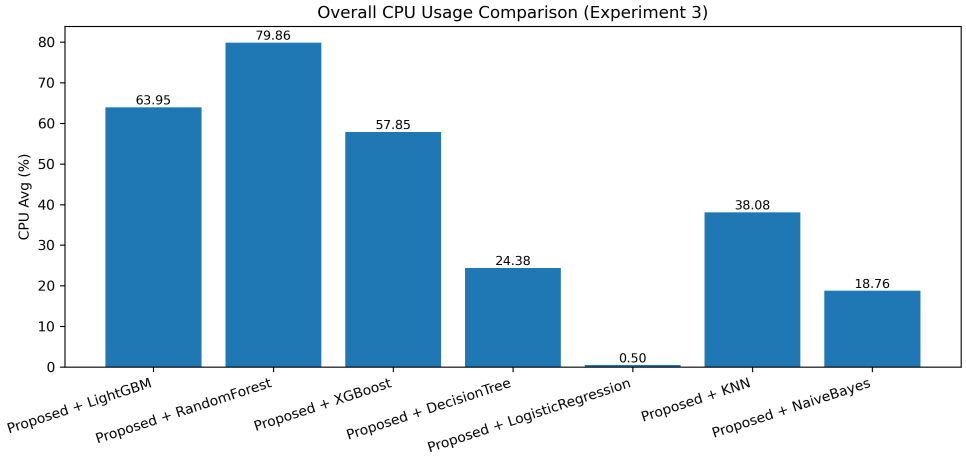

FIGURE 4.23: Average CPU utilization across classifiers

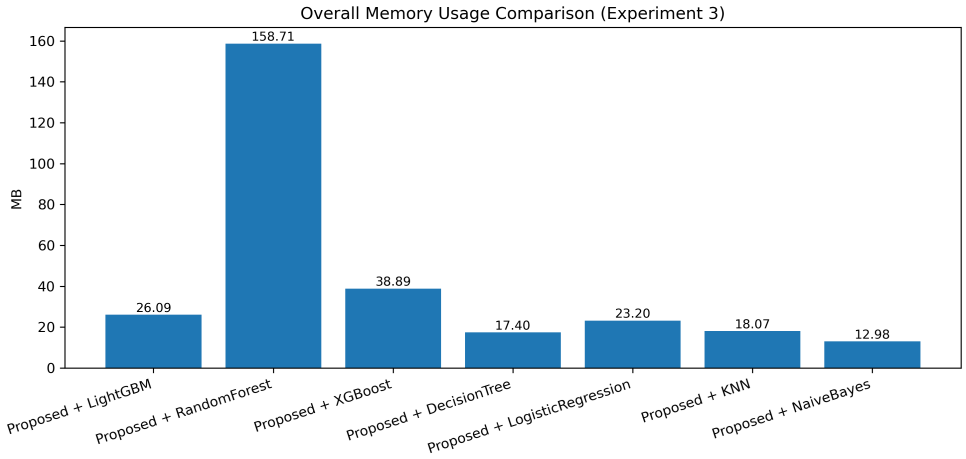

FIGURE 4.24: Overall CPU usage across classifiers

Figures 4.20–4.21–4.22–4.23–4.24 and Table 4.13 show that the suggested hybrid feature selection pipeline continuously increases computational efficiency across all evaluated classifiers. Efficiency improvements in terms of execution time, CPU utilization, and memory consumption are obvious, even though predicted performance varies based on the learning model.

4.8 Experiment 4 :Proposed Pipeline Ablation Study

An ablation study is conducted in Figure 4.25 to examine the contributions of each pipeline component. The goal is to comprehend the effects of each feature selection

step on computing efficiency and detection performance. The same dataset split and benchmarking process is used to assess the following configurations:

- **Baseline:** Only LightGBM without any Feature reduction.
- **L1-LR + LightGBM:** only L1-based feature removal prior to classification.
- **RF + LightGBM:** only Random Forest ranking/selection prior to classification. ,
- **L1-LR + RF + LightGBM** complete suggested pipeline with automatic feature selection.

FIGURE 4.25: Comparison of baseline and feature selection strategies (L1-LR, RF, and L1-LR+RF) with LightGBM for multi-class IoT intrusion detection.)

This experiment's goal is to examine the effects of various feature selecting techniques on the LightGBM classifier's effectiveness and performance.

4.8.1 Results

TABLE 4.14: Experiment 4: Comparison of different feature selection technique using LightGBM

Model	Accuracy	$F1_{mac}$	Train(s)	Test(s)	$CPU_{avg}\%$	RAM(MB)
LightGBM	0.9738	0.9685	22.21	2.02	86.41	65.36
LR + LightGBM	0.9740	0.9687	20.39	1.92	93.96	122.48
RF + LightGBM	0.9737	0.9683	18.54	1.86	93.53	21.70
LR+RF + LightGBM	0.9683	0.9620	14.70	1.30	63.95	26.09

The performance and computational efficiency of several feature selection techniques along with the LightGBM classifier are compared in Table 4.14. Experiment 4's findings show that feature selection technique significantly affects computing efficiency and detection performance. Although there is some efficiency gain from using individual feature selection techniques like Random Forest or Logistic Regression, the suggested hybrid pipeline maintains a more comparable trade-off between accuracy and resource usage.

4.8.2 Accuracy and Efficiency Metrics

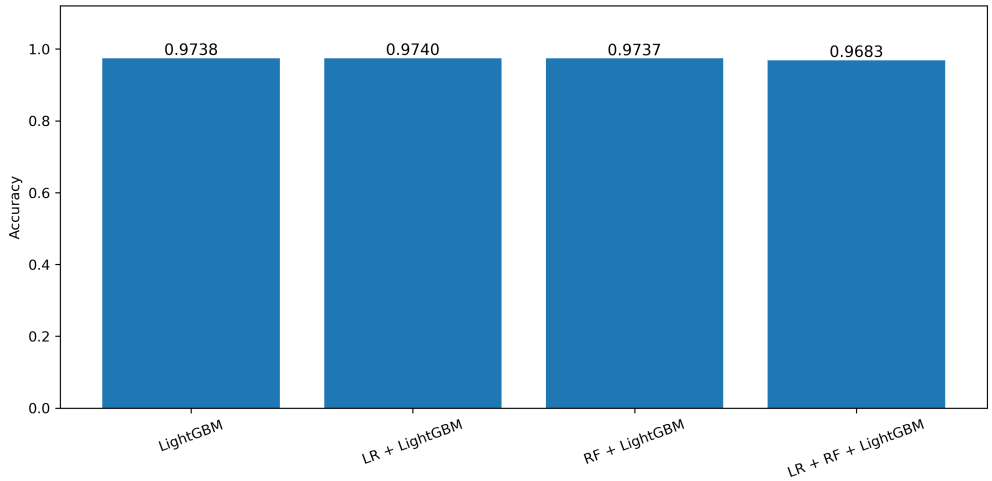

FIGURE 4.26: Accuracy comparison under different feature selection strategies

Figures 4.26, 4.27, 4.28, 4.29, and 4.30, together with Table 4.14, present the comparative evaluation of different feature selection strategies with LightGBM. The proposed LR+RF+LightGBM pipeline efficiently eliminates redundant and weakly informative features with little effect on detection performance, as shown

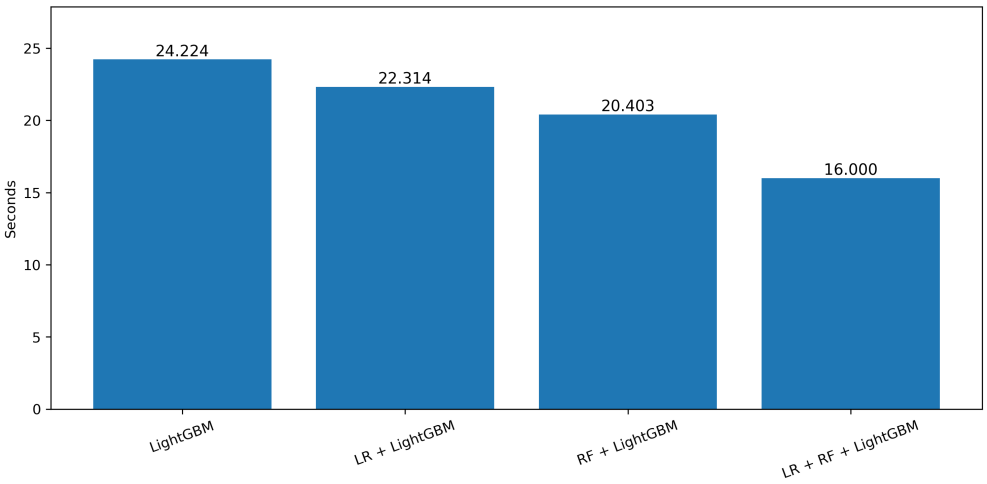

FIGURE 4.27: Overall execution time comparison

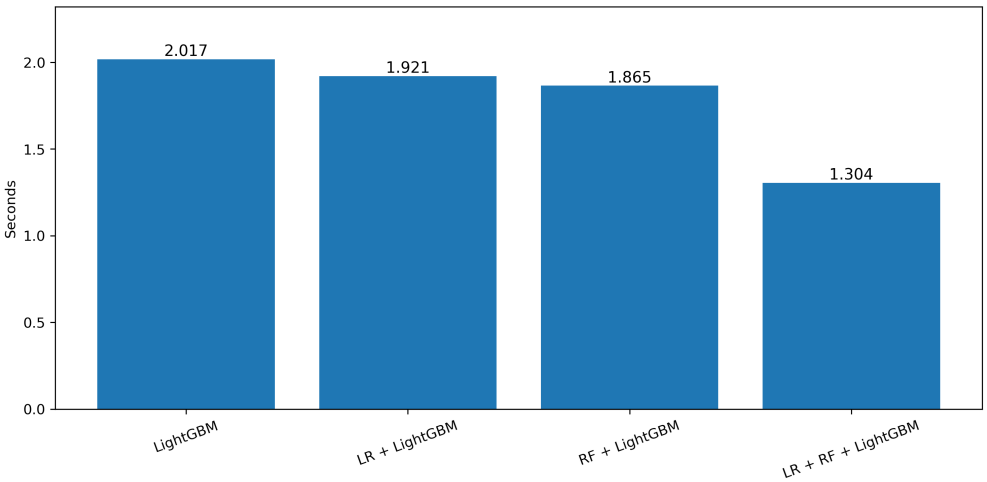

FIGURE 4.28: Testing time comparison

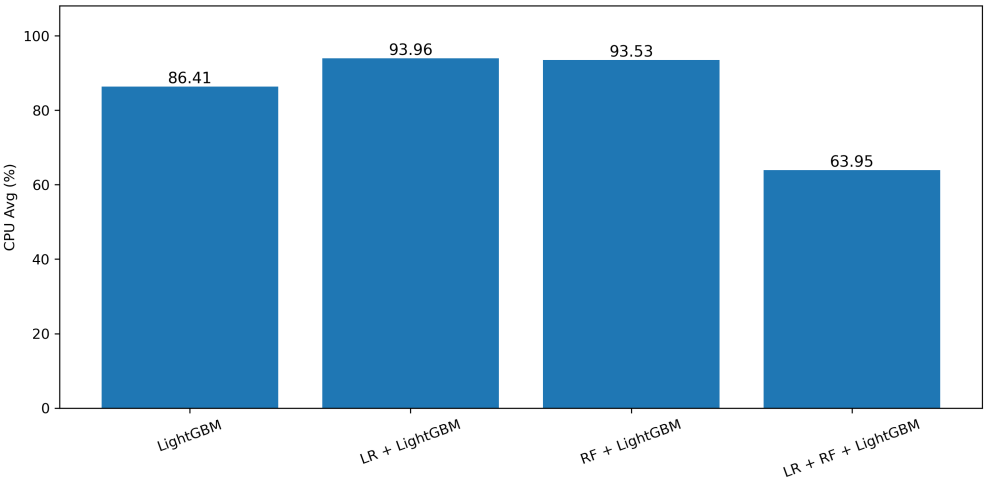

FIGURE 4.29: Average CPU utilization

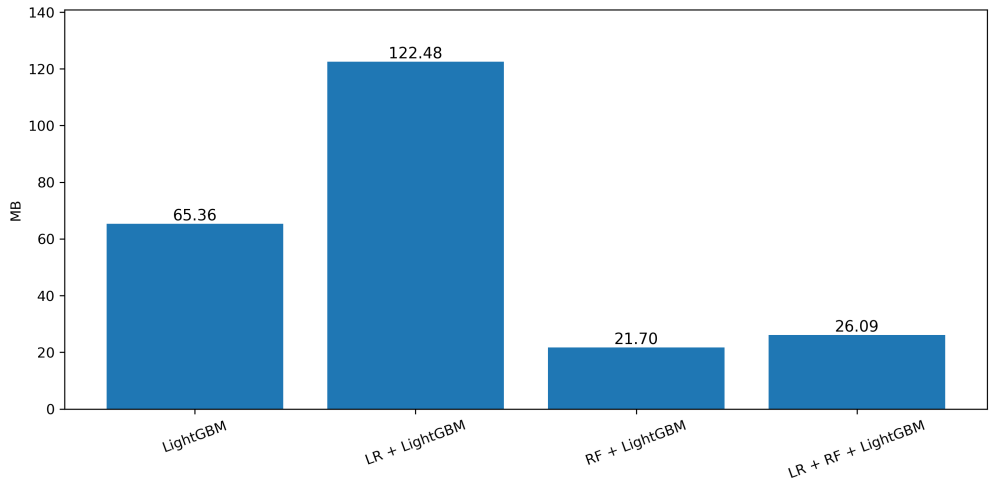

FIGURE 4.30: Overall Memory usage

by a high macro F1-score, even though the baseline and single-stage feature selection models achieve somewhat higher accuracy because they retain more features.

4.9 Experiment 5: Impact of changing number of selected RF-Features

This experiments investigates the effects of the Random Forest stage's feature selection count on the overall accuracy-efficiency tradeoff as shown in fig Figure 4.31

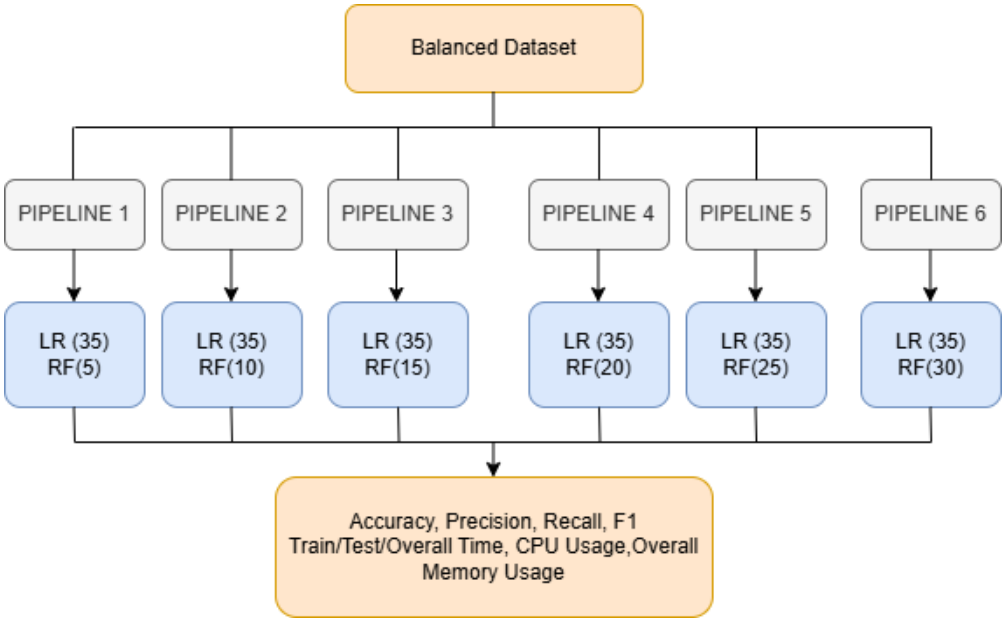

FIGURE 4.31: Impact of changing RF-Selected Features

Following Random Forest feature ranking, the top-k features are chosen for a range of k values (increasing feature counts) rather than choosing a single fixed subset. The same LightGBM classifier is trained and assessed using the standard benchmarking technique for every value of k.

The goal of this analysis is to determine the ideal feature range where accuracy is maintained but time, CPU, and memory consumption are kept to a minimum. It also illustrates how adding features outside of the ideal range raises computation costs without appreciably improving accuracy.

4.9.1 Results of Experiment 5

Evaluation parameters for the experimental findings include accuracy,Precision, Recall and F1-score, training and testing durations, CPU usage, and memory usage.

TABLE 4.15: Performance Impact of Varying RF Feature Count in the pipeline

Method	**Accuracy**	**CPU Avg (%)**	**Peak RAM (MB)**	**Overall Time (s)**
LR+RF(5)+LightGBM	0.8481	95.1	28.4	5.42
LR+RF(10)+LightGBM	0.9222	94.2	36.9	6.68
LR+RF(15)+LightGBM	0.9494	94.5	39.9	8.04
LR+RF(20)+LightGBM	0.9561	92.4	43.7	9.95
LR+RF(25)+LightGBM	0.9689	95.1	49.2	11.04
LR+RF(30)+LightGBM	0.9694	94.9	52.8	11.48

The performance impact of changing the number of RF-selected features in the LR+RF+LightGBM pipeline is shown in Table 4.15. The findings show that as the number of chosen features increases, accuracy continuously improves along with CPU utilization, memory consumption, and execution time. This demonstrates the trade-off between computational efficiency and detection performance, especially for deployment in IoT systems with limited resources.

4.9.2 Accuracy and Efficiency Metrics

The more RF-selected features there are, the better the classification accuracy, as Fig. 4.32 illustrates. Beyond larger feature counts, however, the benefit becomes insignificant, suggesting accuracy concentration and diminishing returns from keeping more features.

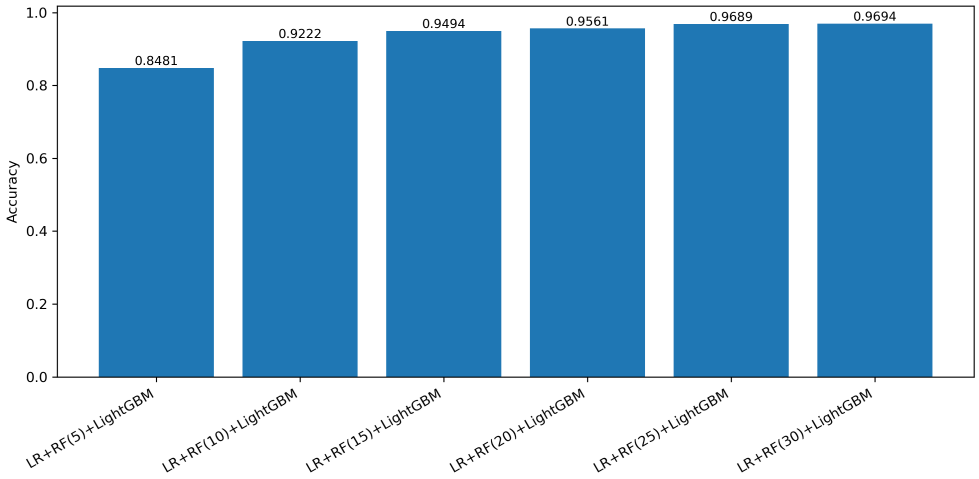

FIGURE 4.32: Accuracy comparison for varying RF feature counts in pipeline.

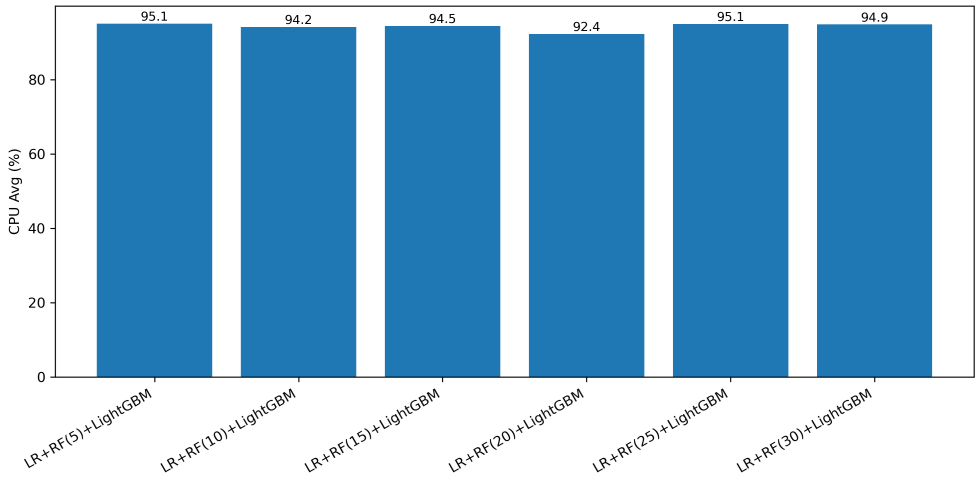

FIGURE 4.33: Average CPU utilization for varying RF feature counts in pipeline.

Fig. 4.33 illustrates how feature count affects CPU utilization, with a higher number of selected features leading to a higher CPU consumption due to the increased computational effort during both training and inference. A similar pattern is shown in Fig. 4.34, which shows that peak RAM consumption increases with feature count because the model must store and analyze more features. It also shows how feature count affects CPU utilization, with more selected features resulting in higher CPU usage because of the increased computational demand during both training and inference.

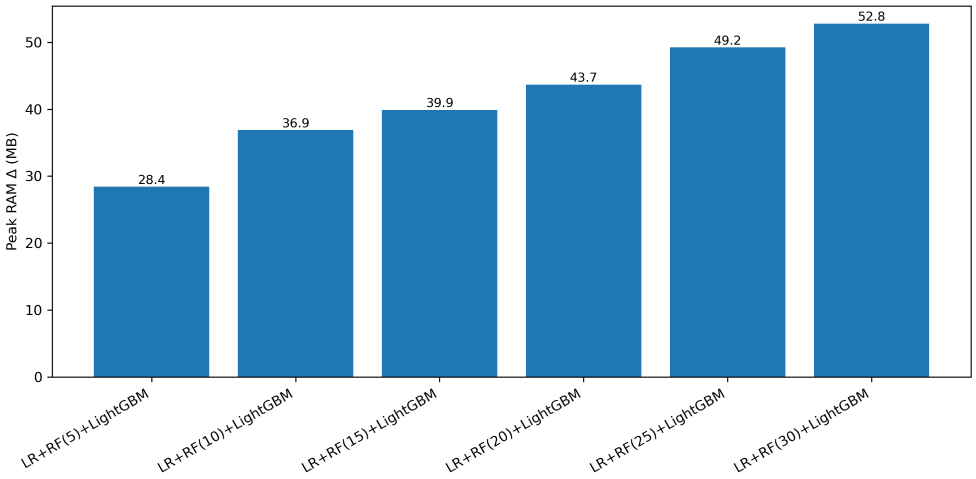

FIGURE 4.34: Overall Memory consumption for varying RF feature counts in pipeline.

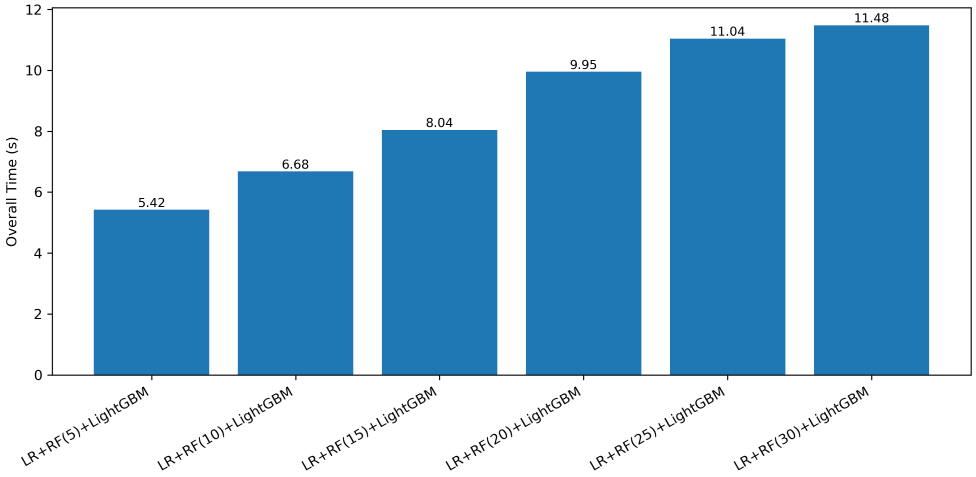

FIGURE 4.35: Overall execution time for varying RF feature counts in the pipeline.

Fig. 4.35 and 4.36 displays a overall time and testing time by changing the no of selected feature from Random Forest.

4.10 Experiment 6: Comparison with Pearson Correlation Based Feature Selection

The suggested hybrid pipeline is contrasted with a different feature selection strategy based on Pearson Correlation Metrics (PCM) in Experiment 6 as shown in Figure 4.37.

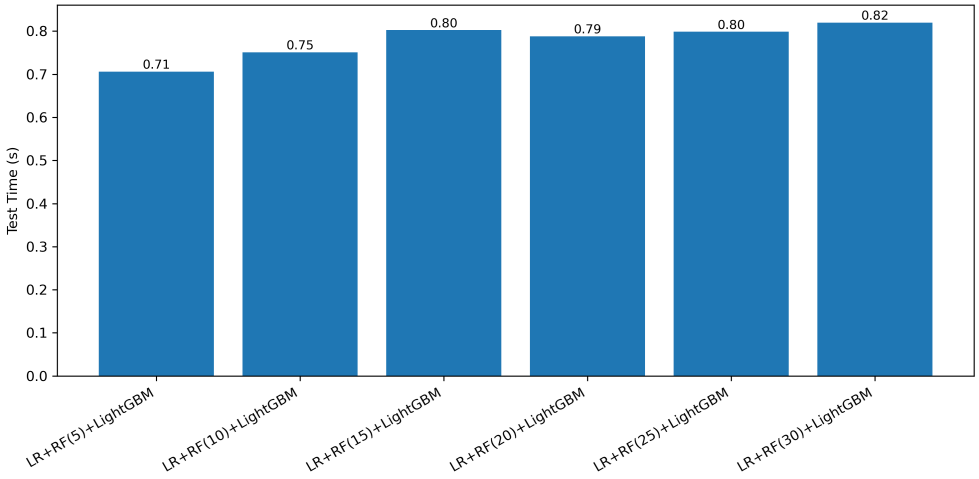

FIGURE 4.36: Inference (test) time for varying RF feature counts in the pipeline.

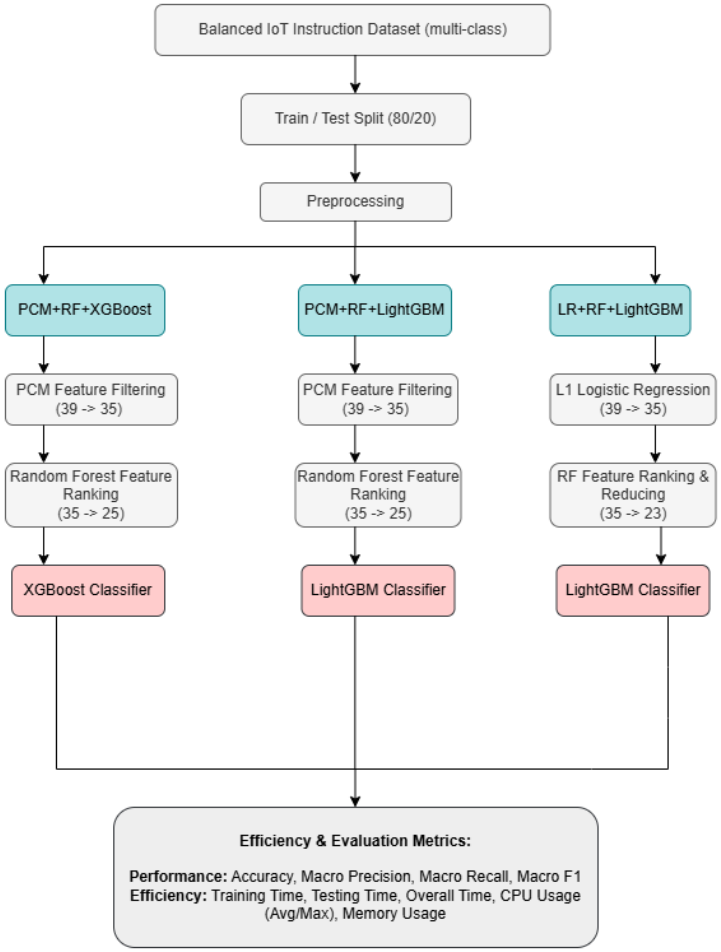

FIGURE 4.37: Comparison with Pearson correlation metrics with different classifiers

The pipelines listed below are assessed using the same benchmarking methodology:

- **PCM + RF + XGBoost,**
- **PCM + RF + LightGBM,**
- **Proposed Pipeline (L1-LR + RF + LightGBM).**

This comparison assesses whether the suggested hybrid strategy outperforms correlation based feature filtering techniques in terms of balancing computational efficiency and multi-class detection performance. The experimental findings from the proposed hybrid machine learning mechanism for intrusion detection in IoT networks are shown and examined in this chapter. The purpose of the experiments was to assess computing efficiency as well as detection performance.

The same dataset split, preparation plan, and benchmarking process outlined in Chapter 3 are used for all experiments. This ensures that the observed variations in outcomes are only attributable to the classifier and feature selection procedures used.

4.10.1 Results of Experiment 6

A comparison of correlation-based and learning-based feature selection pipelines in terms of detection performance and computing efficiency is shown in Table 4.16.

TABLE 4.16: Experiment 6: Comparing correlation-based pipelines' performance and efficiency

Pipeline	Accuracy	$F1_{mac}$	Train(s)	Test(s)	Overall(s)	RAM(MB)
PCM + RF + LightGBM	0.9726	0.9671	18.04	1.99	91.66	30.17
PCM + RF + XGBoost	0.9709	0.9654	27.60	0.46	89.89	35.49
Proposed (Auto) + LightGBM	0.9683	0.9620	14.70	1.30	63.95	26.09

4.10.2 Accuracy and Efficiency Metrics

Figures 4.38 shows the accuracy among different feature selection approach.The model with the highest accuracy, PCM + RF + LightGBM (0.9726), is closely

followed by PCM + RF + XGBoost (0.9709).Although the accuracy of the Proposed (LR + RF + LightGBM) model is slightly lower (0.9683), but all models produce similarly good performance.

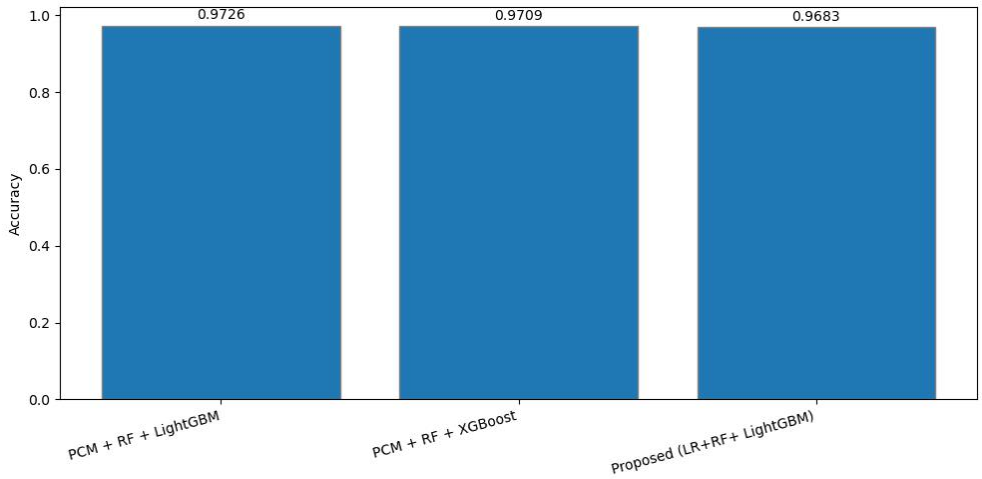

FIGURE 4.38: Accuracy comparison

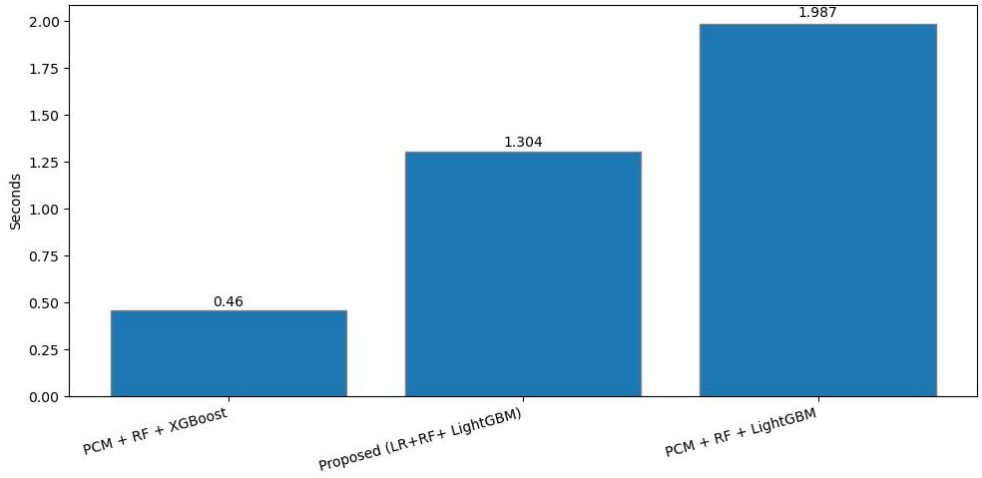

FIGURE 4.39: Testing time comparison

The three ensemble models' testing times in Experiment 5 are contrasted in this figure Figures 4.39. The fastest test time is 0.460 s for PCM + RF + XGBoost, and the longest testing time is 1.987 s for PCM + RF + LightGBM. The Proposed (LR + RF + LightGBM) model outperforms PCM + RF + LightGBM in terms of testing time (1.304 s), but exhibiting a marginally poorer accuracy. The suggested model is more suited for time-sensitive applications because it provides a better balance between accuracy and computational efficiency.

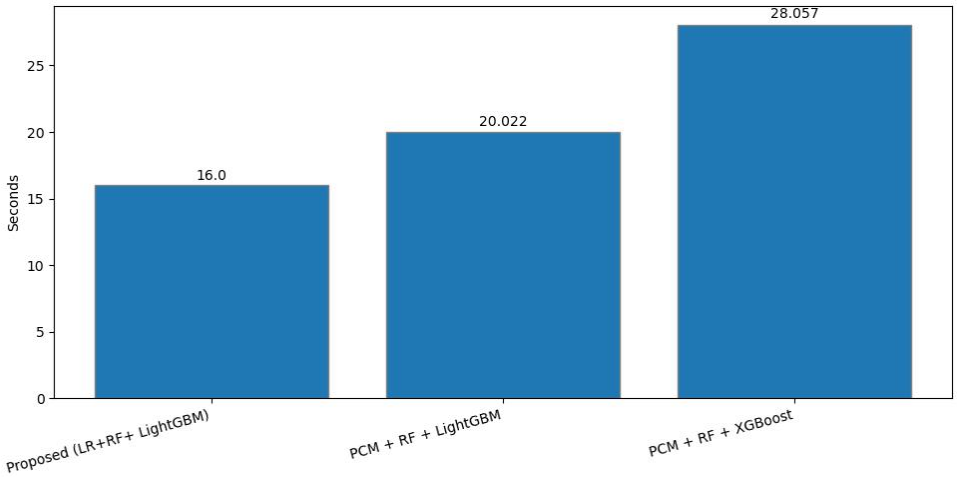

FIGURE 4.40: Overall execution time Comparison

Figure 4.40 shows the overall which is training plus time among different approaches.In this result also our proposed LR+RF+lightGBM gives the best result as compared to others

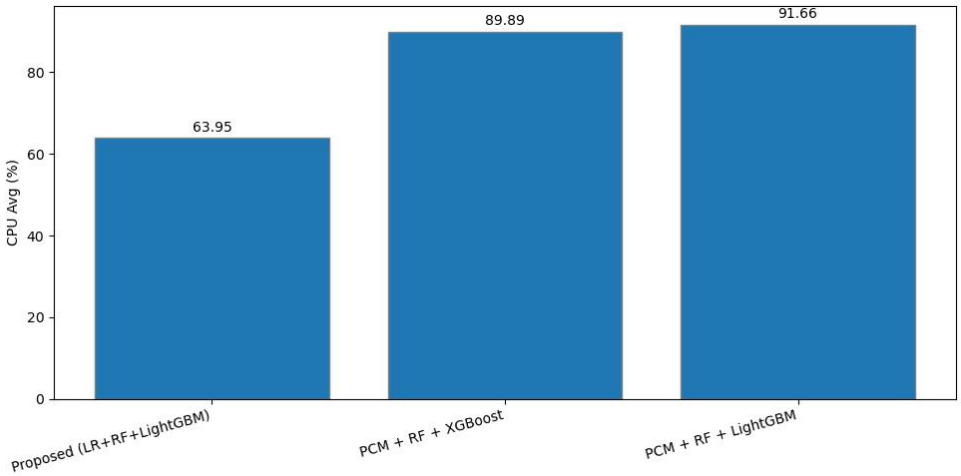

FIGURE 4.41: Average CPU utilization Comparison

Figure 4.41, The average CPU consumption of the models in Experiment 5 is displayed in this diagram. Better computational efficiency is indicated by the Proposed (LR + RF + LightGBM) model, which consumes the fewest CPU resources (63.95PCM + RF + XGBoost (89.89%) and PCM + RF + LightGBM (91.66%) on the other hand, use noticeably more CPU power. Overall, the suggested model is more appropriate for resource-constrained contexts because it obviously offers higher efficiency and reduced CPU consumption, however having somewhat poorer accuracy. Figure 4.42 shows the overall memory which is used

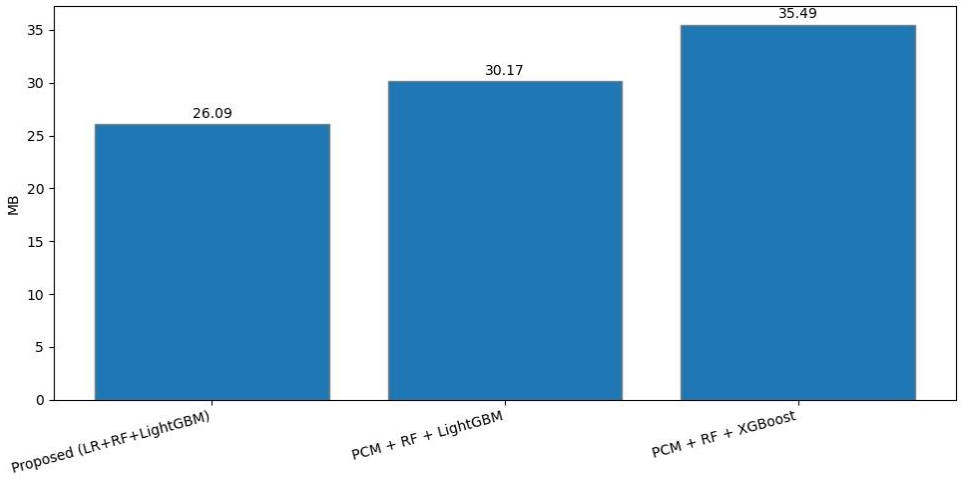

FIGURE 4.42: Overall Memory usage comparison

in training and testing time.In this result also our proposed LR+RF+lightGBM gives the best result as compared to others.

4.11 Chapter Summary

A thorough experimental evaluation of the suggested hybrid machine learning framework for intrusion detection in Internet of Things networks was provided in this chapter. To examine the effects of classifier selection, data imbalance handling, feature selection techniques, and feature reduction levels on detection performance and computing efficiency, a set of six experiments was carried out.

To find the best baseline attack classifier, several machine learning classifiers were first evaluated. The findings showed that LightGBM offers the best trade-off between cheap computational cost and high detection accuracy, which makes it appropriate for IoT contexts with limited resources. Following an analysis of several data balancing strategies, SMOTE-ENN consistently beat SMOTE and class-weight-based approaches by successfully handling noisy samples and class imbalance.

Further studies examined how feature selection can increase effectiveness without significantly affecting detection performance. Significant improvements were demonstrated by individual feature selection methods such as Random Forest and Logistic Regression. However, the suggested hybrid feature selection pipeline,

which combines Random Forest-based feature ranking with sparsity-inducing Logistic Regression, achieved the most balanced trade-off between computational efficiency and accuracy preservation.

The effects of different feature subset sizes were also examined, and the results showed that after a certain point, adding more features significantly increases computational cost while offering little performance advantages. The existence of an ideal feature subset for intrusion detection tasks in Internet of Things networks is confirmed by this observation. Lastly, a comparative analysis using correlation-based feature selection pipelines showed that the suggested learning-based hybrid strategy cuts training time, memory use, CPU utilization, and model size greatly while achieving similar detection performance.

Overall, the findings in this chapter confirm that the suggested hybrid intrusion detection system is reliable and effective. The suggested method provides a higher accuracy–efficiency trade-off by smartly lowering feature dimensionality and utilizing an effective attack classifier, which makes it ideal for real-time deployment in IoT systems with limited resources. The final conclusions and future study directions covered in the following chapter are firmly supported by the knowledge gathered from these studies.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In the literature, many intrusion detection systems (IDS) for Internet of Things (IoT) networks have already been presented; and several of them reports high detection accuracy. A common limitation, though, is that the majority of these methods mainly concentrate on increasing accuracy rather than effectively enhancing computing efficiency. This becomes a significant problem in IoT situations, as devices have limited resources and real-time deployment requires models that are quick and lightweight. We employed an Lightweight IDS pipeline in this thesis that was created to enhance the efficiency-accuracy trade-off. The detection accuracy somewhat declined when we used our feature-reduction process, but efficiency significantly increased like LightGBM obtained an accuracy of **0.97** when it was trained without feature reduction. However, the accuracy deccreased to **0.968** when LightGBM was combined with our feature-reduction strategy. This indicates a negligible drop of **0.002**, however the indicators relating to efficiency demonstrated a notable improvement. The pipeline enhanced suitability for real-time IoT security and decreased the overall resource load in terms of computational performance. The following efficiency advantages were noted as compared to utilizing LightGBM without feature reduction: overall time (training + testing) improved by **8.2** seconds, memory consumption improved by **39.29**%, test time improved by **0.7** seconds, and overall CPU utilization improved by **22.46**%.

These findings show that the suggested feature reduction approach maintains a good balance between accuracy and efficiency, with a significant increase in efficiency and a minimal accuracy decline. In addition, we used LightGBM to analyze each pipeline component independently, and the outcomes consistently shown improvements, hence proving the efficiency of the proposed framework.

5.2 Future Work

Future research should test the proposed intrusion detection methodology on additional IoT datasets to see if it performs well in various scenarios. It will be easier to verify that the model operates consistently across various network types and attack patterns if multiple datasets are used.Efficiency is essential in IoT situations, thus future research should concentrate more on it. Lightweight intrusion detection solutions are highly needed because IoT devices have limited memory, computing power, and energy. The goal of IDS model design should be to reduce resource consumption without sacrificing detection accuracy. Intrusion detection systems can be made more appropriate for practical Internet of Things applications by emphasizing lightweight models and effective feature selection strategies.

Bibliography

[1] J. S. Yalli, M. H. Hasan, and A. A. Badawi, "Internet of things (iot): Origins, embedded technologies, smart applications, and its growth in the last decade," *IEEE Access*, vol. 12, pp. 91357–91382, 2024.

[2] J. S. Yalli, M. H. Hasan, and A. A. Badawi, "Internet of Things (IoT): Origins, embedded technologies, smart applications, and its growth in the last decade," *IEEE access*, vol. 12, pp. 91357–91382, 2024.

[3] D. Dobrilović, V. Brtka, G. Jotanović, Ž. Stojanov, G. Jauševac, and M. Malić, "Architecture of IoT system for smart monitoring and management of traffic noise," in *5th EAI International Conference on Management of Manufacturing Systems*, pp. 251–266, Springer, 2021.

[4] S. Al-Farsi, M. M. Rathore, and S. Bakiras, "Security of blockchain-based supply chain management systems: challenges and opportunities," *Applied Sciences*, vol. 11, no. 12, p. 5585, 2021.

[5] S. Sicato, J. Costa, S. K. Singh, S. Rathore, and J. H. Park, "A comprehensive analyses of intrusion detection system for IoT environment.," *Journal of Information Processing Systems*, vol. 16, no. 4, 2020.

[6] G. Guo, X. Pan, H. Liu, F. Li, L. Pei, and K. Hu, "An IoT intrusion detection system based on TON IoT network dataset," in *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0333–0338, IEEE, 2023.

[7] B. Manjunatha, K. A. Shastry, E. Naresh, P. K. Pareek, and K. T. Reddy, "A network intrusion detection framework on sparse deep denoising auto-encoder

for dimensionality reduction," *Soft Computing*, vol. 28, no. 5, pp. 4503–4517, 2024.

[8] A. Saleem and W. Hamouda, "Lightweight Federated Few-Shot Learning-Based Network Intrusion Detection for Resource-Constrained IoT Devices," *IEEE Internet of Things Journal*, 2026.

[9] J. Joseph, N. T. Aleke, and O. P. Onyeanisi, "Deep learning based intrusion detection system for network security in IOT system," *Int J Edu, Manag Technol*, vol. 3, no. 1, pp. 119–138, 2025.

[10] N. A. Olobo, W. A. Ayuba, A. F. Omojola, I. H. Iyobosa, A. I. Adebayo, A. Obi-Obuoha, and U. P. Afegbai, "Deep learning-based intrusion detection systems for network security in iot system," *Path of Science*, vol. 10, no. 12, pp. 5011–5018, 2024.

[11] A. Pasam, A. Sinha, and A. Mailewa, "Lightweight AI-Based Intrusion Detection Models for IoT Devices: A Comparative Review," in *2025 IEEE International Carnahan Conference on Security Technology (ICCST)*, pp. 1–7, IEEE, 2025.

[12] K. Singal, N. Kandhoul, and S. K. Dhurander, "Evolutionary LightGBM-Based Intrusion Detection System for IoT Networks," *International Journal of Communication Systems*, vol. 38, no. 5, p. e70031, 2025.

[13] A. Nazir, Z. Memon, T. Sadiq, H. Rahman, and I. U. Khan, "A novel feature-selection algorithm in IoT networks for intrusion detection," *Sensors*, vol. 23, no. 19, p. 8153, 2023.

[14] M. Almohaimeed and F. Albalwy, "Enhancing IoT network security using feature selection for intrusion detection systems," *Applied Sciences*, vol. 14, no. 24, p. 11966, 2024.

[15] A. Jahangeer, S. U. Bazai, S. Aslam, S. Marjan, M. Anas, and S. H. Hashemi, "A review on the security of IoT networks: From network layer's perspective," *IEEE Access*, vol. 11, pp. 71073–71087, 2023.

[16] M. S. Habeeb and T. R. Babu, "Coarse and fine feature selection for network intrusion detection systems (IDS) in IoT networks," *Transactions on Emerging Telecommunications Technologies*, vol. 35, no. 4, p. e4961, 2024.

[17] M. Ali, M. Shahroz, M. F. Mushtaq, S. Alfarhood, M. Safran, and I. Ashraf, "Hybrid machine learning model for efficient botnet attack detection in IoT environment," *IEEe Access*, vol. 12, pp. 40682–40699, 2024.

[18] G. K. Baydoğmuş, "Detecting Internet of Things Attacks by Using Hybrid Learning and Feature Selection Method," *Avrupa Bilim ve Teknoloji Dergisi*, no. 29, pp. 19–25, 2021.

[19] P. v. a. Madhavi, V. Kumar, and v. Shariff, "Boosting student performance prediction in e-learning: A hybrid feature selection and multi-tier ensemble modelling framework with federated learning," *Journal of Theoretical and Applied Information Technology*, vol. 103, no. 5, 2025.

[20] M. Al-Omari and Q. A. Al-Haija, "Towards robust IDSs: An integrated approach of hybrid feature selection and machine learning," *J. Internet Serv. Inf. Secur*, vol. 14, no. 3, pp. 47–67, 2024.

[21] M. Nivaashini, P. Thangaraj, S. Sountharrajan, E. Suganya, and R. Soundariya, "Effective Feature Selection for Hybrid Wireless IoT Network Intrusion Detection Systems Using Machine Learning Techniques.," *Ad Hoc Sens. Wirel. Networks*, vol. 49, no. 3-4, pp. 175–206, 2021.

[22] J. B. Awotunde, C. Chakraborty, and A. E. Adeniyi, "Intrusion detection in industrial internet of things network-based on deep learning model with rule-based feature selection," *Wireless communications and mobile computing*, vol. 2021, no. 1, p. 7154587, 2021.

[23] J. Li, M. S. Othman, H. Chen, and L. M. Yusuf, "Optimizing IoT intrusion detection system: feature selection versus feature extraction in machine learning," *Journal of Big Data*, vol. 11, no. 1, p. 36, 2024.

[24] E. M. Maseno and Z. Wang, "Hybrid wrapper feature selection method based on genetic algorithm and extreme learning machine for intrusion detection," *Journal of big data*, vol. 11, no. 1, p. 24, 2024.

[25] E. M. Maseno, Z. Wang, and H. Xing, "A systematic review on hybrid intrusion detection system," *Security and Communication Networks*, vol. 2022, no. 1, p. 9663052, 2022.

[26] A. Khan, M. A. Hussain, and F. Anwer, "A Hybrid Lightweight Deep Learning-Based Intrusion Detection Approach in IoT Utilizing Feature Selection & Explainable Artificial Intelligence," *IEEE Access*, vol. 13, pp. 192451–192466, 2025.

[27] C. Gupta, A. Kumar, and N. K. Jain, "Intrusion defense: Leveraging ant colony optimization for enhanced multi-optimization in network security," *Peer-to-Peer Networking and Applications*, vol. 18, no. 2, p. 98, 2025.

[28] N. Ahmed, M. A. Ngadi, M. S. Rathore, and A. Mahmood, "PCM-RF a Hybrid Feature Selection Mechanism for Intrusion Detection System in IoT," *Security and Privacy*, vol. 8, no. 1, p. e499, 2025.

[29] W. H. Madhloom Kurdi, I. A. Alzuabidi, A. H. Najim, M. N. Kadhim, and A. A. Ahmed, "Efficient Two-Stage Intrusion Detection System Based on Hybrid Feature Selection Techniques and Machine Learning Classifiers.," *International Journal of Intelligent Engineering & Systems*, vol. 18, no. 3, 2025.

[30] M. A. Hossain and M. S. Islam, "A novel hybrid feature selection and ensemble-based machine learning approach for botnet detection," *Scientific Reports*, vol. 13, no. 1, p. 21207, 2023.

[31] H. Bakır and Ö. Ceviz, "Empirical enhancement of intrusion detection systems: a comprehensive approach with genetic algorithm-based hyperparameter tuning and hybrid feature selection," *Arabian Journal for Science and Engineering*, vol. 49, no. 9, pp. 13025–13043, 2024.

[32] L. Hong, K. Wehbi, and T. H. Alsalah, "Hybrid feature selection for efficient detection of DDoS attacks in IoT," in *Proceedings of the 2022 6th International Conference on Deep Learning Technologies*, pp. 120–127, 2022.

[33] G. Logeswari, J. D. Roselind, K. Tamilarasi, and V. Nivethitha, "A comprehensive approach to intrusion detection in IoT environments using hybrid feature selection and multi-stage classification techniques," *IEEE Access*, vol. 13, pp. 24970–24987, 2025.

[34] S. Mohanty, A. A. Acharya, T. Gaber, N. Panda, E. Eldesouky, and I. A. Hameed, "An efficient hybrid feature selection technique toward prediction of suspicious URLs in IoT environment," *IEEE access*, vol. 12, pp. 50578–50594, 2024.

[35] Y. Salami, Y. Ebazadeh, M. Hamrang, and N. Allahbakhshi, "A Novel Approach for Intrusion Detection System in IoT Using Correlation-Based Hybrid Feature Selection and Harris Hawk Optimization Algorithm," *Journal of Optimization of Soft Computing (JOSC)*, vol. 2, no. 3, pp. 7–21, 2024.

[36] R. B. Said, Z. Sabir, and I. Askerzade, "CNN-BiLSTM: A hybrid deep learning approach for network intrusion detection system in software-defined networking with hybrid feature selection," *IEEE Access*, vol. 11, pp. 138732–138747, 2023.

[37] M. Mohamad, A. Selamat, O. Krejcar, R. G. Crespo, E. Herrera-Viedma, and H. Fujita, "Enhancing big data feature selection using a hybrid correlation-based feature selection," *Electronics*, vol. 10, no. 23, p. 2984, 2021.

[38] Y. K. Saheed, A. A. Usman, F. D. Sukat, and M. Abdulrahman, "A novel hybrid autoencoder and modified particle swarm optimization feature selection for intrusion detection in the internet of things network," *Frontiers in Computer Science*, vol. 5, p. 997159, 2023.

[39] S. H. Lavate and P. Srivastava, "A hybrid feature selection approach based on random forest and particle swarm optimization for IoT network traffic analysis," *International Journal of Electrical and Electronics Research*, vol. 11, no. 2, pp. 568–574, 2023.

[40] A. Davahli, M. Shamsi, and G. Abaei, "A lightweight Anomaly detection model using SVM for WSNs in IoT through a hybrid feature selection algorithm based on GA and GWO," *Journal of Computing and Security*, vol. 7, no. 1, pp. 63–79, 2020.

[41] I. Zada, E. Omran, S. Jan, H. Alfraihi, S. Alsalamah, A. Alshahrani, S. Hayat, and N. Phi, "Enhancing IoT cybersecurity through lean-based hybrid feature selection and ensemble learning: A visual analytics approach to intrusion detection," *Plos one*, vol. 20, no. 7, p. e0328050, 2025.

[42] H. K. Bella and S. Vasundra, "Healthcare Intrusion Detection using Hybrid Correlation-based Feature Selection-Bat Optimization Algorithm with Convolutional Neural Network: A Hybrid Correlation-based Feature Selection for Intrusion Detection Systems," *International Journal of Advanced Computer Science & Applications*, vol. 15, no. 1, 2024.

[43] S. Velliangiri and P. Karthikeyan, "Hybrid optimization scheme for intrusion detection using considerable feature selection," *Neural Computing and Applications*, vol. 32, no. 12, pp. 7925–7939, 2020.

[44] M. A. Rahman, A. T. Asyhari, O. W. Wen, H. Ajra, Y. Ahmed, and F. Anwar, "Effective combining of feature selection techniques for machine learning-enabled IoT intrusion detection," *Multimedia Tools and Applications*, vol. 80, no. 20, pp. 31381–31399, 2021.

[45] M. Mohy-Eddine, A. Guezzaz, S. Benkirane, and M. Azrour, "An efficient network intrusion detection model for IoT security using K-NN classifier and feature selection," *Multimedia Tools and Applications*, vol. 82, no. 15, pp. 23615–23633, 2023.

[46] Y. Pourardebil Khah, M. Hosseini Shirvani, and H. Motameni, "A hybrid machine learning approach for feature selection in designing intrusion detection systems (IDS) model for distributed computing networks," *The Journal of Supercomputing*, vol. 81, no. 1, p. 254, 2025.

[47] J. Aswini, K. S. Rekha, R. A. A. Rosaline, and A. Sivaneshkumar, "Enhancing security in cloud computing systems using hybrid feature selection and

ensemble-based machine learning for intrusion detection," *Evolving Systems*, vol. 16, no. 3, p. 101, 2025.

[48] P. Kumar, G. P. Gupta, and R. Tripathi, "Toward design of an intelligent cyber attack detection system using hybrid feature reduced approach for iot networks," *Arabian Journal for Science and Engineering*, vol. 46, no. 4, pp. 3749–3778, 2021.

[49] R. B. Mallela, O. Srinivas, E. A. Goud, A. Rapaka, P. R. Sruthi, and S. N. Yasaswi, "Optimizing IoT DDoS Detection with Hybrid Feature Selection and Ensemble Learning," in *Proceedings of the 2025 3rd International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, pp. 1004–1008, IEEE, 2025.

[50] A. Hajjouz and E. Avksentieva, "Optimizing Intrusion Detection for DoS, DDoS, and Mirai Attack Subtypes Using Hierarchical Feature Selection and CatBoost on the CICIoT2023 Dataset," *Data and Metadata*, vol. 3, p. 577, 2024.

[51] M. A. Mohammed *et al.*, "Enhancing IoT Anomaly Detection Using Hybrid CNN-LSTM Model and Interpretable Feature Selection," *Zanco Journal of Pure and Applied Sciences*, vol. 37, no. 6, pp. 161–181, 2025.

[52] R. Ji, N. Kumar, and D. Padha, "Hybrid enhanced intrusion detection frameworks for cyber-physical systems via optimal features selection," *Indian J Sci Technol*, vol. 17, no. 30, pp. 3069–3079, 2024.

[53] K. R. Samsudiat, "Attack Detection in IoT Networks Using Hybrid Feature Selection and Bayesian Optimization," *Jurnal Nasional Teknik Elektro dan Teknologi Informasi*, vol. 14, no. 3, 2025.

[54] T. Abid, A. Ahmim, F. Maazouzi, D. Chefrour, I. Ullah, M. Ahmim, and R. Almukhlifi, "A Novel IoT Threat Detection Using GWO Feature Selection and CNN-Enhanced LightGBM," *Journal of Cloud Computing*, vol. 14, no. 1, p. 72, 2025.

[55] Q. Gulzar and K. Mustafa, "Enhancing network security in industrial IoT environments: A DeepCLG hybrid learning model for cyberattack detection," *International Journal of Machine Learning and Cybernetics*, pp. 1–19, 2025.

[56] J. B. Awotunde, O. Folorunso, and A. O. Adesina, "Intrusion Detection in Industrial Internet of Things Network Based on Deep Learning Model with Rule-Based Feature Selection," *Wireless Communications and Mobile Computing*, vol. 2021, p. 7154587, 2021.

[57] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, "CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment," *Sensors*, 2023. Dataset available at: https://www.unb.ca/cic/datasets/iotdataset-2023.html.

[58] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, "CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment," *Sensors*, vol. 23, no. 13, p. 5941, 2023.

[59] S. Roy, J. Li, B.-J. Choi, and Y. Bai, "A lightweight supervised intrusion detection mechanism for IoT networks," *Future Generation Computer Systems*, vol. 127, pp. 276–285, 2022.

[60] N. Kumar and S. Sharma, "A hybrid modified deep learning architecture for intrusion detection system with optimal feature selection," *Electronics*, vol. 12, no. 19, p. 4050, 2023.

[61] T. E. Ali, Y.-W. Chong, S. Manickam, M. N. Yusoff, K.-L. A. Yau, and A. D. Zoltan, "A stacking ensemble model with enhanced feature selection for distributed Denial-of-Service detection in Software-Defined networks," *Engineering, Technology & Applied Science Research*, vol. 15, no. 1, pp. 19232–19245, 2025.

[62] G. Logeswari, K. Thangaramya, M. Selvi, and J. D. Roselind, "An improved synergistic dual-layer feature selection algorithm with two type classifier for efficient intrusion detection in IoT environment," *Scientific Reports*, vol. 15, no. 1, p. 8050, 2025.

[63] A. Karunamurthy, K. Vijayan, P. R. Kshirsagar, and K. T. Tan, "An optimal federated learning-based intrusion detection for IoT environment," *Scientific Reports*, vol. 15, no. 1, p. 8696, 2025.

[64] P. Sinha, D. Sahu, S. Prakash, T. Yang, R. S. Rathore, and V. K. Pandey, "A high performance hybrid LSTM CNN secure architecture for IoT environments using deep learning," *Scientific Reports*, vol. 15, no. 1, p. 9684, 2025.

[65] S. D. A. Rihan, M. Anbar, and B. A. Alabsi, "Approach for Detecting Attacks on IoT Networks Based on Ensemble Feature Selection and Deep Learning Models," *Sensors (Basel, Switzerland)*, vol. 23, 2023.

[66] M. M. A. Elasaad, S. G. Sayed, and M. M. El-Dakroury, "AegisGuard: A Multi-Stage Hybrid Intrusion Detection System with Optimized Feature Selection for Industrial IoT Security," *Sensors (Basel, Switzerland)*, vol. 25, 2025.

[67] R. Alkanhel, E.-S. M. El-Kenawy, A. A. Abdelhamid, A. R. A. F. Ibrahim, M. A. Alohali, M. S. A. Abotaleb, and D. S. Khafaga, "Network Intrusion Detection Based on Feature Selection and Hybrid Metaheuristic Optimization," *Computers, Materials & Continua*, 2023.