8-18-2022 11:00 AM

Optimized and Automated Machine Learning Techniques Towards IoT Data Analytics and Cybersecurity

Li Yang, *The University of Western Ontario*

Supervisor: Shami, Abdallah, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Electrical and Computer Engineering

Abstract

The Internet-of-Things (IoT) systems have emerged as a prevalent technology in our daily lives. With the wide spread of sensors and smart devices in recent years, the data generation volume and speed of IoT systems have increased dramatically. In most IoT systems, massive volumes of data must be processed, transformed, and analyzed on a frequent basis to enable various IoT services and functionalities. Machine Learning (ML) approaches have shown their capacity for IoT data analytics. However, applying ML models to IoT data analytics tasks still faces many difficulties and challenges. The first challenge is to process large amounts of dynamic IoT data to make accurate and informed decisions. The second challenge is to automate and optimize the data analytics process. The third challenge is to protect IoT devices and systems against various cyber threats and attacks. To address the IoT data analytics challenges, this thesis proposes various ML-based frameworks and data analytics approaches in several applications.

Specifically, the first part of the thesis provides a comprehensive review of applying Automated Machine Learning (AutoML) techniques to IoT data analytics tasks. It discusses all procedures of the general ML pipeline. The second part of the thesis proposes several supervised ML-based novel Intrusion Detection Systems (IDSs) to improve the security of the Internet of Vehicles (IoV) systems and connected vehicles. Optimization techniques are used to obtain optimized ML models with high attack detection accuracy. The third part of the thesis developed unsupervised ML algorithms to identify network anomalies and malicious network entities (e.g., attacker IPs, compromised machines, and polluted files/content) to protect Content Delivery Networks (CDNs) from service targeting attacks, including distributed denial of service and cache pollution attacks. The proposed framework is evaluated on real-world CDN access log data to illustrate its effectiveness. The fourth part of the thesis proposes adaptive online learning algorithms for addressing concept drift issues (i.e., data distribution changes) and effectively handling dynamic IoT data streams in order to provide reliable IoT services. The development of drift adaptive learning methods can effectively adapt to data distribution changes and avoid data analytics model performance degradation.

Keywords: Machine Learning, AutoML, Data Analytics, Internet of Things, Content Delivery Networks, Network Security, Intrusion Detection System, Concept Drift, Data Stream Analysis, Hyper-parameter Optimization, Ensemble Learning, Transfer Learning.

Lay Summary

The Internet-of-Things (IoT) systems have emerged as a prevalent technology in our daily lives. With the wide spread of sensors and smart devices in recent years, the data generation volume and speed of IoT systems have increased dramatically. In most IoT systems, massive volumes of data must be processed, transformed, and analyzed on a frequent basis to enable various IoT services and functionalities. Machine learning (ML) is a subfield of Artificial Intelligence (AI), enabling machines to learn useful information and patterns from data without explicitly being programmed. ML algorithms have been developed as a promising technique that enables the rapid and accurate processing of massive volumes of data produced by IoT systems to identify patterns required by IoT services.

This thesis focuses on the use of ML algorithms in IoT data analytics and cyber-security applications. To improve ML models' learning performance and reduce human efforts in data analytics tasks, Automated Machine Learning (AutoML) and optimization techniques are studied and developed in this thesis to automatically obtain optimized ML models with the best performance. This thesis can be divided into four distinct parts. Specifically, the first part of the thesis provides a comprehensive review of applying AutoML techniques to IoT data analytics tasks. The second part of the thesis proposes several ML-based novel intrusion detection techniques to identify various types of common network attacks in vehicle networks and protect connected vehicles. The third part of the thesis develops ML algorithms to identify network anomalies (i.e., cyber-attacks) and malicious network entities (e.g., attacker IPs and compromised machines) in massive real-world network log data to protect Content Delivery Networks (CDNs), an essential network for Internet traffic communications, against cyber-attacks. The fourth part of the thesis proposes adaptive online learning algorithms for addressing concept drift issues in dynamic IoT data streams. Concept drift indicates unpredictable events, like the COVID-19 pandemic, which cause data distribution changes and data analytics performance degradation. The proposed methods can effectively handle ever-changing data stream patterns to provide reliable IoT services.

Co-Authorship Statement

The following thesis contains material from previously published papers and manuscripts submitted for publication that have been co-authored by Li Yang, Dr. Abdallah Shami, Dr. Abdallah Moubayed, Dr. Parisa Heidari, Dr. Amine Boukhtouta, Dr. Stere Preda, Adel Larabi, Richard Brunner, Daniel Migault, Dimitrios Michael Manias, Gary Stevens, Stephen DeRusett, and Ismail Hamieh. All the research, experiments, developments, and work presented here were carried out by Li Yang under the guidance of Dr. Abdallah Shami. Original manuscripts which make up parts of Chapters 2-10 in this thesis were also written by Li Yang.

Publications:

[J1] L. Yang and A. Shami, "On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[J2] L. Yang and A. Shami, "A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams," *IEEE Internet of Things Magazine*, vol. 4, no. 2, pp. 96-101, Jun. 2021.

[J3] L. Yang, A. Moubayed and A. Shami, "MTH-IDS: A Multitiered Hybrid Intrusion Detection System for Internet of Vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 616-632, Jan. 2022.

[J4] L. Yang, A. Moubayed, A. Shami, P. Heidari, A. Boukhtouta, A. Larabi, R. Brunner, S. Preda, and D. Migault, "Multi-Perspective Content Delivery Networks Security Framework Using Optimized Unsupervised Anomaly Detection," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 686-705, Mar. 2022.

[J5] L. Yang, A. Moubayed, A. Shami, A. Boukhtouta, P. Heidari, S. Preda, R. Brunner, D. Migault, and A. Larabi, "Forensic Data Analytics for Anomaly Detection in Evolving Networks", To appear in *Digital Forensics*, World Scientific, 2022.

[J6] L. Yang and A. Shami, "IoT Data Analytics in Dynamic Environments: From An Automated Machine Learning Perspective", To appear in *Engineering Applications of Artificial Intelligence*, 2022.

[J7] L. Yang and A. Shami, "A Multi-Stage Automated Online Network Data Stream Analytics Framework for IIoT Systems", Submitted to *IEEE Transactions on Industrial Informatics*, 2022.

[C1] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, "Tree-based Intelligent Intrusion Detection System in Internet of Vehicles," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1-6.

[C2] L. Yang, D. M. Manias, and A. Shami, "PWPAE: An Ensemble Framework for Concept Drift Adaptation in IoT Data Streams," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1-6.

[C3] L. Yang and A. Shami, "A Transfer Learning and Optimized CNN Based Intrusion Detection System for Internet of Vehicles," in *2022 IEEE International Conference on Communications (ICC)*, 2022, pp. 1-6.

[C4] L. Yang, A. Shami, G. Stevens, and S. DeRusett, "LCCDE: A Decision-Based Ensemble Framework for Intrusion Detection in The Internet of Vehicles," Accepted in *2022 IEEE Global Communications Conference (GLOBECOM)*, 2022, pp. 1-6.

Acknowlegements

First and foremost, I would like to express my sincere gratitude and appreciation to my supervisor, Dr. Abdallah Shami, for his professional guidance, unlimited support, and encouragement throughout my Ph.D. studies. His immense knowledge and broad experience have encouraged me throughout my academic research and daily life. It was a wonderful journey to learn from him, and it was my honor to work with him for the past four years.

Second, I would like to thank my thesis examination committee members, Dr. Kenneth McIsaac, Dr. Soodeh Nikan, Dr. Hanan Lutfiyya, and Dr. Chadi Assi, for their time and constructive suggestions on my thesis. I would also like to express my thanks and gratitude to my Master's supervisors, Dr. Radu Muresan and Dr. Arafat Al-Dweik, for leading me to get my foot in the door of research and giving me a lot of help in my Master's studies. I am also grateful to Ericsson's team members whom I have worked with during my Ph.D. studies, Dr. Parisa Heidari, Dr. Amine Boukhtouta, Dr. Stere Preda, Adel Larabi, Richard Brunner, and Daniel Migault, for their great insight, suggestions, and help. All the meetings with them have given me valuable experience in participating in industrial projects.

Third, I would like to express my deepest love and appreciation to my wife, Xin Lin, for her constant love, care, and support. I am glad to marry you during my Ph.D. studies. You have been living every day of my Ph.D., and without you, I would not have had the strength to decide to undertake this long journey. You are who, what, when, where, and why I love. You are my best friend, and you gave meaning to my past and future.

Fourth, I would like to thank all my colleagues and friends that I have had the chance to meet at Western University and The University of Guelph, Sam Aleyadeh, Abdallah Moubayed, Dimitri Michael Manias, Sara Zimmo, MohammadNoor Injadat, Ibrahim Shaer, Ibrahim Tamim, Anas Saci, Xin Huang, Yixuan Wu, Tianxiang Jia, Yanan Liu, Xiao Yang, Mengjie Li, Yifang Tian, Haide Wang, Chen Qiu, and many other friends who have provided their help during my Ph.D. journey. Special thanks to my constant friends in China, Renmin Gan, Shun Liu, and Xiao Luo, for their strong support and encouragement for more than ten years wherever I am.

Fifth, I would like to express my sincere love and gratitude to my family: my father, Shunde Yang, for always teaching me to be a better person and loving me unconditionally; my mother, Qing Li, for giving me life and taking care of me from the first day of my life; my aunt and uncle, Ping Li and Tianpeng Gu, for their unlimited support for my life in Canada; my mother-in-law and father-in-law, Xiubin Cai and De Lin, for taking care of my wife and me; my cousins, Haochuan Lin, Amanda Gu, and Steven Gu, for their company; and all other family members for their support. I highly appreciate their love and support throughout not only this degree but also in my life.

Finally, I am deeply grateful to my late grandparents, Chongsheng Li and Hongxiu Lu, who always encouraged me to pursue higher education and be a professor in the future. Without them, this Ph.D. degree would not have been completed. I miss them very much.

I am deeply grateful to all of you for making this journey memorable!

Contents

List of Figures

List of Tables

List of Abbreviations

Acc	Accuracy
ACE	Adaptive Classifier Ensemble
ADWIN	ADaptive WINdowing
AE	Auto-Encoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
AO	Account-Offering
AONN	Adaptive Online Neural Network
ARF	Adaptive Random Forest
AutoDP	Automated Data Pre-processing
AutoML	Automated Machine Learning
AV	Autonomous Vehicle
AVG	Average
AWE	Accuracy Weighted Ensemble
AWS	Amazon Web Services
BO	Bayesian Optimization
CAN	Controller Area Network
CASH	Combined Algorithm Selection and Hyperparameter tuning
CatBoost	Categorical Boosting
CCN	Content-Centric Networking
CDN	Content Delivery Network
CIC	Canadian Institute of Cybersecurity
CL	Cluster Labeling
CNN	Convolutional Neural Network
COVID-19	Coronavirus Disease 2019
CPA	Cache Pollution Attack
CT	Computational Time
CV	Connected Vehicle

CVFDT	Concept-adapting Very Fast Decision Tree
DBN	Deep Belief Networks
DBSCAN	Density-based Spatial Clustering of Applications with Noise
DD-FS	Drift-based Dynamic Feature Selection
DDM	Drift Detection Method
DDoS	Distributed-Denial-of-Service
DL	Deep Learning
DM	Data Mining
DNN	Deep Neural Networks
DoS	Denial of Service
DR	Detection Rate
DROS	Dynamic Random Over-Sampling
DRUS	Dynamic Random Under-Sampling
DT	Decision Tree
DTEL	Diversity and Transfer-based Ensemble Learning
DWM	Dynamic Weighted Majority
ECU	Electronic Control Unit
EDDM	Early Drift Detection Method
EFB	Exclusive Feature Bundling
EFDT	Extremely Fast Decision Tree
ET	Extra Trees
FAR	False Alarm Rate
FCBF	Fast Correlation-Based Filter
FE	Feature Engineering
FL	Federated Learning
FLA	False-Locality Attacks
FN	False Negative
FP	False Positive
FS	Feature Selection
GA	Genetic Algorithms

GB	GigaBytes
GBDT	Gradient-Boosted Decision Tree
GMM	Gaussian Mixture Model
GOSS	Gradient-based One-Side Sampling
GP	Gaussian Process
GPS	Global Positioning System
GPU	Graphics Processing Unit
GS	Grid Search
HATT	Hoeffding Anytime Tree
HITL	Human-In-The-Loop
HPO	Hyper-Parameter Optimization
HT	Hoeffding Trees
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICN	Information-Centric Networking
ID	Identifier
IDS	Intrusion Detection System
IE	Information Entropy
iForest	Isolation Forest
IG	Information Gain
IIoT	Industrial Internet of Things
IoT	Internet of Things
IoV	Internet of Vehicles
IP	Internet Protocol
ITS	Intelligent Transportation System
IVN	Intra-Vehicle Network
KL	Kullback-Leibler
KNN	K-Nearest Neighbors
KPCA	Kernel Principal Component Analysis
LB	Leverage Bagging

LCCDE	Leader Class and Confidence Decision Ensemble
LDA	Locality-Disruption Attacks
LightGBM	Light Gradient Boosting Machine
LR	Logistic Regression
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
Max	Maximum
MB	MegaBytes
ML	Machine Learning
MLP	Multi-Layer Perceptron
MNIST	Modified National Institute of Standards and Technology dataset
MOA	Massive Online Analysis
MSANA	Multi-Stage Automated Network Analytics
MSE	Mean Squared Error
MTH-IDS	Multi-Tiered Hybrid Intrusion Detection System
NB	Naïve Bayes
NDN	Named Data Networking
NWDAF	Network Data Analytics Function
OASW	Optimized Adaptive and Sliding Windowing
OBD II	On-Board Diagnostics II
OC-SVM	One-Class Support Vector Machine
OPA	Online Passive-Aggressive
PCA	Principal Component Analysis
PL	Paired Learner
Pre	Precision
PSO	Particle Swarm Optimization
PWPAE	Performance Weighted Probability Averaging Ensemble
RBF	Radial Basis Function
Rec	Recall
RF	Random Forest

RFE	Recursive Feature Elimination
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
ROS	Random Over Sampling
RPM	Revolutions Per Minute
RS	Random Search
RUS	Random Under-Sampling
SAM-KNN	Self-Adjusting Memory With K-Nearest Neighbor
SDP	Software-Defined Perimeter
SEA	Streaming Ensemble Algorithm
SGD	Stochastic Gradient Descent
SMOTE	Synthetic Minority Oversampling TEchnique
SRP	Streaming Random Patches
STDeep-Graph	Spatial-Temporal Deep Learning on Communication Graphs
SU	Symmetrical Uncertainty
SVM	Support Vector Machine
SVR	Support Vector Regression
SW	Sliding Window
TL	Transfer Learning
TN	True Negative
TP	True Positive
TPE	Tree Parzen Estimator
URL	Uniform Resource Locator
V2X	Vehicle-to-Everything
VANET	Vehicular Ad Hoc Network
VFDT	Very Fast Decision Tree
WiFI	Wireless Fidelity
W-PWPAE	Window-based Performance Weighted Probability Averaging Ensemble
XGBoost	Extreme Gradient Boosting

Chapter 1

Introduction

1.1 Overview and Motivation

By leveraging rapidly evolving communications technologies, the Internet of Things (IoT) permits the exchange of meaningful information and knowledge across IoT devices and systems to create value for humans [1]. IoT is defined as a network of connected devices and end systems that interact to collect, exchange, and analyze critical data [2]. Typical IoT applications include smart grids, intelligent vehicles, smart homes, smart agriculture, smart healthcare, and so on [2].

The growth and integration of IoT technology into our daily lives have exploded in recent years. According to a Cisco report, there will be more than 30 billion connected devices by the year 2023, growing from 18.4 billion devices in 2018 [3]. Additionally, more than 2.5 quintillion bytes of data are generated from IoT devices daily, and every person produces approximately 1.7 megabytes (MB) of data every second [4] [5]. The enormous increase in the scale of IoT devices and data poses severe challenges to the performance of IoT systems, posing challenges to IoT systems in terms of providing reliable services and making trustworthy decisions [1]. This is because IoT services and functionalities often require fast and accurate data analytics. Effective and efficient data analytics enables IoT systems to make fast decisions, gain rapid insights, discover hidden patterns, and interact with users and other systems [7]. Thus, it is critical to analyze and extract meaningful information from the data collected and generated in IoT systems to make informed decisions and meet users' needs.

On the other hand, due to the rapid development of IoT technology and the expanding connectivity of diverse communication networks and devices, modern networks and IoT systems are exposed to numerous system vulnerabilities, cyber-attacks, and malicious threats [8]. According to the Unit 42 IoT report, 98% of IoT traffic is unencrypted, and 57% of IoT devices are vulnerable to severe cyber-attacks [8]. The number and types of cyber-attacks are also increasing dramatically. For example, the number of Distributed-Denial-of-Service (DDoS) attacks is estimated to be 15.4 million worldwide by the year 2023 [3]. In complex and advanced IoT and modern networks, traditional security mechanisms, like firewalls and authentication mechanisms, are often insufficient to protect against malicious attacks, due to certain limitations like their ineffectiveness in identifying new/zero-day attacks [9].

Therefore, data analytics and cyber-security are two critical challenges in IoT systems. Effective data analytics and security enhancement techniques should be developed to enable IoT systems to provide reliable services to human beings.

Machine Learning (ML) and Data Mining (DM) techniques have been widely used in many IoT data analytics and cyber-security applications. ML can be defined as a group of methods that uses mathematical formulas to automatically explore, investigate, and extract data patterns. Extracting and gaining useful information enables ML models to make informed decisions and predictions. Once trained, an ML model can forecast new input data based on previous data patterns. Modern industry has been investing an increasing amount of money in developing and deploying ML models. According to a Forrester report, the global ML market will increase from $327.5 billion in 2021 to $554.3 billion in 2024 [10].

Due to their powerful data processing capabilities, ML algorithms have become critical contributors to IoT data analytics, enabling the rapid and accurate processing of massive volumes of data produced by IoT systems to identify patterns required by IoT services [2] [11]. Additionally, ML algorithms have become promising solutions for intrusion detection and system security enhancement. This is because ML algorithms can analyze the behaviors and status of IoT devices and network traffic to identify malicious activities and cyber-attacks by analyzing IoT sensor and traffic data. Intrusion Detection Systems (IDSs) developed using ML algorithms can be incorporated into IoT systems and modern networks to effectively identify malicious attacks that can breach firewalls and authentication mechanisms [12].

However, IoT data analytics and IDS development using ML algorithms still face many challenges. The data learning performance of current ML models still has much room for improvement. Even a minor error rate in IoT data analytics may cause severe consequences. For example, in smart healthcare systems, a single diagnostic error may result in delayed illness or even death. Additionally, deploying ML algorithms often requires intensive domain knowledge and human efforts [13].

On the other hand, due to the low-power and low-cost requirements of IoT devices, time and resource limits are the critical problems with IoT data analytics. Moreover, since certain IoT data generated in dynamic IoT environments is dynamic streaming data that changes over time, concept drift issues often occur in IoT data analytics [14]. Concept drift indicates the changes in data patterns over time due to dynamic IoT environments and unpredictable events, causing ML model performance degradation. For network security applications, although many previous studies have had some success constructing IDSs, effective intrusion detection is still a challenging problem due to the massive amount of network data, a large number of available network features, and the variety of cyber-attack patterns [9].

Therefore, many data analysts and ML researchers have been conducting research on Automated Machine Learning (AutoML) technology, which aims to complete data analytics tasks using ML algorithms with minimal human intervention. AutoML techniques are state-of-the-art solutions to automate IoT data analytics processes and reduce human efforts [13]. AutoML is also the process of automatically training and obtaining optimized ML models that can achieve optimal performance on specific datasets/tasks using optimization techniques. AutoML enables people to save valuable resources, including time, financial, and human resources, by automatically making accurate decisions.

Combined Algorithm Selection and Hyperparameter tuning (CASH) is the essential procedure of general AutoML solutions and data analytics pipelines because the suitable ML algorithms and their hyperparameter configurations have a substantial impact on the data learning performance [15]. The CASH procedure can be further divided into the automated model selection and Hyper-Parameter Optimization (HPO). Other components in AutoML pipelines, like data pre-processing and feature engineering, also significantly affect the outcomes of data analytics, but their automation still faces many challenges and usually requires human intervention. On the other hand, since certain IoT data generated in dynamic IoT environments is dynamic streaming data that changes over time, concept drift issues often occur in IoT data analytics [14]. Effective AutoML solutions for IoT dynamic data analytics should also incorporate automated model updating and concept drift-adaptive learning techniques.

To leverage human expertise and knowledge, Human-In-The-Loop (HITL) is introduced to develop ML models by combining machine intelligence with human intelligence [16]. HITL indicates the process that human experts supervise the ML process and help reduce prediction errors. Thus, Human experts can still participate in ML pipelines to make creative decisions, while AutoML techniques can help with tedious, repetitive, and laborious data analytics tasks with higher precision and less human effort [15] [16]. In the application of AutoML in IoT data analytics, certain procedures in AutoML pipelines, like data sampling and feature extraction, can be interfered with by HITL to deal with the high volumes of IoT data and make creative decisions. Human experts can also design promising ML model architecture or help determine the ML model candidates for further selection. After selecting or designing the appropriate ML model, the HPO and automated model updating procedures are the primary ML steps that can benefit from AutoML technology and are the focus of this thesis.

To summarize, this thesis focuses on the automation and optimization of ML algorithms for applications in IoT data analytics and cyber-security. The automation of ML models can largely reduce human efforts, while the optimization of ML models can obtain optimized ML models that can achieve improved performance on data analytics tasks.

1.2 Thesis Objectives

This thesis can be divided into four distinct parts. It focuses on the use and development of machine learning and optimization techniques for data analytics within the IoT system and cyber-security application fields. Accordingly, the first part comprehensively reviews applying AutoML and optimization techniques to IoT data analytics. The second part focuses on the development of intrusion detection systems for the Internet of Vehicles. The third part considers the anomaly detection and security enhancement in CDNs. Lastly, the fourth part focuses on the development of online learning algorithms for addressing concept drift issues in dynamic

IoT data streams.

The first part of the thesis presents a comprehensive review of the application of AutoML and optimization techniques to IoT data analytics problems. Chapter 2 explores the IoT data analytics frameworks, common ML algorithms, optimization methods, online learning methods, and open challenges in the relevant field. Moreover, it conducts a case study of applying optimization and AutoML methods to data analytics tasks.

The second part of the thesis proposes several Intrusion Detection Systems (IDSs) using ML algorithms for the Internet of Vehicles (IoV), as a representative IoT application. Specifically, Chapters 3 and 4 propose different ML models and ensemble learning strategies to develop IDSs for intrusion detection in IoV systems. Chapters 5 and 6 propose advanced ML models and optimization methods to develop robust IDSs that can detect various types of existing and zero-day attacks effectively and efficiently.

The third part of the thesis focuses on network anomaly detection in Content Delivery Networks (CDNs). Chapter 7 proposes unsupervised ML algorithms to identify abnormal events and network entities in CDNs and demonstrates the experimental results on real-world unlabeled network datasets provided by a major CDN operator.

Finally, the fourth part of the thesis proposes the use of online learning and concept drift adaptation methods for dynamic IoT stream analytics. As such, Chapters 8 and 9 propose novel concept drift adaptation methods using optimization and ensemble ML strategies. Chapter 10 proposes a complete pipeline for automated online network data stream analytics. This framework considers the automation of every standard step of online data stream analytics.

1.3 Thesis Outline and Contributions

In summary, the objective of this thesis is to solve IoT data analytics and cyber-security problems by studying, exploring, and developing the following three critical components involved in the AutoML pipeline: model designing/selection, hyperparameter optimization, and automated model updating. The thesis is composed of eleven chapters. An overview of the chapters and their connections are depicted in Fig. 1.1. The content and contributions of each chapter are summarized as follows.

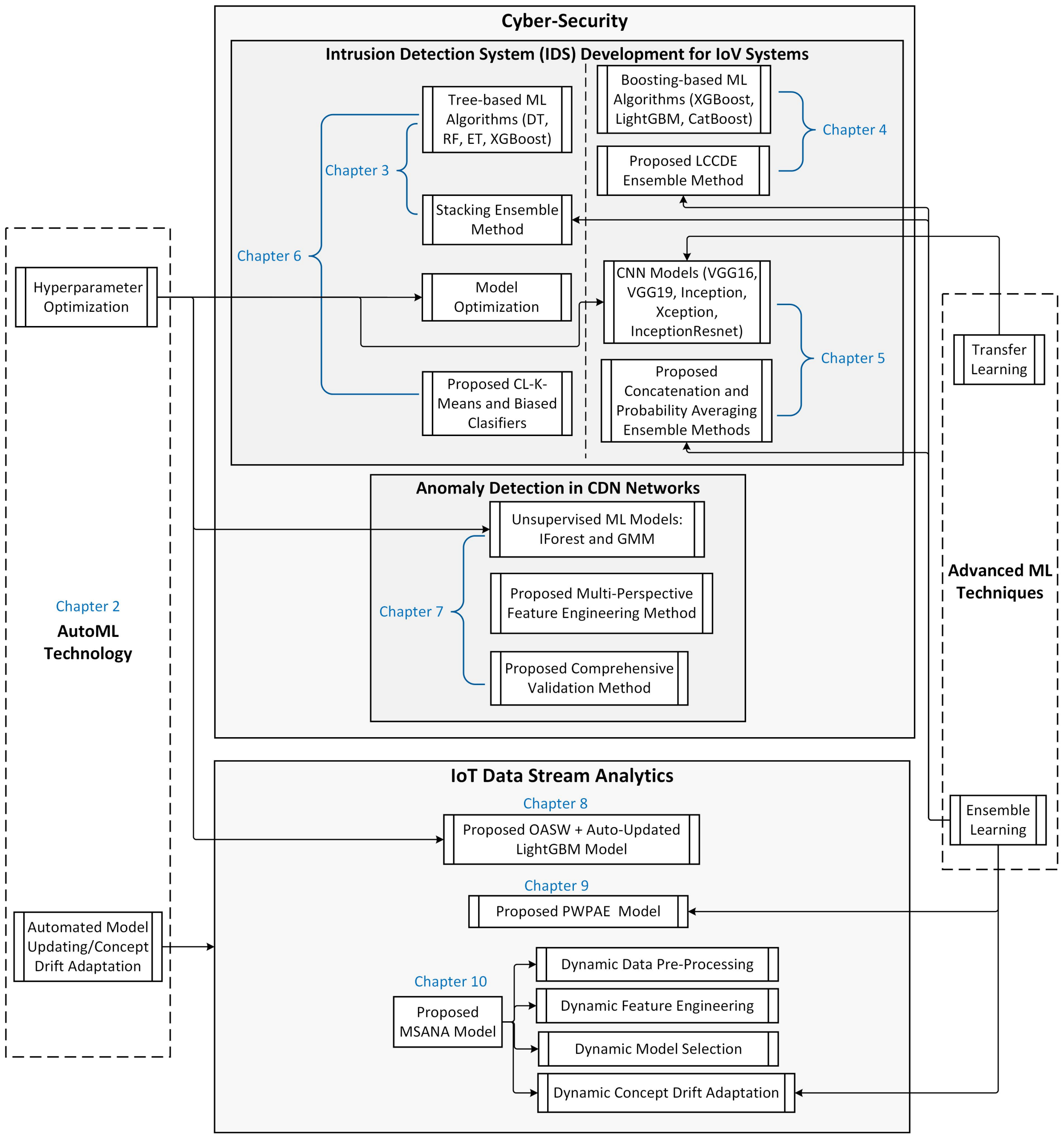
Cyber-Security
Intrusion Detection System (IDS) Development for IoV Systems
Tree-based ML Algorithms (DT, RF, ET, XGBoost)
Stacking Ensemble Method
Chapter 3
Chapter 6
Model Optimization
Proposed CL-K-Means and Biased Clasifiers
Boosting-based ML Algorithms (XGBoost, LightGBM, CatBoost)
Proposed LCCDE Ensemble Method
Chapter 4
CNN Models (VGG16, VGG19, Inception, Xception, InceptionResnet)
Proposed Concatenation and Probability Averaging Ensemble Methods
Chapter 5
Anomaly Detection in CDN Networks
Unsupervised ML Models: IForest and GMM
Proposed Multi-Perspective Feature Engineering Method
Proposed Comprehensive Validation Method
Chapter 7
Chapter 2
AutoML Technology
Hyperparameter Optimization
Automated Model Updating/Concept Drift Adaptation
Advanced ML Techniques
Transfer Learning
Ensemble Learning
IoT Data Stream Analytics
Chapter 8
Proposed OASW + Auto-Updated LightGBM Model
Chapter 9
Proposed PWPAE Model
Chapter 10
Proposed MSANA Model
Dynamic Data Pre-Processing
Dynamic Feature Engineering
Dynamic Model Selection
Dynamic Concept Drift Adaptation

Figure 1.1: An overview of the chapters in this thesis.

Chapter 1 gives a brief introduction to IoT systems, cyber-security, and AutoML technology. Additionally, the challenges in IoT data analytics and AutoML applications are discussed. Moreover, the thesis outline and contributions are provided and discussed. The content and contributions of the remaining ten chapters are summarized as follows.

Chapter 2 provides a comprehensive review of applying ML and optimization techniques to IoT and general data analytics applications. It introduces all the procedures of general AutoML pipelines, including data pre-processing, feature engineering, ML model learning, hyperparameter optimization, and automated model updating/concept drift adaptation. Additionally, it discusses and summarizes the advantages and limitations of the common methods for each AutoML procedure. Finally, it provides two comprehensive case studies to evaluate and compare the discussed methods. This review chapter will help industrial users, data analysts, and researchers to better develop and automate ML models in various data analytics applications. The contributions of this chapter are summarized as follows:

1. Define the overall framework and tasks for IoT data analytics.
2. Review common ML algorithms and their important hyper-parameters.
3. Review existing optimization methods for Hyper-Parameter Optimization (HPO), summarizes their pros and cons, and recommends the most appropriate method for specific situations.
4. Provide a comprehensive review of automated model updating, a novel AutoML procedure, by discussing concept drift detection and adaptation methods.
5. Review existing techniques for other important procedures of general AutoML pipelines, including data pre-processing, feature engineering, and performance metric selection.
6. Conduct comprehensive case studies by applying AutoML and HPO to practical IoT data analytics tasks.
7. Introduce many existing tools and libraries designed for AutoML and IoT data analytics.
8. Discuss the open challenges and research directions in the field of IoT data analytics and AutoML.

Chapter 3 proposes a tree-based intelligent IDS to protect Autonomous Vehicles (AVs) and Connected Vehicles (CVs) by identifying malicious cyber-attacks in network traffic data. Specifically, the proposed IDS trains four ML models, Decision Tree (DT), Random Forest (RF), Extra Trees (ET), and Extreme Gradient Boosting (XGBoost), as base learners, and then uses stacking, an ensemble learning strategy, to construct a robust ensemble learner by integrating the four base learners. The results from the implementation of the proposed IDS on benchmark network traffic datasets indicate that the system has the ability to identify various cyber-attacks in IoV systems. The contributions of this chapter are summarized as follows:

This chapter makes the following contributions:

1. Survey the vulnerabilities and potential attacks in both intra-vehicle and external vehicle networks.
2. Propose an intelligent IDS for both IoV and general networks by using the tree structure ML and ensemble learning methods.
3. Present a comprehensive framework to prepare network traffic data for the purpose of IDS development.
4. Propose an averaging feature selection method using tree structure ML models to improve the efficiency of the proposed IDS and to perform an analysis of network attributes and attacks for network monitoring uses.

Chapter 4 proposes a novel IDS framework named Leader Class and Confidence Decision Ensemble (LCCDE) to protect IoV systems against various types of existing cyber-attacks. It is constructed by determining the best-performing ML model among three advanced ML algorithms (XGBoost, LightGBM, and CatBoost) for every class or type of attack. The class leader models with their prediction confidence values are then utilized to make accurate decisions regarding the detection of various types of cyber-attacks. Experiments on two public IoV security datasets (Car-Hacking [17] and CICIDS2017 [18] datasets) demonstrate the effectiveness of the proposed LCCDE for intrusion detection on both intra-vehicle and external networks. The contributions of this chapter are summarized as follows:

1. Propose a novel ensemble framework, named LCCDE, for effective intrusion detection in IoVs using class leader and confidence decision strategies, as well as gradient-boosting ML algorithms.
2. Evaluate the proposed framework using two public IoV security datasets, the Car-Hacking and CICIDS2017 datasets, representing IVN and external network data, respectively.
3. Compare the performance of the proposed model with other state-of-the-art methods.

Chapter 5 improves the IDS frameworks for IoV systems by introducing two advanced techniques: transfer learning and Hyper-Parameter Optimization (HPO). Specifically, it uses the transfer learning strategy by transferring four state-of-the-art Convolutional Neural Network (CNN) models, named VGG16, VGG19, Xception, Inception, and InceptionResnet, to the IoV intrusion detection tasks by converting network traffic data to images. Thus, a novel data transformation method is also proposed. Moreover, it uses Particle Swarm Optimization (PSO), a powerful HPO method, to automatically tune the hyperparameters of CNN models to obtain optimized CNN models. Lastly, the base CNN models are integrated using two ensemble strategies, confidence averaging and concatenation, to further improve the intrusion detection performance. The contributions of this chapter are summarized as follows:

1. Propose a novel framework for effective cyber-attack detection in both intra-vehicle and external networks through CNN, transfer learning, ensemble learning, and HPO techniques.
2. Propose a data transformation method that can effectively transform vehicle network traffic data into images to more easily distinguish various cyber-attack patterns.
3. Evaluate the proposed method on two benchmark cyber-security datasets that represent intra-vehicle and external network data, and compare the model's performance with other state-of-the-art methods.

Chapter 6 discusses the vulnerabilities of intra-vehicle and external networks, and proposes a multi-tiered hybrid IDS that incorporates a signature-based IDS and an anomaly-based IDS to detect both known and unknown attacks on vehicular networks. The signature-based IDS is developed by extending the tree-based IDS framework proposed in Chapter 3 with the use of two HPO methods, Bayesian Optimization with Gaussian Process (BO-GP) and Bayesian Optimization with Tree Parzen Estimator (BO-TPE), to obtain optimized ML models. The anomaly-based IDS is developed by exploring a Cluster Labeling (CL) k-means method and biased classifiers. Experimental results illustrate that the proposed system can detect various types of known and zero-day attacks with high accuracy on the intra-vehicle network external vehicular network data. The contributions of this chapter are summarized as follows:

1. Propose a novel multi-tiered hybrid IDS that can accurately detect the various surveyed types of cyber-attacks launched on both intra-vehicle and external vehicular networks;
2. Propose a novel feature engineering model based on Information Gain (IG), Fast Correlation-Based Filter (FCBF), and Kernel Principal Component Analysis (KPCA) algorithms;

3. Propose a novel anomaly-based IDS based on CL-k-means and biased classifiers to detect zero-day attacks;
4. Discuss the use of Bayesian optimization techniques to automatically tune the parameters of each tier in the proposed IDS for model optimization;
5. Evaluate the performance and overall efficiency of the proposed model on two state-of-the-art datasets, CAN-intrusion-dataset and CICIDS2017, and discuss its feasibility in real-world IoV devices.

Chapter 7 proposes a multi-perspective unsupervised learning framework for anomaly detection in unlabeled Content Delivery Network (CDN) data. CDNs improve the connectivity and efficiency of global communications, but their caching mechanisms may be breached by cyber-attackers. Among the security mechanisms, effective anomaly detection forms an important part of CDN security enhancement. In the proposed framework, a multi-perspective feature engineering approach, an optimized unsupervised anomaly detection model that utilizes an isolation forest and a Gaussian mixture model, and a multi-perspective validation method, are developed to detect abnormal behaviors in CDNs mainly from the client Internet Protocol (IP) and node perspectives, therefore to identify the Denial of Service (DoS) and Cache Pollution Attack (CPA) patterns. Experimental results are presented based on the analytics of eight days of real-world CDN log data provided by a major CDN operator. Through experiments, the abnormal contents, compromised nodes, malicious IPs, as well as their corresponding attack types, are identified effectively by the proposed framework and validated by multiple cybersecurity experts. This shows the effectiveness of the proposed method when applied to real-world CDN data. The contributions of this chapter are summarized as follows:

1. Summarize the potential patterns and characteristics of DoS and CPA attacks to assist with anomaly detection in CDNs;
2. Propose a comprehensive network feature engineering model that generates features from multiple perspectives, including content, client IP, service provider, and account-offering perspectives;
3. Propose an optimized unsupervised anomaly detection model utilizing iForest, GMM, and Bayesian optimization (BO), to detect cyber-attacks and affected network entities

effectively;

4. Propose a multi-perspective result validation technique that can effectively reduce the false alarm rate and improve the detection rate of unsupervised CDN anomaly detection models.

Chapter 8 proposes an adaptive IoT streaming data analytics framework for anomaly detection use cases based on optimized LightGBM and concept drift adaptation. In recent years, various IoT services and functionalities have been provided by IoT systems based on the analytics of IoT streaming data. However, IoT data analytics faces concept drift challenges due to the dynamic nature of IoT systems and the ever-changing patterns of IoT data streams. A novel drift adaptation method named Optimized Adaptive and Sliding Windowing (OASW) is proposed to adapt to the pattern changes of online IoT data streams. Experiments on two public datasets, IoTID20 [19] and NSL-KDD [20] datasets, show the high accuracy and efficiency of our proposed adaptive LightGBM model compared against other state-of-the-art approaches. The proposed adaptive LightGBM model can perform continuous learning and drift adaptation on IoT data streams without human intervention. The contributions of this chapter are summarized as follows:

1. Discuss the challenges and potential solutions for IoT streaming data analytics.
2. Propose a novel drift adaptation method named OASW to address the concept drift issue. Its performance was evaluated through comparison with other state-of-the-art approaches.
3. Propose an optimized adaptive framework for IoT anomaly detection use cases with offline and online learning functionalities based on LightGBM, PSO, and OASW.

Chapter 9 provides a Performance Weighted Probability Averaging Ensemble (PWPAE) framework for drift adaptive IoT anomaly detection through IoT data stream analytics. The proposed framework is an ensemble learning framework that uses the combinations of two popular drift detection methods, ADaptive WINdowing (ADWIN) and Drift Detection Method (DDM), and two state-of-the-art drift adaptation methods, Adaptive Random Forest (ARF) and Streaming Random Patches (SRP), to construct base learners. The base learners are weighted according to their real-time performance and integrated to construct a robust anomaly detection ensemble model with improved drift adaptation performance. Experiments on two public datasets show the effectiveness of our proposed PWPAE method compared against state-of-the-art methods. The contributions of this chapter are summarized as follows:

1. Investigate concept drift adaptation methods.
2. Propose a novel drift adaptation method named PWPAE to address the performance limitations of current concept drift methods.
3. Evaluate the proposed PWPAE framework on two public IoT cyber-security datasets, IoTID20 [19] and CICIDS2017 [18], for IoT anomaly detection use cases.

Chapter 10 proposes a multi-stage online learning framework, named Multi-Stage Automated Network Analytics (MSANA), for automated data stream analytics and concept drift adaptation for IoT systems. It consists of four stages: dynamic data pre-processing, drift-based dynamic feature selection, base model learning and selection, and online ensemble model development. As a representative application of IIoT data analytics, the proposed framework is evaluated on two benchmark IoT anomaly detection datasets, IoTID20 [19] and CICIDS2017 datasets [18], to solve IIoT security problems. The proposed system can be utilized as a network analytics automation application for IIoT in industry 5.0. The contributions of this chapter are summarized as follows:

1. Propose MSANA, a novel and comprehensive framework for automated data stream analytics in IIoT systems, which includes typical data analytics procedures.
2. Propose the Window-based Performance Weighted Probability Averaging Ensemble (W-PWPAE) method, a novel ensemble drift adaptation strategy for online learning on dynamic data streams.
3. Propose a novel dynamic feature selection method for data stream analytics with concept drift issues.
4. Evaluate the proposed framework on two public IoT security datasets as a case study, and compares it with various state-of-the-art online learning approaches.

Finally, Chapter 11 concludes all the contributions presented in the previous chapters. The future research directions are also discussed in Chapter 11.

Bibliography

[1] T. Yu and X. Wang, “Real-Time Data Analytics in Internet of Things Systems,” in *Handbook of Real-Time Computing*, Singapore: Springer Singapore, 2020, pp. 1–28.

[2] E. Adi, A. Anwar, Z. Baig, and S. Zeadally, “Machine Learning and Data Analytics for the IoT,” *Neural Comput. Appl.*, vol. 32, no. 20, pp. 16205–16233, Oct. 2020.

[3] Cisco Annual Internet Report (2018–2023) White Paper, [online] Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[4] S. S. Goel, A. Goel, M. Kumar, and G. Moltó, “A review of Internet of Things: qualifying technologies and boundless horizon,” *J. Reliab. Intell. Environ.*, vol. 7, no. 1, pp. 23–33, 2021.

[5] Stuart, Product & Data, “IoT + Big Data: Welcoming the Data Revolution [2018 Infographic],” Feb. 2018, Available at: http://www.redpixie.com/blog/iotbig-data.

[6] D. Liu et al., “Sensors Anomaly Detection of Industrial Internet of Things Based on Isolated Forest Algorithm and Data Compression,” *Sci. Program.*, vol. 2021, 2021.

[7] M. Marjani et al., “Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges,” *IEEE Access*, vol. 5, pp. 5247–5261, 2017.

[8] 2020 Unit 42 IoT Threat Report, March 2020, [online] Available: https://unit42.paloaltonetworks.com/iot-threat-report-2020/.

[9] L. Yang, A. Moubayed, and A. Shami, “MTH-IDS: A Multi-Tiered Hybrid Intrusion Detection System for Internet of Vehicles,” *IEEE Internet Things J.*, 2021.

[10] L. Columbus, “Roundup of machine learning forecasts and market estimates, 2020,” *Forbes*, 2020.

[11] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, “Machine learning towards intelligent systems: applications, challenges, and opportunities,” *Artif. Intell. Rev.*, 2021.

[12] F. Salo, M. Injadat, A. B. Nassif, A. Shami, and A. Essex, ”Data mining techniques in intrusion detection systems: A systematic literature review,” in *IEEE Access*, vol. 6, pp. 56046-56058, 2018.

[13] Q. Yao et al., “Taking Human out of Learning Applications: A Survey on Automated Machine Learning,”, *arXiv*, pp. 1–26, 2018.

[14] L. Yang and A. Shami, “A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams,” *IEEE Internet Things Mag.*, vol. 4, no. 2, pp. 96–101, 2021.

[15] X. He, K. Zhao, and X. Chu, “AutoML: A survey of the state-of-the-art,” *Knowledge-Based Syst.*, vol. 212, no. January, p. 106622, 2021.

[16] A. Holzinger, “Interactive machine learning for health informatics: when do we need the human-in-the-loop?,” *Brain Informatics*, vol. 3, no. 2, pp. 119–131, 2016.

[17] E. Seo, H. M. Song, and H. K. Kim, “GIDS: GAN based Intrusion Detection System for In-Vehicle Network,” *2018 16th Annu. Conf. Privacy, Secur. Trust*, pp. 1–6, 2018.

[18] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” *in Proc. Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[19] I. Ullah and Q. H. Mahmoud, “A Scheme for Generating a Dataset for Anomalous Activity Detection in IoT Networks,” in *Advances in Artificial Intelligence*, 2020, pp. 508–520.

[20] J. Liu, B. Kantarci, and C. Adams, “Machine learning-driven intrusion detection for Contiki-NG-based IoT networks exposed to NSL-KDD dataset,” *WiseML 2020 - Proc. 2nd ACM Work. Wirel. Secur. Mach. Learn.*, pp. 25–30, 2020, doi: 10.1145/3395352.3402621.

Chapter 2

Background on IoT Data Analytics and Automated Machine Learning

2.1 Introduction

In this chapter, we discuss how AutoML techniques can be applied to IoT data analytics problems to address the existing challenges. To apply ML models to IoT data analytics tasks, there are several major difficulties and challenges [3] [4]:

1. It is usually time-consuming and requires expert knowledge to manually tune ML models to fit each specific task.
2. Many IoT data streams have concept drift issues, causing ML model performance degradation.
3. The community lacks public and comprehensive benchmarks or practical examples in the AutoML field for IoT data analytics.

To overcome the above difficulties/challenges, the primary achievements and contributions of this chapter are as follows:

1. Review common ML algorithms and their important hyper-parameters.

Parts of the content in this chapter has been published in Neurocomputing [1] and Engineering Applications of Artificial Intelligence [2].

2. Review existing optimization methods for Hyper-Parameter Optimization (HPO), summarizes their pros and cons, and recommends the most appropriate method for specific situations.
3. Provide a comprehensive review of automated model updating, a novel AutoML procedure, by discussing concept drift detection and adaptation methods.
4. Conduct two comprehensive case studies by applying AutoML and HPO to practical IoT data analytics tasks; the code is made open access.

Moreover, this chapter makes the following secondary contributions:

1. Define the overall framework and tasks for IoT data analytics.
2. Review existing techniques for other important procedures of general AutoML pipelines, including data pre-processing, feature engineering, and performance metric selection.
3. Introduce many existing tools and libraries designed for AutoML and IoT data analytics.
4. Discuss the open challenges and research directions in the field of IoT data analytics and AutoML.

This chapter is organized as follows: Section 2.2 presents the properties of IoT data, as well as the layers and tasks of IoT data analytics. Section 2.3 provides an overview of the AutoML and HPO technology. Section 2.4 reviews the common ML algorithms and their important hyperparameters. Section 2.5 covers the various state-of-the-art optimization approaches that have been proposed for HPO and AutoML tasks. Sections 2.6 & 2.7 discuss the automated data pre-processing and feature engineering procedures, respectively. Section 2.8 discusses the automated model updating process by introducing concept drift detection and adaptation methods. Section 2.9 describes the selection of appropriate performance metrics and validation methods for ML tasks. Section 2.10 introduces the tools and libraries for AutoML and time-series analytics. Sections 2.11 & 2.22 present two case studies of applying AutoML and HPO to data analytics problems and discuss the experimental results. Section 2.13 discusses the challenges and research directions of AutoML and IoT data analytics. Section 2.14 concludes the chapter.

Code for this chapter is available at: https://github.com/LiYangHart/Hyperparameter-Optimization-of-Machine-Learning-Algorithms

2.2 IoT Data Analytics

2.2.1 IoT Data Characteristics

Although IoT data is similar to data from many other fields, there are several factors affecting the efficacy of IoT data analytics, such as dynamic IoT environments and time series characteristics. Overall, IoT data has the following characteristics [7]:

1. **High volume**: With the development of large-scale IoT systems, a massive amount of data is continuously generated from a large number of IoT devices. As described in Chapter 1, more than 2.5 quintillion bytes of data are generated from IoT devices daily [8] [9]. Both real-time and historical IoT data must be saved and processed to analyze previous patterns and future trends, resulting in a large volume of IoT data requiring analytics.
2. **High velocity**: IoT data is often generated at high speed by a large number of IoT devices. To achieve real-time analytics, the processing speed of IoT data should be higher than the generation speed of the data. Thus, efficient data analytics techniques should be developed to process the IoT data with high generation speed.
3. **High variability**: Due to the dynamic nature of IoT environments, IoT data is often dynamic data with varying distributions over time. For example, when specific events occur, such as the occurrence of Coronavirus Disease 2019 (COVID-19), the generated IoT data would significantly change.
4. **Time-series/Temporal correlation**: As IoT devices collect data over time, the IoT data is often created with timestamps or time-related information. Due to environmental factors, IoT data is often time-series data with strong temporal correlations. Thus, time-series analysis is often beneficial for IoT data analytics.

In IoT systems, the majority of data is created in the form of time-series data that has temporal correlations (*i.e.,* the sample collected at time t is related to the data samples collected at previous times [$t-1$ to $t-n$]) [11]. A time-series dataset is a collection of measurements or observations collected in chronological order. In time-series data, time is a dependent variable of the target variable. Time-series prediction is the process of predicting future trends using past observations. Global temperature prediction, energy consumption prediction, and IoT device failure detection are typical examples of IoT time-series data analytics tasks.

On the other hand, real-world IoT data is often non-stationary time-series data with varying mean, variance, or autocorrelation [11]. Due to the dynamic nature of IoT settings and environments, IoT streaming data is subject to a range of data distribution adjustments. For instance, the physical events observed by IoT sensors may evolve over time, rendering sensing components outdated or necessitating periodic updates. As such, changes in the distribution of IoT data over time, referred to as concept drift, are often inevitable [10].

Concept drift may hamper the decision-making capabilities of IoT data analytics models, which might have a negative impact on IoT systems [10]. For example, the misleading decision-making process performed by an IoT anomaly detection model with concept drift issues may significantly impair detection accuracy, leaving the IoT system vulnerable to a range of hostile cyber-attacks. When a concept or distribution in IoT data changes, it should be properly handled. Thus, proper analytics approaches should be used to address concept drift issues in dynamic or online IoT data analytics tasks.

2.2.2 IoT Data Analytics Layers

Typical IoT systems consist of three major layers: data collection, transmission, and analytics layers, as shown in Fig. 2.1 [7]. The data collection layer is comprised of IoT end devices that are used to detect, collect, and store sensor data. IoT devices can form sub-systems, like smart homes and Intelligent Transportation Systems (ITSs). IoT end devices and sub-systems are the basic and core components of IoT systems, which directly interact with their physical IoT environments through sensors and actuators. The transmission layer is enabled by gateways to transmit data between IoT end devices and edge/cloud servers. Common transmission strategies include cellular networks, Wireless Fidelity (WiFi), Bluetooth, Zigbee, and so on [10]. The analytics layer is responsible for processing and analyzing data from IoT devices, which can be completed in both cloud and edge servers.

Edge and cloud computing are modern computing and storage paradigms for intelligent IoT data analytics [6]. Firstly, IoT data can be processed locally on IoT end devices or edge

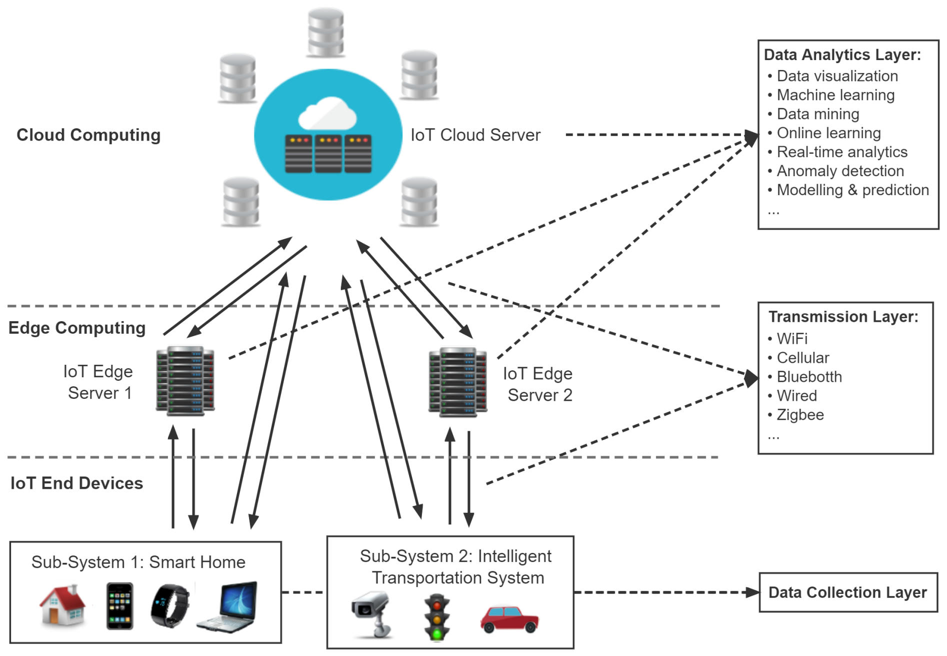

Figure 2.1: An overview of IoT data analytics architecture.

servers through edge computing. Edge computing enables fundamental IoT data analytics inside the local network of the data source to avoid long-distance transmissions and return real-time processing responses. However, to reduce IoT system costs, IoT end devices are usually constructed with limited computational power and resources. Hence, IoT end devices or edge servers can only perform basic and initial data processing due to their limited resources. It is usually difficult for these resource-constrained IoT devices to perform computational-intensive data analytics tasks.

On the other hand, cloud servers can be used to handle large-scale IoT data and perform complex data analytics tasks. Cloud computing is a paradigm in which data is stored, gathered, managed, and processed on remotely placed computing servers connected over the Internet [5]. Cloud computing support many services, including Software as a Service (SaaS), Infrastructure as a Service (IaaS), and Platform as a Service (PaaS). As IoT systems can generate a massive amount of data, cloud servers are required for the storage and analytics of large and complex IoT data. One major limitation of cloud computing is the high overhead of processing huge amounts of data, since transmitting IoT data streams between cloud and edge devices would require additional costs, bandwidth, and power. Thus, it is usually difficult to achieve real-time analytics by only using cloud servers.

Thus, collaborative computing, including both edge and cloud computing, is often used in modern IoT systems for large IoT data analytics tasks [12]. AutoML methods can be operated at both central servers and local edge servers for IoT data analytics applications to improve learning performance and reduce human effort. In IoT systems, any device with computational, communication, and storage capabilities can be used as edge computing devices, such as smart gateways and lightweight base stations. AutoML methods can be implemented in these edge computing devices to perform fundamental and preliminary analytics on the local sensor data. Many preliminary data analytics procedures, such as data pre-processing, feature engineering, storage, and initial analytics, can be conducted at edge devices, which reduces the burden of communication and resource consumption on the cloud servers.

Although many IoT edge devices enable local processing of IoT data, only basic functionalities are feasible in local processing due to the limited computing capability of many IoT end devices [5]. Therefore, large IoT data is often transmitted to the cloud server for comprehensive analytics. AutoML methods can be implemented in the central machines with strong computational power on the cloud. After receiving the pre-processed data transferred from the edge devices, the central machines on the cloud servers will comprehensively process and analyze the data using AutoML techniques and return the analytics results to IoT end devices. Cloud servers can process data faster and more accurately than edge devices due to their powerful computational capabilities, but they can incur additional latency owing to the data transmission process [12]. Thus, it is required to strike a balance between the data analytics time and transmission time.

To summarize, cloud computing is suitable for complicated, large-scale, and delay-tolerant data analytics tasks due to its high computational power, while edge computing is suitable for low-latency and real-time data analytics tasks owing to its ability to process data locally. For example, in IoT anomaly detection applications, if multiple edge devices or nodes are under different types of attacks, each edge server can collect and analyze local IoT traffic data using AutoML methods to detect the type of attack that it suffered as a fundamental attack detection, while the cloud server can collect the attack data transmitted from multiple edge servers using AutoML methods to detect various types of attacks as a comprehensive attack detection. The local and fundamental attack detection results can protect each local device from the types of attacks it has suffered, while the comprehensive attack detection results from the cloud server can prevent each local device from both the old and new types of attacks. Deploying AutoML techniques using collaborative computing can provide IoT devices and users with rapid and reliable services.

2.2.3 IoT Data Analytics Tasks

IoT data analytics tasks can be classified into classification, regression, clustering, and anomaly detection tasks [13]:

1. **Classification**: Given a collection of labeled time-series data, the objective of a classification task is to train a classifier capable of assigning correct labels to new time-series data samples. Training on labeled samples enables a classifier to identify distinctive characteristics that can be used to distinguish different classes.
2. **Regression**: Regression tasks aim to determine the relationship between a series of input features and a continuous target variable. For IoT time-series data analytics, regression tasks are often prediction tasks that aim to predict future trends and measurements using historical data samples with temporal dependencies.
3. **Clustering**: Clustering is the process of dividing data samples into a number of groups according to their natural characteristics and patterns. The purpose of clustering is to group similar data samples into the same group and separate the different data samples into different groups.
4. **Anomaly detection**: In IoT time-series analytics, anomaly detection is the process of detecting abnormal sequences in a time series to analyze abnormal events. The conventional anomaly detection process is to model normal patterns and then identify sequences that deviate from them.

Moreover, IoT data analytics algorithms can also be classified as batch learning and online learning algorithms, depending on the type of IoT data to be processed [10].

1. **Batch learning**: Batch learning methods analyze static IoT data in batches and often need access to the entire dataset prior to model training. Traditional ML algorithms can

effectively solve batch learning tasks [14]. Although batch learning models often achieve high performance due to their ability to learn diverse data patterns, it is often difficult to update these models once created. Therefore, batch learning faces two significant challenges: model degradation and data unavailability.

2. **Online learning**: Online learning techniques are able to train models using continuously incoming online IoT data streams in dynamic IoT environments [15]. By learning a single data sample at a time, online learning models can reduce memory requirements for data storage and learn new data patterns. Additionally, online learning models can often achieve real-time processing and address concept drift issues. Thus, when applied to dynamic data streams or when inadequate data is available, online learning is often more effective than batch learning.

2.3 AutoML Overview

Although there are many existing ML algorithms that are commonly used in IoT data analytics applications to analyze IoT data and make decisions, a randomly-selected ML model with default architecture or hyperparameter configuration usually cannot achieve the optimal analytics results and make accurate decisions. Thus, experienced data scientists are required in many procedures of ML pipelines, including preparing appropriate and clean data, selecting the most suitable ML algorithm, tuning hyperparameters, and determining whether the model needs to be updated. Data scientists often conduct experiments using a variety of ML algorithms and hyperparameter values in order to determine the most efficient combination. These procedures are labor-intensive, time-demanding, and require specialized expertise in ML and data analysis [16]. The process of automating this ML design and tuning process is referred to as AutoML. Thus, AutoML refers to the fully automated process of applying machine learning to real-world and practical applications. AutoML can be used by both beginners and experts to apply ML models efficiently. It has the potential to significantly improve the performance and effectiveness of ML models by shortening work cycles, enhancing model performance, and even possibly eliminating the necessity for data scientists. Hence, AutoML is a promising solution to data scientist shortage and high labor costs. In this Section, the basic concept and common optimization techniques of AutoML technology are discussed.

There are three significant benefits of using AutoML:

1. It increases efficiency and reduces computational costs by automating repetitive ML processes.
2. It assists in avoiding mistakes caused by human labor.
3. It lowers the threshold of implementing ML models by requiring less ML expertise and experience.

Moreover, manually selecting, designing, and tuning ML algorithms for IoT data analytics is usually more time-consuming than AutoML [17]. The objective of AutoML is to enable the automation of the tedious and time-consuming ML model selecting, designing, and tuning procedures by machines instead of humans. Thus, using AutoML techniques will not increase the IoT data analytics time compared with using traditional ML models, as they automate and simplify the selecting, designing, and tuning procedures of traditional data analytics models.

An overview of the AutoML pipeline for IoT time-series data analytics is illustrated in Fig. 2.2. It consists of four stages: automated data pre-processing, automated feature engineering, automated model learning, and automated model updating [16]. Automated model learning can be further divided into automated model selection and Hyper-Parameter Optimization (HPO). AutoML begins with data pre-processing, which aims to transform the original data into a sanitized version. It is a time-consuming and important procedure that has a significant effect on learning performance. The next step is feature engineering, which includes feature extraction and selection. This stage preserves important patterns of datasets while enhancing learning generalization. The following step is model selection, which uses an optimization technique to identify the optimal ML algorithm that produces the most accurate predictions. The process of tuning hyperparameters, referred to as hyper-parameter optimization, aims to further enhance the model learning performance. AutoML systems often need to use a range of optimization algorithms to carry out these phases in the pipeline. Sections 2.4 - 2.9 will explain every step of the AutoML pipeline in depth.

To apply ML algorithms to real-world problems, the primary challenge is selecting and configuring ML. In a prediction task, the accuracy of different models with different configu-

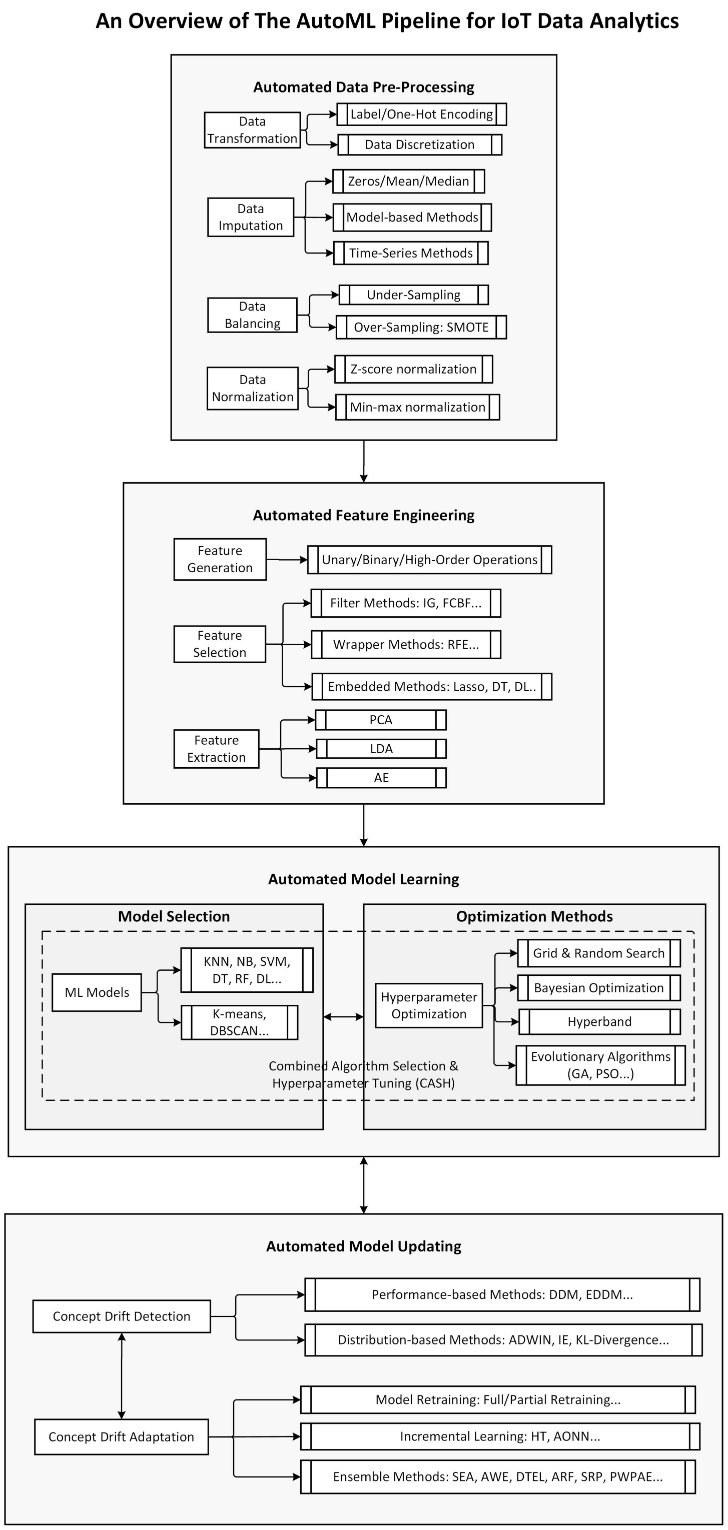
An Overview of The AutoML Pipeline for IoT Data Analytics
Automated Data Pre-Processing
Data Transformation
Label/One-Hot Encoding
Data Discretization
Data Imputation
Zeros/Mean/Median
Model-based Methods
Time-Series Methods
Data Balancing
Under-Sampling
Over-Sampling: SMOTE
Data Normalization
Z-score normalization
Min-max normalization
Automated Feature Engineering
Feature Generation
Unary/Binary/High-Order Operations
Feature Selection
Filter Methods: IG, FCBF...
Wrapper Methods: RFE...
Embedded Methods: Lasso, DT, DL..
Feature Extraction
PCA
LDA
AE
Automated Model Learning
Model Selection
ML Models
KNN, NB, SVM, DT, RF, DL...
K-means, DBSCAN...
Optimization Methods
Hyperparameter Optimization
Grid & Random Search
Bayesian Optimization
Hyperband
Evolutionary Algorithms (GA, PSO...)
Combined Algorithm Selection & Hyperparameter Tuning (CASH)
Automated Model Updating
Concept Drift Detection
Performance-based Methods: DDM, EDDM...
Distribution-based Methods: ADWIN, IE, KL-Divergence...
Concept Drift Adaptation
Model Retraining: Full/Partial Retraining...
Incremental Learning: HT, AONN...
Ensemble Methods: SEA, AWE, DTEL, ARF, SRP, PWPAE...

Figure 2.2: The overview of AutoML pipeline for IoT time-series data.

rations might vary significantly [18]. Therefore, it is critical to determine the most appropriate ML model with the optimal hyperparameter configuration.

Hyperparameters are the parameters of ML algorithms that determine the architecture of ML models and must be specified prior to model learning [4]. Hyperparameters can be classified into three types based on their domains: continuous hyperparameters (*e.g.,* the learning rate of neural networks), discrete hyperparameters (*e.g.,* the number of clusters in k-means), and categorical hyperparameters (*e.g.,* the kernel type in support vector machines). Additionally, certain hyperparameter configurations have conditionality. For example, the two important hyperparameters in DBSCAN, the scan radius and the minimum included points, have strong correlations to determine the data density together [4]. Conditional hyperparameters must be tuned together in order to identify the optimal configuration.

In ML model learning, selecting appropriate ML algorithms and hyperparameter values can be seen as a search problem. All the potential ML models and their hyperparameter combinations define a search space, and a single model with a hyperparameter configuration can be seen as a point in the search space. Detecting the optimal point in the search space is a global optimization problem.

Therefore, using optimization techniques to automatically detect the best ML algorithm with the optimal hyperparameter configuration is defined as a Combined Algorithm Selection and Hyperparameter (CASH) optimization problem [19]. CASH is a core component of current AutoML systems. CASH systems are divided into two stages: model selection and HPO. At the first stage, suitable ML models are selected with their default hyperparameters. At the second stage, model-specific hyperparameters are tuned to obtain the optimal final model [18].

In general, the CASH problem is defined as finding the ML algorithm and hyperparameter configuration that minimizes the loss function. It can be described as follows [19]:

$$A^{\star}, \lambda_{\star} \in \operatorname*{argmin}_{A(j)\in\mathcal{A}, \lambda\in\Lambda^{(j)}} \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}\left(A_{\lambda}^{(j)}, D_{\text{train}}^{(i)}, D_{\text{valid}}^{(i)}\right) \tag{2.1}$$

where $A^{\star} \in A$ is the algorithm to be chosen, λ is the hyperparameters of the algorithms to be tuned, D_{train} and D_{valid} denote the training and validation sets, K denotes k-fold cross-validation.

To summarize, CASH is the process of applying optimization methods to select ML models and tune their hyperparameters in order to achieve optimal or near-optimal performance based on defined metrics within a certain time budget [20]. In this thesis, the focus is the HPO procedure for ML model optimization, while model selection and design procedures still require human expertise. The existing optimization techniques for CASH and HPO problems are discussed in Section 2.5.

2.4 Machine Learning Algorithm Overview

In general, ML models can be classified as supervised and unsupervised learning algorithms, based on whether they are built to model labeled or unlabeled datasets [21]. Supervised learning algorithms are a set of machine learning algorithms that map input features to a target by training on labeled data, and mainly include linear models, k-nearest neighbors (KNN), support vector machines (SVM), naïve Bayes (NB), decision-tree-based models, and deep learning (DL) algorithms [22]. Unsupervised learning algorithms are used to find patterns from unlabeled data and can be divided into clustering and dimensionality reduction algorithms based on their aims. Clustering methods mainly include k-means, density-based spatial clustering of applications with noise (DBSCAN), and expectation-maximization (EM); while principal component analysis (PCA) is the most representative dimensionality reduction algorithm [23]. Moreover, there are several ensemble learning methods that combine different singular models to further improve model performance, like voting, bagging, and AdaBoost. To optimize ML models, firstly, we need to find out what the key hyper-parameters are that people need to tune to fit the ML models into specific problems or datasets. Thus, in this chapter, the important hyper-parameters of common ML models are studied based on their names in Python libraries, including scikit-learn (sklearn) [24], XGBoost [25], and Keras [26].

2.4.1 Supervised Learning Algorithms

In supervised learning, both the input x and the output y are available, and the goal is to obtain an optimal predictive model function f^* to minimize the cost function $\mathcal{L}(f(x), y)$ that models the error between the estimated output and ground-truth labels. The predictive model function

f varies based on its model structure. With limited model architectures determined by different hyper-parameter configurations, the domain of the ML model function f is restricted to a set of functions F. Thus, the optimal predictive model f^* can be obtained by [27]:

$$f^* = \arg\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x_i\right), y_i\right) \tag{2.2}$$

where n is the number of training data points, x_i is the feature vector of the i-th instance, y_i is the corresponding actual output, and L is the cost function value of each sample.

Many different loss functions exist in supervised learning algorithms, including the square of Euclidean distance, cross-entropy, information gain, etc. [27]. On the other hand, different ML algorithms generate different predictive model architectures based on different hyper-parameter configurations, which will be discussed in detail in this subsection.

Linear Models

In general, supervised learning models can be classified as regression and classification techniques when used to predict continuous or discrete target variables, respectively. Linear regression [28] is a typical regression model that predicts a target y by the following equation:

$$\hat{y}(\mathbf{w}, \mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_p x_p, \tag{2.3}$$

where the target variable y is expected to be a linear combination of p input features $\mathbf{x} = \left(x_1, \cdots x_p\right)$, and $\hat{y}$ is the predicted value. The weight vector $\mathbf{w} = \left(w_1, \cdots w_p\right)$ is designated as an attribute 'coef_', and w_0 is defined as another attribute 'intercept_' in the linear model of sklearn. Usually, no hyper-parameter needs to be tuned in linear regression. A linear model's performance mainly depends on how well the problem or data follows a linear distribution.

To improve the original linear regression models, ridge regression was proposed in [29]. Ridge regression imposes a penalty on the coefficients, and aims to minimize the objective function [30]:

$$\alpha \|w\|_2^2 + \sum_{i=1}^{p} \left(y_i - w_i \cdot x_i\right)^2, \tag{2.4}$$

where $\|w\|_2$ is the L_2-norm of the coefficient vector, and α is the regularization strength. A larger value of α indicates a larger amount of shrinkage; thus, the coefficients are also more robust to collinearity.

Lasso regression [31] is another linear model used to estimate sparse coefficients, consisting of a linear model with an L_1 priori added regularization term. It aims to minimize the objective function [30]:

$$\alpha||w||_1 + \sum_{i=1}^{p} (y_i - w_i \cdot x_i)^2 , \tag{2.5}$$

where α is the regularization strength and $||w||_1$ is the L_1-norm of the coefficient vector. Therefore, the regularization strength α is an crucial hyper-parameter of both ridge and lasso regression models.

Logistic regression (LR) [32] is a linear model used for classification problems. In LR, its cost function may be different, depending on the regularization method chosen for the penalization. There are three main types of regularization methods in LR: L_1-norm, L_2-norm, and elastic-net regularization [33].

Therefore, the first hyper-parameter that needs to be tuned in LR is to the regularization method used in the penalization, ’l1’, ’l2’, ’elasticnet’ or ’none’, which is called ’penalty’ in sklearn. The coefficient, ’C’, is another essential hyper-parameter that determines the regularization strength of the model. In addition, the ’solver’ type, representing the optimization algorithm type, can be set to ’newton-cg’, ’lbfgs’, ’liblinear’, ’sag’, or ’saga’ in LR. The ’solver’ type has correlations with ’penalty’ and ’C’, so they are conditional hyper-parameters.

KNN

K-nearest neighbor (KNN) is a simple ML algorithm that is used to classify data points by calculating the distances between different data points [34]. In KNN, the predicted class of each test sample is set to the class to which most of its k-nearest neighbors in the training set belong.

Assuming the training set $T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$, x_i is the feature vector of an instance, and $y_i \in \{c_1, c_2, \cdots, c_m\}$ is the class of the instance, $i = (1, 2, \cdots n)$, for a test instance

x, its class y can be denoted by [34]:

$$y = \arg\max_{c_j} \sum_{x_i \in N_k(x)} I\left(y_i = c_j\right), i = 1, 2, \cdots, n; j = 1, 2, \cdots, m, \tag{2.6}$$

where $I(x)$ is an indicator function, $I = 1$ when $y_i = c_j$, otherwise $I = 0$; $N_k(x)$ is the field involving the k-nearest neighbors of x.

In KNN, the number of considered nearest neighbors, k, is the most crucial hyper-parameter [35]. If k is too small, the model will be under-fitting; if k is too large, the model will be over-fitting and require high computational time. In addition, the weighted function used in the prediction can also be chosen from 'uniform' (points are weighted equally) or 'distance' (points are weighted by the inverse of their distance), depending on specific problems. The distance metric and the power parameter of the Minkowski metric can also be tuned as it can result in minor improvement. Lastly, the 'algorithm' used to compute the nearest neighbors can also be chosen from a ball tree, a k-dimensional (KD) tree, or a brute force search. Typically, the model can determine the most appropriate algorithm itself by setting the 'algorithm' to 'auto' in sklearn [24].

SVM

A support vector machines (SVM) [36] is a supervised learning algorithm that can be used for both classification and regression problems. SVM algorithms are based on the concept of mapping data points from low-dimensional into high-dimensional space to make them linearly separable; a hyperplane is then generated as the classification boundary to partition data points [37]. Assuming there are n data points, the objective function of SVM is [38]:

$$\arg\min_{\mathbf{w}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \max\left\{0, 1 - y_i f\left(x_i\right)\right\} + C\mathbf{w}^T\mathbf{w} \right\}, \tag{2.7}$$

where $\mathbf{w}$ is a normalization vector; C is the penalty parameter of the error term, which is an important hyper-parameter of all SVM models.

The kernel function $f(x)$, which is used to measure the similarity between two data points x_i and x_j, can be chosen from multiple types of kernels in SVM models. Therefore, the kernel type would be a vital hyper-parameter to be tuned. Common kernel types in SVM include linear kernels, radial basis function (RBF), polynomial kernels, and sigmoid kernels.

The different kernel functions can be denoted as follows [39]:

1. Linear kernel:
$$f(x) = x_i^T x_j; \tag{2.8}$$
2. Polynomial kernel:
$$f(x) = \left(\gamma x_i^T x_j + r\right)^d; \tag{2.9}$$
3. RBF kernel:
$$f(x) = \exp\left(-\gamma \|x - x'\|^2\right); \tag{2.10}$$
4. Sigmoid kernel:
$$f(x) = \left(\tanh\left(\gamma x_i^T x_j + r\right)\right); \tag{2.11}$$

As shown in the kernel function equations, a few other different hyper-parameters need to be tuned after a kernel type is chosen. The coefficient γ, denoted by 'gamma' in sklearn, is the conditional hyper-parameter of the 'kernel type' hyper-parameter when it is set to polynomial, RBF, or sigmoid; r, specified by 'coef0' in sklearn, is the conditional hyper-parameter of polynomial and sigmoid kernels. Moreover, the polynomial kernel has an additional conditional hyper-parameter d representing the 'degree' of the polynomial kernel function. In support vector regression (SVR) models, there is another hyper-parameter, 'epsilon', indicating the distance error to of its loss function [24].

Naïve Bayes

Naïve Bayes (NB) [40] algorithms are supervised learning algorithms based on Bayes' theorem. Assuming there are n dependent features $x_1, \cdots x_n$ and a target variable y, the objective function of naïve Bayes can be denoted by:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P\left(x_i | y\right), \tag{2.12}$$

where $P(y)$ is the probability of a value y, and $P(x_i|y)$ is the posterior probabilities of x_i given the values of y. Regarding the different assumptions of the distribution of $P(x_i|y)$, there are different types of naïve Bayes classifiers. The four main types of NB models are: Bernoulli NB, Gaussian NB, multinomial NB, and complement NB [41].

For Gaussian NB [42], the likelihood of features is assumed to follow a Gaussian distribution:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{\left(x_i - \mu_y\right)^2}{2\sigma_y^2}\right). \tag{2.13}$$

The maximum likelihood method is used to calculate the mean value, μ_y, and the variance, σ_y^2. Normally, there is not any hyper-parameter that needs to be tuned for Gaussian NB. The performance of a Gaussian NB model mainly depends on how well the dataset follows Gaussian distributions.

Multinomial NB [43] is designed for multinomially-distributed data based on the naïve Bayes algorithm. Assuming there are n features, and θ_{yi} is the distribution of each value of the target variable y, which equals the conditional probability $P(x_i|y)$ when a feature value i is involved in a data point belonging to the class y. Based on the concept of relative frequency counting, θ_y can be estimated by a smoothed version of θ_{yi} [24]:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}, \tag{2.14}$$

where N_{yi} is the number of times when feature i is in a data point belonging to class y, and N_y is the sum of all N_{yi} $(i = 0, 1, 2, \cdots, n)$. The smoothing priors $\alpha \geq 0$ are used for features that are not in the learning samples. When $\alpha = 1$, it is called Laplace smoothing; when $\alpha < 1$, it is called Lidstone smoothing.

Complement NB [44] is an improved version of the standard multinomial NB algorithm and is suitable for processing imbalanced data, while Bernoulli NB [45] requires samples to have binary-valued feature vectors so that the data can follow multivariate Bernoulli distributions. They both have the additive (Laplace/Lidstone) smoothing parameter, α, as the main hyper-parameter that needs tuning. To conclude, for naïve Bayes algorithms, developers often do not need to tune hyper-parameters or only need to tune the smoothing parameter α, which is a continuous hyper-parameter.

Tree-based Models

Decision tree (DT) [46] is a common classification method that uses a tree-structure to model decisions and possible consequences by summarizing a set of classification rules from the data. A DT has three main components: a root node representing the entire data; multiple decision nodes indicating decision tests and sub-node splits over each feature; and several leaf nodes representing the result classes [47]. DT algorithms recursively split the training set with better feature values to achieve good decisions on each subset. Pruning, which means removing some of the sub-nodes of decision nodes, is used in DT to avoid over-fitting. Since a deeper tree has more sub-trees to make more accurate decisions, the maximum tree depth, 'max_depth', is an essential hyper-parameter that controls the complexity of DT algorithms [48].

There are many other important HPs to be tuned to build effective DT models [49]. Firstly, the quality of splits can be measured by setting a measuring function, denoted by 'criterion' in sklearn. Gini impurity and information gain are the two main types of measuring functions. The split selection method, 'splitter', can also be set to 'best' to choose the best split, or 'random' to select a random split. The number of considered features to generate the best split, 'max_features', can also be tuned as a feature selection process. Moreover, there are several discrete hyper-parameters related to the splitting process: the minimum number of data points to split a decision node or to obtain a leaf node, denoted by 'min_samples_split' and 'min_samples_leaf', respectively; the 'max_leaf_nodes', indicating the maximum number of leaf nodes, and the 'min_weight_fraction_leaf' that means the minimum weighted fraction of the total weights, can also be tuned to improve model performance [24] [49].

Based on the concept of DT models, many decision-tree-based ensemble algorithms have been proposed to improve model performance by combining multiple decision trees, including random forest (RF), extra trees (ET), and extreme gradient boosting (XGBoost) models. RF [50] is an ensemble learning method that uses the bagging method to combine multiple decision trees. In RF, basic DTs are built on many randomly-generated subsets, and the class with the majority voting will be selected to be the final classification result [51]. ET [52] is another tree-based ensemble learning method that is similar to RF, but it uses all samples to build DTs and randomly selects the feature sets. In addition, RF optimizes splits on DTs while ET randomly makes the splits. XGBoost [25] is a popular tree-based ensemble model designed for speed and performance improvement, which uses the boosting and gradient descent methods to combine basic DTs. In XGBoost, the next input sample of a new DT will be related to the results of previous DTs. XGBoost aims to minimize the following objective function [48]:

$$Obj = -\frac{1}{2}\sum_{j=1}^{t}\frac{G_j^2}{H_j+\lambda} + \gamma t, \tag{2.15}$$

where t is the number of leaves in a decision tree, G and H are the sums of the first and second order gradient statistics of the cost function, γ and λ are the penalty coefficients.

Light Gradient Boosting Machine (LightGBM) [10] is a high-performance tree-based model that is constructed from an ensemble of DTs. In comparison to other ML techniques, the primary strength of LightGBM is its ability to deal with large-scale and high-dimensional data. It is achieved via the use of two strategies: gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB). GOSS is a downsampling technique that maintains only data samples with high gradients during model training to save time and memory. By combining mutually exclusive characteristics, the EFB approach significantly reduces training time without compromising crucial information. By including GOSS and EFB, LightGBM's time and space complexity has been significantly lowered from $O(NF)$ to $O(N'F')$, where N and N' denote the original and the reduced number of instances, respectively; and F and F' denote the original and bundled number of features, respectively [10].

Since tree-based ensemble models are built with decision trees as base learners, they have the same hyper-parameters as DT models, as described in this subsection. Apart from these hyper-parameters, RF, ET, XGBoost, and LightGBM all have another crucial hyper-parameter to be tuned, which is the number of decision trees to be combined, denoted by 'n_estimators' in sklearn. XGBoost has several additional hyper-parameters, including [53]: 'min_child_weight' which means the minimum sum of weights in a child node; 'subsample' and 'colsample_bytree' used to control the subsampling ratio of instances and features, respectively; and four continuous hyper-parameters — 'gamma', 'alpha', 'lambda', and 'learning_rate' — indicating the minimum loss reduction for a split, L_1, and L_2 regularization term on weights, and the learning rate, respectively.

Ensemble Learning Algorithms

Apart from tree-based ensemble models, there are several other generic ensemble learning methods that combine multiple singular ML models to achieve better model performance than any singular algorithms alone. Three common ensemble learning models — voting, bagging, and AdaBoost — are introduced in this subsection [54].

Voting [54] is a basic ensemble learning algorithm that uses the majority voting rule to combine singular estimators and generate a comprehensive estimator with improved accuracy. In sklearn, the voting method can be set to be 'hard' or 'soft', indicating whether to use majority voting or averaged predicted probabilities to determine the classification result. The list of selected single ML estimators and their weights can also be tuned in certain cases. For instance, a higher weight can be assigned to a better-performing singular ML model in a voting model.

Bootstrap aggregating [54], also named bagging, trains multiple base estimators on different randomly-extracted subsets to construct a final predictor [55]. When using bagging methods, the first consideration should be the type and number of base estimators in the ensemble, denoted by 'base_estimator' and 'n_estimators', respectively. Then, the 'max_samples' and 'max_features', indicating the sample size and feature size to generate different subsets, can also be tuned.

AdaBoost [54], short for adaptive boosting, is an ensemble learning method that trains multiple base learners consecutively (weak learners), and later learners emphasize the misclassified samples of previous learners; ultimately, a final strong learner is obtained. During this process, incorrectly-classified instances are retrained with other new instances, and their weights are adjusted so that the subsequent classifiers focus more on difficult cases, thereby gradually building a stronger classifier. In AdaBoost, the type of base estimator, 'base_estimator', can be set to a decision tree or other methods. In addition, the maximum number of estimators at which boosting is terminated, 'n_estimators', and the learning rate that shrinks the contribution of each classifier, should also be tuned to achieve a trade-off between these two hyper-parameters.

Deep Learning Models

Deep learning (DL) algorithms are widely applied to various areas — like computer vision, natural language processing, and machine translation — since they have had great success solving many types of problems. DL models are based on the theory of artificial neural networks (ANNs). Common types of DL architectures include deep neural networks (DNNs), Multi-Layer Perceptron (MLP), deep belief networks (DBNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), AutoEncoders (AEs) and many more [56]. All these DL models have similar hyper-parameters since they have similar underlying neural network architecture. Compared with other ML models, DL models benefit more from HPO since they often have many hyper-parameters that require tuning.

The first set of hyper-parameters is related to the construction of a DL model; hence, named model design hyper-parameters. Since all neural network models have an input layer and an output layer, the complexity of a deep learning model mainly depends on the number of hidden layers and the number of neurons in each layer, which are two main hyper-parameters to build DL models [57]. These two hyper-parameters are set and tuned according to the complexity of the datasets or the problems. DL models need to have enough capacity to model objective functions (or prediction tasks) while avoiding over-fitting. At the next stage, certain function types need to be set or tuned. The first function type to configure is the loss function type, which is chosen mainly based on the problem type (*e.g.*, binary cross-entropy for binary classification problems, multi-class cross-entropy for multi-classification problems, and RMSE for regression problems). Another important hyper-parameter is the activation function type used to model non-linear functions, which be set to ’softmax’, ’rectified linear unit (ReLU)’, ’sigmoid’, ’tanh’, or ’softsign’. Lastly, the optimizer type can be set to Stochastic Gradient Descent (SGD), adaptive moment estimation (Adam), root mean square propagation (RMSprop), etc. [58].

On the other hand, some other hyper-parameters are related to the optimization and training process of DL models; hence, categorized as optimizer hyper-parameters. The learning rate is one of the most important hyper-parameters in DL models [59]. It determines the step size at each iteration, which enables the objective function to converge. A large learning rate speeds up the learning process, but the gradient may oscillate around a local minimum value or even cannot converge. On the other hand, a small learning rate converges smoothly, but will largely increase model training time by requiring more training epochs. An appropriate learning rate should enable the objective function to converge to a global minimum in a reasonable amount of time. Another common hyper-parameter is the drop-out rate. Drop-out is a standard regularization method for DL models proposed to reduce over-fitting. In drop-out, a proportion of neurons are randomly removed, and the percentage of neurons to be removed should be tuned.

Mini-batch size and the number of epochs are the other two DL hyper-parameters that represent the number of processed samples before updating the model, and the number of complete passes through the entire training set, respectively [60]. Mini-batch size is affected by the resource requirements of the training process and the number of iterations. The number of epochs depends on the size of the training set and should be tuned by slowly increasing its value until validation accuracy starts to decrease, which indicates over-fitting. On the other hand, DL models often converge within a few epochs, and the following epochs may lead to unnecessary additional execution time and over-fitting, which can be avoided by the early stopping method. Early stopping is a form of regularization whereby model training stops in advance when validation accuracy does not increase after a certain number of consecutive epochs. The number of waiting epochs, called early stop patience, can also be tuned to reduce model training time.

Apart from traditional DL models, transfer learning (TL) is a technology that obtains a pre-trained model on the data in a related domain and transfers it to other target tasks [61]. To transfer a DL model from one problem to another problem, a certain number of top layers are frozen, and only the remaining layers are retrained to fit the new problem. Therefore, the number of frozen layers is a vital hyper-parameter to tune if TL is used.

2.4.2 Unsupervised Learning Algorithms

Unsupervised learning algorithms are a set of ML algorithms used to identify unknown patterns in unlabeled datasets. Clustering and dimensionality-reduction algorithms are the two main types of unsupervised learning methods. Clustering methods include k-means, DBSCAN, EM, etc.; while PCA is a representative dimensionality reduction algorithm [23].

Clustering Algorithms

In most clustering algorithms — including k-means and EM — the number of clusters is the most important hyper-parameter to tune [62].

The k-means algorithm [63] uses k prototypes, indicating the centroids of clusters, to cluster data. In k-means algorithms, the number of clusters, 'n_clusters', must be specified, and is determined by minimizing the sum of squared errors [63]:

$$\sum_{i=0}^{n_k} \min_{u_j \in C_k} \left(\mathbf{x}_i - u_j\right)^2, \tag{2.16}$$

where $(\mathbf{x}_1, \cdots, \mathbf{x}_n)$ is the data matrix; u_j, also called the centroid of the cluster C_k, is the mean of the samples in the cluster; and n_k is the number of sample points in the cluster C_k.

To tune k-means, 'n_clusters' is the most crucial hyper-parameter. Besides this, the method for centroid initialization, 'init', could be set to 'k-means++', 'random' or a human-defined array, which slightly affects model performance. In addition, 'n_init', denoting the number of times that the k-means algorithm will be executed with different centroid seeds, and the 'max_iter', the maximum number of iterations in a single execution of k-means, also have slight impacts on model performance [24].

The expectation-maximization (EM) algorithm [64] is an iterative algorithm used to detect the maximum likelihood estimation of parameters. Gaussian Mixture model is a clustering method that uses a mixture of Gaussian distributions to model data by implementing the EM method. Similar to k-means, its major hyper-parameter to be tuned is 'n_components', indicating the number of clusters or Gaussian distributions. Additionally, different methods can be chosen to constrain the covariance of the estimated classes in Gaussian mixture models, including 'full covariance', 'tied', 'diagonal' or 'spherical' [65]. Other hyper-parameters could also be tuned, including 'max_iter' and 'tol', representing the number of EM iterations to perform and the convergence threshold, respectively [24].

DBSCAN [66] is a density-based clustering method that determines the clusters by dividing data into clusters with sufficiently high density. Unlike other clustering models, the number of clusters does not need to be configured before training. Instead, DBSCAN has two significant conditional hyper-parameters — the scan radius represented by 'eps', and the minimum number of considered neighbor points represented by 'min_samples' — which define the cluster density together [67]. DBSCAN works by starting with an unvisited point and detecting all its neighbor points within a pre-defined distance 'eps'. If the number of neighbor points reaches the value of 'min_samples', this unvisited point and all its neighbors are defined as a cluster. The procedures are executed recursively until all data points have been visited. A higher 'min_samples' or a lower 'eps' indicates a higher density to form a cluster.

Dimensionality Reduction Algorithms

The increasing amount of collected data provides ample information, but also increases problem complexity. In real-world applications, many features are irrelevant or redundant to predict target variables. Dimensionality reduction algorithms often serve as feature engineering methods to extract important features and eliminate insignificant or redundant features. Principal component analysis (PCA) is the most representative dimensionality reduction algorithm. In PCA, the number of features to be extracted, represented by 'n_components' in sklearn, is the main hyper-parameter to be tuned.

Principal component analysis (PCA) [68] is a widely used linear dimensionality reduction method. PCA is based on the concept of mapping the original n-dimensional features into k-dimension features as the new orthogonal features, also called the principal components. PCA works by calculating the covariance matrix of the data matrix to obtain the eigenvectors of the covariance matrix. The matrix comprises the eigenvectors of k features with the largest eigenvalues (*i.e.*, the largest variance). Consequently, the data matrix can be transformed into a new space with reduced dimensionality. Singular value decomposition (SVD) [69] is a popular method used to obtain the eigenvalues and eigenvectors of the covariance matrix of PCA. Therefore, in addition to 'n_components', the SVD solver type is another hyper-parameter of PCA to be tuned, which can be assigned to 'auto', 'full', 'arpack' or 'randomized' [24].

2.5 Optimization Methods

The key process of machine learning is to solve optimization problems. To build a ML model, its weight parameters are initialized and optimized by an optimization method until the objective function approaches a minimum value or the accuracy approaches a maximum value [70]. Similarly, hyper-parameter optimization methods aim to optimize the architecture of a ML model by detecting the optimal hyper-parameter configurations. In this section, the concepts of optimization and the common HPO techniques are discussed.

2.5.1 Hyper-parameter Optimization Problem Statement

During the design process of ML models, effectively searching the hyper-parameters' space using optimization techniques can identify the optimal hyper-parameters for the models. The hyper-parameter optimization process consists of four main components: an estimator (a regressor or a classifier) with its objective function, a search space (configuration space), a search or optimization method used to find hyper-parameter combinations, and an evaluation function to compare the performance of different hyper-parameter configurations.

The domain of a hyper-parameter can be continuous (*e.g.*, learning rate), discrete (*e.g.*, number of clusters), binary (*e.g.*, whether to use early stopping or not), or categorical (*e.g.*, type of optimizer). Therefore, hyper-parameters are classified as continuous, discrete, and categorical hyper-parameters. For continuous and discrete hyper-parameters, their domains are usually bounded in practical applications [71] [72]. On the other hand, the hyper-parameter configuration space sometimes contains conditionality. A hyper-parameter may need to be used or tuned depending on the value of another hyper-parameter, called a conditional hyper-parameter [73]. For instance, in SVM, the degree of the polynomial kernel function only needs to be tuned when the kernel type is chosen to be polynomial.

In simple cases, all hyper-parameters can take unrestricted real values, and the feasible set X of hyper-parameters can be a real-valued n-dimensional vector space. However, in most cases, the hyper-parameters of a ML model often take on values from different domains and have different constraints, so their optimization problems are often complex constrained optimization problems [74]. For instance, the number of considered features in a decision tree should be in the range of 0 to the number of features, and the number of clusters in k-means should not be larger than the size of data points. Additionally, categorical features can often only take several certain values, like the limited choices of the activation function and the optimizer of a neural network. Therefore, the feasible domain of X often has a complex structure, which increases the problems' complexity [74].

In general, for a hyper-parameter optimization problem, the aim is to obtain [75]:

$$x^* = \arg\min_{x \in X} f(x), \tag{2.17}$$

where $f(x)$ is the objective function to be minimized, such as the error rate or the root mean squared error (RMSE); x^* is the hyper-parameter configuration that produces the optimum value of $f(x)$; and a hyper-parameter x can take any value in the search space X.

The aim of HPO is to achieve optimal or near-optimal model performance by tuning hyper-parameters within the given budgets [76]. The mathematical expression of the function f varies, depending on the objective function of the chosen ML algorithm and the performance metric function. Model performance can be evaluated by various metrics, like accuracy, RMSE, F1-score, and false alarm rate. On the other hand, in practice, time budgets are an essential constraint for optimizing HPO models and must be considered. It often requires a massive amount of time to optimize the objective function of a ML model with a reasonable number of hyper-parameter configurations. Every time a hyper-parameter value is tested, the entire ML model needs to be retrained, and the validation set needs to be processed to generate a score that reflects the model performance.

The main process of HPO is as follows [73]:

1. Select the objective function and the performance metrics;
2. Select the hyper-parameters that require tuning, summarize their types, and determine the appropriate optimization technique;
3. Train the ML model using the default hyper-parameter configuration or common values as the baseline model;
4. Start the optimization process with a large search space as the hyper-parameter feasible domain determined by manual testing and/or domain knowledge;

5. Narrow the search space based on the regions of currently-tested well-performing hyper-parameter values, or explore new search spaces if necessary.
6. Return the best-performing hyper-parameter configuration as the final solution.

2.5.2 Model-free Algorithms

Babysitting

Babysitting, also called 'Trial and Error' or grad student descent (GSD), is a basic hyper-parameter tuning method [17]. This method is implemented by 100% manual tuning and widely used by students and researchers. The workflow is simple: after building a ML model, a student tests many possible hyper-parameter values based on experience, guessing, or the analysis of previously-evaluated results; the process is repeated until this student runs out of time (often reaching a deadline) or is satisfied with the results. As such, this approach requires a sufficient amount of prior knowledge and experience to identify optimal hyper-parameter values with limited time.

Manual tuning is infeasible for many problems due to several factors, like a large number of hyper-parameters, complex models, time-consuming model evaluations, and non-linear hyper-parameter interactions [77]. These factors inspired increased research into techniques for the automatic optimization of hyper-parameters [78].

Grid Search

Grid search (GS) is one of the most commonly-used methods to explore hyper-parameter configuration space [79]. GS can be considered an exhaustive search or a brute-force method that evaluates all the hyper-parameter combinations given to the grid of configurations [80]. GS works by evaluating the Cartesian product of a user-specified finite set of values [81].

GS cannot exploit the well-performing regions further by itself. Therefore, to identify the global optimums, the following procedure needs to be performed manually [82]:

1. Start with a large search space and step size.
2. Narrow the search space and step size based on the previous results of well-performing hyper-parameter configurations.

3. Repeat step 2 multiple times until an optimum is reached.

GS can be easily implemented and parallelized. However, the main drawback of GS is its inefficiency for high-dimensionality hyper-parameter configuration space, since the number of evaluations increases exponentially as the number of hyper-parameters grows. This exponential growth is referred to as the curse of dimensionality [84]. For GS, assuming that there are k parameters, and each of them has n distinct values, its computational complexity increases exponentially at a rate of $O(n^k)$ [75]. Thus, only when the hyper-parameter configuration space is small can GS be an effective HPO method.

Random Search

To overcome certain limitations of GS, random search (RS) was proposed in [83]. RS is similar to GS; but, instead of testing all values in the search space, RS randomly selects a pre-defined number of samples between the upper and lower bounds as candidate hyper-parameter values, and then trains these candidates until the defined budget is exhausted. The theoretical basis of RS is that if the configuration space is large enough, then the global optimums, or at least their approximations, can be detected. With a limited budget, RS is able to explore a larger search space than GS [83].

The main advantage of RS is that it is easily parallelized and resource-allocated since each evaluation is independent. Unlike GS, RS samples a fixed number of parameter combinations from the specified distribution, which improves system efficiency by reducing the probability of wasting much time on a small poor-performing region. Since the number of total evaluations in RS is set to a fixed value n before the optimization process starts, the computational complexity of RS is $O(n)$ [85]. In addition, RS can detect the global optimum or the near-global optimum when given enough budgets [81].

Although RS is more efficient than GS for large search spaces, there are still a large number of unnecessary function evaluations since it does not exploit the previously well-performing regions [82].

To conclude, the main limitation of both RS and GS is that every evaluation in their iterations is independent of previous evaluations; thus, they waste massive time evaluating poorly-performing areas of the search space. This issue can be solved by other optimization methods, like Bayesian optimization that uses previous evaluation records to determine the next evaluation [86].

2.5.3 Gradient-based Optimization

Gradient descent [87] is a traditional optimization technique that calculates the gradient of variables to identify the promising direction and moves towards the optimum. After randomly selecting a data point, the technique moves towards the opposite direction of the largest gradient to locate the next data point. Therefore, a local optimum can be reached after convergence. The local optimum is also the global optimum for convex functions. Gradient-based algorithms have a time complexity of $O(n^k)$ for optimizing k hyper-parameters [88].

For specific machine learning algorithms, the gradient of certain hyper-parameters can be calculated, and then the gradient descent can be used to optimize these hyper-parameters. Although gradient-based algorithms have a faster convergence speed to reach local optimum than certain other methods like GS, they have several limitations. Firstly, they can only be used to optimize continuous hyper-parameters because other types of hyper-parameters, like categorical hyper-parameters, do not have gradient directions. Secondly, they are only efficient for convex functions because the local instead of a global optimum may be reached for non-convex functions [82]. Therefore, the gradient-based algorithms can only be used in some cases where it is possible to obtain the gradient of hyper-parameters; *e.g.*, optimizing the learning rate in neural networks (NN) [89]. Still, it is not guaranteed for ML algorithms to identify global optimums using gradient-based optimization techniques.

2.5.4 Bayesian Optimization

Bayesian optimization (BO) [90] is an iterative algorithm that is popularly used for HPO problems. Unlike GS and RS, BO determines the future evaluation points based on the previously-obtained results. To determine the next hyper-parameter configuration, BO uses two key components: a surrogate model and an acquisition function [50]. The surrogate model aims to fit all the currently-observed points into the objective function. After obtaining the predictive distribution of the probabilistic surrogate model, the acquisition function determines the usage of different points by balancing the trade-off between exploration and exploitation. Exploration is to sample the instances in the areas that have not been sampled, while exploitation is to sample in the currently promising regions where the global optimum is most likely to occur, based on the posterior distribution. BO models balance the exploration and the exploitation processes to detect the current most likely optimal regions and avoid missing better configurations in the unexplored areas [91].

The basic procedures of BO are as follows [90]:

1. Build a probabilistic surrogate model of the objective function.
2. Detect the optimal hyper-parameter values on the surrogate model.
3. Apply these hyper-parameter values to the real objective function to evaluate them.
4. Update the surrogate model with new results.
5. Repeat steps 2 - 4 until the maximum number of iterations is reached.

Thus, BO works by updating the surrogate model after each evaluation on the objective function. BO is more efficient than GS and RS since it can detect the optimal hyper-parameter combinations by analyzing the previously-tested values, and running a surrogate model is often much cheaper than running the entire objective function.

However, since Bayesian optimization models are executed based on the previously-tested values, they belong to sequential methods that are difficult to parallelize; but they can usually detect near-optimal hyper-parameter combinations within a few iterations [92].

Common surrogate models for BO include Gaussian process (GP) [93], and the tree Parzen estimator (TPE) [71]. Therefore, there are three main types of BO algorithms based on their surrogate models: BO-GP, BO-TPE.

BO-GP

Gaussian process (GP) is a standard surrogate model for objective function modeling in BO [90]. Assuming that the function f with a mean μ and a covariance σ^2 is a realization of a GP, the predictions follow a normal distribution [94]:

$$p(y|x, D) = N\left(y|\hat{\mu}, \hat{\sigma}^2\right), \tag{2.18}$$

where D is the configuration space of hyper-parameters, and $y = f(x)$ is the evaluation result of each hyper-parameter value x. After obtaining a set of predictions, the points to be evaluated next are then selected from the confidence intervals generated by the BO-GP model. Each newly-tested data point is added to the sample records, and the BO-GP model is re-built with the new information. This procedure is repeated until termination.

Applying a BO-GP to a size n dataset has a time complexity of $O(n^3)$ and space complexity of $O(n^2)$ [95]. One main limitation of BO-GP is that the cubic complexity to the number of instances limits the capacity for parallelization [76]. Additionally, it is mainly used to optimize continuous variables.

BO-TPE

Tree-structured Parzen estimator (TPE) [71] is another common surrogate model for BO. Instead of defining a predictive distribution used in BO-GP, BO-TPE creates two density functions, $l(x)$ and $g(x)$, to act as the generative models for all domain variables [76]. To apply TPE, the observation results are divided into good results and poor results by a pre-defined percentile y^*, and the two sets of results are modeled by simple Parzen windows [71]:

$$p(x|y, D) = \begin{cases} l(x), & if \quad y < y^* \\ g(x), & if \quad y > y^* \end{cases} . \tag{2.19}$$

After that, the expected improvement in the acquisition function is reflected by the ratio between the two density functions, which is used to determine the new configurations for evaluation. The Parzen estimators are organized in a tree structure, so the specified conditional dependencies are retained. Therefore, TPE naturally supports specified conditional hyper-parameters [94]. The time complexity of BO-TPE is $O(nlogn)$, which is lower than the complexity of BO-GP [76].

BO methods are effective for many HPO problems, even if the objective function f is stochastic, non-convex, or non-continuous. However, the main drawback of BO models is that, if they fail to achieve the balance between exploration and exploitation, they might only reach a local instead of a global optimum. RS does not have this limitation since it does not focus on any specific area. Additionally, it is difficult to parallelize BO models since their intermediate results are dependent on each other [92].

2.5.5 Multi-fidelity Optimization Algorithms

One major issue with HPO is the long execution time, which increases with a larger hyper-parameter configuration space and larger datasets. The execution time may be several hours, several days, or even more [96]. Multi-fidelity optimization techniques are common approaches to solve the constraint of limited time and resources. To save time, people can use a subset of the original dataset or a subset of the features [97]. Multi-fidelity involves low-fidelity and high-fidelity evaluations and combines them for practical applications [98]. In low-fidelity evaluations, a relatively small subset is evaluated at a low cost but with poor generalization performance. In high-fidelity evaluations, a relatively large subset is evaluated with better generalization performance but at a higher cost than low-fidelity evaluations. In multi-fidelity optimization algorithms, poorly-performing configurations are discarded after each round of hyper-parameter evaluation on generated subsets, and only well-performing hyper-parameter configurations will be evaluated on the entire training set.

Bandit-based algorithms categorized to multi-fidelity optimization algorithms have shown success dealing with deep learning optimization problems [76]. Two common bandit-based techniques are successive halving [99] and Hyperband [100].

Successive Halving

Theoretically speaking, exhaustive methods are able to identify the optimal hyper-parameter combination by evaluating all the given combinations. However, many factors, including limited time and resources, should be considered in practical applications. These factors are called budgets (B). To overcome the limitations of GS and RS and to improve efficiency, successive halving algorithms were proposed in [99].

The main process of using successive halving algorithms for HPO is as follows. Firstly, it is presumed that there are n sets of hyper-parameter combinations, and that they are evaluated with uniformly-allocated budgets ($b = B/n$). Then, according to the evaluation results for each iteration, half of the poorly-performing hyper-parameter configurations are eliminated, and the better-performing half is passed to the next iteration with double budgets ($b_{i+1} = 2 * b_i$). The above process is repeated until the final optimal hyper-parameter combination is detected.

Successive halving is more efficient than RS, but is affected by the trade-off between the number of hyper-parameter configurations and the budgets allocated to each configuration [81]. Thus, the main concern of successive halving is how to allocate the budget and how to determine whether to test fewer configurations with a higher budget for each or to test more configurations with a lower budget for each [82].

Hyperband

Hyperband [100] is then proposed to solve the dilemma of successive halving algorithms by dynamically choosing a reasonable number of configurations. It aims to achieve a trade-off between the number of hyper-parameter configurations (n) and their allocated budgets by dividing the total budgets (B) into n pieces and allocating these pieces to each configuration ($b = B/n$). Successive halving serves as a subroutine on each set of random configurations to eliminate the poorly-performing hyper-parameter configurations and improve efficiency. The main steps of Hyperband algorithms are shown in Algorithm 1 [82].

Algorithm 1: Hyperband

```
Input: $b_{\max}, b_{\min}$
1: $s_{\max} = \log\left(\frac{b_{\max}}{b_{\min}}\right)$
2: for $s \in \{b_{\max}, b_{\min} - 1, \ldots, 0\}$ do
3:     $n = DetermineBudget(s)$
4:     $\gamma = SampleConfigurations(n)$
5:     $SuccessiveHalving(\gamma)$
6: end for
7: return The best configuration so far.
```

Firstly, the budget constraints b_{min} and b_{max} are determined by the total number of data points, the minimum number of instances required to train a sensible model, and the available budgets. After that, the number of configurations n and the budget size allocated to each configuration are calculated based on b_{min} and b_{max} in steps 2-3 of Algorithm 1. The configurations are sampled based on n and b, and then passed to the successive halving model demonstrated in steps 4-5. The successive halving algorithm discards the identified poorly-performing configurations and passes the well-performing configurations on to the next iteration. This process is repeated until the final optimal hyper-parameter configuration is identified. By involving the successive halving searching method, Hyperband has a computational complexity of $O(nlogn)$ [100].

2.5.6 Metaheuristic Algorithms

Metaheuristic algorithms [101] are a set of algorithms mainly inspired by biological theories and widely used for optimization problems. Unlike many traditional optimization methods, metaheuristics have the capacity to solve non-convex, non-continuous, and non-smooth optimization problems.

Population-based optimization algorithms (POAs) are a major type of metaheuristic algorithm, including genetic algorithms (GAs), evolutionary algorithms, evolutionary strategies, and particle swarm optimization (PSO). POAs start by creating and updating a population as each generation; each individual in every generation is then evaluated until the global optimum is identified [86]. The main differences between different POAs are the methods used to generate and select populations [3]. POAs can be easily parallelized since a population of N individuals can be evaluated on at most N threads or machines in parallel [81]. Genetic algorithms and particle swarm optimization are the two main POAs that are popularly-used for HPO problems.

Genetic Algorithm

Genetic algorithm (GA) [102] is one of the common metaheuristic algorithms based on the evolutionary theory that individuals with the best survival capability and adaptability to the environment are more likely to survive and pass on their capabilities to future generations. The next generation will also inherit their parents' characteristics and may involve better and worse individuals. Better individuals will be more likely to survive and have more capable offspring, while the worse individuals will gradually disappear. After several generations, the individual with the best adaptability will be identified as the global optimum [103].

To apply GA to HPO problems, each chromosome or individual represents a hyper-parameter, and its decimal value is the actual input value of the hyper-parameter in each evaluation. Every chromosome has several genes, which are binary digits; and then crossover and mutation operations are performed on the genes of this chromosome. The population involves all possible values within the initialized chromosome/parameter ranges, while the fitness function characterizes the evaluation metrics of the parameters [103].

Since the randomly-initialized parameter values often do not include the optimal parameter values, several operations, including selection, crossover, and mutation operations, must be performed on the well-performing chromosomes to identify the optimums [102]. Chromosome selection is implemented by selecting those chromosomes with good fitness function values. To keep the population size unchanged, the chromosomes with good fitness function values are passed to the next generation with higher probability, where they generate new chromosomes with the parents' best characteristics. Chromosome selection ensures that good characteristics of each generation can be passed to later generations. Crossover is used to generate new chromosomes by exchanging a proportion of genes in different chromosomes. Mutation operations are also used to generate new chromosomes by randomly altering one or more genes of a chromosome. Crossover and mutation operations enable later generations to have different characteristics and reduce the chance of missing good characteristics [76].

The main procedures of GA are as follows [101]:

1. Randomly initialize the population, chromosomes, and genes, which represent the entire search space, hyper-parameters, and hyper-parameter values, respectively.
2. Evaluate the performance of each individual in the current generation by calculating the fitness function, which indicates the objective function of a ML model.
3. Perform selection, crossover, and mutation operations on the chromosomes to produce a new generation involving the next hyper-parameter configurations to be evaluated.
4. Repeat steps 2 & 3 until the termination condition is met.
5. Terminate and output the optimal hyper-parameter configuration.

Among the above steps, the population initialization step is an important step of GA and PSO since it provides an initial guess of the optimal values. Although the initialized values will be iteratively improved in the optimization process, a suitable population initialization method can significantly improve the convergence speed and performance of POAs. A good initial population of hyper-parameters should involve individuals that are close to global optimums by covering the promising regions and should not be localized to an unpromising region of the search space [104].

To generate hyper-parameter configuration candidates for the initial population, random initialization that simply creates the initial population with random values in the given search space is often used in GA [105]. Thus, GA is easily implemented and does not necessitate good initializations, because its selection, crossover, and mutation operations lower the possibility of missing the global optimum.

Hence, it is useful when the data analyst does not have much experience determining a potential appropriate initial search space for the hyper-parameters. The main limitation of GA is that the algorithm itself introduces additional hyper-parameters to be configured, including the fitness function type, population size, crossover rate, and mutation rate. Moreover, GA is a sequential execution algorithm, making it difficult to parallelize. The time complexity of GA is $O(n^2)$ [106]. As a result, sometimes, GA may be inefficient due to its low convergence speed.

Particle Swarm Optimization

Particle swarm optimization (PSO) [107] is another set of evolutionary algorithms that are commonly used for optimization problems. PSO algorithms are inspired by biological populations that exhibit both individual and social behaviors [3]. PSO works by enabling a group of particles (swarm) to traverse the search space in a semi-random manner [77]. PSO algorithms identify the optimal solution through cooperation and information sharing among individual particles in a group.

In PSO, there are a group of n particles in a swarm $\mathbf{S}$ [82]:

$$\mathbf{S} = (S_1, S_2, \cdots, S_n), \tag{2.20}$$

and each particle S_i is represented by a vector:

$$S_i = < \overrightarrow{x_i}, \overrightarrow{v_i}, \overrightarrow{p_i} >, \tag{2.21}$$

where $\overrightarrow{x_i}$ is the current position, $\overrightarrow{v_i}$ is the current velocity, and $\overrightarrow{p_i}$ is the known best position of the particle so far.

After initializing the position and velocity of each particle,very particle evaluates the current position and records the position with its performance score. In the next iteration, the velocity $\overrightarrow{v_i}$ of each particle is changed based on the previous position $\overrightarrow{p_i}$ and the current global optimal position $\overrightarrow{p}$:

$$\overrightarrow{v_i} := \overrightarrow{v_i} + U(0, \varphi_1)(\overrightarrow{p_i} - \overrightarrow{x_i}) + U(0, \varphi_2)(\overrightarrow{p} - \overrightarrow{x_i}), \tag{2.22}$$

where $U(0, \varphi)$ is the continuous uniform distributions based on the acceleration constants φ_1 and φ_2.

After that, the particles move based on their new velocity vectors:

$$\overrightarrow{x_i} := \overrightarrow{x_i} + \overrightarrow{v_i}. \tag{2.23}$$

The above procedures are repeated until convergence or termination constraints are reached.

Compared with GA, it is easier to implement PSO, since PSO does not have certain additional operations like crossover and mutation. In GA, all chromosomes share information with each other, so the entire population moves uniformly toward the optimal region; while in PSO, only information on the individual best particle and the global best particle is transmitted to others, which is a one-way flow of information sharing, and the entire search process follows the direction of the current optimal solution [82]. The computational complexity of PSO algorithm is $O(nlogn)$ [108]. In most cases, the convergence speed of PSO is faster than of GA. In addition, particles in PSO operate independently and only need to share information with each other after each iteration, so this process is easily parallelized to improve model efficiency [77].

The main limitation of PSO is that it requires proper population initialization; otherwise, it might only reach a local instead of a global optimum, especially for discrete hyper-parameters

[109]. Proper population initialization requires developers' prior experience or using population initialization techniques. Many population initialization techniques have been proposed to improve the performance of evolutionary algorithms, like the opposition-based optimization algorithm [105] and the space transformation search method [110]. Involving additional population initialization techniques will require more execution time and resources.

Table 2.1: The comparison of common HPO algorithms (n is the number of hyper-parameter values and k is the number of hyper-parameters)

HPO Method	**Strengths**	**Limitations**	**Time Complexity**
GS	· Simple.	· Time-consuming, · Only efficient with categorical HPs.	$O(n^k)$
RS	· More efficient than GS. · Enable parallelization.	· Not consider previous results. · Not efficient with conditional HPs.	$O(n)$
Gradient-based models	· Fast convergence speed for continuous HPs.	· Only support continuous HPs. · May only detect local optimums.	$O(n^k)$
BO-GP	· Fast convergence speed for continuous HPs.	· Poor capacity for parallelization. · Not efficient with conditional HPs.	$O(n^3)$
BO-TPE	· Efficient with all types of HPs. · Keep conditional dependencies.	· Poor capacity for parallelization.	$O(nlogn)$
Hyperband	· Enable parallelization.	· Not efficient with conditional HPs. · Require subsets with small budgets to be representative.	$O(nlogn)$
GA	· Efficient with all types of HPs. · Not require good initialization.	· Poor capacity for parallelization.	$O(n^2)$
PSO	· Efficient with all types of HPs. · Enable parallelization.	· Require proper initialization.	$O(nlogn)$

2.5.7 Apply HPO Algorithms to ML Models

The strengths and limitations of the hyper-parameter optimization algorithms involved in this chapter are summarized in Table 2.1. Since there are many different HPO methods for different use cases, it is crucial to select the appropriate optimization techniques for different ML models. ML algorithms can be classified by the characteristics of their hyper-parameter configurations. Appropriate optimization algorithms can be chosen to optimize the hyper-parameters based on these characteristics.

One Discrete Hyper-parameter

Commonly for some ML algorithms, like certain neighbor-based, clustering, and dimensionality reduction algorithms, only one discrete hyper-parameter needs to be tuned. For KNN, the major hyper-parameter is *k*, the number of considered neighbors. The most essential hyper-parameter of k-means and EM is the number of clusters. Similarly, for dimensionality reduction algorithms, such as PCA, their basic hyper-parameter is 'n_components', the number of features to be extracted.

In these situations, Bayesian optimization is the best choice, and the three surrogates could be tested to find the best one. Hyperband is another good choice, which may have a fast execution speed due to its capacity for parallelization. In some cases, people may want to fine-tune the ML model by considering other less important hyper-parameters, like the distance metric of KNN and the SVD solver type of PCA; so BO-TPE, GA, or PSO could be chosen for these situations.

One Continuous Hyper-parameter

Some linear models, including ridge and lasso algorithms, and some naïve Bayes algorithms, involving multinomial NB, Bernoulli NB, and complement NB, generally only have one vital continuous hyper-parameter to be tuned. In ridge and lasso algorithms, the continuous hyper-parameter is 'alpha', the regularization strength. In the three NB algorithms mentioned above, the critical hyper-parameter is also named 'alpha', but it represents the additive (Laplace/Lidstone) smoothing parameter. In terms of these ML algorithms, BO-GP is the best choice, since it is good at optimizing a small number of continuous hyper-parameters. Gradient-based algorithms can also be used, but might only detect local optimums, so they are less effective than BO-GP.

A Few Conditional Hyper-parameters

It is noticeable that many ML algorithms have conditional hyper-parameters, like SVM, LR, and DBSCAN. LR has three correlated hyper-parameters, 'penalty', 'C', and the solver type. Similarly, DBSCAN has 'eps' and 'min_samples' that must be tuned in conjunction. SVM is more complex, since after setting a different kernel type, there is a separate set of conditional hyper-parameters that need to be tuned next. Hence, some HPO methods that cannot effectively optimize conditional hyper-parameters, including GS, RS, BO-GP, and Hyperband, are not suitable for ML models with conditional hyper-parameters. For these ML methods, BO-TPE is the best choice if we have pre-defined relationships among the hyper-parameters. GA and PSO can be used, as well.

A Large Hyper-parameter Configuration Space with Multiple Types of Hyper-parameters

Tree-based algorithms, including DT, RF, ET, and XGBoost, as well as DL algorithms, like DNN, CNN, RNN, are the most complex types of ML algorithms to bed tuned, since they have many hyper-parameters with various, different types. For these ML models, PSO is the best choice since it enables parallel executions to improve efficiency, particularly for DL models that often require massive training time. Some other techniques, like GA and BO-TPE can also be used, but they may cost more time than PSO, since it is difficult to parallelize these techniques.

Categorical Hyper-parameters

This category of hyper-parameters is mainly for ensemble learning algorithms, since their major hyper-parameter is a categorical hyper-parameter. For bagging and AdaBoost, the categorical hyper-parameter is 'base_estimator', which is set to be a singular ML model. For voting, it is 'estimators', indicating a list of ML singular models to be combined. The voting method has another categorical hyper-parameter, 'voting', which is used to choose whether to use a hard or soft voting method. If we only consider these categorical hyper-parameters, GS would be sufficient to detect their suitable base machine learners. On the other hand, in many cases, other hyper-parameters need to be considered, like 'n_estimators', 'max_samples', and 'max_features' in bagging, as well as 'n_estimators' and 'learning_rate' in AdaBoost; consequently, BO algorithms would be a better choice to optimize these continuous or discrete hyper-parameters.

In conclusion, when tuning a ML model to achieve high model performance and low computational costs, the most suitable HPO algorithm should be selected based on the properties of its hyper-parameters. A comprehensive summary of applying hyper-parameter optimization techniques to ML models is shown in Table 2.2.

2.6 Data Pre-Processing

2.6.1 Overview

Data pre-processing aims to improve the quality of data for ML model development. Common data quality issues include outliers, missing values, and class imbalance [18]. Data pre-processing procedures guarantee that ML models can learn meaningful patterns from the quality data, but they are time-consuming and tedious. Therefore, Automated Data Pre-processing (AutoDP) is a critical component of AutoML [18].

Data pre-processing tasks can be divided into the following four categories [111]:

1. **Transformation**: The process of transforming categorical features into continuous features using encoding techniques, or transforming continuous features into categorical features using discretization techniques.
2. **Imputation**: The process of handling missing values using imputation methods.
3. **Balancing**: The process of balancing a dataset's class distribution through over-sampling or under-sampling methods.
4. **Normalization**: The process of converting continuous characteristics to a comparable or same range of values.

2.6.2 Data Transformation

Data transformation indicates the transformation between numerical features and categorical features. Firstly, in real-world applications, many data values are generated as words or strings

Table 2.2: A comprehensive overview of common ML models, their hyper-parameters, suitable optimization techniques, and available Python libraries

ML Algorithm	Main HPs	Optional HPs	HPO methods	Libraries
Linear regression	-	-	-	-
Ridge & lasso	alpha	-	BO-GP	Skpot
Logistic regression	penalty, c, solver	-	BO-TPE	Hyperopt
KNN	n_neighbors	weights, p, algorithm	BOs, Hyperband	Skpot, Hyperopt, Hyperband
SVM	C, kernel, epsilon (for SVR)	gamma, coef0, degree	BO-TPE	Hyperopt
NB	alpha	-	BO-GP	Skpot
DT	criterion, max_depth, min_samples_split, min_samples_leaf, max_features	splitter, min_weight_fraction_leaf, max_leaf_nodes	GA, PSO, BO-TPE	TPOT, Optunity
RF & ET	n_estimators max_depth, criterion, min_samples_split, min_samples_leaf, max_features	splitter, min_weight_fraction_leaf, max_leaf_nodes	GA, PSO, BO-TPE	TPOT, Optunity
XGBoost	n_estimators, max_depth, learning_rate, subsample, colsample_bytree	min_child_weight, gamma, alpha, lambda	GA, PSO, BO-TPE	TPOT, Optunity
LightGBM	n_estimators, max_depth, learning_rate, num_leaves	min_data_in_leaf, boosting_type, feature_fraction	GA, PSO, BO-TPE	TPOT, Optunity
Voting	estimators, voting	weights	GS	sklearn
Bagging	base_estimator, n_estimators	max_samples, max_features	GS, BOs	sklearn, Skpot, Hyperopt
AdaBoost	base_estimator, n_estimators, learning_rate	-	BO-TPE	Hyperopt
Deep learning	number of hidden layers, 'units' per layer, optimizer, Activation, learning_rate, dropout rate, epochs, batch_size, early stop patience	number of frozen layers (if transfer learning is used)	PSO	Optunity
K-means	n_clusters	init, n_init, max_iter	BOs, Hyperband	Skpot, Hyperopt, Hyperband
DBSCAN	eps, min_samples	-	BO-TPE	Hyperopt
Gaussian mixture	n_components	covariance_type, max_iter	BO-GP	Skpot
PCA	n_components	svd_solver	BOs, Hyperband	Skpot, Hyperopt, Hyperband

to make them human-readable. Data encoding is the process of converting string features to numerical features that machine learning models can understand and process [18]. Common encoding techniques include label encoding, one-hot encoding, and target encoding. For better interpretation in ML models, label coding and one-hot encoding assign incremental values or a new column to each string value of categorical features, respectively. However, the transformed values only represent a unique label instead of containing meaning information [112]. Target encoding is to replace categorical values with the mean or median of the target variable. Target encoding can generate meaningful values, like the fraction of the samples in different classes, to replace string values [18].

Many AutoML tools have data transformation functionalities. For example, Auto-Sklearn [113] uses one-hot encoding, and H2O.AI [114] uses target encoding to encode data [18].

On the other hand, data discretization is the process of converting numerical features to categorical features by setting multiple intervals [115]. Data discretization can better handle outliers and simplify the calculations.

2.6.3 Data Imputation

Real-world datasets often have missing values as a result of data inaccessibility or collecting difficulties. Null values, whitespace, NaNs, and incorrect data types are all examples of missing values. Most ML models are incapable of directly handling missing values or are adversely affected by them in the learning process [18].

While dropping the related features or observations with missing values is the easiest solution, it may result in the loss of significant information. As a result, missing values are often resolved using imputation techniques. The purpose of data imputation is to replace missing information with reasonable values. Several basic imputation methods replace missing values with the same value. For numerical features, basic imputation methods replace all missing values in this column with zero, the mean, or the median value of each column [18]. For categorical features, the basic method is called mode imputation, which involves replacing missing values with the most common category in each feature.

However, since the sequential values in IoT time-series data often have strong correlations, basic imputation methods, such as zero, mean, and median imputation methods, are often ineffective in dealing with missing values of IoT time-series data [116]. Thus, many advanced imputation methods for time-series data have been proposed, such as backward/forward filling and the moving window [117]. Backward and forward filling methods replace each missing value with its most recent or next observation, respectively. As a result, they enable missing values to be distributed according to time series distributions. However, imputed values may be misleading for sudden changes in observations. The moving window is another time-series imputation method that replaces each missing value with the average of its previous n observations, indicating a moving window with size *n*. The primary difficulty with the moving window is determining the window size *n*.

Model-based imputation techniques, like linear regression, KNN, and XGBoost imputation, can estimate the missing values as the target variable by learning other feature values [18]. XGboost imputation method is used in several AutoML tools, such as Auto-WEKA [19] and TPOT [118]. Additionally, Datawig [119], a DL-based method, is developed for data imputation. Model-based imputation techniques often outperform model-free methods as imputed values estimated by ML models are often closer to actual values. However, implementing machine learning models often takes much longer than other methods. Moreover, DL-based methods also require high computational power and a relatively large-sized dataset for accurate imputation. The pros and cons of common imputation methods are summarized in Table 2.3.

The main procedures for automating the imputation process are as follows:

1. Calculate the total number of missing values and their percentage in a given dataset to evaluate whether data imputation is necessary;
2. Select a suitable imputation method to handle missing data according to the requirements of specific tasks. If execution speed is the top priority, model-free imputation methods are the most efficient choices. If the model performance is more important than the execution speed, model-based imputation methods can provide more accurate imputed data. On the other hand, if the given dataset is a time-series dataset, time-series imputation methods (*e.g.,* forward/backward filling and moving window methods) are better choices.
3. Optimize the parameters of imputation methods if it is necessary (*e.g.,* the window size in

Table 2.3: The comparison of common imputation methods.

Type	Imputation Method	Pros	Cons
Model-free methods	Dropping	· Easy to implement	· May lose important information
	Basic imputation methods (zero, mode, mean, median imputation)	· Easy to implement · Work well with small datasets	· May generate misleading values · Not consider feature correlations
	Forward/ backward filling	· Work well with time-series datasets	· Not effective for consecutive nulls and sudden changes
	Moving window	· Work well on time-series datasets	· Not effective for a large number of nulls · Need to determine the window size
Model-based methods	Model-based imputation (linear regression, KNN, XGBoost, etc.)	· Perform better than basic methods	· Time-consuming on large datasets · Need to tune hyperparameters
	Datawig (DL imputation)	· Perform better than basic methods	· Require high computational power · Not work well with small datasets

the moving window method and the hyperparameters of ML algorithms in model-based imputation).

2.6.4 Data Balancing

With the growth of IoT data streams, it is becoming more difficult to maintain uniform distributions of all classes for classification problems, resulting in class imbalance. Class imbalance indicates that the distributions of classes in a dataset are highly imbalanced, causing ML model degradation. Severe class imbalance occurs when certain classes have an extremely small number of instances. Many ML algorithms, including SVM, DT-based algorithms, and neural network models, are very sensitive to class imbalance [14]. Learning imbalanced datasets often causes unjustified bias in majority classes, which has an adverse effect on the prediction accuracy of minority classes [18]. Class imbalance problems can be solved by resampling techniques, including over-sampling and under-sampling [120].

Under-Sampling Methods

Under-sampling methods solve class imbalance by reducing the number of samples in the majority classes. Random Under-Sampling (RUS) is a basic under-sampling method for data balancing by randomly discarding samples from the majority classes [121]. By reducing the size of the data, under-sampling techniques can increase model learning efficiency. However, by removing a fraction of data samples, critical information contained in the majority classes may be lost [121].

Over-Sampling Methods

Since under-sampling algorithms may ignore certain critical instances in the majority class, resulting in model performance degradation [122], over-sampling methods are often utilized to resolve class imbalance. Random Over Sampling (ROS) and Synthetic Minority Oversampling TEchnique (SMOTE) [123] are the two common over-sampling methods used to create new instances in the minority classes. Unlike ROS, which simply replicates the instances, SMOTE analyzes the original instances and synthesizes new instances using the principle of KNN. For each instance X in the minority class, assuming its k nearest neighbors are $X_1, X_2, \cdots, X_k$, and X_i is a randomly selected sample from the k nearest neighbors, the new synthetic instance is denoted by [124],

$$X_n = X + rand(0,1) * (X_i - X), i = 1, 2, \cdots, k, \tag{2.24}$$

where $rand(0, 1)$ denotes a random number in the range of 0 to 1.

As SMOTE can solve the majority of class imbalance problems, it can be used as the default method on imbalanced datasets [125].

2.6.5 Data Normalization

ML models often treat features with larger values as more important. If the scales of features are significantly different, data normalization should be used to prevent creating biased models. This is especially important for ML algorithms that use distance calculations, such as k-means, KNN, PCA, and SVM. Z-score and min-max normalization are two of the most often used normalization approaches in ML model learning.

In Z-score normalization, the normalized value of each data point, x_n, is denoted by [126],

$$x_n = \frac{x - \mu}{\sigma}, \tag{2.25}$$

where x is the original value, μ and σ are the mean and standard deviation of the data points.

In min-max normalization, the normalized value of each data point, x_n, is denoted by [126],

$$x_n = \frac{x - min}{max - min}, \tag{2.26}$$

where x is the original feature value, min and max are the minimum and maximum values of each original feature.

Min-max normalization ensures that all features have the same scale of 0-1, but it does not handle outliers well. In contrast, Z-score normalization can handle outliers, although the feature ranges may be slightly different. Thus, these two techniques can be automatically chosen depending on the occurrence and the percentage of outliers.

2.7 Feature Engineering

2.7.1 Overview

Although research into the automated model selection and HPO has made significant progress, Feature Engineering (FE), as a vital component of the ML pipeline, has been ignored in many AutoML applications. Using original feature values to train ML models often cannot obtain the best prediction results. In this case, new features need to be created, and misleading features should be removed to fit specific tasks [18]. The objective of feature engineering is to provide data with optimal input features for ML models. The upper limit of ML applications is determined by FE [4].

Manual feature engineering is tedious and time-consuming, and often requires domain knowledge. Automated Feature Engineering (AutoFE) enables the automation of feature engi-

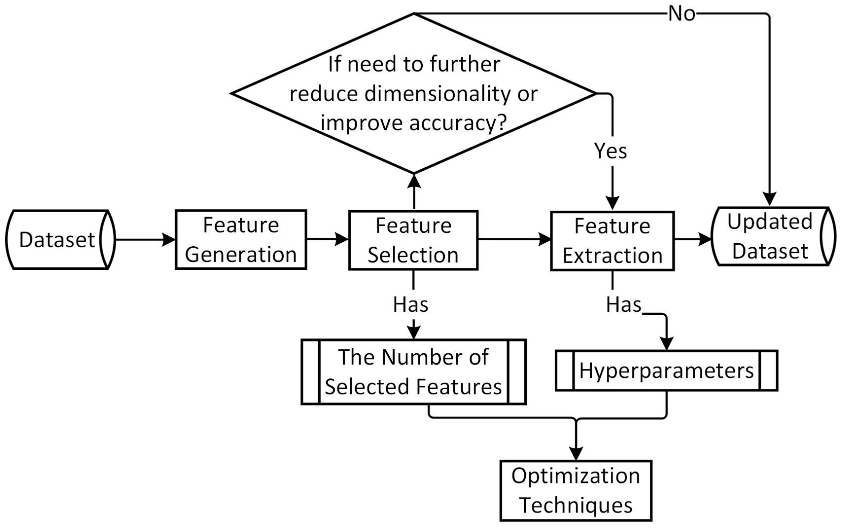

Figure 2.3: An automated feature engineering framework.

neering by automatically generating and selecting relevant features using a generic framework applicable to different problems. AutoFE is more efficient and reproducible than manual feature engineering, enabling faster development of more accurate learning models.

FE methods can be classified into three categories: feature generation, feature selection, and feature extraction. Feature generation is the process of creating new features through the combination or transformation of original features to expand the feature spaces. Feature selection is used to reduce feature redundancy by selecting relevant and significant features. Similar to feature generation, feature extraction can create new features, but its primary purpose is to reduce the dimensionality of original features through mapping functions. AutoFE is essentially a dynamic combination of these three components.

Most state-of-the-art AutoFE approaches, like AutoFeat, use the generate-and-select strategy. In this strategy, an exhaustive feature pool is generated and then valuable features are selected from it [127]. Certain procedures, like feature selection and extraction, may require the use of optimization techniques to identify appropriate parameters that can return the optimal model. As shown in Fig. 2.3, the main procedures for the AutoFE framework with the generate-and-select strategy are as follows:

1. Generate a variety of candidate features using common operations;
2. Select important features using feature selection methods;

3. Determine the optimal number of features using optimization techniques;
4. Further extract features and reduce the data dimensionality if it is still high or the learning performance still needs to be further improved.

2.7.2 Feature Generation

Feature generation is the process of generating new features by transforming or combining existing features to improve the generalizability and robustness of a ML model [4]. In real-world IoT systems, data is often scattered over many devices and files and must be combined into a single database with rows for observations and columns for features [18]. Although the feature generation step often requires domain knowledge from experts, certain features can still be created automatically to extract useful information.

Common feature generation operations can be classified as follows [82]:

1. **Unary operations**: Numerical feature discretization or normalization, time expansion, or mathematical operations like a logarithm.
2. **Binary operations**: The combination of two features using feature correlations or mathematical operations (*e.g.,* addition, subtraction, multiplication, division, etc.).
3. **High-order operations**: The calculation of multiple records for one feature, like the maximum, minimum, average, or median values.

It is challenging to manually produce all meaningful and useful features. The process of generating valuable features usually requires human expertise. For automated feature generation, the general process is to automatically generate a large number of features using various operations and use feature selection techniques to select the relevant and useful features. Although generating numerous features is often time-consuming, automated feature generation can largely reduce human efforts and overhead by getting rid of the dependence on human expertise. Decision tree-based and GA-based methods described in [128] can be used to simplify the feature generation process by defining and exploring the feature space.

2.7.3 Feature Selection

While feature generation may create a large number of features, some of them may be irrelevant or redundant. For specific tasks, certain features have a great impact on the target variable prediction, while other features may have a minimal or negative effect on the prediction [18]. Therefore, Feature Selection (FS) should be implemented to identify the most appropriate features for use in constructing a more efficient and accurate learning model [14]. FS is a time-consuming and challenging procedure in ML pipelines, especially for high-dimensional datasets.

Automated feature selection is the process of automatically selecting a subset of the original feature set to improve ML model performance and training speed by removing irrelevant and redundant features [82]. To achieve AutoFS, the FS problem can be framed as an optimization problem [129]. For a small number of input features, all combinations of the features can be evaluated to detect the best performing feature set. On the other hand, for a large number of features, optimization techniques can be utilized to explore the feature search space and identify the optimal feature set.

Existing FS methods can be divided into three categories: filter methods, wrapper methods, and embedded methods.

Filter methods assign a score to each feature by calculating its importance, and then select a subset of features based on a given threshold (*e.g.,* the number of selected features or the accumulated importance). Each feature's score can be estimated using a variety of measures, including Information Gain (IG), the chi-square test, Pearson correlation coefficient, variance, etc. For example, the Fast Correlation Based Filter (FCBF) is a popular filter method that measures the correlation of features and selects features by calculating the Symmetrical Uncertainty (SU) [125].

Wrapper methods make predictions based on a selected subset of features, and then evaluate the feature set according to prediction accuracy. Recursive Feature Elimination (RFE) is a wrapper method that recursively evaluates subsets of features to remove irrelevant features until a desired number of features are selected [130].

Embedded methods indicate the FS process included in the learning process of ML models, like Lasso regularization, DT-based algorithms, and DL models. Thus, using those ML algorithms with embedded FS functionality often does not need additional FS procedures.

These three types of FS methods have different advantages and limitations. The major advantage of filter methods is that they can be completed prior to model training, which results in a relatively fast execution time [130]. However, since filter approaches derive their features only via statistical measurements, they may not be optimal for all ML models. Embedded methods can select the relevant features for a specific ML model in its construction process [130]. Thus, it can often achieve the optimal performance on this specific ML model. However, the selected features are often only beneficial for the same type of model. Additionally, applying embedded methods to select features for another ML model is often time-consuming. Therefore, using an embedded method in the ML model where it is embedded is often the most appropriate choice. Wrapper techniques can be used for various ML models to select the most relevant features [130]. However, wrapper methods are often more time-intensive than filter and embedded methods, as they need to continually train a ML algorithm on different subsets of features until the termination conditions are met.

In conclusion, when choosing feature selection approaches, a trade-off between the time complexity and learning performance must be made. Different FS approaches should be selected in different situations or tasks. Filter methods are often utilized in tasks with strict time constraints, while wrapper techniques often work better for tasks that demand great performance. Embedded methods are often used in certain ML algorithms that already have these embedded FS functionalities.

2.7.4 Feature Extraction

Feature extraction is the process of reducing dimensionality using mapping functions. Unlike feature generation, which preserves the original features, feature extraction alters the original features to extract more informative features that can replace the original features [14]. Through feature extraction, a more concise representation of the original dataset can be obtained. Additionally, model learning efficiency may be increased by dimensionality reduction [128]. Common feature extraction methods include PCA and AE [4]. Feature extraction is not a required procedure in the general feature engineering process. It is often utilized only when the feature set produced after feature generation and selection is still high dimensional or under-performing, since feature extraction can further reduce dimensionality and misleading feature components.

2.8 Automated Model Updating by Handling Concept Drift

2.8.1 Model Drift in IoT Systems

Because of the dynamic IoT environments, IoT online data analysis often encounters concept drift issues when data distributions shift over time. Concept drift often impairs the performance of IoT data analytics models, posing significant threats to IoT services. To deal with concept drift, a successful data analytics model must reliably identify and respond to detected drifts in order to retain high prediction accuracy.

Concept drift refers to the conceptual and unpredictable changes in data streams [131]. The presence of concept drift has brought significant challenges to the development of ML models. As the majority of ML models are built with the premise that the data is collected in a static environment, they lack the adaptability to learn streaming data with concept drift [132]. Thus, in an ever-changing environment with concept drift issues, ML models' performance may gradually degrade. When concept drift occurs, it is necessary to upgrade the current ML model in order to preserve or enhance model performance [133]. This process is also referred to as automated model updating. To ensure high performance in a non-stationary environment, an IoT data analytics model should be updated automatically when concept drift occurs.

2.8.2 Concept Drift Definition

In non-stationary and dynamically changing environments, the distribution of input data often changes over time, causing concept drift. Given an instance $(\boldsymbol{X}, y)$ with the input features $\boldsymbol{X}$ and the target variable y, concept drift that occurs between the time points t_0 and t_1 can be denoted by [134]:

$$\exists X : P_{t_0}(\boldsymbol{X}, y) \neq P_{t_1}(\boldsymbol{X}, y) \tag{2.27}$$

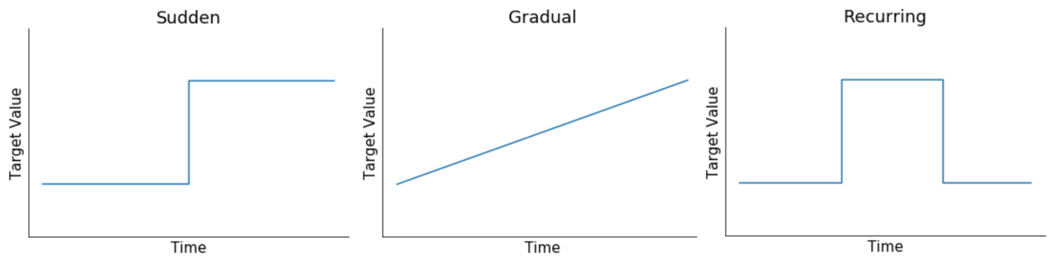

Figure 2.4: Concept drift types.

where P_{t_0} represents the joint distribution between $\boldsymbol{X}$ and y at time t_0.

The joint probability $P_t(\boldsymbol{X}, y)$ can be calculated by [131]:

$$P_t(\boldsymbol{X}, y) = P_t(\boldsymbol{X}) \times P_t(y \mid \boldsymbol{X}) \tag{2.28}$$

where $P_t(\boldsymbol{X})$ denotes the marginal probability and $P_t(y \mid \boldsymbol{X})$ represents the posterior probability.

Although changes in different probabilities can result in concept drift, only the distribution changes that affect the performance of learning models should be dealt with for data learning purposes [134]. Thus, the changes in the posterior probability, $P_t(y \mid \boldsymbol{X})$, are referred to as real concept drift because they cause model decision boundary changes and model performance degradation; other types of drift, like changes in $P_t(\boldsymbol{X})$, are referred to as virtual concept drift and are not taken into account in the learning system adaptation procedures [131].

As illustrated in Fig. 2.4, there are three major types of data distribution changes that can cause concept drift: sudden, gradual, and recurring drift.

1. Sudden drift is the term used to describe the rapid and irreversible changes that occur in a short period of time.
2. Gradual drift occurs when a new data distribution gradually replaces an older one over time.
3. Recurring drift is a temporary change in the distribution of data. The distribution will return to its previous state within a certain period of time.

Due to the occurrence of concept drift, the learning system must detect drift in time and update itself by adapting to the detected drift; hence, accurate predictions can be made on the continuously arriving data streams. Therefore, in addition to the training and prediction procedures in traditional ML models, there are two additional procedures for analyzing streaming data with concept drift: drift detection (detect the occurrence and the time of drift) and drift adaptation (handle the detected drift) [131]. In this Section, the state-of-the-art drift detection and adaptation methods are described. Additionally, the strengths and limitations of the drift detection and adaptation methods discussed in this chapter are summarized in Table 2.4.

2.8.3 Drift Detection

To design a model capable of dealing with concept drift, it should be able to effectively detect drift nodes and address the drift rapidly. Thus, drift detection is critical functionality for adaptive ML models capable of resolving concept drift problems.

Drift detection methods are generally classified into two main categories: distribution-based methods and performance-based methods. Distribution-based methods identify concept drift by detecting the changes in data distributions. Statistical variables, such as the mean, variance, and class imbalance, can be used to quantify data distribution changes. In model-based methods, concept drift is measured based on the changes in the metrics used to assess model performance. For example, accuracy degradation and error rate increase are common indicators of concept drift. The severity of concept drift can be measured by the degree of model performance degradation.

Distribution-based methods

Distribution-based drift detection techniques are developed by measuring and comparing the data distributions of old and new data in time windows. Significant data distribution changes often cause concept drift and trigger model updates [131]. There are several approaches to measure the data distributions of different time windows, like mean, variance, information entropy, Kullback-Leibler (KL) divergence, etc.

ADaptive WINdowing (ADWIN) [135] is a distribution-based approach for detecting concept drift using variable-size sliding windows and characteristic values (*e.g.,* mean, variance). If no noticeable drift or distribution change is identified when the streaming data enters the model, the window size is dynamically enlarged, whereas the window size is reduced when

Table 2.4: The comparison of concept drift methods for automated model updating.

Task	Category	Methods	Strengths	Limitations
Drift Detection	Distribution -based Methods	ADWIN	· Work well with gradual drifts. · Good interpretability.	· A single ADWIN is limited to one-dimensional data. · Characteristic values used by ADWIN are not always effective.
		IE, KL Divergence	· Good interpretability. · Can work with unlabeled data.	· High computational cost. · May detect virtual drifts. · Require pre-defined time periods.
	Performance -based Methods	DDM	· Work well with sudden drifts. · Can ensure all detected drifts are real drifts.	· Slow reaction time. · Ineffective for gradual drifts. · Need to tune the drift and warning thresholds.
		EDDM	· Work better with gradual drifts than DDM.	· Sensitive to noise. · Need to tune the drift and warning thresholds.
Drift Adaptation	Model Retraining	Full Retraining	· Easy to understand and implement. · Can retain all existing concepts.	· Time-consuming due to unnecessary retrainings. · May become extremely slow as data increases.
		Partial Retraining	· Easy to understand and implement. · Can remove outdated samples. · Faster than full retraining.	· May lose historical patterns. · Unnecessary retrainings.
		Instance Weighting	· Can retain all the existing concepts. · Can better adapt to drifts than full and partical retrainings.	· Time-consuming due to unnecessary retrainings. · Need to choose an updateable learner capable of weighted learning.
	Incremental Learning	HT, VFDT, CVFDT	· Can be continuously updated. · Fast training speed due to partial updating.	· Incapable of directly addressing concept drift. · Limited ML algorithms support incremental learning.
		AONN	· Strong adaptability to drifts.	· Ineffective for sudden drifts. · Time-consuming.
	Ensemble Learning	SEA, AWE, ACE	· Can retain historical concepts. · Strong adaptability to drifts. · Good generalizability.	· Need to determine a proper chunk size. · Time consuming. · Outdated concepts may be misleading.
		ARF, SRP, LB, PWPAE	· Strong adaptability to drifts. · Good generalizability.	· Time-consuming. · Require high memory space.

concept drift is identified [133]. The main procedures of ADWIN are as follows:

1. For a sliding window W, the characteristic values (e.g., mean, variance) between its two sub-windows, W_1 and W_2, is computed and compared. W_1 and W_2 represent earlier and more recent data, respectively.
2. A concept drift alarm will be triggered if the characteristic values of W_1 and W_2 diverge significantly enough (*i.e.,* the difference exceeds a specific threshold).
3. Once a drift has been detected, the sliding window size is adjusted to the newer sub-window, W_2, while the older subwindow, W_1, is dropped.

ADWIN is well-suited for data streams with gradual drift because the sliding window can be enlarged to a large size window for detecting long-term changes. On the other hand, a single ADWIN model can only handle one-dimensional data. As a result, multiple ADWIN base models with discrete windows for each dimension are required for multi-dimensional data [11]. Furthermore, the mean value is not always a suitable method to define changes.

Distribution-based approaches can be used to detect concept drift using other metrics. The Information Entropy (IE) [136] is a widely-used distance metric to quantify how much information is included in a data distribution. The entropy of a data distribution X can be calculated by:

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{2.29}$$

Assuming the two probability distributions are p and q, the IE-based method calculates their distance based on the difference between their entropy values:

$$D_{IE}(p||q) = |H(p) - H(q)| \tag{2.30}$$

KL divergence [137] is another common distant metric to compute the distance between two distributions.

For the two probability distributions p and q, KL divergence estimates the distance by:

$$D_{KL}(p||q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} \tag{2.31}$$

In IE or KL divergence-based methods, concept drift occurs when $D_{IE}(p||q)$ or $D_{KL}(p||q)$ is greater than a threshold α.

Distribution-based methods are often used in systems with limited memory, like IoT devices, since they need only the most recent samples to be stored. Additionally, distribution-based approaches can provide a high level of interpretability by illustrating how the data distribution varies over time. They can also generally specify the exact time and place of the drift, which is beneficial for drift adaptation. However, distribution-based approaches often incur higher computational costs than performance-based approaches and often require the use of predefined historical and new time periods.

Performance-based methods

Model performance-based methods track changes in learners' prediction error rates to identify concept drift [138]. If the data distribution is stationary and without drift, the error rate of a learner should gradually decrease or remain constant as more data samples are learned. Conversely, if a learner's error rate rises dramatically as more data is processed, this often reveals the occurrence of concept drift.

Drift Detection Method (DDM) is a popular performance-based method that measures model error rate and standard deviation changes using two pre-defined thresholds, the warning threshold and the drift threshold [139]. With an estimating error rate at time t is p_t, the standard deviation at time t can be calculated based on the Binomial distribution:

$$S_t = \sqrt{p_t(1-p_t)/t} \tag{2.32}$$

The error rate and standard deviation can be used to determine whether the warning level or drift level have been exceeded:

$$\begin{cases} \text{if } p_t + s_t \geq p_{\min} + 2 * s_{\min} \rightarrow \text{ Warning Level} \\ \text{if } p_t + s_t \geq p_{\min} + 3 * s_{\min} \rightarrow \text{ Drift Level} \end{cases} \tag{2.33}$$

where $p_{\min}$ and $s_{\min}$ are the current error rate's minimum and standard deviation's minimum, respectively. If the warning threshold is exceeded, newly arrived samples will be archived for potential drift adaption. If the drift level is exceeded, the learning model will be updated with the newly collected data.

DDM often performs well on data streams with sudden drift, but its reaction time is often too slow for detecting gradual drift [140].

Early Drift Detection Method (EDDM) is an improved version of DDM that uses the same warning and drift mechanism as DDM to identify concept drift [141]. Unlike DDM, EDDM detects drift by calculating the change rate of the learner's error rate, rather than the error rate itself. Although EDDM often outperforms DDM, it is still inferior to distribution-based approaches on the gradual drift. Additionally, since it is sensitive to noise, it may misclassify noise as drift, resulting in false alarms.

Both DDM and EDDM have three primary hyperparameters that have a direct impact on the accuracy of drift detection: 1) the warning threshold; 2) the drift threshold; 3) the minimum number of incoming samples before detecting the first drift [142].

2.8.4 Drift Adaptation

After detecting concept drift, it is necessary to update or reconstruct the existing models to handle the drift using proper drift adaptation methods. Drift adaptation techniques can be classified into three categories: 1) Model retraining; 2) Incremental learning methods; 3) Ensemble learning methods.

Model Retraining

Model retraining is a simple and straightforward method for reacting to concept drift [131]. Owing to the fact that pre-trained or offline models cannot always precisely predict the future incoming streaming data due to concept drift, they can be retrained on the newly arrived data streams to maintain high performance. A conventional online learning strategy for analyzing data streams without concept drift detection is to update the learner regularly to fit the most recent data. However, using this method can result in unnecessary model retrainings or drift adaptation delays. Therefore, an appropriate drift detection method should be used together with learning models to determine when to retrain the learning model for timely and necessary updates.

Model retraining strategies include full retraining, partial retraining, and instant weighting. Full retraining is the process of retraining the learning model on the entire dataset involving all available samples. Full retraining is easy to implement but often time-consuming.

Partial retraining is developed by retraining the model on only certain parts of data to improve model updating efficiency. Using window-based strategies that retain the model only on the recent data can reduce the training time but may result in the loss of historical data patterns. Thus, it is critical to choose a proper window size. ADWIN [135] is a well-performing drift detection method for model retraining since it uses a dynamic window to fit new data. Optimized Adaptive and Sliding Windowing (OASW) [10] is another partial retraining model for IoT data stream analytics. It uses adaptive and sliding windows to detect concept drift and collect samples of a new concept. Thus, the learning model can be partially retrained on only the new concept samples to save training time.

Instead of directly retraining the learning model on new data, the instance weighting method is another popular model retraining technique [132] [143]. It adjusts the weights of data samples according to their timestamp or retention time. Recent samples will be assigned a higher weight, while old data samples will be assigned a reduced weight or even deleted from the training set. This method is based on the assumption that as time passes, outdated data samples become less relevant, and new data samples become more critical. As a result, an existing learning model can adjust to concept drift by retraining on weighted samples. For weighted model retraining, an updateable learner capable of weighted learning should be used.

Incremental Learning methods

In data stream analytics, new data samples are continuously being added to the learning system. Rather than learning offline on static data, an effective model needs to be continuously updated to adapt to the changing data distributions. Therefore, incremental learning has become a widely-used strategy in data stream analytics research. Incremental learning is the process of learning data samples sequentially and updating the learning model with each instance is processed [144].

When new samples arrive, incremental learning approaches often partially update the learning model to fit the new samples [145]. Due to the progressive learning ability of incremental learning approaches, they do not need a sufficient amount of data prior to the training process. However, only a small number of ML algorithms enable partial updates, including MLP, multinomial NB, etc. Thus, several new incremental learning methods for concept drift adaptation, such as Hoeffding Trees (HT) [146] based methods and Adaptive Online Neural Network (AONN) [147], were proposed. HT algorithms based on Hoeffding's inequality are one of the most common incremental learning methods for data stream analytics. By using the Hoeffding bound to calculate the number of samples required to determine the split node, the nodes in HTs can be partially updated as new samples arrive [138]. There are several variants of HTs, including the Very Fast Decision Tree (VFDT), Concept-adapting Very Fast Decision Tree (CVFDT), Extremely Fast Decision Tree (EFDT), etc. [134].

VFDT is a technique for creating classification decision trees in a data stream mining environment by using Hoffding inequality [148]. It is constructed by continuously replacing leaf nodes with branch nodes to preserve an essential statistic at each decision node, and the splitting test is performed when the statistic of the node reaches a certain threshold. VFDT is an efficient method since it only has to process the data stream once. Additionally, it can often achieve high performance comparable to typical ML techniques. The primary drawback is that it is incapable of effectively addressing the concept drift issue.

CVFDT extends VFDT to rapidly tackle the concept drift issue associated with data streams [146]. CVFDT's basic principle is to replace the historical subtree with a new subtree that has a lower error rate. It uses a sliding window to choose test data samples and updates the resulting decision tree as data flows into and out of the time frame.

EFDT [149], also known as the Hoeffding Anytime Tree (HATT), is a modified version of the HT that divides nodes as soon as the confidence level is reached, rather than identifying the optimal split in the HT. This splitting method enables the EFDT to adjust more precisely to concept drifts than the HT, although its performance still has much room for improvement.

AONN is another incremental learning method based on neural network models [147]. In AONN, a model update is triggered when the model's error increases. The AONN network is updated by either increasing the number of neurons in the output layers or by changing the weights of neurons using a batch of online data samples. Incremental methods can often adapt to new data patterns by continually learning from newly received data samples. However, they are not specifically designed to address concept drift, as the old concepts and model components are still retained. Thus, they are ineffective in addressing certain types of drifts, like sudden drifts, which often need a completely new learner.

Ensemble Learning methods

Ensemble learning techniques have been developed to generate powerful learners for data stream analytics in order to achieve greater concept drift adaptation. Ensemble learning is a ML technique that combines multiple base learners to tackle the same problem [145]. In ensemble learning, base learners can be constructed using different algorithms, different hyperparameter configurations, or different subsets. As ensemble learning models aggregate the outputs of multiple base learners, they often have better generalizability than single models. For concept drift adaptation, reusing existing models in an ensemble is much more efficient than training new models on data streams with recurring concept drift [131]. Ensemble methods for data stream analytics can be further classified as block-based ensembles and online ensembles [150].

Block-based ensembles divide the data streams into discrete blocks with defined sizes and train a base learner on each block. When a new block is added, the existing base learners are evaluated and upgraded. Many block-based ensemble learning methods have been designed for concept drift adaptation, including Streaming Ensemble Algorithm (SEA), Accuracy Weighted Ensemble (AWE), Adaptive Classifier Ensemble (ACE), Learn++.NSE, Dynamic Weighted Majority (DWM), Diversity and Transfer-based Ensemble Learning (DTEL), etc.

SEA [151] is an ensemble learning model that adapts to concept changes by changing its structure. It constructs an ensemble of N base learners, each trained on a batch of data samples. The final result is computed using the majority voting technique, which combines the prediction outcomes of base learners with the same weight. SEA limits the maximum number of base learners by the use of a threshold. Once the threshold is reached, the newly trained base learner will replace the worst-performing base learner according to the error rate and diversity. Experimental studies show that SEA is effective when the ensemble has no more than 25 base learners [140].

AWE [152] is another ensemble learning approach that trains a base learner on each data chunk and combines the base learners, but it improves the technique of base learner replacement. Each incoming data chunk will be used to train a new base learner and evaluate the other existing base learners. The top *n* best-performing base learners will be chosen to create a new ensemble model. Thus, the outdated base learners will be removed from the ensemble, leaving only those capable of effectively predicting the data with the new concept. The AWE method outperforms other methods when dealing with streaming data that contains recurring concept drift, and its performance on large streaming data will continue to improve [140]. However, AWE's chunk-size selection remains a concern. Additionally, a noisy new data block may result in a biased ensemble [153].

ACE [154] is a variant of AWE that is designed to deal with gradual drift. It continuously monitors the error rate change of each base learner in response to new input data, and removes the base models with degrading performance. ACE is effective at handling gradual drift, but struggles with sudden and recurring concept drifts.

DWM [155] is another ensemble model that trains multiple base learners but weights them differently based on their prediction performance. When a base learner makes an inaccurate prediction, its weight is slightly reduced. Additionally, if the ensemble model makes an incorrect prediction, a new base learner will be trained and given the highest weight among the base learners. The primary advantage of DWM is that it is capable of preserving historical models built on existing concepts. However, it may be resource-intensive, particularly when dealing with huge volumes of streaming data [153].

Learn++.NSE [156] is an ensemble learning model that consists of multiple incrementally trained neural network models. Each base learner is trained on a single batch of incoming data. The Learn++.NSE model dynamically weights base neural network models depending on their error rates on the most recent batch of data. Additionally, the incorrectly predicted instances will be assigned a higher weight, allowing learners to concentrate on the challenging instances. When the ensemble model's prediction error rate exceeds a predefined threshold, a new base learner is trained and added to the ensemble [153]. Learn++.NSE can handle sudden, gradual, and recurring drifts, because the base learners can be deactivated and reactivated by adjusting their weights [131].

DTEL [144] is an ensemble learning model that trains and stores each historical model from the initial models and then uses a transfer learning strategy to transfer the initial or historical models to new incoming data. To maintain model diversity in DTEL, it is crucial to train base models on a number of diverse data distributions or concepts. DTEL works effectively in the presence of recurring drift because historical models can be preserved and directly transferred to a new drift. On the other hand, owing to the utilization of transfer learning, DTEL often has a high learning efficiency and a quick response time to drift.

Paired Learner (PL) technique [157] is an effective drift adapter that pairs a steady online learner with a reactive one to deal with concept drift. A stable learner makes predictions based on its entire experience, while an active learner makes predictions based on its most recent experience. Thus, the proper learner can be chosen for different scenarios. Comparative studies have revealed that the PL approach outperformed a wide variety of other ensemble methods or achieved equivalent performance at a much lower computational cost.

Online ensembles can enhance learning performance by integrating multiple incremental learning models, such as HTs. Gomes *et al.* [158] introduced the Adaptive Random Forest (ARF) technique, which makes use of HTs as base learners and ADWIN as the default drift detector for each tree. The drift detection process substitutes new trees that fit the new concept for underperforming base trees. ARF often outperforms a wide variety of other techniques, since the random forest method is also a well-performing ML technique. Additionally, ARF makes optimal use of resampling and is adaptable to a wide range of drift types.

Gomes *et al.* [159] have presented a unique adaptive ensemble approach for streaming data analytics called Streaming Random Patches (SRP). SRP makes predictions using a combination of random subspace and online bagging techniques. SRP is similar to ARF in principle but employs a global subspace randomization mechanism rather than ARF's local subspace randomization. Global subspace randomization is a more flexible method of boosting the diversity of base learners. While SRP's prediction accuracy is often slightly higher than that of ARF, its execution time is frequently longer. The number of base learners and the embedded drift detector (*e.g.,* ADWIN, DDM, EDDM, etc.) are the two significant hyperparameters of SRP and ARF models.

Leverage Bagging (LB) [160] is another popular online ensemble that uses bootstrap samples to construct base learners. It employs the Poisson distribution to increase the diversity of input data and maximize bagging performance. While LB is simple to build, it often performs worse than SRP and ARF.

Performance Weighted Probability Averaging Ensemble (PWPAE) [138] is a novel online ensemble framework for concept drift adaptation. It uses the weighted prediction probabilities to integrate four base online learners: ARF-ADWIN, ARF-DDM, SRP-ADWIN, and SRP-DDM. PWPAE outperforms other compared drift adaptation methods as it uses dynamic weights to take advantage of other online learning models. However, the computational complexity of PWPAE is also higher than other methods.

Although ensemble learning models often perform well when dealing with gradual and recurring drifts, they are incapable of coping with abrupt drifts owing to the ensemble learner's limited impact on a new base learner. In comparison to a single learner, however, using an ensemble learning model often increases computing complexity and costs. Thus, ensemble models that use the local learning strategy to train each base learner on a small local subset are more efficient in streaming data analytics [145].

2.9 Selection of Evaluation Metrics and Validation Methods

2.9.1 Evaluation Metrics Selection

To evaluate the learning model on a given IoT dataset, appropriate metrics should be selected in the AutoML pipeline, as they have a significant impact on model selection and HPO procedures.

The performance metrics are mainly chosen according to the types of problems (*e.g.,* accuracy, precision, recall, and F1-score for classification problems; Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) for regression problems) [125] [161] [162].

Classification Metrics

Accuracy is the most basic metric, defined as the proportion of correctly categorized test instances to the total number of test instances [163]. It is applicable to the majority of classification problems but is less useful when dealing with imbalanced datasets. Accuracy can be calculated by using True Positives (TPs), True Negatives (TNs), False Positives (FPs), and False Negatives (FNs):

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.34}$$

Precision is the metric used to quantify the correctness of classification. Precision indicates the ratio of correct positive classifications to expected positive classifications. The larger the proportion, the more accurate the model, indicating that it is more capable of correctly identifying the positive class.

$$Precision = \frac{TP}{TP + FP} \tag{2.35}$$

Recall is a measure of the percentage of accurately recognized positive instances to the total number of positive instances.

$$Recall = \frac{TP}{TP + FN} \tag{2.36}$$

The F1 score is calculated as the harmonic mean of the Recall and Precision scores, therefore balancing their respective strengths.

$$F1 = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{2.37}$$

The Receiver Operating Characteristic curve (ROC curve) plots the true positive rate against the false positive rate. AUC-ROC stands for Area Under Receiver Operating Characteristics, and a larger area indicates a more accurate model.

If class imbalance occurs, the F1-score or AUC-ROC should be used instead of accuracy to determine the optimal solution. Otherwise, a biased model may be returned.

Regression Metrics

In contrast to classification models, which produce discrete output variables, regression models aim to predict continuous output variables [162]. As a result, relevant measures for evaluating regression models are appropriately established.

MSE is a straightforward measure that computes the difference between the actual and anticipated values (error), squares it, and then delivers the mean of all errors. MSE is very sensitive to outliers and will display a very large error rate even if a few outliers exist in otherwise well-fitted model predictions. Assuming y is the real value and $\hat{y}$ is the estimated value, the MSE for a dataset of size n can be denoted by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.38}$$

RMSE is the root of MSE. The advantage of RMSE is that it assists in reducing the magnitude of the mistakes to more interpretable numbers.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.39}$$

MAE is the average of the absolute error numbers (actuals – expectations).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.40}$$

MAE is the preferred method when outlier values need to be ignored, since it considerably reduces the penalty associated with outliers by deleting the square terms.

Unsupervised Learning Metrics

The Silhouette Coefficient quantifies how close or distant each point in one cluster is to each point in the other clusters [63]. Higher Silhouette values (closer to +1) indicate a strong separation of sample points from two different clusters. While a value of 0 indicates that the points are close to the decision boundary, values closer to -1 indicate that the points were incorrectly assigned to the cluster.

Execution Time & Memory

Due to the fact that IoT systems often face strict time and memory limits, the execution time and memory usage of the AutoML model should also be considered [148]. The execution time comprises the time spent training and updating the model, as well as the time spent testing each instance. This can be used to determine if the learning model meets the requirements for real-time processing. Memory consumption can be used to determine if the size of the learning model and the memory used by it are smaller than the system memory on the IoT device, which is often used for edge computing; otherwise, the model must be implemented on a cloud server to ensure enough computational power and resources.

2.9.2 Validation Method Selection

Hold-out Evaluation

Hold-out evaluation is a frequently used evaluation method for ML algorithms [134]. In hold-out evaluation, a hold-out subset is separated from the original dataset before the model training process. After training a model, its generalizability on the previously unseen dataset will be validated using the hold-out subset. For streaming data with concept drift, a hold-out evaluation will assess a learner at time t by generating a hold-out subset that has the same concept at t. The current learning model is evaluated on the test sets at regular time intervals. Thus, for dynamic IoT data stream analytics, hold-out evaluation is only able to evaluate the learner's performance on synthetic data with predefined drift times [131].

Cross-Validation

Cross-validation is an effective and popular evaluation method [164]. It uses a resampling strategy to evaluate ML models and assess how a model performs on different partitions of a given dataset. The k-fold cross-validation method is conducted by dividing the original dataset into k equal subsets for k different experiments; in each experiment, each of the subsets is selected as the validation set, while the other $(k - 1)$ subsets are used as the training set. The average performance of the learning model in k experiments is calculated as the final prediction performance. Unlike hold-out validation, which only assesses ML models on a subset of data and may produce biassed models that only perform well on a subset of data, using cross-validation can evaluate ML models on all subsets of data to avoid over-fitting. Cross-validation in IoT time-series data enables the evaluation of a learner on different time periods, assisting in the development of a complete and robust online learning model.

The order of the data is critical for time-series-related problems. For time-related datasets, random split or k-fold split of data into train and validation may not yield good results. For the time-series dataset, the split of data into train and validation sets is according to the time, also referred to as the forwarding chaining method or rolling cross-validation. For a particular iteration, the future instances of train data can be treated as validation data.

Prequential Evaluation

Prequential evaluation, also named test-and-train validation, is one of the most appropriate methods to evaluate model learning performance on data streams generated in dynamic environments [163] [165]. In prequential evaluation, each incoming instance is firstly predicted by the learning model to update the metrics, and then learned by the model for model updating [131]. The prequential error E can be calculated by the sum of a loss function:

$$E = \sum_{t=1}^{n} f(y_t, \hat{y}_t) \tag{2.41}$$

where n is the total number of incoming samples, y_t and $\hat{y}_t$ are the true and predicted values of the t_{th} sample, the loss function f can be selected from the metrics introduced in Section 2.9.1, based on the problem type.

The prequential error is dynamically updated as new data samples arrive. Thus, using prequential evaluation can monitor the real-time performance of a learning model using metrics that change dynamically with each new data sample. Prequential evaluation can often be used to evaluate real-time model performance and take the most advantage of streaming data.

2.10 Tools and Libraries

2.10.1 AutoML & HPO Tools

Auto-Weka [19] is recognized as the first framework for AutoML. It is built on top of Weka, a well-known Java library package that contains a large number of ML methods. Bayesian optimization methods are the core strategies of Auto-Weka for both model selection and HPO procedures.

Auto-Sklearn [113] is a Python package for AutoML and CASH that is developed on top of Scikit-Learn. Auto-Sklearn introduced the concept of meta-learning for the model selection and HPO procedures. BO and ensemble approaches are employed in Auto-Sklearn to optimize the output models' performance. Both meta-learning and ensemble approaches can enhance the performance of model optimization.

Hyperopt-Sklearn [166] is AutoML framework built on the Scikit-learn library. Hyperopt-Sklearn utilizes Hyperopt to establish the search space for possible Scikit-Learn core components, such as the HPO and preprocessing approaches. Hyperopt supports a variety of optimization techniques for CASH, including random search and Bayesian optimization, for exploring search spaces including different types of variables.

Auto-Keras [167] is an open-source AutoML library. It is developed on top of Keras, a well-known DL library. Auto-Keras implements HPO methods to design optimal DL models.

TPOT [118] is a tree-based optimization framework for AutoML applications built on top of Scikit-Learn. It uses genetic algorithms to explore potential configurations by feature engineering and CASH procedures, thus finding the best solution.

H2O [114] is a AutoML platform that supports both Python and R languages. H2O is capable of automating a wide variety of complex ML tasks, including feature engineering, model selection, model tuning, model visualization, and model validation.

Amazon SageMaker [168] is a AutoML tool built on Amazon Web Services (AWS). It involves automated model tuning as a major module. In Amazon SageMaker, RS and BO methods are used to optimize ML models. It enables large-scale parallel optimization of complicated models and datasets.

Scikit-optimize (Skopt) [169] is a HPO library that is built on top of the scikit-learn [24] library. It implements several sequential model-based optimization models, including RS and BO-GP. The methods exhibit good performance with small search space and proper initialization.

Hyperopt [166] is a HPO framework that involves RS and BO-TPE as the optimization algorithms. Unlike some of the other libraries that only support a single model, Hyperopt is able to use multiple models to model hierarchical hyper-parameters. In addition, Hyperopt is parallelizable since it uses MongoDb as the central database to store the hyper-parameter combinations.

Optunity [84] is a popular HPO framework that provides several optimization techniques, including GS, RS, PSO, and BO-TPE. In Optunity, categorical hyper-parameters are converted to discrete hyper-parameters by indexing, and discrete hyper-parameters are processed as continuous hyper-parameters by rounding them; as such, it supports all types of hyper-parameters.

Hyperband [100] is a Python package for tuning hyper-parameters by Hyperband, a bandit-based approach. Similar to ’GridSearchCV’ and ’RandomizedSearchCV’ in scikit-learn, there is a class named ’HyperbandSearchCV’ in Hyperband that can be combined with sklearn and used for HPO problems. In ’HyperbandSearchCV’ method, cross-validation is used for evaluation.

TPOT [118] is a Python tool for auto-ML that uses genetic programming to optimize ML pipelines. TPOT is built on top of sklearn, so it is easy to implement TPOT on ML models. ’TPOTClassifier’ is its principal function, and several additional hyper-parameters of GA must be set to fit specific problems.

2.10.2 Online Learning and Concept Drift Adaptation Tools

Several tools and frameworks are available for analyzing streaming data and resolving concept drift issues.

Massive Online Analysis (MOA) [170] is an open-source tool for streaming data analysis. It is developed in Java and is based on the Waikato Environment for Knowledge Analysis (WEKA) platform. MOA is capable of detecting and adapting to concept drift using a number of strategies, including DDM, EDDM, and Hoeffding tree. Additionally, MOA contains various classes for creating streaming data, such as those for the Agrawal, Hyperplane, and Waveform datasets.

Scikit-multiflow (Skmultiflow) [171] is a Python package for streaming data learning and concept drift adaptation. It provides many state-of-the-art streaming data learning algorithms, data generators, concept drift detection methods, and algorithms. The included drift detection methods are ADWIN, DDM, EDDM, and Page Hinkley. Streaming data learners for concept drift adaptation include KNN+ADWIN, Hoeffding adaptive tree, ARF, Oze bagging, etc. Stream data generators include Agrawal, Hyperplane, Led, Mixed, Random Tree, Waveform, etc. Skmultiflow supports both prequential and hold-out evaluations of models and all regularly used machine learning measures, such as accuracy, Kappa, and MSE.

River [172] is a Python library for data stream analytics and addressing concept drift through online ML models. All accessible learning models in River can be updated with a single incoming instance, allowing these methods to learn from data streams. It also includes a variety of streaming datasets, such as AirlinePassengers, Bananas, Bikes, ChickWeights, CreditCard, and Elec2. Additionally, it incorporates several well-known ML algorithms that support incremental learning, such as KNN, NB, and MLP.

Scikit-learn (Sklearn) [24] is a popular ML library written in Python. Although Sklearn is primarily used for batch learning problems, it also provides several incremental learning methods for online learning and streaming data analytics, including multinomial NB, stochastic gradient descent (SGD), MLP, incremental PCA, etc.

2.11 Case Study 1: HPO Method Comparison

To put theory into practice, several experiments have been conducted based on the theory in previous sections. This section provides the experiments of applying seven different HPO techniques to three common and representative ML algorithms on two benchmark datasets. In the first part of this section, the experimental setup and the main process of HPO are discussed. In the second part, the results of utilizing different HPO methods are compared and analyzed.

The sample code of the experiments has been published online at GitHub to illustrate the process of applying hyper-parameter optimization to ML models.

2.11.1 Experimental Setup

The experiments are conducted based on the following procedures.

Firstly, two standard benchmarking datasets provided by the sklearn library [24], namely, the Modified National Institute of Standards and Technology dataset (MNIST) and the Boston housing dataset, are selected as the benchmark datasets for HPO method evaluation on data analytics problems. MNIST is a hand-written digit recognition dataset used as a multi-classification problem, while the Boston housing dataset contains information about the price of houses in various places in the city of Boston and can be used as a regression dataset to predict the housing prices.

At the next stage, the ML models with their objective function need to be configured. In Section 2.5.7, all common ML models are divided into five categories based on their hyper-parameter types. Among those ML categories, ”one discrete hyper-parameter”, ”a few conditional hyper-parameters”, and ”a large hyper-parameter configuration space with multiple types of hyper-parameters” are the three most common cases. Thus, three ML algorithms, KNN, SVM, and RF, are selected as the target models to be optimized, since their hyper-parameter types represent the three most common HPO cases: KNN has one important hyper-parameter, the number of considered nearest neighbors for each sample; SVM has a few conditional hyper-parameters, like the kernel type and the penalty parameter C; RF has multiple hyper-parameters of different types, as discussed in Section 2.4.1. Moreover, KNN, SVM, and RF can all be applied to solve both classification and regression problems.

In the next step, the performance metrics and evaluation methods are configured. For each experiment on the selected two datasets, 3-fold cross validation is implemented to evaluate the involved HPO methods. The two most commonly-used performance metrics are used in our experiments. For classification models, accuracy is used as the classifier performance metric, which is the proportion of correctly classified data; while for regression models, the

Code is available at: https://github.com/LiYangHart/Hyperparameter-Optimization-of-Machine-Learning-Algorithms

Table 2.5: Configuration space for the hyper-parameters of tested ML models

ML Model	**Hyper-parameter**	**Type**	**Search Space**
RF Classifier	n_estimators	Discrete	[10,100]
	max_depth	Discrete	[5,50]
	min_samples_split	Discrete	[2,11]
	min_samples_leaf	Discrete	[1,11]
	criterion	Categorical	[’gini’, ’entropy’]
	max_features	Discrete	[1,64]
SVM Classifier	C	Continuous	[0.1,50]
	kernel	Categorical	[’linear’, ’poly’, ’rbf’, ’sigmoid’]
KNN Classifier	n_neighbors	Discrete	[1,20]
RF Regressor	n_estimators	Discrete	[10,100]
	max_depth	Discrete	[5,50]
	min_samples_split	Discrete	[2,11]
	min_samples_leaf	Discrete	[1,11]
	criterion	Categorical	[’mse’, ’mae’]
	max_features	Discrete	[1,13]
SVM Regressor	C	Continuous	[0.1,50]
	kernel	Categorical	[’linear’, ’poly’, ’rbf’, ’sigmoid’]
	epsilon	Continuous	[0.001,1]
KNN Regressor	n_neighbors	Discrete	[1,20]

mean squared error (MSE) is used as the regressor performance metric, which measures the average squared difference between the predicted values and the actual values. Additionally, the computational time (CT) , the total time needed to complete a HPO process with 3-fold cross-validation, is also used as the model efficiency metric [48]. In each experiment, the optimized ML model architecture that has the highest accuracy or the lowest MSE and the optimal hyper-parameter configuration will be returned.

After that, to fairly compare different optimization algorithms and frameworks, certain constraints should be satisfied. Firstly, we compare different HPO methods using the same hyper-parameter configuration space. For KNN, the only hyper-parameter to be optimized, ’n_neighbors’, is set to be in the same range of 1 to 20 for each optimization method evaluation. The hyper-parameters of SVM and RF models for classification and regression problems are also set to be in the same configuration space for each type of problem. The specifics of the configuration space for ML models are shown in Table 2.5. The selected hyper-parameters and their search space are determined based on the concepts in Section 2.4, domain knowledge, and manual testings [79]. The hyper-parameter types of each ML algorithm are also summarized in Table 2.5.

On the other hand, to fairly compare the performance metrics of optimization techniques, the maximum number of iterations for all HPO methods is set to 50 for RF and SVM model optimizations, and 10 for KNN model optimization based on manual testings and domain knowledge. Moreover, to avoid the impacts of randomness, all experiments are repeated ten times with different random seeds, and results are averaged for regression problems or given the majority vote for classification problems.

In Section 2.5, more than ten HPO methods are introduced. In our experiments, seven representative HPO approaches are selected for performance comparison, including GS, RS, BO-GP, BO-TPE, Hyperband, GA, and PSO.

All experiments were conducted using Python 3.5 on a machine with 6 Core i7-8700 processor and 16 gigabytes (GB) of memory. The involved ML and HPO algorithms are evaluated using multiple open-source Python libraries and frameworks introduced in Section 2.10, including Sklearn [24], Skopt [169], Hyperopt [166], Optunity [84], Hyperband [100], and TPOT [118].

2.11.2 Performance Comparison

The experiments of applying seven different HPO methods to ML models are summarized in Tables 2.6 to 2.11. Tables 2.6 to 2.8 provide the performance of each optimization algorithm when applied to RF, SVM, and KNN classifiers evaluated on the MNIST dataset after a complete optimization process; while Tables 2.9 to 2.11 demonstrate the performance of each HPO method when applied to RF, SVM, and KNN regressors evaluated on the Boston-housing dataset. In the first step, each ML model with its default hyper-parameter configuration is trained and evaluated as the baseline model. After that, each HPO algorithm is implemented on the ML models to evaluate and compare their accuracies for classification problems, or MSEs for regression problems, as well as their computational time (CT).

From Tables 2.6 to 2.11, we can see that using the default HP configurations do not yield the best model performance in our experiments, which emphasizes the importance of utilizing HPO methods. GS and RS can be seen as baseline models for HPO problems. From the results in Tables 2.6 to 2.11, it is shown that the computational time of GS is often much higher than

Table 2.6: Performance evaluation of applying HPO methods to the RF classifier on the MNIST dataset

Optimization Algorithm	**Accuracy** (%)	**CT** (s)
Default HPs	90.65	0.09
GS	93.32	48.62
RS	93.38	16.73
BO-GP	93.38	20.60
BO-TPE	93.88	12.58
Hyperband	93.38	8.89
GA	93.83	19.19
PSO	93.73	12.43

Table 2.7: Performance evaluation of applying HPO methods to the SVM classifier on the MNIST dataset

Optimization Algorithm	**Accuracy** (%)	**CT** (s)
Default HPs	97.05	0.29
GS	97.44	32.90
RS	97.35	12.48
BO-GP	97.50	17.56
BO-TPE	97.44	3.02
Hyperband	97.44	11.37
GA	97.44	16.89
PSO	97.44	8.33

Table 2.8: Performance evaluation of applying HPO methods to the KNN classifier on the MNIST dataset

Optimization Algorithm	**Accuracy** (%)	**CT** (s)
Default HPs	96.27	0.24
GS	96.22	7.86
RS	96.33	6.44
BO-GP	96.83	1.12
BO-TPE	96.83	2.33
Hyperband	96.22	4.54
GA	96.83	2.34
PSO	96.83	1.73

other optimization methods. With the same search space size, RS is faster than GS, but both of them cannot guarantee to detect the near-optimal hyper-parameter configurations of ML models, especially for RF and SVM models which have a larger search space than KNN.

The performance of BO and multi-fidelity models is much better than GS and RS. The

Table 2.9: Performance evaluation of applying HPO methods to the RF regressor on the Boston-housing dataset

Optimization Algorithm	**MSE**	**CT** (s)
Default HPs	31.26	0.08
GS	29.02	4.64
RS	27.92	3.42
BO-GP	26.79	17.94
BO-TPE	25.42	1.53
Hyperband	26.14	2.56
GA	26.95	4.73
PSO	25.69	3.20

Table 2.10: Performance evaluation of applying HPO methods to the SVM regressor on the Boston-housing dataset

Optimization Algorithm	**MSE**	**CT** (s)
Default HPs	77.43	0.02
GS	67.07	1.33
RS	61.40	0.48
BO-GP	61.27	5.87
BO-TPE	59.40	0.33
Hyperband	73.44	0.32
GA	60.17	1.12
PSO	58.72	0.53

Table 2.11: Performance evaluation of applying HPO methods to the KNN regressor on the Boston-housing dataset

Optimization Algorithm	**MSE**	**CT** (s)
Default HPs	81.48	0.004
GS	81.53	0.12
RS	80.77	0.11
BO-GP	80.77	0.49
BO-TPE	80.83	0.08
Hyperband	80.87	0.10
GA	80.77	0.33
PSO	80.74	0.19

computation time of BO-GP is often higher than other HPO methods due to its cubic time complexity, but it can obtain better performance metrics for ML models with small-size continuous hyper-parameter space, like KNN. Conversely, hyperband is often not able to obtain the highest accuracy or the lowest MSE among the optimization methods, but their computational time is low because it works on the small-sized subsets. The performance of BO-TPE is often better than others, since they can detect the optimal or near-optimal hyper-parameter configurations within a short computational time.

For metaheuristics methods, GA and PSO, their accuracies are often higher than other HPO methods for classification problems, and their MSEs are often lower than other optimization techniques. However, their computational time is often higher than BO-TPE and multi-fidelity models, especially for GA, which does not support parallel executions.

To summarize, it is simple to implement GS and RS, but they often cannot detect the optimal hyper-parameter configurations or cost much computational time. BO-GP and GA also cost more computational time than many other HPO methods, but BO-GP works well on small configuration space, while GA is effective for large configuration space. Hyperband's computational time is low, but it cannot guarantee to detect the global optimums. For ML models with large configuration space, BO-TPE and PSO often work well.

2.12 Case Study 2: Complete AutoML Pipeline

With the introduction of IoT data analytics and AutoML techniques, a case study is presented in this section to illustrate the capabilities and advantages of adopting AutoML techniques. A comprehensive AutoML pipeline is used in this case study to solve IoT anomaly detection problems.

This section provides the experimental results of applying the complete AutoML pipeline to an IoT anomaly detection use case using real-world IoT datasets. The first subsection introduces the use case. In the second part of this section, the experimental setup of the AutoML pipeline is described. In the last part, the results of offline IoT data analytics using traditional ML algorithms and dynamic IoT data analytics utilizing online adaptive algorithms are presented and analyzed.

2.12.1 Use Case

With the introduction of IoT data analytics and AutoML techniques, a case study is presented in this section to illustrate the capabilities and advantages of AutoML. A comprehensive AutoML pipeline is used in this case study to solve IoT anomaly detection problems.

This section provides the experimental results of applying the complete AutoML pipeline to an IoT anomaly detection use case using real-world IoT datasets. The first subsection discusses the use case. In the second part of this section, the experimental setup of the AutoML pipeline is described. In the last part, the results of offline IoT data analytics using traditional ML algorithms and dynamic IoT data analytics utilizing online adaptive algorithms are presented and analyzed.

2.12.2 Use Case

With the rapid development of IoT systems, numerous cyber-threats have extended from the Internet to people's everyday devices. Current IoT systems are vulnerable to most existing cyber-threats, due to the limited IoT device capability, gigantic scale, and vulnerable environments [173]. Due to the paucity of IoT security mechanisms capable of dealing with IoT threats, it is critical for IoT system protection to develop advanced approaches for detecting and identifying abnormal IoT devices and events. Thus, IoT anomaly detection has become an important use case in recent IoT systems for detecting compromised IoT devices and malicious IoT attacks [174].

For the purpose of enhancing IoT security, supervised ML algorithms can be used as effective mechanisms to distinguish malicious attack traffic from normal traffic. As discussed in Section 2.2.3, IoT anomaly detection problems can be classified into batch learning problems and online learning problems based on whether their environment is static or dynamic. In static IoT environments, traditional ML algorithms can be used to construct a conventional AutoML pipeline for static IoT data analytics. In dynamic IoT environments, online learning techniques can be used to construct a drift-adaptive AutoML pipeline for IoT streaming data analytics.

Two public IoT anomaly detection datasets are used in this work to evaluate the proposed AutoML pipeline. The first dataset is the IoTID20 dataset proposed in [175]. This dataset was created by using normal and attack virtual machines as network platforms, simulating IoT services with the node-red tool, and extracting features with the ISCXFlow meter program. A typical smart home environment was established for generating this dataset using five IoT devices or services: a smart fridge, a smart thermostat, motion-activated lights, a weather station, and a remotely-activated garage door. Thus, the traffic data samples of normal and abnormal IoT devices are collected in Pcap files.

The second dataset utilized in this chapter is the CICIDS2017 dataset [176]. It is generated by the Canadian Institute of Cybersecurity (CIC) and has the most updated network threats. The CICIDS2017 dataset is close to real-world network data since it has a large amount of network traffic data, a variety of network features (80), various types of attacks (14), and highly imbalanced classes.

The proposed AutoML pipeline is evaluated using a reduced IoTID20 dataset with 62,578 entries and a reduced NSL-KDD dataset with 28,307 records for the purpose of this work.

2.12.3 Experimental Setup

The studies use a comprehensive AutoML pipeline to solve the IoT anomaly detection problem, including the AutoDP, AutoFE, automated model selection, and HPO procedures. The specifications of each procedure in the AutoML pipeline are presented in Table 2.12.

AutoDP involves automated encoding, imputation, normalization, and balancing procedures. The automated encoding procedure identifies and converts string features to numerical features to make the data more understandable for ML models. The automated imputation procedure includes detecting missing values and imputing missing values using the mean imputation method introduced in Section 2.6.3. The automated normalization process automatically chooses an appropriate normalization method from Z-score and min-max normalization methods based on their performance in anomaly detection.

As the CICIDS2017 and IoTID20 datasets are both highly-imbalanced datasets, with an abnormal/normal ratio of 19%/81% and 6%/94%, respectively, an automated data balancing technique is also implemented in the proposed AutoML pipeline to balance the datasets. The system will evaluate whether the incoming dataset is imbalanced (the abnormal/normal ratio is smaller than a threshold (*e.g.,* 50%)); and if it is, the SMOTE technique introduced in Section 2.6.4 will be automatically implemented to synthesize new samples for the minority class to balance the data. The quality of the datasets is significantly enhanced after the AutoDP

Table 2.12: The specifications of the proposed AutoML pipeline.

Category	Procedure	Method	Aim/Operation
AutoDP	Encoding	Label Encoding	Identify and transform string features into numerical features to make the data more readable by ML models
	Imputation	Mean Imputation	Detect and impute missing values to improve data quality
	Normalization	Z-Score or Min-Max Normalization	Normalize the range of features to a similar scale to improve data quality
	Balancing	SMOTE	Generate minority class samples to solve class-imbalance and improve data quality
AutoFE	Feature Selection	IG	Remove irrelevant features to improve model efficiency
		Pearson Correlation	Remove redundant features to improve model efficiency and accuracy
Automated Model Learning	Model Selection	NB	Select the best-performing model among five common ML models by evaluating their learning performance
		MLP	
		KNN	
		RF	
		LightGBM	
	Hyperparameter Optimization	BO-TPE	Tune the hyperparameters of the learning models to obtain the optimized models
Automated Model Updating	Adaptive Model Selection	HT	Select the best-performing model among four online adaptive models to adapt to dynamic data streams with concept drift issues
		EFDT	
		ARF	
		SRP	

procedures.

As both datasets used in the experiments have a large number of features totaling more than 80, the AutoFE technique in the proposed AutoML pipeline focuses primarily on feature selection in order to obtain a sanitized and optimal feature subset. In the first step of AutoFE, an IG-based method is used to remove irrelevant or unimportant features by measuring the importance of each feature. In the second step, a Pearson correlation-based method is used to remove redundant and noising features by calculating the correlation between different features. By removing unnecessary and redundant information using AutoFE, the learning model becomes more efficient and accurate at detecting anomalies.

Automated model selection is an essential procedure in the development of AutoML pipelines. For batch learning in static IoT environments, the learning model will be chosen from five conventional ML algorithms (NB, MLP, KNN, RF, and LightGBM); while for online learning in dynamic IoT environments, the learning model will be chosen from four drift-adaptive online

Table 2.13: The HPO configuration of well-performing learning models.

Model	Hyperparameter Name	Configuration Space	Optimal Value on CICIDS2017	Optimal Value on IoTID20
LightGBM	n_estimators	[50,500]	360	440
	max_depth	[5,50]	36	38
	learning_rate	(0, 1)	0.957	0.456
	num_leaves	[100,2000]	1100	1200
	min_child_samples	[10,50]	50	25
RF	n_estimators	[50,500]	460	220
	max_depth	[5,50]	26	14
	min_samples_split	[2,11]	8	2
	min_samples_leaf	[1,11]	1	4
	criterion	['gini', 'entropy']	'entropy'	'entropy'
ARF	n_models	[3, 20]	18	15
	drift_detector	['ADWIN', 'DDM']	'DDM'	'DDM'
SRP	n_models	[3, 20]	14	10
	drift_detector	['ADWIN', 'DDM']	'DDM'	'DDM'

learning algorithms (HT, EFDT, ARF, and SRP), as it needs to be updated automatically based on data distribution changes (concept drift). After evaluating the performance of each learning model based on accuracy and F1-scores, the best-performing and the second best-performing models using default hyperparameters will be selected for further evaluations using HPO. By selecting not only the best-performing model but also the second-best-performing model, the probability of missing the real optimal model can be decreased.

After selecting the top-two learning models, their hyperparameters are tuned by the HPO technique to obtain the two optimized models and then select the final optimal model. As shown in Tables 2.13 - 2.17, the two best-performing batch learning algorithms on both datasets are RF and LightGBM, while the two best-performing online learning methods are ARF and SRP. Thus, the hyperparameters of these four algorithms are optimized. Table 2.13 illustrates the search space and the detect optimal values for the hyperparameters of these learning algorithms. Continuous hyperparameters are assigned a search range, while categorical hyperparameters are assigned all possible values/choices. BO-TPE is used to optimize learning models because it works well with a variety of different hyperparameters and has a low time complexity of $O(nlogn)$ [76].

The evaluation method for the learning models is determined by the tasks and environments. For offline learning in static environments, 5-fold cross-validation is used in the experiments since it can help develop a generic and robust learning model. For online learning in dynamic environments, prequential evaluation introduced in Section 2.9.2 is used in the experiments to evaluate the long-term learning performance of the drift-adaptive models on IoT time-series data.

Lastly, since the used datasets are highly imbalanced, the F1-score is chosen as the primary performance metric for evaluating the proposed AutoML pipeline. This is because F1-score considers both precision and recall metrics to give a fair view of anomaly detection results and minimize bias [125]. To ensure comprehensiveness of the evaluation results, the performance of each learning method is presented using four different metrics (accuracy, precision, recall, and F1-score), as shown in Tables 2.14 - 2.17. Additionally, the learning time of each model, including model training and testing, is also presented in Tables 2.14 - 2.17 to allow for a comparison of model efficiency.

The experiments were conducted on a machine with an i7-8700 processor and 16 GB of memory, representing an IoT server machine that supports large IoT data analytics. The techniques and methods utilized in the studies are implemented using the Python packages: Auto-Sklearn [113], Hyperopt [166], Skmultiflow [154], and River [155].

2.12.4 Experimental Results and Analysis

In this work, two series of experiments were conducted to validate the effectiveness of the AutoML framework. The first series of experiments were conducted to assess an offline AutoML pipeline in static IoT environments, while the second series of experiments were conducted to evaluate an online AutoML pipeline in dynamic IoT environments. The primary difference between the offline and online AutoML pipelines is the learning models used in the framework (traditional ML models versus adaptive online models).

To assess the AutoML framework's performance, we evaluated the accuracy, precision, recall, F1-score, and model learning time when using AutoML versus when not using AutoML in both offline and online learning tests. The experimental results for offline learning on the CICIDS2017 and IoTID20 datasets are shown in Tables 2.14 and 2.15, while the experimental

Table 2.14: The experimental results of offline learning on the CICIDS2017 dataset using 5-fold cross-validation.

AutoML Procedures	Learning Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Model Learning Time (s)
No	NB	72.545	37.837	62.253	47.896	0.2
	MLP	88.536	94.277	43.701	58.830	63.5
	KNN	97.238	92.081	93.782	92.923	9.1
	RF	**99.703**	**99.577**	**98.830**	**99.248**	**17.2**
	LightGBM	**99.816**	**99.543**	**99.506**	**99.525**	**1.4**
AutoDP & AutoFE	NB	73.316	39.690	73.080	51.435	0.1
	MLP	85.968	92.069	26.563	44.831	55.2
	KNN	97.058	92.024	92.831	92.423	9.1
	RF	**99.735**	**99.632**	**99.012**	**99.294**	**13.3**
	LightGBM	**99.844**	**99.616**	**99.579**	**99.598**	**0.9**
All	RF	99.760	99.578	99.141	99.368	62.3
	LightGBM	**99.866**	**99.670**	**99.634**	**99.653**	**1.0**

results for online learning on the CICIDS2017 and IoTID20 datasets are presented in Tables 2.16 and 2.17.

Specifically, in each Table, three different sets of results are shown to demonstrate the performance of the proposed AutoML pipeline. The first set of results compares the performance of original ML algorithms with default hyperparameter configurations (without AutoML) as baseline models for comparison purposes. The second set of results shows the performance of ML algorithms after implementing the proposed AutoDP & AutoFE procedures to illustrate the impact of data quality improvement by using AutoML. The third set of results presents the performance of a complete AutoML pipeline, which comprises AutoDP, AutoFE, automated model selection of the top-2 ML algorithms, and HPO. The proposed AutoML pipeline starts by implementing AutoDP and AutoFE, and then automatically selects the two best-performing learning models based on their F1-scores shown in the second set of results as the automated model selection procedure. After that, the hyperparameters of the two selected models are optimized to obtain a final optimal model with the best F1-score, as shown in the third set of results. The well-performing configurations in each set experiment are highlighted with boldface in Tables 2.14 - 2.17.

Table 2.14 summarizes the experimental results from the first series of experiments on offline learning using the CICIDS2017 dataset. For original ML models without using AutoML,

Table 2.15: The experimental results of offline learning on the IoTID20 dataset using 5-fold cross-validation.

AutoML Procedures	**Learning Algorithm**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Model Learning Time (s)**
No	NB	89.603	95.886	92.882	94.359	0.5
	MLP	99.202	96.800	98.319	97.742	125.7
	KNN	97.445	98.258	99.027	98.641	43.3
	RF	**99.920**	**99.913**	**100.0**	**99.953**	**25.5**
	LightGBM	**99.984**	**99.985**	**99.998**	**99.991**	**2.5**
AutoDP & AutoFE	NB	93.628	93.629	99.998	96.709	0.2
	MLP	95.009	95.099	98.943	97.781	113.6
	KNN	97.865	98.534	99.196	98.864	44.3
	RF	**99.976**	**99.973**	**100.0**	**99.989**	**16.3**
	LightGBM	**99.986**	**99.986**	**99.998**	**99.992**	**1.2**
All	RF	99.984	99.980	100.0	99.991	29.5
	LightGBM	**99.992**	**99.993**	**99.998**	**99.996**	**2.3**

five ML models (NB, MLP, KNN, RF, and LightGBM) demonstrate largely different performances. The F1-scores of NB and MLP models are at a low level (47.896% and 58.830%), because they are simple models and under-fitting on the complex CICIDS2017 dataset, as discussed in Table 2.1. RF and LightGBM models achieve high F1-scores of 99.248% and 99.525% due to their strong capacity to process complex and imbalanced datasets. After implementing the proposed AutoDP and AutoFE procedures, the RF and LightGBM models are still the two best-performing models. Their F1-scores are more than 6% higher than those of the other three compared ML models. Compared with the ML models without AutoDP and AutoFE, the F1-scores of the RF and LightGBM models have improved from 99.248% to 99.294% and from 99.525% to 99.598%, respectively. This is because the data quality has been improved by using SMOTE to balance the dataset and using FS methods to remove noisy features. Additionally, the learning time for RF and LightGBM has been reduced from 17.2s to 13.3s and from 1.4s to 0.9s, respectively. This is because the number of features of the CICIDS2017 dataset has been reduced from 80 to 19 after implementing the AutoFE technique. Furthermore, after implementing the HPO procedure to optimize the RF and LightGBM models to complete the entire AutoML pipeline, the performance of the learning models has been further improved, and the optimal LightGBM model with the highest F1-score of 99.653% is returned as the final model.

Table 2.16: The experimental results of online learning on the CICIDS2017 dataset using prequential evaluation.

AutoML Procedures	Learning Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Model Learning Time (s)
No	Offline LightGBM	88.033	90.538	36.794	50.510	7.4
	HT [146]	88.676	95.266	43.451	59.681	17.1
	EFDT [149]	95.132	86.438	88.674	87.541	17.4
	ARF [158]	**96.228**	**92.784**	**87.228**	**89.920**	**30.8**
	SRP [159]	**95.772**	**92.204**	**85.292**	**88.614**	**231.7**
AutoDP & AutoFE	Offline LightGBM	85.023	76.967	18.323	29.599	6.2
	HT	88.496	76.749	57.894	66.001	7.7
	EFDT	94.181	84.880	84.966	84.923	7.9
	ARF	**94.912**	**89.045**	**83.948**	**86.421**	**26.6**
	SRP	**94.547**	**90.314**	**80.342**	**85.037**	**101.8**
All	ARF	98.593	96.259	96.455	96.357	29.3
	SRP	**98.990**	**97.801**	**96.944**	**97.371**	**139.4**

Similarly, as shown in Table 2.15, the RF and LightGBM models outperform the other three compared ML models on the IoTID20 dataset due to their ability to analyze complex and imbalanced IoT anomaly detection data. Their F1-scores have slightly improved from 99.953% to 99.989% and 99.991% to 99.992% after implementing AutoDP and AutoFE, as the SMOTE method has been implemented to balance the dataset and the FS methods have been implemented to remove irrelevant and noisy features. As the number of features has been reduced from 83 to 31, the learning time has also been reduced for each ML model. After conducting the HPO procedures on the two best-performing models, RF and LightGBM, the optimized LightGBM model achieves the highest F1-score of 99.996% on the IoTID20 dataset and is selected as the final optimal model.

To summarize, implementing the AutoML pipeline can obtain a better offline learning model with 0.128% and 0.005% F1-score improvement as well as 28.6% and 8.0% reduction in learning time, when compared to the best performing learning model obtained without AutoML on the CICIDS2017 and IoTID20 datasets, respectively.

In the second series of experiments for online learning, the results on the CICIDS2017 and IoTID20 datasets are shown in Tables 2.16 and 2.17, respectively. To justify the necessity of online adaptive learning, the best-performing static ML model in the offline learning experi-

Table 2.17: The experimental results of online learning on the IoTID20 dataset using prequential evaluation.

AutoML Procedures	**Learning Algorithm**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Model Learning Time (s)**
No	Offline LightGBM	98.500	98.793	99.618	99.204	11.6
	HT [146]	98.220	98.981	99.120	99.051	37.2
	EFDT [149]	**99.478**	**99.571**	**99.873**	**99.722**	**37.3**
	ARF [158]	98.195	98.434	99.659	99.043	65.3
	SRP [159]	**99.208**	**99.367**	**99.790**	**99.578**	**492.1**
AutoDP & AutoFE	Offline LightGBM	99.037	99.005	99.983	99.492	9.4
	HT	98.922	99.092	99.763	99.426	12.1
	EFDT	99.280	99.359	99.877	99.617	12.2
	ARF	**99.501**	**99.541**	**99.928**	**99.734**	**51.1**
	SRP	**99.494**	**99.539**	**99.922**	**99.730**	**159.2**
All	ARF	99.664	99.658	99.985	99.821	58.5
	SRP	**99.705**	**99.726**	**99.960**	**99.843**	**199.7**

ments, offline LightGBM, is also evaluated for comparison purposes. As shown in Table 2.16, the offline LightGBM model shows the worst F1-scores of 50.510% among the five evaluated models (offline LightGBM, HT, EFDT, ARF, and SRP) on the CICIDS2017 dataset. This is because many new or zero-day attacks were launched in the creation process of the CICIDS2017 dataset, causing several concept drift points. As offline LightGBM cannot adapt to concept drift and can only detect existing types of attacks, its performance has been gradually degrading over time. On the other hand, the four online learning methods (HT, EFDT, ARF, and SRP), especially ARF and SRP, can adapt to concept drift and maintain high performance. Among the four online learning algorithms, ARF and SRP are the two best-performing learning models, with F1-scores of 86.421% and 85.037% after implementing the AutoDP and AutoFE procedures. This is because they are strong online ensemble models with high drift adaptability, as discussed in Section 2.8.4. The two incremental learning methods, HT and EFDT, are not as robust as ensemble models due to their relatively low model complexity.

Compared with the learning models without using AutoDP and AutoFE, although F1-scores of the learning models with AutoDP and AutoFE are slightly lower, their model learning time has largely reduced from 30.8s to 26.6s and from 231.7s to 101.8s, respectively. The F1-scores are slightly lower because the number of features of the CICIDS2017 dataset has been largely reduced from 80 to 19 after implementing the AutoFE method, which eliminates many less important features. As online learning usually starts with a small number of samples, the features have a more significant impact on online learning performance than offline learning. Although removing less important features may ignore certain data patterns and slightly reduce the learning performance, it can largely reduce the model learning time as the complexity and size of the dataset have been significantly reduced. To achieve real-time online learning in IoT data analytics, it is crucial to take learning speed into account. Moreover, after implementing the HPO procedure to complete the entire AutoML pipeline, the F1-scores of the ARF and SRP models have significantly improved to 96.357% and 97.371%, respectively. Thus, the proposed AutoML pipeline can return the optimal SRP model with the highest F1-score of 97.371%. This justifies that the overall AutoML procedures can still improve both model learning effectiveness and efficiency.

For the IoTID20 dataset, as shown in Table 2.17, the ARF and SRP models have also achieved higher F1-scores than the other two learning models after executing the AutoDP and AutoFE procedures, of 99.734% and 99.730%, respectively, although the best performing original learning model without AutoML is EFDT (99.722%). This shows that different online learning models will perform differently in specific IoT data analytics tasks. Additionally, the offline LightGBM still achieves a relatively low F1-score of 99.204% on the IoTID20 dataset, which is much lower than the F1-scores of most online learning methods.

After implementing AutoDP and AutoFE, the F1-scores of ARF and SRP have increased by 0.691% and 0.152%, while their model learning time has reduced by 21.8% and 67.7%, respectively. This is because the dataset remains balanced during the learning process after implementing the SMOTE method, and the feature size has been largely reduced from 83 to 31 after implementing the IG and Pearson-based FS methods. Finally, after implementing the HPO procedure, the optimal SRP model with the highest F1-score of 99.843% can be returned. Therefore, implementing the AutoML pipeline can obtain a better online learning model with 8.757% and 0.265% F1-score improvement as well as 39.8% and 59.4% learning time reduction than the same learning model obtained without AutoML on the CICIDS2017 and IoTID20 datasets, respectively.

In conclusion, the proposed AutoML pipeline enables us to obtain an optimal learning model with high effectiveness and efficiency for IoT anomaly detection tasks in both offline and online environments using the IoTID20 and CICIDS2017 datasets. Furthermore, the experimental results in Tables 2.14 and 2.17 have justified the following assumptions and theoretical analysis in previous sections:

1. Different ML methods have different performances in specific tasks, as demonstrated by the performance of the five offline ML models and four online learning models in the experimental results. This supports the necessity of selecting appropriate models.
2. Hyperparameter tuning and optimization have a direct impact on the model performance, as shown in the third set of results in each of Tables 2.14 - 2.17. The performance of learning models has been improved significantly by implementing the HPO method ("All" versus "AutoDP & AutoFE"). This supports the necessity of hyperparameter optimization or automated model tuning.
3. Data pre-processing and feature engineering methods affect learning performance. The performance of most learning models has been improved significantly by implementing AutoDP and AutoFE ("AutoDP & AutoFE" versus "No"). This supports the necessity of AutoDP and AutoFE.
4. Concept drift issues will cause model performance degradation, but automated model updating and concept drift adaptation methods can address model performance degradation. As shown in Tables 2.16 and 2.17, the best-performing offline models (offline LightGBM) still perform the worst when compared to other online adaptive learning models due to the occurrence of concept drift. This supports the necessity of automated model updating and concept drift adaptation.

2.13 Open Challenges and Research Directions

To effectively apply AutoML methods to IoT streaming data analytics problems, many challenges need to be addressed. In this Section, we discuss the open challenges and research directions in this domain. These challenges are classified into three major categories: IoT data analytics challenges, AutoML application challenges, and concept drift challenges, as summarized in Table 2.18.

Table 2.18: The challenges and research directions of applying AutoML to IoT data analytics

Category	Challenge	Brief Description
IoT Data Analytics Challenges	IoT Data Quality	Data quality has a direct effect on data analytics performance. However, IoT data is often collected from different data sources, it is often challenging to ensure data quality.
	IoT Data Privacy	The collection process of IoT data streams often faces privacy issues, as they are often from different IoT devices/systems. Federated Learning (FL) techniques can be used to protect data privacy.
	IoT Data Analytics Speed	IoT data analytics models often require fast processing speed to achieve real-time processing in IoT systems.
AutoML Challenges	Automated Model Updating	The automated model updating process is often ignored in many AutoML systems. This step is important in real-world IoT data analytics applications, as IoT data is often dynamic streaming data.
	Data Pre-Processing and Feature Engineering	Most AutoML systems only focus on automated model selection and HPO procedures. Thus, data pre-processing and feature engineering need more attention, as they also have a significant effect on model performance.
	Large Scale AutoML	It is challenging to apply AutoML models on large-scale data, such as ImageNet, as the learning models often need to be trained many times to identify the optimal solution.
	Explainability	AutoML solutions are often black boxes, so their explainability needs more research.
	Transfer Learning in AutoML	High complexity is a common issue in AutoML systems. Transfer learning techniques can be used to save model learning time.
	Benchmarking and Comparability	A benchmark should be agreed upon by the community for a fair comparison of different AutoML techniques.
Concept Drift Challenges	Unsupervised Learning	More research should be conducted on unsupervised or semi-supervised drift detection and adaptation, as most existing methods are developed for supervised learning.
	Drift Analysis	A comprehensive analysis should be conducted on the detected drifts, such as the timing and severity of each drift.
	ML Model Integration	Appropriate drift methods should be selected and integrated with specific ML algorithms to develop effective automated drift adaptation functionality.
	Accurate Drift Detection	Drift detection methods should be capable of accurately detecting different types of drift (*e.g.,* abrupt and gradual drift.)

2.13.1 IoT Data Analytics Challenges

Although data analytics contributes significantly to IoT applications, it is still in its early stages [177]. Numerous challenges must be addressed before IoT data can be properly used in IoT applications [178], such as the quality, privacy, and analytics speed of IoT data.

IoT Data Quality

Firstly, the quality of data has a direct effect on the performance of data analytics models. Thus, it is critical to have high-quality data [178]. However, as IoT data is often collected from different data sources and is highly variable, maintaining data quality is usually challenging. The high generation speed and volume of IoT data are also significant problems [177]. Effective data integration has also become a challenge for creating high-quality datasets from different IoT devices.

IoT Data Analytics Speed

Due to the massive amount of data generated by IoT devices, time constraints are a primary challenge of IoT data analytics. Many IoT applications have real-time requirements, such as autonomous vehicles and e-health systems [179]. In these applications, real-time feedback on environmental changes is required. However, many factors, like transmission delays and model learning time, have increased the reaction time of IoT systems. Therefore, it is essential for data analytics methods to achieve real-time analytics on large amounts of IoT data generated at high speeds [178]. Real-time analytics allows IoT devices to make real-time decisions and provide services. As described in Section 2.2.2, edge computing and collaborative computing techniques are promising solutions to achieve real-time analytics, but better architecture should be designed to balance data analytics efficiency and accuracy.

On the other hand, to avoid unfeasible IoT data analytics due to time or memory constraints, distributed ML is a promising solution that allocates the learning process over multiple workstations [180]. Unlike traditional data processing systems that collect data from multiple sources for central processing, in distributed ML, every IoT end device or edge server can store and process its own data in itself or only share data with trusted devices to avoid the leakage of private and sensitive data. Distributed ML can also improve data analytics speed by enabling parallel execution and reducing data transmission time. This is because the computational time of analyzing a large central database is much higher than dividing it into multiple sub-tasks that analyze data in parallel [180]. Thus, both data security and processing efficiency can be improved by using distributed ML techniques.

IoT Data Security and Privacy

With the advent of data analytics techniques for IoT data, data security has emerged as a critical concern [177]. As a comprehensive IoT dataset is often generated from different data sources, certain personal or sensitive business data may be derived during the data collection process [178]. Thus, it is crucial for IoT systems to solve data privacy issues.

Cybersecurity mechanisms, like data encryption and device authentication, can improve the privacy of IoT data. However, these techniques introduce additional overhead to IoT systems. On the other hand, distributed ML and Federated Learning (FL) techniques can be a potential solution for IoT data privacy [181]. By distributed ML approaches, the data is stored only in local IoT devices to make data unavailable to unauthorized devices, thus maintaining data security and privacy. FL techniques can train ML models without direct access to local data by exchanging model parameter values between edge and central servers [182]. Thus, by the employment of FL approaches, the privacy of IoT data can be protected without compromising learning performance.

2.13.2 AutoML Challenges

AutoML has made considerable strides in the previous decade in automating model construction and development, especially for supervised learning tasks. However, to be widely applied to real-world IoT applications, AutoML still faces many challenges [183].

Automated Model Updating

Despite the development of AutoML, most AutoML solutions are offline models designed for static datasets. However, many real-world applications, like IoT systems, face concept drift issues throughout the data analytics process, and most existing AutoML solutions only update models using new data samples [10]. Therefore, this chapter considers automated model updates by addressing concept drift issues in the AutoML pipeline, which is a novel contribution to AutoML applications. Considering automated model updates can help construct robust AutoML models that maintain effectiveness over time.

Data Pre-Processing and Feature Engineering

Although there are many existing AutoML solutions, the majority of them focus on automated model selection and HPO. Researchers have paid little attention to automated data pre-processing and feature engineering [183]. However, data pre-processing and feature engineering are critical components of the AutoML pipeline and have a direct influence on system performance. It is often challenging to generalize and automate the feature engineering process because it is very task- and dataset-dependent [82]. Appropriate feature engineering often requires specialized domain knowledge or a significant amount of effort. Therefore, automated feature engineering is a critical but challenging subject that needs further research.

Large Scale AutoML

Applying AutoML to large-scale data is still an unsolved issue. Due to the fact that AutoML pipelines often need a significant number of model trainings to identify the optimum final learner, the majority of AutoML solutions are developed on small datasets, with just a few capable of large-scale data learning. For instance, research on AutoML solutions for the ImageNet problem is currently rather limited, owing to the dataset's massive size [81].

Explainability

In general, AutoML solutions are black boxes that attempt to explore the space of possible models and discover the optimal solution. Despite AutoML's advances, the community has not explored the prospect of transparent AutoML systems. AutoML models should have mechanisms for explaining and understanding them, since this would considerably improve AutoML's accessibility.

Transfer Learning in AutoML

Due to the high complexity of existing AutoML solutions in terms of time and space, transfer learning methods can be utilized to increase AutoML's efficiency. This is because transfer learning enables the reduction of unnecessary model retrainings via the usage of existing models. While meta-learning processes are a subtype of transfer learning, transfer learning can also refer to the transfer of knowledge about the optimization process (*e.g.,* transferring information on the dynamics of the optimization process from task to task) [184].

Benchmarking and Comparability

As various AutoML systems have distinct benefits and drawbacks in different IoT applications, the community should agree on a set of common benchmarks that allow a fair comparison of different techniques [81]. Similarly, code sharing and processes that facilitate the replication of AutoML discoveries may have a substantial impact on the field's maturity.

2.14 Conclusion

Machine Learning (ML) and Deep Learning (DL) algorithms have achieved great success in data analytics tasks for IoT applications, such as intelligent transportation systems, smart homes, e-health, and IoT security. However, developing effective ML models for specific IoT tasks requires a high level of human expertise, which limits their applicability. Thus, Automated ML (AutoML) has become a promising solution for constructing ML models without or with minimal human intervention. In this chapter, we have comprehensively discussed the procedures of the standard AutoML pipeline, including automated data pre-processing, automated feature engineering, automated model selection, Hyper-Parameter Optimization (HPO), and automated model updating with concept drift adaptation. Moreover, we have explored the IoT data analytics tasks , as well as the ML and DL models that are often employed in IoT data analytics. Existing tools and libraries for implementing AutoML and IoT data analytics are also presented in this chapter. Additionally, a case study of IoT anomaly detection is conducted in this work to demonstrate the procedures of AutoML applications. Experimental results have shown the benefits of using AutoML frameworks in IoT data analytics problems. Finally, we discuss the open challenges and research directions related to the existing AutoML and IoT data analytics tasks.

Bibliography

[1] L. Yang and A. Shami, “On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[2] L. Yang and A. Shami, “IoT Data Analytics in Dynamic Environments: From An Automated Machine Learning Perspective”, *Engineering Applications of Artificial Intelligence*, 2022.

[3] Q. Yao et al., “Taking Human out of Learning Applications: A Survey on Automated Machine Learning,”, *arXiv*, pp. 1–26, 2018.

[4] X. He, K. Zhao, and X. Chu, “AutoML: A survey of the state-of-the-art,” *Knowledge-Based Syst.*, vol. 212, no. January, p. 106622, 2021.

[5] T. Yu and X. Wang, “Real-Time Data Analytics in Internet of Things Systems,” in *Handbook of Real-Time Computing*, Singapore: Springer Singapore, 2020, pp. 1–28.

[6] E. Adi, A. Anwar, Z. Baig, and S. Zeadally, “Machine Learning and Data Analytics for the IoT,” *Neural Comput. Appl.*, vol. 32, no. 20, pp. 16205–16233, Oct. 2020.

[7] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, “Deep Learning for IoT Big Data and Streaming Analytics: A Survey,” *IEEE Commun. Surv. Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[8] Cisco Annual Internet Report (2018–2023) White Paper, [online] Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[9] S. S. Goel, A. Goel, M. Kumar, and G. Moltó, “A review of Internet of Things: qualifying technologies and boundless horizon,” *J. Reliab. Intell. Environ.*, vol. 7, no. 1, pp. 23–33, 2021.

[10] L. Yang and A. Shami, “A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams,” *IEEE Internet Things Mag.*, vol. 4, no. 2, pp. 96–101, 2021.

[11] A. A. Cook, G. Mısırlı, and Z. Fan, “Anomaly Detection for IoT Time-Series Data: A Survey,” *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6481–6494, 2020.

[12] S. K. Sharma and X. Wang, “Live Data Analytics With Collaborative Edge and Cloud Processing in Wireless IoT Networks,” *IEEE Access*, vol. 5, pp. 4621–4635, 2017.

[13] C. Yin, S. Zhang, J. Wang, and N. N. Xiong, “Anomaly Detection Based on Convolutional Recurrent Autoencoder for IoT Time Series,” *IEEE Trans. Syst. Man, Cybern. Syst.*, pp. 1–11, 2020.

[14] A. L’Heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz, “Machine Learning with Big Data: Challenges and Approaches,” *IEEE Access*, vol. 5, pp. 7776–7797, 2017.

[15] B. I. F. Maciel, J. I. G. Hidalgo, and R. S. M. de Barros, “An Ultimately Simple Concept Drift Detector for Data Streams,” in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2021, pp. 625–630.

[16] A. Mustafa and M. Rahimi Azghadi, “Automated Machine Learning for Healthcare and Clinical Notes Analysis,” *Computers*, vol. 10, no. 2, 2021.

[17] S. Abreu, “Automated Architecture Design for Deep Neural Networks,” *arXiv*, 2019.

[18] K. Chauhan et al., “Automated Machine Learning: The New Wave of Machine Learning,” *2nd Int. Conf. Innov. Mech. Ind. Appl. ICIMIA 2020 - Conf. Proc.*, no. Icimia, pp. 205–212, 2020.

[19] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. Part F1288, pp. 847–855, 2013.

[20] R. Elshawi and S. Sakr, “Automated Machine Learning: Techniques and Frameworks,” in *Big Data Management and Analytics*, 2020, pp. 40–69.

[21] A. Moubayed, M. Injadat, A. Shami, and H. Lutfiyya, “DNS Typo-Squatting Domain Detection: A Data Analytics & Machine Learning Based Approach,” *2018 IEEE Glob. Commun. Conf.*, pp. 1–7, 2019.

[22] R. Caruana and A. Niculescu-Mizil, “An empirical comparison of supervised learning algorithms,” *ACM Int. Conf. Proceeding Ser.*, vol. 148, pp. 161–168, 2006.

[23] O. Kramer, Scikit-Learn, in *Machine Learning for Evolution Strategies*, Cham, Switzerland:Springer, vol. 20, 2016.

[24] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[25] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[26] F. Chollet, Keras, 2015. https://github.com/fchollet/keras.

[27] C. Gambella, B. Ghaddar, and J. Naoum-Sawaya, “Optimization Models for Machine Learning: A Survey,” arXiv, pp. 1–40, 2019.

[28] C. M. Bishop, ”Pattern Recognition and Machine Learning”, Berlin, Germany:Springer-Verlag, ISBN: 978-0-387-31073-2.

[29] A. E. Hoerl and R. W. Kennard, “Ridge Regression: Applications to Nonorthogonal Problems,” *Technometrics*, vol. 12, no. 1, pp. 69–82, 1970.

[30] L. E. Melkumova and S. Y. Shatskikh, “Comparing Ridge and LASSO estimators for data analysis,” *Procedia Eng.*, vol. 201, pp. 746–755, 2017.

[31] R. Tibshirani, “Regression Shrinkage and Selection Via the Lasso,” *J. R. Stat. Soc. Ser. B*, vol. 58, no. 1, pp. 267–288, 1996.

[32] D. W. Hosmer and S. Lemeshow, “Applied logistic regression,” *Appl. Logist. Regres.*, no. October, pp. 118–128, 2000.

[33] J. O. Ogutu, T. Schulz-Streeck, and H. P. Piepho, “Genomic selection using regularized linear regression models: ridge regression,” *BMC proceedings. BioMed Cent.*, vol. 6, no. Suppl 2, 2012.

[34] J. M. Keller and M. R. Gray, “A Fuzzy K-Nearest Neighbor Algorithm,” *IEEE Trans. Syst. Man Cybern.*, vol. SMC-15, no. 4, pp. 580–585, 1985.

[35] W. Zuo, D. Zhang, and K. Wang, “On kernel difference-weighted k-nearest neighbor classification,” *Pattern Anal. Appl.*, vol. 11, no. 3–4, pp. 247–257, 2008.

[36] H. Drucker, C. J. C. Surges, L. Kaufman, A. Smola, and V. Vapnik, “Support vector regression machines,” *Adv. Neural Inf. Process. Syst.*, vol. 1, pp. 155–161, 1997.

[37] L. Yang, R. Muresan, A. Al-Dweik, and L. J. Hadjileontiadis, “Image-Based Visibility Estimation Algorithm for Intelligent Transportation Systems,” *IEEE Access*, vol. 6, pp. 76728–76740, 2018.

[38] L. Yang, Comprehensive Visibility Indicator Algorithm for Adaptable Speed Limit Control in Intelligent Transportation Systems, M.A.Sc. thesis, University of Guelph, 2018.

[39] O. S. Soliman and A. S. Mahmoud, “A classification system for remote sensing satellite images using support vector machine with non-linear kernel functions,” *2012 8th Int. Conf. Informatics Syst. INFOS 2012*, p. BIO-181-BIO-187, 2012.

[40] I. Rish, “An empirical study of the naive Bayes classifier,” *IJCAI 2001 Work. Empir. methods Artif. Intell.*, vol. 22230, pp. 41–46, 2001.

[41] J. N. Sulzmann, J. Fürnkranz, and E. Hüllermeier, “On pairwise naive bayes classifiers,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4701 LNAI, pp. 371–381, 2007.

[42] C. Bustamante, L. Garrido, and R. Soto, “Comparing fuzzy Naive Bayes and Gaussian Naive Bayes for decision making in RoboCup 3D,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4293 LNAI, pp. 237–247, 2006.

[43] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes, "Multinomial naive bayes for text categorization revisited," *Lect. Notes Artif. Intell. (Subseries Lect. Notes Comput. Sci.*, vol. 3339, pp. 488–499, 2004.

[44] J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger, "The Poor Assumptions of Naive Bayes Classifiers," *Proc. Twent. Int. Conf. Mach. Learn. (ICML).*, no. 1973, 2003.

[45] V. Narayanan, I. Arora, and A. Bhatia, "Fast and accurate sentiment classification using an enhanced Naive Bayes model," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8206 LNCS, pp. 194–201, 2013.

[46] S. Rasoul and L. David, "A Survey of Decision Tree Classifier Methodology," *IEEE Trans. Syst. Man. Cybern.*, vol. 21, no. 3, pp. 660–674, 1991.

[47] D. M. Manias et al., "Machine Learning for Performance-Aware Virtual Network Function Placement," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[48] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, "Tree-Based Intelligent Intrusion Detection System in Internet of Vehicles," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[49] S. Sanders and C. Giraud-Carrier, "Informing the use of hyperparameter optimization through metalearning," *Proc. - IEEE Int. Conf. Data Mining, ICDM*, vol. 2017-Novem, no. 1, pp. 1051–1056, 2017.

[50] M. Injadat, F. Salo, A. B. Nassif, A. Essex, and A. Shami, "Bayesian Optimization with Machine Learning Algorithms Towards Anomaly Detection," 2*018 IEEE Glob. Commun. Conf.*, pp. 1–6, 2019.

[51] F. Salo, M. N. Injadat, A. Moubayed, A. B. Nassif, and A. Essex, "Clustering Enabled Classification using Ensemble Feature Selection for Intrusion Detection," *2019 Int. Conf. Comput. Netw. Commun. ICNC 2019*, pp. 276–281, 2019.

[52] K. Arjunan and C. N. Modi, "An enhanced intrusion detection framework for securing network layer of cloud computing," *ISEA Asia Secur. Priv. Conf. 2017, ISEASP 201*7, pp. 1–10, 2017.

[53] Y. Xia, C. Liu, Y. Y. Li, and N. Liu, "A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring," *Expert Syst. Appl.*, vol. 78, pp. 225–241, 2017.

[54] T. G. Dietterich, "Ensemble methods in machine learning," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 1857 LNCS, pp. 1–15, 2000.

[55] A. Moubayed, E. Aqeeli, and A. Shami, "Ensemble-based Feature Selection and Classification Model for DNS Typo-squatting Detection," in *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2020.

[56] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative Study of CNN and RNN for Natural Language Processing,", *arXiv*, 2017.

[57] A. Koutsoukas, K. J. Monaghan, X. Li, and J. Huan, "Deep-learning: Investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data," *J. Cheminform.*, vol. 9, no. 1, pp. 1–13, 2017.

[58] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," *IJCAI Int. Jt. Conf. Artif. Intell.*, vol. 2015-Janua, pp. 3460–3468, 2015.

[59] Y. Ozaki, M. Yano, and M. Onishi, "Effective hyperparameter optimization using Nelder-Mead method in deep learning," *IPSJ Trans. Comput. Vis. Appl.*, vol. 9, no. 1, 2017.

[60] F. C. Soon, H. Y. Khaw, J. H. Chuah, and J. Kanesan, "Hyper-parameters optimisation of deep CNN architecture for vehicle logo recognition," *IET Intell. Transp. Syst.*, vol. 12, no. 8, pp. 939–946, 2018.

[61] D. Han, Q. Liu, and W. Fan, "A new image classification method using CNN transfer learning and web data augmentation," *Expert Syst. Appl.*, vol. 95, pp. 43–56, 2018.

[62] C. Di Francescomarino et al., "Genetic algorithms for hyperparameter optimization in predictive business process monitoring," *Inf. Syst.*, vol. 74, pp. 67–83, 2018.

[63] A. Moubayed, M. Injadat, A. Shami, and H. Lutfiyya, "Student Engagement Level in e-Learning Environment: Clustering Using K-means," *Am. J. Distance Educ.*, vol. 34, no. 02, pp. 1–20, 2020.

[64] T.K. Moon, "The Expectatio Maximization Algorithm," *IEEE Signal Process. Mag.*, pp. 47–60, 1996.

[65] S. Brahim-Belhouari, A. Bermak, M. Shi, and P. C. H. Chan, "Fast and Robust gas identification system using an integrated gas sensor technology and Gaussian mixture models," *IEEE Sens. J.*, vol. 5, no. 6, pp. 1433–1444, 2005.

[66] K. Khan, S. U. Rehman, K. Aziz, S. Fong, S. Sarasvady, and A. Vishwa, "DBSCAN: Past, present and future," *5th Int. Conf. Appl. Digit. Inf. Web Technol. ICADIWT 2014, pp. 232–238*, 2014.

[67] H. Zhou, P. Wang, and H. Li, "Research on adaptive parameters determination in DBSCAN algorithm," *J. Inf. Comput. Sci.*, vol. 9, no. 7, pp. 1967–1973, 2012.

[68] J. Shlens, "A Tutorial on Principal Component Analysis,", *arXiv*, 2014.

[69] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, 2011.

[70] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A Survey of Optimization Methods from a Machine Learning Perspective,", arXiv, pp. 1–30.

[71] J. Bergstra et al., “Algorithms for Hyper-Parameter Optimization Algorithms for Hyper-Parameter Optimization,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 2546–2554.

[72] B. Shahriari, A. Bouchard-Côté, and N. de Freitas, “Unbounded Bayesian Optimization via Regularization,” in Proc. Artif. Intell. Statist., 2016, pp. 1168–1176.

[73] G. Luo, “A review of automatic selection methods for machine learning algorithms and hyper-parameter values,” *Netw. Model. Anal. Heal. Informatics Bioinforma.*, vol. 5, no. 1, pp. 1–16, 2016.

[74] G. I. Diaz, A. Fokoue-Nkoutche, G. Nannicini, and H. Samulowitz, “An effective algorithm for hyperparameter optimization of neural networks,” *IBM J. Res. Dev.*, vol. 61, no. 4, pp. 1–20, 2017.

[75] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor, “Particle Swarm Optimization for Hyper-Parameter Selection in Deep Neural Networks,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 481–488.

[76] R. E. Shawi, M. Maher, S. Sakr, ”Automated machine learning: State-of-the-art and open challenges”, *arXiv*, 2019.

[77] O. S. Steinholtz, “A Comparative Study of Black-box Optimization Algorithms for Tuning of Hyper-parameters in Deep Neural Networks,” M.S. thesis, Dept. Elect. Eng., Luleå Univ. Technol., 2018.

[78] I. Ilievski, T. Akhtar, J. Feng, and C. A. Shoemaker, “Efficient hyperparameter optimization of deep learning algorithms using deterministic RBF surrogates,” *31st AAAI Conf. Artif. Intell. AAAI 2017*, pp. 822–829, 2017.

[79] M. N. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, “Systematic ensemble model selection approach for educational data mining,” *Knowledge-Based Syst.*, vol. 200, p. 105992, 2020.

[80] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, “Multi-split Optimized Bagging Ensemble Model Selection for Multi-class Educational Data Mining,” *Springer’s Appl. Intell.*, 2020.

[81] F. Hutter, L. Kotthoff, and J. Vanschoren, ”Automated machine learning: methods, systems, challenges,” *Springer*, 2019.

[82] M. A. Zöller and M. F. Huber, “Benchmark and Survey of Automated Machine Learning Frameworks,” *J. Artif. Intell. Res.*, vol. 70, no. 1993, pp. 409–472, 2021.

[83] B. James and B. Yoshua, “Random Search for Hyper-Parameter Optimization,” *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, 2012.

[84] M. Claesen, J. Simm, D. Popovic, Y. Moreau, and B. De Moor, “Easy Hyperparameter Search Using Optunity,” *arXiv*, 2014.

[85] C. Witt, “Worst-Case and Average-Case Approximations by Simple Randomized Search Heuristics,” in *STACS 2005*, 2005, pp. 44–56.

[86] K. Eggensperger et al., “Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters,” *BayesOpt Work.*, pp. 1–5, 2013.

[87] Y. Bengio, “Gradient-based optimization of hyperparameters,” *Neural Comput.*, vol. 12, no. 8, pp. 1889–1900, 2000.

[88] H. H. Yang and S. I. Amari, “Complexity Issues in Natural Gradient Descent Method for Training Multilayer Perceptrons,” *Neural Comput.*, vol. 10, no. 8, pp. 2137–2157, 1998.

[89] D. Maclaurin, D. Duvenaud, and R. P. Adams, “Gradient-based Hyperparameter Optimization through Reversible Learning,” *arXiv*, 2015.

[90] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian Optimization of Machine Learning Algorithms,” in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, 2012, pp. 2951–2959.

[91] E. Hazan, A. Klivans, and Y. Yuan, “Hyperparameter Optimization: A Spectral Approach,” *arXiv*, 2017.

[92] N. Decastro-García, Á. L. Muñoz Castañeda, D. Escudero García, and M. V. Carriegos, “Effect of the Sampling of a Dataset in the Hyperparameter Optimization Phase over the Efficiency of a Machine Learning Algorithm,” *Complexity*, vol. 2019, 2019.

[93] M. Seeger, “Gaussian Processes for Machine Learning,” *Int. J. Neural Syst.*, vol. 14, pp. 69–109, 2004.

[94] I. Dewancker, M. McCourt, and S. Clark, “Bayesian Optimization Primer,”, arXiv pp. 1–4, 2015.

[95] J. Hensman, N. Fusi, and N. D. Lawrence, “Gaussian processes for big data,” *Uncertain. Artif. Intell. - Proc. 29th Conf. UAI 2013*, no. September 2013, pp. 282–290, 2013.

[96] M. Claesen and B. De Moor, “Hyperparameter Search in Machine Learning,” *arXiv*, pp. 1–5, 2015.

[97] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” *Proc. COMPSTAT 2010 - 19th Int. Conf. Comput. Stat. Keynote, Invit. Contrib. Pap.*, pp. 177–186, 2010.

[98] S. Zhang, J. Xu, E. Huang, and C. H. Chen, “A new optimal sampling rule for multi-fidelity optimization via ordinal transformation,” *IEEE Int. Conf. Autom. Sci. Eng.*, vol. 2016-Novem, pp. 670–674, 2016.

[99] Z. Karnin, T. Koren, and O. Somekh, “Almost optimal exploration in multi-armed bandits,” *30th Int. Conf. Mach. Learn. ICML 2013*, vol. 28, no. PART 3, pp. 2275–2283, 2013.

[100] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, “Hyperband: A novel bandit-based approach to hyperparameter optimization,” *J. Mach. Learn. Res.*, vol. 18, pp. 1–52, 2018.

[101] A. Gogna and A. Tayal, “Metaheuristics: Review and application,” *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 25, no. 4. Taylor & Francis, pp. 503–526, 2013.

[102] S. Lessmann, R. Stahlbock, and S. F. Crone, “Optimizing hyperparameters of support vector machines by genetic algorithms,” *Proc. 2005 Int. Conf. Artif. Intell. ICAI'05*, vol. 1, pp. 74–80, 2005.

[103] F. Itano, M. A. De Abreu De Sousa, and E. Del-Moral-Hernandez, “Extending MLP ANN hyper-parameters Optimization by using Genetic Algorithm,” *Proc. Int. Jt. Conf. Neural Networks*, vol. 2018-July, pp. 1–8, 2018.

[104] B. Kazimipour, X. Li, and A. K. Qin, “A Review of Population Initialization Techniques for Evolutionary Algorithms,” *2014 IEEE Congr. Evol. Comput.*, pp. 2585–2592, 2014.

[105] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, “A novel population initialization method for accelerating evolutionary algorithms,” *Comput. Math. with Appl.*, vol. 53, no. 10, pp. 1605–1614, 2007.

[106] F. G. Lobo, D. E. Goldberg, and M. Pelikan, “Time complexity of genetic algorithms on exponentially scaled problems,” *Proc. Genet. Evol. Comput. Conf.*, no. June, pp. 151–158, 2000.

[107] E. Engineering and M. St, “Parameter Selection in Particle Swarm Optimization Department of Electrical Engineering lndiana University Purdue University Indianapolis,” *Evol. Program. VII,* vol. 1447, pp. 591–600, 1998.

[108] X. Yan, F. He, and Y. Chen, “A Novel Hardware / Software Partitioning Method Based on Position Disturbed Particle Swarm Optimization with Invasive Weed Optimization,” *Journal of Computer Science and Technology*, vol. 32, no. 61472289, pp. 340–355, 2017.

[109] M. Y. Cheng, K. Y. Huang, and M. Hutomo, “Multiobjective Dynamic-Guiding PSO for Optimizing Work Shift Schedules,” *J. Constr. Eng. Manag.*, vol. 144, no. 9, pp. 1–7, 2018.

[110] H. Wang, Z. Wu, J. Wang, X. Dong, S. Yu, and G. Chen, “A new population initialization method based on space transformation search,” *5th Int. Conf. Nat. Comput. ICNC 2009*, vol. 5, no. 2, pp. 332–336, 2009.

[111] J. Giovanelli, B. Bilalli, and A. Abelló, “Effective data pre-processing for AutoML,” C*EUR Workshop Proc.*, vol. 2840, pp. 1–10, 2021.

[112] E. Jackson and R. Agrawal, “Performance Evaluation of Different Feature Encoding Schemes on Cybersecurity Logs,” in *2019 SoutheastCon*, 2019, pp. 1–9.

[113] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, “Auto-sklearn: Efficient and Robust Automated Machine Learning,” i*n Automated Machine Learning: Methods, Systems, Challenges, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Cham: Springer International Publishing*, 2019, pp. 113–134.

[114] A. Candel, V. Parmar, E. LeDell, and A. Arora, “Deep learning with H2O,” *H2O. ai Inc*, pp. 1–21, 2016.

[115] C.-F. Tsai and Y.-C. Chen, “The optimal combination of feature selection and data discretization: An empirical study,” *Inf. Sci. (Ny).*, vol. 505, pp. 282–293, 2019.

[116] Y. Ma, J. Jin, Q. Huang, and F. Dan, “Data Preprocessing of Agricultural IoT Based on Time Series Analysis,” in *Intelligent Computing Theories and Application*, 2018, pp. 219–230.

[117] E. Law, “Impyute Documentation,” 2017.

[118] R. S. Olson and J. H. Moore, “TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning,” in *Automated Machine Learning: Methods, Systems, Challenges, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Cham: Springer International Publishing*, pp. 151–160, 2019.

[119] F. Bießmann et al., “DataWig: Missing value imputation for tables,” *J. Mach. Learn. Res.*, vol. 20, pp. 1–6, 2019.

[120] Z. Chen et al., “Machine learning based mobile malware detection using highly imbalanced network traffic,” *Inf. Sci. (Ny).*, vol. 433–434, pp. 346–364, 2018.

[121] P. Kaur and A. Gosain, “Comparing the Behavior of Oversampling and Undersampling Approach of Class Imbalance Learning by Combining Class Imbalance Problem with Noise,” in *ICT Based Innovations*, 2018, pp. 23–30.

[122] Q. Kang, X. S. Chen, S. S. Li, and M. C. Zhou, “A Noise-Filtered Under-Sampling Scheme for Imbalanced Classification,” *IEEE Trans. Cybern.*, vol. 47, no. 12, pp. 4263–4274, 2017.

[123] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic minority over-sampling technique,” *J. Artif. Intell. Res.*, vol. 16, no. February 2017, pp. 321–357, 2002.

[124] X. Tan et al., “Wireless sensor networks intrusion detection based on SMOTE and the random forest algorithm,” *Sensors (Switzerland)*, vol. 19, no. 1, 2019.

[125] L. Yang, A. Moubayed, and A. Shami, “MTH-IDS: A Multi-Tiered Hybrid Intrusion Detection System for Internet of Vehicles,” *IEEE Internet Things J.*, 2021.

[126] A. Pandey and A. Jain, “Comparative Analysis of KNN Algorithm using Various Normalization Techniques,” *Int. J. Comput. Netw. Inf. Secur.*, vol. 9, no. 11, pp. 36–42, 2017.

[127] H. Eldeeb, S. Amashukeli, and R. El Shawi, "An Empirical Analysis of Integrating Feature Extraction to Automated Machine Learning Pipeline," in *Pattern Recognition. ICPR International Workshops and Challenges*, 2021, pp. 336–344.

[128] P. Sondhi, "Feature construction methods: a survey," *Sifaka. Cs. Uiuc. Edu*, vol. 69, pp. 70–71, 2010.

[129] T. Thaher, M. Mafarja, H. Turabieh, P. A. Castillo, H. Faris, and I. Aljarah, "Teaching Learning-Based Optimization With Evolutionary Binarization Schemes for Tackling Feature Selection Problems," *IEEE Access*, vol. 9, pp. 41082–41103, 2021.

[130] A. Bauer, M. Züfle, N. Herbst, A. Zehe, A. Hotho, and S. Kounev, "*Time Series Forecasting for Self-Aware Systems*," Proc. IEEE, vol. 108, no. 7, pp. 1068–1093, 2020.

[131] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under Concept Drift: A Review," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 12, pp. 2346–2363, 2019.

[132] A. S. Iwashita and J. P. Papa, "An Overview on Concept Drift Learning," *IEEE Access*, vol. 7, no. Section III, pp. 1532–1547, 2019.

[133] K. Wadewale et al., "Survey on Method of Drift Detection and Classification for time varying data set," *Comput. Biol. Med.*, vol. 32, no. 11, pp. 1–7, 2016.

[134] J. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A Survey on Concept Drift Adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, Mar. 2014.

[135] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," *Proc. 7th SIAM Int. Conf. Data Min.*, pp. 443–448, 2007.

[136] P. Vorburger and A. Bernstein, "Entropy-based concept shift detection," *Proc. - IEEE Int. Conf. Data Mining, ICDM*, pp. 1113–1118, 2006.

[137] L. I. Kuncheva, "Change detection in streaming multivariate data using likelihood detectors," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 5, pp. 1175–1180, 2013.

[138] L. Yang, D. M. Manias, and A. Shami, "PWPAE: An Ensemble Framework for Concept Drift Adaptation in IoT Data Streams," in *IEEE Global Communications Conference (GlobeCom)*, 2021, pp. 1–6.

[139] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3171, pp. 286–295, 2004.

[140] P. B. Dongre and L. G. Malik, "A review on real time data stream classification and adapting to various concept drift scenarios," *Souvenir 2014 IEEE Int. Adv. Comput. Conf. IACC 2014*, pp. 533–537, 2014.

[141] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, "Early Drift Detection Method," *4th ECML PKDD Int. Work. Knowl. Discov. from Data Streams*, vol. 6, pp. 77–86, 2006.

[142] S. G. T. C. Santos, R. S. M. Barros, and P. M. Gonçalves, "Optimizing the parameters of drift detection methods using a genetic algorithm," *Proc. - Int. Conf. Tools with Artif. Intell. ICTAI*, vol. 2016-Janua, pp. 1077–1084, 2016.

[143] I. Žliobaitė, "Learning under Concept Drift: an Overview," *arXiv*, pp. 1–36, 2010.

[144] Y. Sun, K. Tang, Z. Zhu, and X. Yao, "Concept drift adaptation by exploiting historical knowledge," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 29, no. 10, pp. 4822–4832, 2017.

[145] R. Seraj and M. Ahmed, "Concept Drift for Big Data," in *Combating Security Challenges in the Age of Big Data: Powered by State-of-the-Art Artificial Intelligence Techniques*, Springer International Publishing, 2020, pp. 29–43.

[146] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," *Proc. Seventh ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. 18, pp. 97–106, 2001.

[147] M. Żarkowski, "Adaptive online neural network for face identification with concept drift," *Adv. Intell. Syst. Comput.*, vol. 323, pp. 703–712, 2015.

[148] M. M. Yacoub, A. Rezk, and M. B. Senousy, "Adaptive classification in data stream mining," *J. Theor. Appl. Inf. Technol.*, vol. 98, no. 13, pp. 2637–2645, 2020.

[149] C. Manapragada, G. I. Webb, and M. Salehi, "Extremely fast decision tree," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 1953–1962, 2018.

[150] Y. Sun, Z. Wang, H. Liu, C. Du, and J. Yuan, "Online Ensemble Using Adaptive Windowing for Data Streams with Concept Drift," *Int. J. Distrib. Sens. Networks*, vol. 12, no. 5, p. 4218973, 2016.

[151] W. Nick Street and Y. S. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," *Proc. Seventh ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 377–382, 2001.

[152] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 226–235, 2003.

[153] S. Wares, J. Isaacs, and E. Elyan, "Data stream mining: methods and challenges for handling concept drift," *SN Appl. Sci.*, vol. 1, no. 11, pp. 1–19, 2019.

[154] K. Nishida, K. Yamauchi, and T. Omori, "ACE: Adaptive Classifiers-Ensemble System for Concept-Drifting Environments," in *Multiple Classifier Systems*, 2005, pp. 176–185.

[155] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: A new ensemble method for tracking concept drift," *Proc. - IEEE Int. Conf. Data Mining*, ICDM, pp. 123–130, 2003.

[156] R. Polikar, L. Udpa, S. S. Udpa, and V. Honavar, “Learn++: An incremental learning algorithm for supervised neural networks,” *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 31, no. 4, pp. 497–508, 2001.

[157] S. H. Bach and M. A. Maloof, “Paired learners for concept drift,” *Proc. - IEEE Int. Conf. Data Mining*, ICDM, pp. 23–32, 2008.

[158] H. M. Gomes et al., “Adaptive random forests for evolving data stream classification,” *Mach. Learn.*, vol. 106, no. 9–10, pp. 1469–1495, 2017.

[159] H. M. Gomes, J. Read, and A. Bifet, “Streaming random patches for evolving data stream classification,” *Proc. - IEEE Int. Conf. Data Mining*, ICDM, vol. 2019-Novem, no. Icdm, pp. 240–249, 2019.

[160] A. Bifet, G. Holmes, and B. Pfahringer, “Leveraging bagging for evolving data streams,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6321 LNAI, no. PART 1, pp. 135–150, 2010.

[161] F. Salo, M. Injadat, A. B. Nassif, A. Shami, and A. Essex, “Data mining techniques in intrusion detection systems: A systematic literature review,” *IEEE Access*, vol. 6, pp. 56046–56058, 2018.

[162] D. Chicco, M. J. Warrens, and G. Jurman, “The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation,” *PeerJ Comput. Sci.*, vol. 7, p. e623, Jul. 2021.

[163] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, “Ensemble learning for data stream analysis: A survey,” *Inf. Fusion*, vol. 37, pp. 132–156, 2017.

[164] C. Bergmeir and J. M. Benítez, “On the use of cross-validation for time series predictor evaluation,” *Inf. Sci. (Ny).*, vol. 191, pp. 192–213, 2012.

[165] J. I. G. Hidalgo, B. I. F. Maciel, and R. S. M. Barros, “Experimenting with prequential variations for data stream learning evaluation,” *Comput. Intell.*, vol. 35, no. 4, pp. 670–692, 2019.

[166] B. Komer, J. Bergstra, and C. Eliasmith, “Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn,” *Proc. 13th Python Sci. Conf.*, no. Scipy, pp. 32–37, 2014.

[167] H. Jin, Q. Song, and X. Hu, “Auto-Keras: An Efficient Neural Architecture Search System,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1946–1956.

[168] V. Perrone et al., “Amazon SageMaker Automatic Model Tuning: Scalable Gradient-Free Optimization,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 3463–3471, 2021.

[169] S. Markov, “SKOPT Documentation,” 2017.

[170] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive Online Analysis," *J. Mach. Learn. Res.*, vol. 11, pp. 1601–1604, 2010.

[171] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A Multi-output Streaming Framework," *J. Mach. Learn. Res.*, vol. 19, pp. 1–5, 2018.

[172] J. Montiel et al., "River: Machine learning for streaming data in python," *J. Mach. Learn. Res.*, vol. 22, pp. 1–8, 2021.

[173] C. Wheelus and X. Zhu, "IoT Network Security: Threats, Risks, and a Data-Driven Defense Framework," *IoT*, vol. 1, no. 2, pp. 259–285, 2020.

[174] L. Yang et al., "Multi-Perspective Content Delivery Networks Security Framework Using Optimized Unsupervised Anomaly Detection," *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 1, pp. 686–705, 2022. https://doi.org/10.1109/TNSM.2021.3100308.

[175] I. Ullah and Q. H. Mahmoud, "A Scheme for Generating a Dataset for Anomalous Activity Detection in IoT Networks," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12109 LNAI, pp. 508–520, 2020.

[176] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," no. Cic, pp. 108–116, 2018.

[177] M. Marjani et al., "Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges," *IEEE Access*, vol. 5, pp. 5247–5261, 2017.

[178] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for internet of things data analysis: a survey," *Digit. Commun. Networks*, vol. 4, no. 3, pp. 161–175, 2018.

[179] F. Hussain, R. Hussain, S. A. Hassan, and E. Hossain, "Machine Learning in IoT Security: Current Solutions and Future Challenges," *IEEE Commun. Surv. Tutorials*, vol. 22, no. 3, pp. 1686–1721, 2020.

[180] D. Peteiro-Barral and B. Guijarro-Berdiñas, "A survey of methods for distributed machine learning," *Prog. Artif. Intell.*, vol. 2, no. 1, pp. 1–11, 2013.

[181] D. M. Manias and A. Shami, "Making a Case for Federated Learning in the Internet of Vehicles and Intelligent Transportation Systems," *IEEE Netw.*, vol. 35, no. 3, pp. 88–94, 2021.

[182] D. M. Manias, I. Shaer, L. Yang, and A. Shami, "Concept Drift Detection in Federated Networked Systems," in *IEEE Global Communications Conference (GlobeCom)*, 2021, pp. 1–6.

[183] H. J. Escalante, "Automated Machine Learning – a brief review at the end of the early years," *arXiv*, pp. 1–17, 2020.

[184] L. Yang and A. Shami, “A Transfer Learning and Optimized CNN Based Intrusion Detection System for Internet of Vehicles,” in *2022 IEEE International Conference on Communications (ICC)*, 2022, pp. 1–6.

Chapter 3

Tree-based Intelligent Intrusion Detection System in Internet of Vehicles

3.1 Introduction

With more vehicles, devices, and infrastructures involved, the conventional vehicular ad hoc networks (VANETs) are gradually evolving into the Internet of Vehicles (IoV) [2]. In intelligent transportation systems (ITSs), VANETs enable wireless communications between vehicles and devices, then transform the vehicles and devices into wireless routers or mobile nodes [3]. Autonomous vehicles (AV) technology is a fast-paced technology which provides an ideal solution to reduce traffic collisions and related costs. Vehicle-to-everything (V2X) [3] technology is at the core of AV, consisting of Vehicle-to-Vehicle (V2V), Vehicle-to-Pedestrian (V2P), Vehicle-to-Infrastructure (V2I), and Vehicle-to-Network (V2N) technologies to enable local and wide area cellular network communications between vehicles, pedestrians, and infrastructures.

V2X technology, with the means of wireless communications, aims to involve more Internet of things (IoT) devices. However, some of these devices lack security mechanisms such as firewalls and gateways [4]. AVs are prone to network threats with severe consequences because attacking or maliciously controlling the vehicles on the road poses serious threat to human lives. The potential networking threats include the following attacks. A common type of attack is Denial of services (DoS) attacks launched by sending a large number of irrelevant

A version of this chapter has been published in IEEE GlobeCom 2019 [1].

messages or requests to occupy the node and take control of the resources of the vehicle [5]. The attackers perform spoofing attacks such as GPS spoofing to masquerade as legitimate users and provide the nodes with fake GPS information [5]. Sniffing attacks such as port scan attacks is another type of attack, which is launched to obtain confidential or sensitive data of the vehicles systems and users [6]. In addition, brute-force attacks are launched by attackers to crack passwords in the vehicle networks or systems [6]. To intrude into the web interface of vehicles or servers, web attacks including SQL injection attacks and cross-site scripting (XSS) can also be implemented by the attackers [6].

Apart from external communication threats, AVs are also prone to intra-vehicle communication attacks. Controller Area Network (CAN) [7] is a bus communication protocol which enables in-vehicle communications between all Electronic Control Units (ECUs). It provides an efficient error detection mechanism for stable transmission and can reduce wiring cost, weight, and complexity [7]. However, all the ECUs communicate with each other through the CAN bus, which makes the ECUs vulnerable to various attacks if the CAN bus is compromised. In CAN bus communications, attackers can inject malicious messages to monitor the network traffic or launch other hostile attacks and the nodes will inevitably deal with the messages without validating their origins. The message injection attacks on CAN bus can be classified by their aims. Similar to external networks, DoS and spoofing attacks can also be launched on the CAN bus to occupy the resources or provide malicious information such as gear and RPM (revolutions per minute) information. In addition, fuzzy attacks are another common type of attack launched on CAN bus by injecting arbitrary messages to cause the vehicles to show unintended states or malfunction [8].

All the above vulnerabilities and threats call for a robust protection system that can repel possible attacks that pose threats to intra-vehicle and external communications in AV systems. Intrusion detection systems (IDSs) are an effective security mechanism to identify the abnormal information and attacks through the network traffic data during the communication between vehicles and other devices. Intrusion detection is often considered as a classification problem; machine learning (ML) methods have been widely used to develop IDSs [9]. An intelligent IDS is proposed in this chapter for network attack detection that can be applied to not only Controller Area Network (CAN) bus of AVs but also on general IoVs. The proposed IDS utilizes tree-based ML algorithms including decision tree (DT), random forest (RF), extra trees (ET), and Extreme Gradient Boosting (XGBoost). A qualified IDS needs to not only achieve a high detection rate, but also have a low computational cost. Therefore, an ensemble learning model, namely stacking, is used to improve accuracy and the feature selection methods are also implemented to reduce computational time. The performance of the proposed IDS is evaluated using multiple standard open-source data sets with results showing high accuracy in detecting intrusions.

This chapter makes the following contributions:

1. Surveys the vulnerabilities and potential attacks in CAN and AV networks.
2. Proposes an intelligent IDS for both AV and general networks by using the tree structure ML and ensemble learning methods.
3. Presents a comprehensive framework to prepare network traffic data for the purpose of IDS development.
4. Proposes an averaging feature selection method using tree structure ML models to improve the efficiency of the proposed IDS and to perform an analysis of network attributes and attacks for network monitoring uses.

This chapter is organized as follows: Section 3.2 presents the system overview of the proposed IDS and its architecture. Section 3.3 discusses the proposed IDS framework in detail. Section 3.4 provides the results, performance comparison and feature analysis. Finally, Section 3.5 concludes the chapter.

3.2 System Design

3.2.1 Problem Statement

Due to the fact that the AV systems are vulnerable to many types of network threats through different communication mediums, a comprehensive IDS system is proposed to be implemented

Code for the major modules is available at: https://github.com/Western-OC2-Lab/Intrusion-Detection-System-Using-Machine-Learning

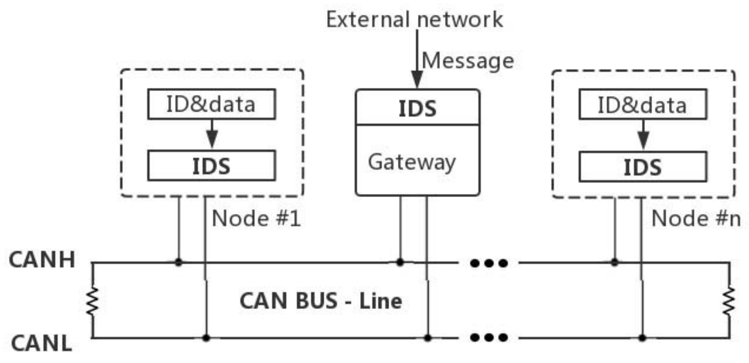

Figure 3.1: The proposed IDS-protected AV architecture

for both intra-vehicle and external communication networks. This is done to better protect not only the vehicle components but also all the IoT devices involved in the entire IoV.

The proposed IDS should be able to detect various common intrusions launched on CAN bus and external networks. The main attacks to be considered in this work include DoS attacks on both the intra-vehicle and external communication networks, fuzzy attacks and spoofing attacks on the CAN-bus, sniffing, brute force and web attacks launched on the external networks. The IDS should have a high detection rate to effectively identify most of the attacks and low computational time to improve efficiency.

3.2.2 IDS System Overview and Architecture

To provide protection for both the intra-vehicle and external communications, the proposed IDS is implemented in multiple locations within the AV system. To detect threats on the CAN bus and secure it, the IDS can be placed on the top of the CAN bus to process every transmitted message and ensure the nodes are not compromised [10]. Alternatively, the proposed IDS can be placed inside the gateway to secure the external communication networks [11]. The topology of IDS implementation on vehicle systems is shown in Fig. 3.1.

On the CAN bus, all the nodes are linked, and when a node receives a message mainly consisting of an ID and data field sent by another device connected to the CAN bus, the message is passed through the IDS to check when the signal line of the CAN bus changes from CANH to CANL. Similarly, when a message is transmitted from external network to intranet, it is passed through the IDS inside the gateway and checked.

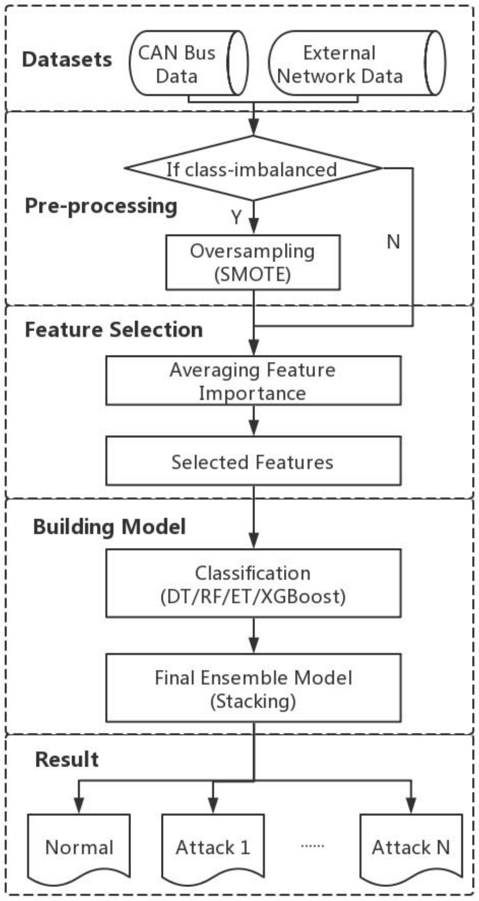

Figure 3.2: The proposed IDS framework

To design the IDS, an overview of the tree-based ML classification framework for intrusion detection is provided in Fig. 3.2. The process of the proposed model is as follows. Firstly, sufficient network traffic data is collected. Secondly, if the classes of the data set are imbalanced, oversampling is implemented to reduce its impact. At the next stage, feature selection based on averaging feature importance is done to reduce computational cost. After that, four base-models are built to be the input of the stacking ensemble model. At the end, the final model is built to classify the data.

3.3 Proposed IDS Framework

3.3.1 Data Pre-Processing

The first step to develop an IDS is to collect sufficient amount of network traffic data under both the normal state and the abnormal state caused by different types of attacks. The data can be collected by the packet sniffers, but they should have suitable network attributes, or named network features, for the purpose of IDS development.

Firstly, to design an IDS for CAN bus intrusion, since the main CAN bus threats are message injection attacks, the data of CAN messages/frames should be collected and the main features correlated to attacks are CAN IDs and the data field of frames [8].

Regarding the external networks, since they belong to general networks and are prone to various regular network threats, the data with more network attributes should be collected to develop an effective IDS that can detect various types of cyber-attacks. Most of the regular network attributes such as packet length, data transfer rate, throughput, inter-arrival time, flags of TCP and their counts, segment size, and active/idle time should be considered [6]. However, the computational complexity of the proposed IDS may increase dramatically due to the high data dimensionality. Thus, further feature analysis should be done for the external network data.

The collected network data would be pre-processed after a few steps to be better suited for IDS development purpose. Firstly, the data can be encoded with one-hot-vector because it has a certain threshold to help separate normal data and anomalies [7]. On the other hand, ML training is often more efficient with normalized data [4]. Thus, each feature with numerical values is set to the range of 0.0 to 1.0, and each value after normalization x_n is indicated as:

$$x_n = \frac{x - min}{max - min}, \tag{3.1}$$

where x is the original value, max and min are the maximum and minimum values of each feature.

In addition, network data are often class imbalanced because in real life, networks maintain the normal state most of the time and the attack-label instances are often not enough. To overcome the issue of class-imbalanced data which often results in a low anomaly detection rate, random oversampling and Synthetic Minority Oversampling Technique (SMOTE) [12] can be used to generate more data in the minority classes that do not have enough data. The basic strategy of random oversampling is to simply make multiple copies of the samples to increase samples in the minority classes. However, the random oversampling method can easily result in over-fitting because the information learned would be very specific instead of generic. On the other hand, the SMOTE algorithm analyzes the minority classes and generates new samples based on them by using the idea of K nearest neighbors. Therefore, SMOTE can generate high-quality samples and is used for the minority classes in the proposed system.

3.3.2 The Proposed ML Approaches

In the proposed system, to detect various cyber-attacks, developing the IDS can be considered as a multi-classification problem, and machine learning algorithms are widely used to solve such classification problems [13] [14]. The selected ML algorithms are based on tree structure, including decision tree, random forest, extra trees, and XGBoost.

Decision tree (DT) [15] is a common classification method based on divide and conquer strategy. A DT comprises decision nodes and leaf nodes, and they represent a decision test over one of the features, and the result class, respectively. Random forest (RF) [16] is an ensemble learning classifier based on the majority voting rule that the class with the highest votes by decision trees is selected to be the classification result. Similarly, extra trees (ET) [17] is another ensemble model based on a collection of randomized decision trees generated by processing different subsets of data set. In contrast, XGBoost [18] is a ensemble learning algorithm designed for speed and performance improvement by using the gradient descent method to combine many decision trees.

Apart from the proposed algorithms, other models such as K-nearest neighbor (KNN) and support vector machine (SVM) [19] are also common for classification problems. For the purpose of model selection, the computational complexity of common supervised ML algorithms is calculated. Assuming the number of training instances is N, the number of features is P, and the number of trees is T, we have the following approximations. The complexity of DT is $O(N^2P)$ while the complexity of RF is $O(N^2\sqrt{P}T)$. In addition, ET and XGBoost have a similar complexity of $O(NPT)$. On the other hand, KNN's complexity is $O(NP)$ and the SVM algorithm's complexity is $O(N^2P)$ [13] [20]. However, unlike KNN and SVM, the proposed four tree-based models, DT, RF, ET, and XGBoost, enable multi-threading to save training time. Assuming the maximum number of participating threads of a computer is M, the time complexity of DT, RF, ET, and XGBoost reduces to $O(\frac{N^2P}{M})$, $O(\frac{N^2\sqrt{P}T}{M})$, $O(\frac{NPT}{M})$, and $O(\frac{NPT}{M})$, respectively. Therefore, although the original time complexity of the considered algorithms are similar, the four tree structure algorithms have lower computational time due to multi-threading, which is an important reason for choosing these four algorithms.

The other reasons for choosing these algorithms are as follows: 1) Most of the tree structure ML models are using ensemble learning so they often show better performance than other single models such as KNN. 2) They have the ability to handle non-linear and high dimensional data that the proposed network data belongs to. 3) The feature importance calculations are done during the building process of those models, which is beneficial when performing feature selection.

It is noteworthy that there are some hyper-parameters of the proposed algorithms that need be tuned to achieve better performance. For the DT algorithm, the split measure function is set to be Gini impurity, and the classification and regression trees (CART) model is then built, which shows better performance than using information gain theory to build an ID3 tree. Assuming that S denotes the set of all sub-trees, CART selects the tree in S that minimizes [15]:

$$C(S) = \hat{L}_n(S) + \alpha|S|, \tag{3.2}$$

where $|S|$ is the cardinality of the tree, α is a constant and $\hat{L}_n(S)$ is the empirical risk using the tree S. Since the deeper tree has more sub-trees, tree depth D is an important parameter of the CART algorithm.

For RF and ET, since their results are based on the majority voting of many decision trees, the number of decision trees T is another important parameter affecting the performance. T could also be tuned in XGBoost which is also based on the ensemble of numerous trees. Specifically, XGBoost minimizes the following regularized objective function [18]:

$$Obj = -\frac{1}{2}\sum_{j=1}^{t}\frac{G_j^2}{H_j+\lambda}+\gamma t, \tag{3.3}$$

where G and H represent the sums of the first and second order gradient statistics of the loss function, t is the number of leaves in a decision tree, λ and γ are the penalty coefficients. Since the gradient statistics are based on the sum of the predicted scores of the T trees, and the number of leaves increases with the increase of the tree depth D, T and D have direct impacts on the objective function value of XGBoost.

If the parameters T and D are too small, it leads to under-fitting and if they are too large, it causes over-fitting and additional computational costs.

The grid search method [21] is utilized to find the optimal values. To be precise, the training of the models starts from a small number of trees and a short depth. Then, these two values are slowly increased with accuracy evaluated until over-fitting, which is indicated by the dropping of accuracy. Finally, the tree depth D is tuned to 8 and the number of trees T is set to 200. Similarly, the minimum sample split and minimum sample leaf are both tuned from 1 to 10, and finally set to 8 and 3, respectively. Note that the parameter tuning process can be further improved by using other optimization techniques.

3.3.3 Ensemble Learning and Feature Selection

For the purpose of further accuracy improvement, an ensemble learning technique, stacking, is implemented. Stacking [22] is a common ensemble method consisting of two layers where the first layer contains a few trained base predictors, and their output serves as the input of a meta-learner in the second layer to build a strong classifier. The four trained tree structure algorithms serve as the base models in the first layer of the stacking ensemble method, and the singular algorithm with highest accuracy among the four base models is selected to be the meta-classifier in the second layer.

To improve the confidence of the selected features, an ensemble feature selection (FS) technique is utilized by calculating the average of feature importance lists generated by the four selected tree-based ML models. They are chosen for feature selection because tree-based algorithms calculate the importance of each feature based on each single tree, and then average the output of the trees to make the result more reliable. Additionally, different traditional feature selection methods such as information gain, entropy, and Gini coefficient are utilized in the system by setting different parameters in tree-based methods to generate the convincing feature importance. The sum of the total feature importance is 1.0. To select features, the features are ranked with their importance and each feature is added into the feature list from high importance to low importance till the sum of importance reaches 0.9. The other features with the sum of importance less than 0.1 will be discarded to reduce the computational costs.

3.3.4 Validation Metrics

Each considered data set is split into five subsets and 5-fold cross-validation is implemented to evaluate the proposed models. Accuracy (Acc), detection rate (DR)/recall, false alarm rate (FAR), and F1 score are the main metrics used to evaluate the proposed method, with their formulas proposed in [23]. The accuracy is the proportion of correctly classified data. However, the data sets may exhibit class imbalance, resulting in a high accuracy of normal data classification but a low attack detection rate. Thus, the detection rate, the ratio between the detected attack data and the total abnormal data, is also calculated for evaluation. A qualified IDS should have a high DR to ensure most of the attacks can be detected and a low FAR to confirm the system does not misreport data for higher DR [16]. In addition, the F1 score considers both the precision and recall by calculating their harmonic average, which can be used to evaluate the overall performance of the methods. Also, the execution time, mainly the model training time, is used to provide insights into the computational performance.

3.4 Performance Evaluation

3.4.1 Datasets Description

To evaluate the proposed IDS, this work considers two different datasets for the purpose of its implementation in both intra-vehicle and external networks. The first data set is called ”Car-Hacking Dataset” or ”CAN-intrusion dataset” which was proposed in 2018 for the purpose of

Table 3.1: Data Type and Size of The CAN-Intrusion Dataset

Class label	Number of instances	Class label	Number of instances
Normal	14,037,293	RPM spoofing	654,897
DoS	587,521	Gear spoofing	597,252
Fuzzy	491,847	-	-

Table 3.2: Data Type and Size of The CICIDS2017 Dataset

Class label	Number of instances	Class label	Number of instances
BENIGN	2,273,097	Web-Attack	2,180
DoS	380,699	Botnet	1,966
Port-Scan	158,930	Infiltration	36
Brute-Force	13,835	-	-

IDS development on CAN bus [7]. On the other hand, to build a comprehensive IDS that can also be effective in external communication networks, a standard IDS data set containing the most updated attack scenarios, named ”CICIDS2017” [6], is considered in this work. To prepare better datasets for the IDS development, minor tasks including data combination, missing value removal and new label assignments were done for both datasets based on the methods proposed in [24]. The specifics of the improved datasets are shown in Tables 3.1 and 3.2.

3.4.2 Performance Analysis

The proposed system was implemented using Python 3.5 and the experiments were carried out on a machine with 6 Core i7-8700 processor and 16 GB of memory.

The results of testing different algorithms on CAN-intrusion data set and CICIDS2017 data

Table 3.3: Performance Evaluation of IDS on CAN-intrusion Dataset

Method	Acc (%)	DR (%)	FAR (%)	F1 Score	Execution Time (S)
KNN [9]	97.4	96.3	5.3	0.934	911.6
SVM [9]	96.5	95.7	4.8	0.933	13765.6
DT	99.99	99.99	0.006	0.999	328
RF	99.99	99.99	0.0003	0.999	506.8
ET	99.99	99.99	0.0005	0.999	216.3
XGBoost	99.98	99.98	0.012	0.999	3499.1
Stacking	100	100	0.0	1.0	1237.1
FS RF	99.99	99.99	0.0013	0.999	99.6
FS Stacking	99.99	99.99	0.0006	0.999	325.6

Table 3.4: Performance Evaluation of IDS on CICIDS2017 Dataset

Method	Acc (%)	DR (%)	FAR (%)	F1 Score	Execution Time (S)
KNN [6]	96.6	96.4	5.6	0.966	9243.6
SVM [25]	98.01	97.58	1.48	0.978	49953.1
DT	99.72	99.3	0.029	0.998	126.7
RF	98.37	98.29	0.039	0.983	2421.6
ET	93.43	93.35	0.001	0.934	2349.6
XGBoost	99.78	99.76	0.069	0.997	1637.2
Stacking	99.86	99.8	0.012	0.998	4519.3
FS XGBoost	99.7	99.55	0.077	0.996	995.9
FS Stacking	99.82	99.75	0.011	0.997	2774.8

set are shown in Tables 3.3 and 3.4, respectively. According to Table 3.3, when testing on the CAN-intrusion data set, the tree-based algorithms including DT, RF, ET, and XGBoost used in the proposed system are 2.5% more accurate than KNN and 3.4% more accurate than SVM used in [9]. In addition, multi-threading is enabled in DT, RF, ET, and XGBoost, resulting in a lower execution time compared with KNN and SVM. Since XGBoost has the lowest accuracy and longest execution time among the four tree-based ML models, only the other three algorithms, DT, RF, and ET were selected into the stacking ensemble model, and the single model with best performance, RF, was selected to be the meta-classifier in the second layer. After using stacking to combine the three tree-based models, the accuracy, detection rate and F1 score reaches 100%, which means that all the trained attacks can be detected.

Similarly in CICIDS2017 data set, as shown in Table 3.4, most of the used tree-based algorithms shows 1.8-3.2% higher accuracy, detection rate and F1 score than KNN [6] and SVM [25] except for ET. Hence, only DT, RF, and XGBoost were chosen in the proposed stacking method, and XGBoost was selected to be the meta-classifier of the stacking model. Although the execution time of stacking is longer than any singular tree-based models, it reaches the highest accuracy among the models (99.86%).

After training the data sets with all the features to obtain the highest performance, the feature selection method proposed in Section 3.3.3 was implemented to reduce execution time. From Tables 3.3 and 3.4, it can be seen that RF and XGBoost are the singular models with best accuracy on CAN-intrusion data set and CICIDS2017 data set, respectively. As single base models have lower execution time than the ensemble model, these two singular models were also tested after feature selection. For CAN-intrusion data set, the top-4 important features, 'CAN ID', 'DATA[5]', 'DATA[3]', 'DATA[1]' were selected for the tests. The results of RF and stacking on CAN-intrusion data set after feature selection, named "FS RF" and "FS Stacking", can be seen from Table 3.3. It shows that the RF and stacking models saved 80.3% and 73.7% of the execution time after feature selection, respectively, while still maintaining a high accuracy (99.99%). For CICIDS2017 data set, 36 out of 78 features were selected and the accuracy of XGBoost and stacking models only decreased by 0.08% and 0.04% while saving 39.2% and 38.6% of the execution time, respectively, as shown in Table 3.4. Therefore, the tree-based averaging feature selection method enables the IDS to save execution time while maintaining high accuracy.

3.4.3 Feature Analysis

Table 3.5: Top-3 Feature Importance by Each Attack

Label	**Feature**	**Weight**
DoS	Bwd Packet Length Std	0.1723
	Average Packet Size	0.1211
	Destination Port	0.0785
Port-Scan	Total Length of Fwd Packets	0.3020
	Average Packet Size	0.1045
	PSH Flag Count	0.1019
Brute-Force	Destination Port	0.3728
	Fwd Packet Length Min	0.1022
	Packet Length Variance	0.0859
Web-Attack	Init_Win_bytes_backward	0.2643
	Average Packet Size	0.1650
	Destination Port	0.0616
Botnet	Destination Port	0.2364
	Bwd Packet Length Mean	0.1240
	Avg Bwd Segment Size	0.1104
Infiltration	Total Length of Fwd Packets	0.2298
	Subflow Fwd Bytes	0.1345
	Destination Port	0.1149

In order to analyze the features, the proposed feature selection method was tested on the subsets of each attack. The list of top-3 important features and their corresponding weights of each attack is shown in Table 3.5. As Table 3.5 shows, the destination port can reflect a DoS, brute force, web attack, and botnet attack. The size of packets is another important feature. For example, the average packet size indicates a DoS attack, port scan attack, and web attack. The packet length in the forward direction is related to port scan, brute-force, and infiltration attacks, while the packet length in the backward direction reflects a DoS, web attack, and botnet. In addition, the variance of the length of the packets in both forward and backward directions reflects the brute force attack, and the count of pushing flags indicates port scan attack. After getting the feature importance list of each attack, the IDSs with other aims like designing a dedicated system for detecting a single type of attack can be developed by selecting the important features based on the list. On the other hand, the important features can be selected as key attributes to be monitored by network supervisors. If those attributes change abnormally, the attacks can be detected quickly.

3.5 Conclusion

As autonomous vehicles and connected vehicles are vulnerable to various cyber-attacks, IDS is one of the efficient solutions to detect the launched network attacks and secure the vehicle networks. In this chapter, we presented an IDS based on tree-based machine learning algorithms to detect threats both on CAN bus and external networks. The SMOTE oversampling method and tree-based averaging feature selection approaches were also introduced to reduce the impact of class imbalance and computational cost. To evaluate the proposed IDS, it was tested on two data sets for both intra-vehicle and external networks. The results on both data sets show that the proposed system has 2-3% higher accuracy, detection rate, F1 score and lower false alarm rate than the methods proposed in the literature. Moreover, unlike other proposed methods, our work combines all the subsets of the data sets and develops an IDS that can detect various attacks instead of only single type of attack on each run. The accuracy of CAN intrusion and CICIDS2017 data set reaches 100% and 99. 86%, while the computational time was reduced by 73.7% to 325.6s and by 38.6% to 2774.8s, respectively. In future work, the results of the proposed IDS on CICIDS2017 data set can be further improved by using optimization techniques such as particle swarm optimization and Baysian optimization to tune the hyper-parameters of the proposed algorithms.

Bibliography

[1] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, “Tree-based Intelligent Intrusion Detection System in Internet of Vehicles,” *proc. 2019 IEEE Glob. Commun. Conf.*, pp. 1–6, Hawaii, USA, 2019.

[2] A. Awang, K. Husain, N. Kamel and S. Aïssa, ”Routing in Vehicular Ad-hoc Networks: A Survey on Single- and Cross-Layer Design Techniques, and Perspectives,” in *IEEE Access*, vol. 5, pp. 9497-9517, 2017.

[3] F. Yang, S. Wang, J. Li, Z. Liu, and Q. Sun, ”An overview of Internet of Vehicles,” *China Commun.*, vol. 11, no. 10, pp. 1-15, 2014.

[4] K. M. Ali Alheeti and K. Mc Donald-Maier, ”Intelligent intrusion detection in external communication systems for autonomous vehicles,” *Syst. Sci. Control Eng.*, vol. 6, no. 1, pp. 48-56, 2018.

[5] H. P. Dai Nguyen and R. Zoltán, ”The Current Security Challenges of Vehicle Communication in the Future Transportation System,” *SISY 2018 - IEEE 16th Int. Symp. Intell. Syst. Informatics, Proc*, Subotica, pp. 161-166, 2018.

[6] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, ”Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” no. Cic, pp. 108-116, 2018.

[7] E. Seo, H. M. Song, and H. K. Kim, ”GIDS: GAN based Intrusion Detection System for In-Vehicle Network,” *2018 16th Annu. Conf. Privacy, Secur. Trust*, pp. 1-6, 2018.

[8] H. Lee, S. H. Jeong, and H. K. Kim, ”OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame,” *Proc. 15th Annu. Conf. Privacy, Secur. Trust. PST 2017*, pp. 57-66, 2017.

[9] A. Alshammari, M. A. Zohdy, D. Debnath, and G. Corser, ”Classification Approach for Intrusion Detection in Vehicle Systems,” *Wirel. Eng. Technol.*, vol. 09, no. 04, pp. 79-94, 2018.

[10] B. Groza and P. Murvay, ”Efficient Intrusion Detection With Bloom Filtering in Controller Area Networks,” *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 4, pp. 1037-1051, 2019.

[11] U. E. Larson, D. K. Nilsson and E. Jonsson, ”An approach to specification-based attack detection for in-vehicle networks,” *IEEE Intell. Veh. Symp. Proc*, Eindhoven, pp. 220-225, 2008.

[12] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer, ”SMOTE: Synthetic Minority Over-Sampling Technique,” *J. Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002.

[13] A. Moubayed, M. Injadat, A. Shami and H. Lutfiyya, ”DNS Typo-Squatting Domain Detection: A Data Analytics & Machine Learning Based Approach,” *2018 IEEE Glob. Commun. Conf.*, Abu Dhabi, United Arab Emirates, pp. 1-7, Dec. 2018.

[14] D. M. Manias, M. Jammal, H. Hawilo, A. Shami, et. al., ”Machine Learning for Performance-Aware Virtual Network Function Placement,” *2019 IEEE Glob. Commun. Conf.*, Waikolao, HI, USA, Dec. 2019.

[15] Y. Ping, ”Hybrid fuzzy SVM model using CART and MARS for credit scoring,” *2009 Int. Conf. Intell. Human-Machine Syst. Cybern. IHMSC 2009*, vol. 2, pp. 392-395, 2009.

[16] M. Injadat, F. Salo, A. B. Nassif, A. Essex, and A. Shami, ”Bayesian Optimization with Machine Learning Algorithms Towards Anomaly Detection,” *2018 IEEE Glob. Commun. Conf.*, pp. 1-6, 2018.

[17] K. Arjunan and C. N. Modi, ”An enhanced intrusion detection framework for securing network layer of cloud computing,” *ISEA Asia Secur. Priv. Conf. 2017, ISEASP 2017*, pp. 1-10, 2017.

[18] D. Zhang, L. Qian, B. Mao, C. Huang, B. Huang, and Y. Si, ”A Data-Driven Design for Fault Detection of Wind Turbines Using Random Forests and XGboost,” in *IEEE Access*, vol. 6, pp. 21020-21031, 2018.

[19] L. Yang, R. Muresan, A. Al-Dweik and L. J. Hadjileontiadis, ”Image-Based Visibility Estimation Algorithm for Intelligent Transportation Systems,” in *IEEE Access*, vol. 6, pp. 76728-76740, 2018.

[20] MJ. Kearns, ”The computational complexity of machine learning,” *MIT press*, 1990.

[21] A. Pattawaro and C. Polprasert, ”Anomaly-Based Network Intrusion Detection System through Feature Selection and Hybrid Machine Learning Technique,” *2018 16th Int. Conf. ICT&KE*, Bangkok, pp. 1-6, 2018.

[22] M. Mohammed, H. Mwambi, B. Omolo, and M. K. Elbashir, ”Using stacking ensemble for microarray-based cancer classification,” *Int. Conf. Comput. Control. Electr. Electron. Eng. ICCCEEE 2018*, pp. 1-8, 2018.

[23] F. Salo, M. Injadat, A. B. Nassif, A. Shami, and A. Essex, ”Data mining techniques in intrusion detection systems: A systematic literature review,” in *IEEE Access*, vol. 6, pp. 56046-56058, 2018.

[24] R. Panigrahi and S. Borah, ”A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems,” *Int. J. Eng. Technol.*, vol. 7, no. 3.24, pp. 479-482, 2018.

[25] S. U. Jan, S. Ahmed, V. Shakhov, and I. Koo, ”Toward a Lightweight Intrusion Detection System for the Internet of Things,” in *IEEE Access*, vol. 7, pp. 42450-42471, 2019.

Chapter 4

LCCDE: A Decision-Based Ensemble Framework for Intrusion Detection in The Internet of Vehicles

4.1 Introduction

With the fast development of the Internet of Things (IoT) and the Internet of Vehicles (IoV) technologies, network-controlled automobiles, such as Autonomous Vehicles (AVs) and Connected Vehicles (CVs), have begun replacing conventional vehicles [2]. IoV systems typically consist of intra-vehicle networks (IVNs) and external networks. In IVNs, the Controller Area Network (CAN) bus is the core infrastructure enabling communication between Electronic Control Units (ECUs) to implement various functionalities [3]. External vehicular networks, on the other hand, utilizes Vehicle-To-Everything (V2X) technology to enable the connection between smart cars and other IoV entities, such as roadside units, infrastructures, and smart devices [4].

Due to the expanded network attack surfaces of IoV systems, the growing connectivity of vehicular networks has resulted in numerous security threats [5]. In addition, there are not authentication or encryption mechanisms involved in the processing of CAN packets owing to

A version of this chapter has been accepted in IEEE GlobeCom 2022 [1].

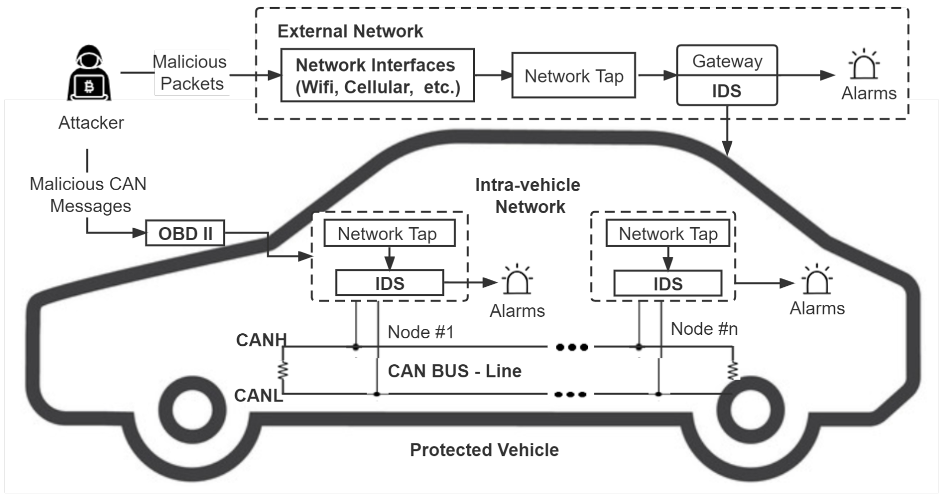

Figure 4.1: The IDS-protected vehicle architecture.

their short length [6]. In the absence of basic security mechanisms, cybercriminals are able to insert malicious messages into IVNs and execute a variety of attacks, such as Denial of Service (DoS), fuzzy, and spoofing attacks [5]. On the other hand, the emergence of connectivity between connected cars and external networks has made these vehicles vulnerable to a number of conventional cyber-attacks. Figure 4.1 depicts the IoV attack scenarios, including IVN and external network attacks. Intrusion Detection Systems (IDSs) have been developed as promising solutions for detecting intrusions and defending Internet of Vehicles (IoV) systems and smart automobiles from cyberattacks [7].

The potential deployment of IDSs in IoV systems is also shown in Fig. 4.1. To protect IVNs, IDSs can be placed on top of the CAN-bus to identify malicious CAN messages [6]. IDSs can also be incorporated into the gateways to detect malicious packets coming from external networks [2].

Due to the advancement of Machine Learning (ML) methods, ML-driven IDSs in IoV applications have recently drawn the interest of researchers and automotive manufacturers [8]. Through network traffic data analytics, ML approaches are commonly employed to construct classifier-based IDSs that can differentiate between benign network events and various cyber-attacks [9].

To apply ML models to IDS systems, it is common to observe that the prediction performance of different ML models varies significantly for different types of cyber-attack detection. Thus, a novel ensemble approach named Leader Class and Confidence Decision Ensemble (LCCDE) is proposed in this chapter to obtain optimal performance on all types of attack detection by integrating three advanced gradient-boosting ML algorithms, including Extreme Gradient Boosting (XGBoost) [10], Light Gradient Boosting Machine (LightGBM) [11], and Categorical Boosting (CatBoost) [12]. LCCDE aims to achieve optimal model performance by identifying the best-performing base ML model with the highest prediction confidence for each class.

This chapter mainly makes the following contributions:

1. It proposes a novel ensemble framework, named LCCDE, for effective intrusion detection in IoVs using class leader and confidence decision strategies, as well as gradient-boosting ML algorithms.
2. It evaluates the proposed framework using two public IoV security datasets, Car-Hacking [13] and CICIDS2017 [14] datasets, representing IVN and external network data, respectively.
3. It compares the performance of the proposed model with other state-of-the-art methods.

The remainder of the chapter is organized as follows. Section 4.2 introduces the related work about IoV intrusion detection using ML and ensemble models. Section 4.3 presents the proposed LCCDE framework in detail. Section 4.4 presents and discusses the experimental results. Finally, Section 4.5 concludes the chapter.

4.2 Related Work

The recent surge in the number of intelligent cars has led to an increase in the development of ML models as effective solutions for IoV intrusion detection and security enhancement [15]. Song *et al.* [16] proposed a deep convolutional neural network model framework for intrusion detection in in-vehicle networks. It shows high performance on the Car-Hacking dataset. Zhao *et al.* [17] proposed an IDS framework for IoT systems based on lightweight

code is available at: https://github.com/Western-OC2-Lab/Intrusion-Detection-System-Using-Machine-Learning deep neural network models. It also uses Principal Component Analysis (PCA) to reduce feature dimensionality and computational cost.

Several existing works have focused on IDS development using ensemble techniques. Yang *et al.* [18] proposed a stacking ensemble framework for network intrusion detection in IoV systems using tree-based ML models. The stacking ensemble model shows high accuracy on the CAN-Intrusion and CICIDS2017 datasets. Elmasry *et al.* [19] proposed an ensemble model for network intrusion detection using three deep learning models: Deep Neural Networks (DNN), Long Short-Term Memory (LSTM), and Deep Belief Networks (DBN). Chen *et al.* [20] proposed a novel ensemble IDS framework, named All Predict Wisest Decides (APWD), to detect intrusions and make decisions based on the wisest model for each class. However, it only achieves an accuracy of 79.7% on the NSL-KDD dataset.

Although many of the related works achieve high performance in intrusion detection tasks of IoV systems, there is still much room for performance improvement. Existing IDS frameworks can be improved with the use of more advanced ML algorithms and ensemble strategies. To the best of our knowledge, our proposed LCCDE framework is the first technique that leverages both leader class and prediction confidence strategies to construct ensemble IDSs. The use of three advanced gradient-boosting algorithms also improves the effectiveness of intrusion detection.

4.3 Proposed Framework

4.3.1 System Overview

The purpose of this work is to develop an ensemble IDS framework that can effectively detect various types of attacks on both IVN and external vehicular networks. Figure 4.2 demonstrates the overall framework of the proposed system, consisting of two phases: model training and model prediction. At the model training stage, three advanced ML algorithms, XGBoost [10], LightGBM [11], and CatBoost [12], are trained on the IoV traffic dataset to obtain the leader models for all classes/types of attacks. At the model prediction stage, the class leader models and their prediction confidences are used to accurately detect attacks. The algorithm details are

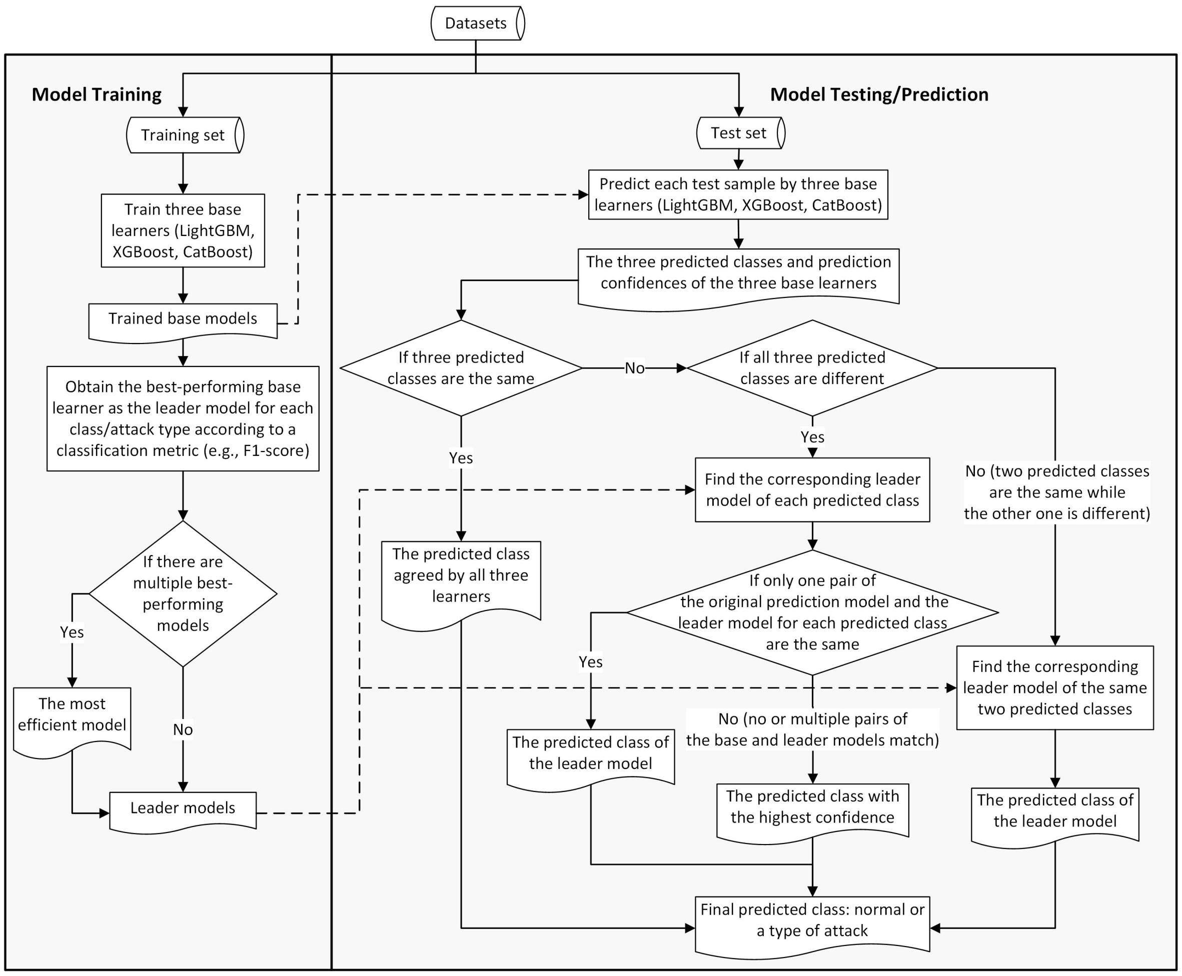
Datasets
Model Training
Training set
Train three base learners (LightGBM, XGBoost, CatBoost)
Trained base models
Obtain the best-performing base learner as the leader model for each class/attack type according to a classification metric (e.g., F1-score)
If there are multiple best-performing models
Yes
The most efficient model
No
Leader models
Model Testing/Prediction
Test set
Predict each test sample by three base learners (LightGBM, XGBoost, CatBoost)
The three predicted classes and prediction confidences of the three base learners
If three predicted classes are the same
No
If all three predicted classes are different
Yes
The predicted class agreed by all three learners
Yes
Find the corresponding leader model of each predicted class
No (two predicted classes are the same while the other one is different)
If only one pair of the original prediction model and the leader model for each predicted class are the same
Yes
The predicted class of the leader model
No (no or multiple pairs of the base and leader models match)
The predicted class with the highest confidence
Find the corresponding leader model of the same two predicted classes
The predicted class of the leader model
Final predicted class: normal or a type of attack

Figure 4.2: The framework of the proposed LCCDE model.

provided in this section.

4.3.2 Base Machine Learning Models

A decision tree (DT) is a basic ML algorithm that uses a tree structure to make decisions based on the divide and conquer technique [18]. In DTs, the decision nodes represent the decision tests, while the leaves indicate the result classes. Gradient Boosting Decision Tree (GBDT) is an iterative DT algorithm that constructs multiple DTs and aggregates their prediction outputs. To improve the performance of basic GBDTs, three advanced gradient-boosting algorithms, XGBoost [10], LightGBM [11], and CatBoost [12], have been developed and widely used in many applications. These three gradient-boosting algorithms are used in the proposed system to build the LCCDE ensemble framework.

XGBoost is a popular gradient-boosting DT algorithm designed for the speed and performance improvement of GBDTs [10]. XGBoost uses a regularization term and a Second-Order Taylor Approximation for the summation of the squared errors to minimize the loss function and reduce over-fitting. XGBoost has a low computational complexity of $O(Kd||\mathbf{x}|| \log n)$, where K is the number of trees, d is the maximum tree depth, $||\mathbf{x}||$ is the number and non-missing samples, and n is the data size [10]. Additionally, XGBoost support parallel execution to save model learning time.

LightGBM is a fast and robust ensemble ML model constructed by multiple DTs [11]. LightGBM's key advantage over other ML methods is its capacity to efficiently handle large-scale and high-dimensional data. Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) are the two core strategies of LightGBM [11]. GOSS is a down-sampling method that only preserves data samples with large gradients and randomly discards small gradient samples to accelerate model training and reduce memory consumption. EFB is a feature engineering method that regroups mutually exclusive features into bundles as single features to minimize feature size and improve model training efficiency. By employing GOSS and EFB, the data size can be reduced significantly without the loss of critical information. The time and space complexity of LightGBM has also been reduced to $O(N'F')$, where N' is the reduced number of samples after using GOSS, F' is the bundled number of features after employing EFB [21].

CatBoost is another advanced gradient-boosting algorithm designed to process categorical features more effectively [12]. CatBoost, in comparison to existing gradient-boosting models, includes three significant model enhancement components: symmetric trees, ordered boosting, and native feature support. In symmetric trees, leaves are split under the same condition as in prior trees, and the pair of feature splits with the lowest loss is applied to all nodes. Using symmetric trees can improve model prediction speed and reduce over-fitting. Ordered boosting is a permutation-driven technique that prevents overfitting on small datasets by training a model on a subset while calculating residuals on another subset. CatBoost's native feature support indicates that it can directly process all types of features, such as numerical, textual, and categorical features, without the need for extra pre-processing. CatBoost is an ensemble model with low computational complexity of $O(SN)$, where S is the number of permutations of the subsets and *N* is the number of base DT models [12].

The primary reasons for selecting XGBoost, LightGBM, and CatBoost as base learners are as follows [10] - [12]:

1. These three ML models are all robust ensemble models that have had great success in a variety of data analytics applications [22].
2. These three ML models can automatically generate feature importance scores and select features during their training process, which saves time and resources by avoiding the need for extra feature engineering.
3. These three ML models are fast models with relatively low computational complexity. Additionally, they all support parallelization and Graphics Processing Unit (GPU) execution, which can further improve model learning speed.
4. These three ML models include randomness in their model construction process, enabling people to develop a robust ensemble model with high diversity and generalizability.

4.3.3 LCCDE: Proposed Ensemble Algorithm

The performance of different ML models often varies on different types of attack detection tasks. For example, when applying multiple ML models on the same network traffic dataset, a ML model perform the best for detecting the first type of attack (*e.g.,* DoS attacks), while another ML model may outperform other models for detecting the second type of attack (*e.g.,* sniffing attacks). Therefore, this work aims to propose an ensemble framework that can achieve optimal model performance for the detection of every type of attack. Ensemble learning is a technique that combines multiple base ML models to improve learning performance and generalizability [4]. The proposed ensemble model is constructed using XGBoost, LightGBM, and CatBoost, three advanced gradient-boosting ML methods introduced in Section 4.3.2.

Figure 4.2 demonstrates the process of the proposed LCCDE framework in two phases: model training and model prediction. The detailed procedures of the training and prediction phases are also described in Algorithms 2 & 3, respectively. At the training stage, the LCCDE framework aims to obtain leader models for all classes via the following steps:

Algorithm 2: Leader Class and Confidence Decision Ensemble (LCCDE) - Model Training

```
Input:
  $D_{train}$: the training set,
  $M = \{M_1, M_2, M_3\}$: the base ML model list, including $M_1$ = LightGBM, $M_2$ = XGBoost,
 $M_3$ = CatBoost,
  $c = 1, 2, \ldots, n$: the class list for $n$ different classes.
Output:
  $M = \{M_1, M_2, M_3\}$: the trained base model list,
  $LM = \{LM_1, LM_2, \ldots, LM_n\}$: the leader model list for all classes.
1  $M_1 \leftarrow Training(M_1, D_{train})$;                    // Train the LightGBM model
2  $M_2 \leftarrow Training(M_2, D_{train})$;                     // Train the XGBoost model
3  $M_3 \leftarrow Training(M_3, D_{train})$;                     // Train the CatBoost model
4  for $c = 1, 2, \ldots, n$ do      // For each class (normal or a type of attack), find the leader model
5      $Mlist_c \leftarrow BestPerforming(M_1, M_2, M_3, c)$;   // Find the best-performing model for each class
         (e.g., has the highest F1-score)
6      if $Len(Mlist_c) == 1$ then                          // If only one model has the highest F1
7          $LM_c \leftarrow Mlist_c[0]$;                 // Save this model as the leader model for the class $c$
8      else                                     // If multiple ML models have the same highest F1-score
9          $LM_c \leftarrow MostEfficient(Mlist_c)$;    // Save the fastest or most efficient model as the leader
             model for the class $c$
10     end
11     $LM \leftarrow LM \cup \{LM_c\}$;                        // Collect the leader model for each class
12 end
```

1. *Train three base learners.* The three base ML models (XGBoost, LightGBM, and CatBoost) are trained on the training set to obtain base learners.
2. *Evaluate base learners.* The performance of the three ML models for each class (normal or a type of attack) is evaluated using cross-validation and F1-scores. F1-scores are chosen because it is a comprehensive performance metric and works well with imbalanced datasets.
3. *Determine the leader model for each class.* For each class, the best-performing ML model with the highest F1-score is selected as the leader model for each class. If multiple top-performing ML models have identical highest F1-scores, the most efficient ML model with the highest speed is chosen as the final leader model.

After the training process, the trained leader models for all classes are utilized for model prediction. At the model prediction stage, the LCCDE framework predicts each test sample based on the following steps:

1. *Make initial predictions.* The three trained base ML models obtained from the training

Algorithm 3: Leader Class and Confidence Decision Ensemble (LCCDE) - Model Prediction

```
Input:
  $D_{test}$: the test set,
  $M = \{M_1, M_2, M_3\}$: the trained base ML model list, including $M_1$ = LightGBM, $M_2$ = XGBoost, $M_3$ = CatBoost,
  $c = 1, 2, \ldots, n$: the class list for $n$ different classes.
Output:
  $L_{test}$: the prediction classes for all test samples in $D_{test}$.
1  for each data sample $x_i \in D_{test}$ do                                  // For each test sample
2      $L_{i1}, p_{i1} \leftarrow Prediction(M_1, x_i)$;  // Use the trained LightGBM model to predict the sample, and save the predicted class & confidence
3      $L_{i2}, p_{i2} \leftarrow Prediction(M_2, x_i)$;                       // Use XGBoost to predict
4      $L_{i3}, p_{i3} \leftarrow Prediction(M_3, x_i)$;                       // Use CatBoost to predict
5      if $L_{i1} == L_{i2} == L_{i3}$ then       // If the predicted classes of all the three models are the same
6          $L_i \leftarrow L_{i1}$;                      // Use this predicted class as the final predicted class
7      else if $L_{i1} != L_{i2} != L_{i3}$ then    // If the predicted classes of all the three models are different
8          for $j = 1, 2, 3$ do                                              // For each prediction model
9              if $M_j == LM_{L_{i,j}}$ then // Check if the predicted class's original ML model is the same as its leader model
10                 $L\_list_i \leftarrow L\_list_i \cup \{L_{i,j}\}$;                     // Save the predicted class
11                 $p\_list_i \leftarrow p\_list_i \cup \{p_{i,j}\}$;                        // Save the confidence
12             end
13         end
14         if $Len(L\_list_i) == 1$ then // If only one pair of the original model and the leader model for each predicted class is the same
15             $L_j \leftarrow L\_list_i[0]$;  // Use the predicted class of the leader model as the final prediction class
16         else      // If no pair or multiple pairs of the original prediction model and the leader model for each predicted class are the same
17             if $Len(L\_list_i) == 0$ then
18                 $p\_list_i \leftarrow \{p_{i1}, p_{i2}, p_{i3}\}$;                  // Avoid empty probability list
19             end
20             $p\_max_i \leftarrow \max(p\_list_i)$;                             // Find the highest confidence
21             if $p\_max_i == p_{i1}$ then    // Use the predicted class with the highest confidence as the final prediction class
22                 $L_i \leftarrow L_{i1}$;
23             else if $p\_max_i == p_{i2}$ then
24                 $L_i \leftarrow L_{i2}$;
25             else
26                 $L_i \leftarrow L_{i3}$;
27             end
28         end
29     else                              // If two predicted classes are the same and the other one is different
30         $n \leftarrow mode(L_{i1}, L_{i2}, L_{i3})$;                // Find the predicted class with the majority vote
31         $L_i \leftarrow Prediction(M_n, x_i)$;   // Use the predicted class of the leader model as the final prediction class
32     end
33     $L_{test} \leftarrow L_{test} \cup \{L_i\}$;                  // Save the predicted classes for all tested samples;
34 end
```

process are used to make initial predictions. Their predicted classes and the corresponding prediction confidences are retained for further analysis. Confidence is a probability value used to quantify how confident the model is about its predictions.

2. *Check if the three predicted classes are the same.* If they are the same, the predicted class agreed by all three base learners is used as the final predicted class.
3. *Check if the three predicted classes are different.* If they are all different, the corresponding leader model for each predicted class is compared to the base learner that has predicted this class. If only one pair of the leader and base models match, their predicted class is used as the final predicted class; otherwise, the predicted class with the highest prediction confidence is used as the final predicted class.
4. *Check if two predicted classes are the same and the other one is different.* If so, the corresponding leader model of the same two predicted classes is used to make the final prediction.

In brief, LCCDE detects attacks based on the following three principles:

1. It uses the trained ML models to generate initially predicted classes.
2. It uses the leader models for each class to make the final predictions.
3. If there are multiple leader models for different classes, it selects the leader model with the highest prediction confidence to make final decisions.

The computational complexity of LCCDE is $O(NCK)$, where N is the data size, C is the number of classes, K is the complexity of base models. Thus, its complexity mainly depends on the complexity of all base ML models. In the proposed framework, three fast gradient-boosting ML algorithms are used to achieve low overall complexity. These three algorithms can be replaced by other ML algorithms using the same generic LCCDE strategy according to specific tasks. LCCDE is designed to address the difficult samples that cannot be correctly predicted by individual ML models. By using LCCDE, the final ensemble model can achieve optimal performance for detecting every type of attack.

4.4 Performance Evaluation

4.4.1 Experimental Setup

To develop the proposed IDS, the models were implemented using Scikit-learn, Xgboost [10], Lightgbm [11], and Catboost [12] libraries in Python. The experiments were conducted on a Dell Precision 3630 computer with an i7-8700 processor and 16 GB of memory, representing an IoV server machine.

The proposed LCCDE framework is evaluated on two public benchmark IoV network security datasets, Car-Hacking [13] and CICIDS2017 [14] datasets, representing the IVN and external network data, respectively. The Car-Hacking dataset [13] is created by transmitting CAN messages into a real vehicle's CAN bus. It has nine features (*i.e.,* CAN ID and the eight bits of the CAN message data field) and four types of attacks (*i.e.,* DoS, fuzzy, gear spoofing, and Revolutions Per Minute (RPM) spoofing attacks). The CICIDS2017 dataset [14] is a state-of-the-art general cyber-security dataset including the most updated types of attacks (*i.e.,* DoS, sniffing, brute-force, web-attacks, botnets, and infiltration attacks).

To evaluate the proposed LCCDE model, five-fold cross-validation is used in the training process to select leader class models, and 80%/20% hold-out validation is then used in the testing process to evaluate the model on the unseen test set. As network traffic data is often highly imbalanced and contains only a small proportion of attack samples, four performance measures, including accuracy, precision, recall, and F1-scores, are utilized to evaluate the model performance [4]. The execution time, including the model training and test time, is used to evaluate the efficiency of the model.

4.4.2 Experimental Results and Discussion

The experimental results of evaluating the three base ML models (LightGBM, XGBoost, CatBoost) and the proposed LCCDE model on the Car-Hacking and CICIDS2017 datasets are shown in Tables 4.1 - 4.4. Tables 4.1 and 4.2 illustrate the performance of the four models for detecting every type of attack in the two datasets based on their F1-scores. It is noticeable that the F1-scores of different base ML models vary for different types of attack detection.

Table 4.1: Model Performance Comparison for Each Class in The Car-Hacking Dataset

Method	**Car-Hacking Dataset**				
	F1 (%) of Class 1: Normal	**F1 (%) of Class 2: DoS**	**F1 (%) of Class 3: Fuzzy**	**F1 (%) of Class 4: Gear Spoofing**	**F1 (%) of Class 5: RPM Spoofing**
LightGBM [11]	**99.9998**	**100.0**	**99.995**	**100.0**	**100.0**
XGBoost [10]	99.9996	**100.0**	99.990	**100.0**	**100.0**
CatBoost [12]	99.9996	**100.0**	99.990	**100.0**	**100.0**
Proposed LCCDE	**99.9998**	**100.0**	**99.995**	**100.0**	**100.0**

Table 4.2: Model Performance Comparison for Each Class in The CICIDS2017 Dataset

Method	**CICIDS2017 Dataset**						
	F1 (%) of Class 1: Normal	**F1 (%) of Class 2: DoS**	**F1 (%) of Class 3: Sniffing**	**F1 (%) of Class 4: Brute-Force**	**F1 (%) of Class 5: Web Attack**	**F1 (%) of Class 6: Botnets**	**F1 (%) of Class 7: Infiltra-tion**
LightGBM [11]	99.863	**100.0**	**99.889**	99.222	**99.354**	**100.0**	**85.714**
XGBoost [10]	99.863	**100.0**	**99.889**	**99.351**	99.137	**100.0**	**85.714**
CatBoost [12]	99.794	99.754	99.557	99.094	**99.354**	**100.0**	**85.714**
Proposed LCCDE	**99.876**	**100.0**	**99.889**	**99.351**	**99.354**	**100.0**	**85.714**

Table 4.3: Performance Evaluation of Models on Car-hacking Dataset

Method	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Execution Time (s)**
KNN [23]	97.4	96.3	98.2	93.4	195.6
SVM [23]	96.5	95.7	98.3	93.3	1345.3
LSTM-AE [24]	99.0	99.0	99.9	99.0	-
DCNN [16]	99.93	99.84	99.84	99.91	-
LightGBM [11]	99.9997	99.9997	99.9997	99.9997	10.7
XGBoost [10]	99.9994	99.9994	99.9994	99.9994	45.3
CatBoost [12]	99.9994	99.9994	99.9994	99.9994	88.6
Proposed LCCDE	**99.9997**	**99.9997**	**99.9997**	**99.9997**	185.1

Table 4.4: Performance Evaluation of Models on CICIDS2017

Method	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Execution Time (s)**
KNN [14]	96.3	96.2	93.7	96.3	1558.3
RF [14]	98.82	98.8	99.955	98.8	135.1
DBN [19]	98.95	95.82	95.81	95.81	-
Stacking [18]	99.80	99.75	99.89	99.70	278.6
LightGBM [11]	99.794	99.795	99.794	99.792	14.3
XGBoost [10]	99.794	99.795	99.794	99.792	44.7
CatBoost [12]	99.683	99.684	99.683	99.680	73.7
Proposed LCCDE	**99.813**	**99.814**	**99.913**	**99.811**	168.9

For example, on the CICIDS2017 dataset, LightGBM achieves the highest F1-score among the three base learners for detecting normal samples, DoS, sniffing, webattacks, botnets, and infiltration attacks, while XGBoost outperforms LightGBM for the brute-force attack detection. As shown in Table 4.1 and 4.2, the proposed LCCDE ensemble model can achieve the highest F1-score for every class. Thus, as shown in Tables 4.3 and 4.4, the overall F1-scores of the proposed model are also the highest among the four utilized ML models on the two datasets. The proposed LCCDE model achieves a near-perfect F1-score on the Car-Hacking dataset (99.9997%), and improved its F1-score from 99.792% to 99.811% on the CICIDS2017 dataset. This demonstrates the benefits of identifying the best-performing base models for each class to construct the LCCDE ensemble model.

Tables 4.3 and 4.4 also compare the performance of the proposed technique with existing state-of-the-art methods [14] [16] [18] [19] [23] [24] on the two datasets. The proposed LCCDE model outperforms other methods by at least 0.09% and 0.11% F1-score improvements on the Car-Hacking and CICIDS2017 datasets, respectively. As an ensemble approach, the proposed LCCDE model has a longer execution time than the other three base gradient-boosting models, but it is still faster than many other ML algorithms, such as K-Nearest Neighbors (KNN) and Support Vector Machine (SVM). This is because the proposed ensemble model is built using low complexity ML models with parallel execution and GPU support. To summarize, the proposed model can achieve the highest F1-scores among the compared methods with relatively low execution time on the two benchmark datasets.

4.5 Conclusion

For the purpose of enhancing IoV security, Machine Learning (ML) algorithms have been used as promising solutions to detect various types of cyber-attacks. However, ML models often perform differently for different types of attack detection. To achieve optimal performance on all types of attack detection in IoV networks, a novel ensemble method, namely Leader Class and Confidence Decision Ensemble (LCCDE), is proposed in this chapter. It identifies the best-performing ML models for each type of attack detection as the leader class models to construct a robust ensemble model. Additionally, the prediction confidence information is utilized to help determine the final prediction classes. Three advanced gradient-boosting ML algorithms, XGBoost, LightGBM, and CatBoost, are utilized to construct the proposed LC-CDE ensemble model due to their high effectiveness and efficiency. Through the experiments, the proposed IDS framework achieves high F1-scores of 99.9997% and 99.811% on the Car-Hacking and CICIDS2017 datasets, representing intra-vehicle and external vehicular network data, respectively. Moreover, the proposed model's F1-scores are higher than other compared ML methods for detecting every type of attack. This illustrates the benefits of the proposed leader class-based strategy.

Bibliography

[1] L. Yang, A. Shami, G. Stevens, and S. DeRusett, "LCCDE: A Decision-Based Ensemble Framework for Intrusion Detection in The Internet of Vehicles," submitted to *2022 IEEE Global Communications Conference (GLOBECOM)*, 2022, pp. 1-6.

[2] H. Bangui and B. Buhnova, "Recent Advances in Machine-Learning Driven Intrusion Detection in Transportation: Survey," *Procedia Comput. Sci.*, vol. 184, pp. 877–886, 2021.

[3] O. Y. Al-Jarrah, C. Maple, M. Dianati, D. Oxtoby, and A. Mouzakitis, "Intrusion Detection Systems for Intra-Vehicle Networks: A Review," *IEEE Access*, vol. 7, pp. 21266–21289, 2019.

[4] L. Yang and A. Shami, "A Transfer Learning and Optimized CNN Based Intrusion Detection System for Internet of Vehicles," *in 2022 IEEE Int. Conf. Commun. (ICC)*, 2022, pp. 1–6.

[5] K. Kim, J. S. Kim, S. Jeong, J.-H. Park, and H. K. Kim, "Cybersecurity for autonomous vehicles: Review of attacks and defense," *Comput. Secur.*, vol. 103, p. 102150, 2021.

[6] L. Yang, A. Moubayed, and A. Shami, "MTH-IDS: A Multitiered Hybrid Intrusion Detection System for Internet of Vehicles," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 616–632, 2022.

[7] J. Jiang, F. Liu, W. W. Y. Ng, Q. Tang, W. Wang, and Q.-V. Pham, "Dynamic Incremental Ensemble Fuzzy Classifier for Data Streams in Green Internet of Things," *IEEE Trans. Green Commun. Netw.*, p. 1, 2022.

[8] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, "Machine learning towards intelligent systems: applications, challenges, and opportunities," *Artif. Intell. Rev.*, 2021.

[9] L. Yang, D. M. Manias, and A. Shami, "PWPAE: An Ensemble Framework for Concept Drift Adaptation in IoT Data Streams," in *proc. 2021 IEEE Glob. Commun. Conf.*, pp. 1–6, 2021.

[10] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[11] G. Ke *et al.*, "LightGBM: A highly efficient gradient boosting decision tree," *Adv. Neural Inf. Process. Syst.*, vol. 2017-December, no. Nips, pp. 3147–3155, 2017.

[12] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, “Catboost: Unbiased boosting with categorical features,” *Adv. Neural Inf. Process. Syst.*, vol. 2018-December, no. Section 4, pp. 6638–6648, 2018.

[13] E. Seo, H. M. Song, and H. K. Kim, “GIDS: GAN based Intrusion Detection System for In-Vehicle Network,” *2018 16th Annu. Conf. Privacy, Secur. Trust*, pp. 1–6, 2018.

[14] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” *in Proc. Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[15] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, “Multi-Stage Optimized Machine Learning Framework for Network Intrusion Detection,” *IEEE Trans. Netw. Serv. Manag.*, vol. 4537, no. c, pp. 1–14, 2020.

[16] H. M. Song, J. Woo, and H. K. Kim, “In-vehicle network intrusion detection using deep convolutional neural network,” *Veh. Commun.*, vol. 21, p. 100198, 2020.

[17] R. Zhao *et al.*, “A Novel Intrusion Detection Method Based on Lightweight Neural Network for Internet of Things,” *IEEE Internet Things J.*, p. 1, 2021.

[18] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, “Tree-Based Intelligent Intrusion Detection System in Internet of Vehicles,” in *proc. 2019 IEEE Glob. Commun. Conf.*, pp. 1–6, 2019.

[19] W. Elmasry, A. Akbulut, and A. H. Zaim, “Evolving deep learning architectures for network intrusion detection using a double PSO metaheuristic,” *Comput. Networks*, vol. 168, 2020.

[20] Z. Chen, M. Simsek, B. Kantarci, and P. Djukic, “All Predict Wisest Decides: A Novel Ensemble Method to Detect Intrusive Traffic in IoT Networks,” in *proc. 2021 IEEE Glob. Commun. Conf.*, pp. 1–6, 2021.

[21] L. Yang and A. Shami, “A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams,” *IEEE Internet Things Mag.*, vol. 4, no. 2, pp. 96–101, 2021.

[22] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[23] A. Alshammari, M. A. Zohdy, D. Debnath, and G. Corser, “Classification Approach for Intrusion Detection in Vehicle Systems,” *Wirel. Eng. Technol.*, vol. 09, no. 04, pp. 79–94, 2018.

[24] J. Ashraf, A. D. Bakhshi, N. Moustafa, H. Khurshid, A. Javed, and A. Beheshti, “Novel Deep Learning-Enabled LSTM Autoencoder Architecture for Discovering Anomalous Events From Intelligent Transportation Systems,” *IEEE Trans. Intell. Transp. Syst.*, pp. 1–12, 2020.

Chapter 5

A Transfer Learning and Optimized CNN Based Intrusion Detection System for Internet of Vehicles

5.1 Introduction

With the rapid development of the Internet of Things (IoT) and the Internet of Vehicles (IoV) technologies, modern vehicles have been evolving to network-controlled vehicles, including Autonomous Vehicles (AVs) and Connected Vehicles (CVs) [2]. Typical IoV systems involve intra-vehicle networks (IVNs) and external networks. In IVNs, Controller Area Network (CAN) bus is the central system that enables communications between Electronic Control Units (ECUs) to perform actions and adopt functionalities. On the other hand, external vehicular networks allow the connection between smart vehicles and other entities in IoV, including road-side units, infrastructures, and road users [3].

However, the improving connectivity and accessibility of vehicular networks has increased the cyber-attack surfaces of modern vehicles [4]. Additionally, due to the limited length of CAN packets, there is no authentication or encryption strategies involved in the processing of these packets [2]. The lack of fundamental security measures enables cyber-attackers to

A version of this chapter has been published in IEEE ICC 2022 [1].

inject malicious messages to IVNs and launch different types of attacks, like Denial of Service (DoS), fuzzy, and spoofing attacks. On the other hand, the emerging cellular connections between connected vehicles and external networks have made these vehicles vulnerable to various conventional cyber-attacks [5]. Therefore, it is crucial to develop Intrusion Detection Systems (IDSs) to protect IoV systems and smart vehicles by identifying cyber-attacks [6].

Recently, due to the progression of Machine Learning (ML) and Deep Learning (DL) techniques, their applications in cyber-security and vehicle systems have attracted the attention of researchers and automotive manufacturers [7] [8]. ML and DL techniques are widely used to develop classifier-based IDSs that can distinguish between normal network traffic and different cyber-attacks through traffic data analytics [9]. In this chapter, an intelligent IDS model based on optimized Convolutional Neural Networks (CNNs), transfer learning, and ensemble learning techniques is proposed to protect IoV systems. Five advanced CNN models, including VGG16, VGG19, Xception, Inception, and InceptionResnet [10], are used to train base learners on vehicle network traffic data. The hyper-parameters of the CNN models are tuned using Particle Swarm Optimization (PSO), a hyper-parameter optimization (HPO) method, to obtain optimized learning models [11]. The base CNN models are then integrated using two ensemble strategies, confidence averaging and concatenation, to further improve the intrusion detection performance. The effectiveness and efficiency of the proposed IDS framework are evaluated using two public vehicle network datasets: the Car-Hacking dataset [12] and the CICIDS2017 dataset [13].

This chapter mainly makes the following contributions:

1. It proposes a novel framework for effective cyber-attack detection in both intra-vehicle and external networks through CNN, transfer learning, ensemble learning, and HPO techniques.
2. It proposes a data transformation method that can effectively transform vehicle network traffic data into images to more easily distinguish various cyber-attack patterns.
3. It evaluates the proposed method on two benchmark cyber-security datasets that represent intra-vehicle and external network data, and compares the model's performance with

Code is available at: https://github.com/Western-OC2-Lab/Intrusion-Detection-System-Using-CNN-and-Transfer-Learning other state-of-the-art methods.

To the best of our knowledge, no previous work proposed such an optimized IDS model that integrates CNN, transfer learning, ensemble learning, and HPO techniques to effectively detect various types of attacks on both intra-vehicle and external networks.

The rest of the chapter is organized as follows. Section 5.2 introduces the related work that uses ML and DL algorithms for vehicle network intrusion detection. Section 5.3 presents the proposed framework, including data transformation, CNN, transfer learning, ensemble learning, and HPO techniques. Section 5.4 presents and discusses the experimental results. Finally, Section 5.5 summarizes the chapter.

5.2 Related Work

ML and DL models have been widely used in IoV intrusion detection tasks. Rosay *et al.* [5] proposed a DL-based IDS for connected vehicles using Multi-Layer Perceptron (MLP). The MLP model was evaluated on an automotive microprocessor using the CICIDS2017 dataset. Yang *et al.* [3] [4] proposed a tree-based stacking algorithm for network traffic analysis in IoV environments. The proposed stacking method shows high performance on the IoV and CICIDS2017 datasets.

Several existing works focused on CNN-based IDS development for vehicular networks. Mehedi *et al.* [2] proposed the P-LeNet method for in-vehicle network intrusion detection based on deep transfer learning. The P-LeNet model achieved a high F1-score of 97.83% on the Car-Hacking dataset. Hossain *et al.* [6] proposed a one-dimensional CNN (1D-CNN) based IDS for intra-vehicle intrusion detection, as 1D-CNN models work well in many time-series data analytics problems. Song *et al.* [8] proposed a deep CNN (DCNN) based IDS model using reduced InceptionResnet to detect attacks in IVNs. The DCNN model shows high accuracy on the Car-Hacking dataset.

Although the above methods achieve high accuracy in IoV cyber-attack detection tasks, there is still much room for performance improvement. The proposed solution aims to construct an optimal IDS framework using state-of-the-art CNN models optimized using HPO and ensemble learning strategies. Additionally, transfer learning techniques are used to improve

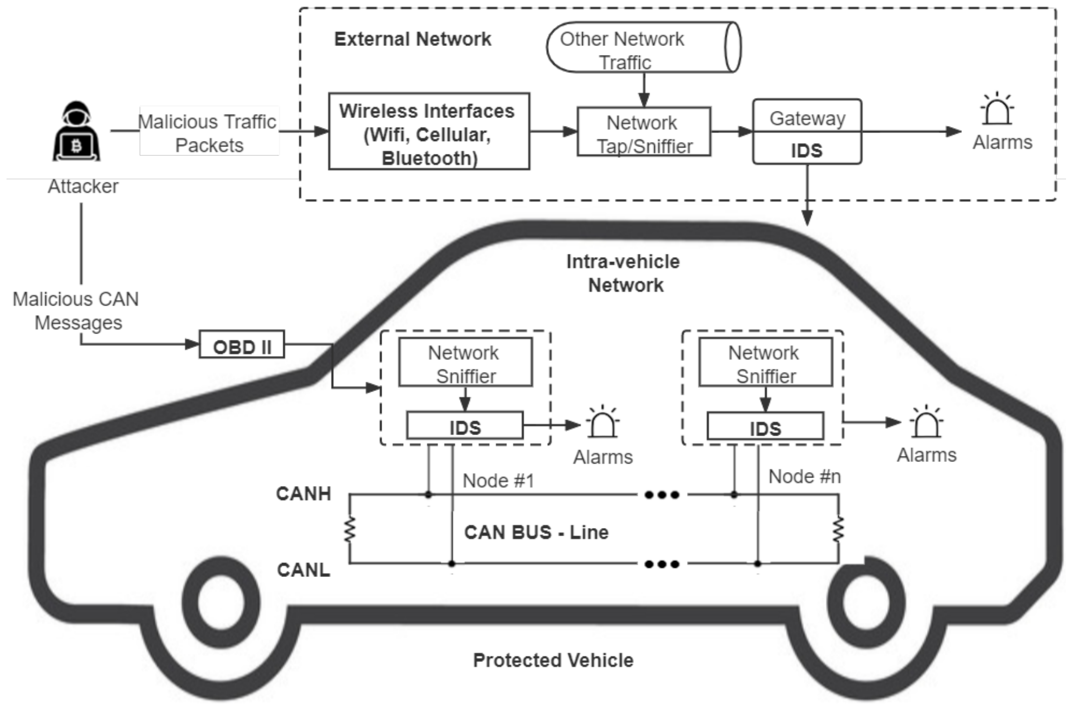

Figure 5.1: The IDS-protected vehicle architecture.

the model training efficiency.

5.3 Proposed Framework

5.3.1 System Overview

The purpose of this work is to develop an IDS that can detect various types of attacks in intra-vehicle and external vehicular networks to protect them both. The typical attack scenario and the architecture of an IDS-protected vehicle are shown in Fig. 5.1. Cyber-attackers can launch internal attacks on IVNs through the On-Board Diagnostics II (OBD II) interface and launch external attacks to external vehicular networks through wireless interfaces by sending malicious traffic packets. Thus, the proposed IDS should be deployed in both IVNs and external networks. In IVNs, the proposed IDS can be deployed on top of the CAN-bus to detect abnormal CAN messages and generate alarms [4]. In external networks, the proposed IDS can be incorporated into the gateways to identify and block all malicious packets that aim to breach the vehicles [3].

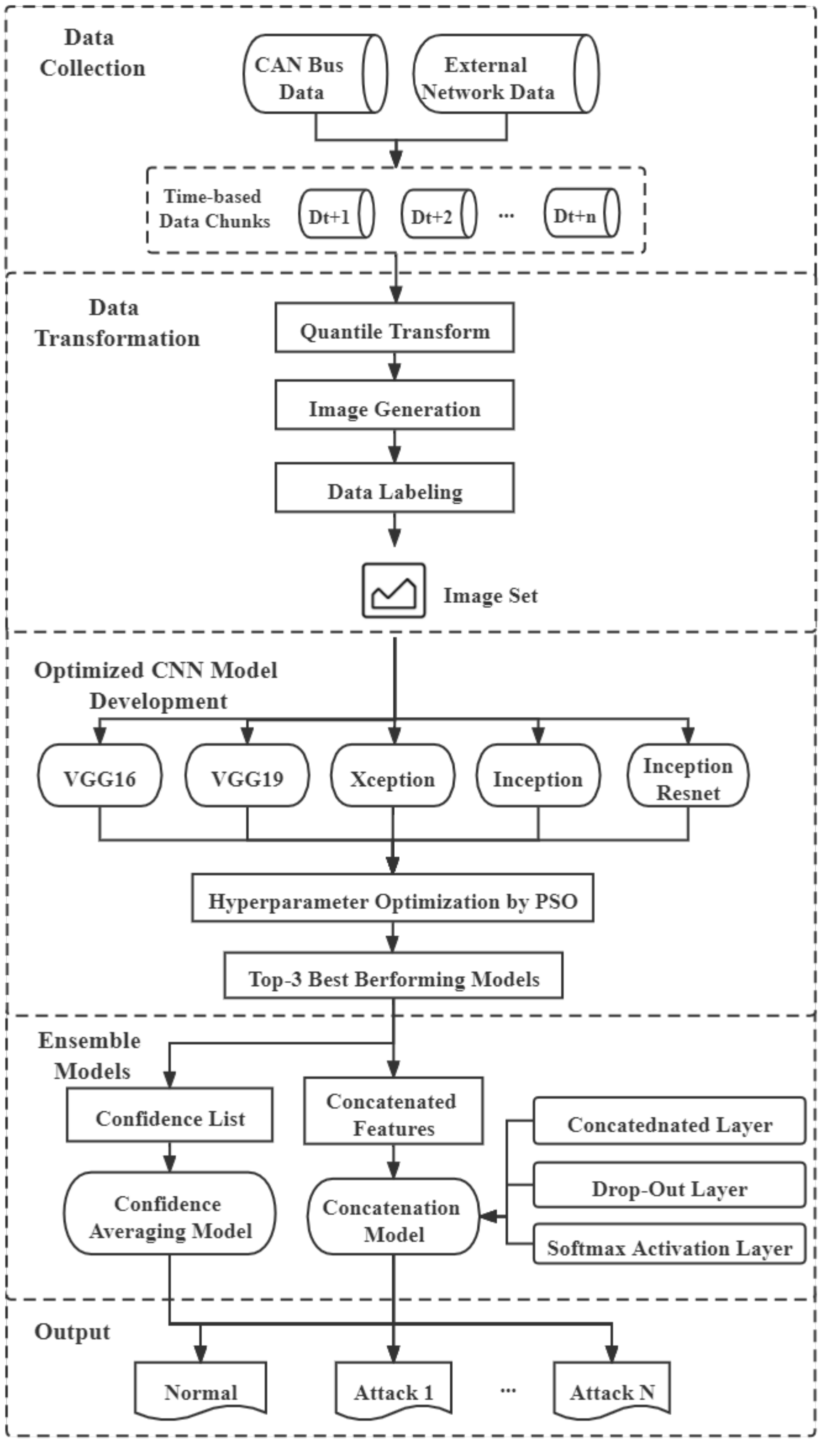

Figure 5.2: The proposed optimized CNN-based IDS framework.

In this chapter, a novel optimized CNN and transfer learning-based IDS is proposed to detect various types of attacks in IoV systems. Figure 5.2 demonstrates the overview of the proposed IDS framework. Firstly, the intra-vehicle and external network data are collected in time-based chunks and then transformed into images using the quantile transform method. At the next stage, the generated image set is trained by five state-of-the-art CNN models (VGG16, VGG19, Xception, Inception, and InceptionResnet) to construct base learners. The CNN models are optimized by PSO, a HPO method that can automatically tune the hyper-parameters. After that, the top-3 best performing CNN models are selected as the three base CNN models to construct the ensemble learning models. Lastly, two ensemble strategies, confidence averaging and concatenation, are used to construct ensemble models for final detection.

5.3.2 Data Description and Transformation

To develop the proposed IDS for both IVNs and external vehicular networks, two datasets are used in this work. The first dataset is the Car-Hacking dataset [12] that represents intra-vehicle data, as it is generated by transmitting CAN packets into the CAN-bus of a real vehicle. The CAN identifier (ID) and 8-bit data field of CAN packets (DATA[0]-DATA[7]) are the main features of the dataset. The Car-Hacking dataset involves four main types of attacks: DoS, fuzzy, gear spoofing, and Revolutions Per Minute (RPM) spoofing attacks. The second dataset used is the CICIDS2017 dataset [13] that represents external network data, as it is a state-of-the-art network security dataset that includes the most updated attack patterns. According to the dataset analysis in [4] [14], the attack patterns in the CICIDS2017 dataset can be summarized into five main types of attacks: DoS attacks, port-scan attacks, brute-force attacks, web-attacks, and botnets.

After acquiring the data, it should be pre-processed to generate a proper input for the proposed IDS. As CNN models work better on image sets and vehicular network traffic datasets are usually tabular data, the original network data should be transformed into image forms [15].

The data transformation process starts with data normalization. Since the pixel values of images range from 0 to 255, the network data should also be normalized into the scale of 0-255. Among the normalization techniques, min-max and quantile normalization are the two commonly used methods that can convert data values to the same range. As min-max normalization does not handle outliers well and may cause most data samples to have extremely small values, quantile normalization is used in the proposed framework [16]. The quantile normalization method transforms the feature distribution to a normal distribution and re-calculates all the features values based on the normal distribution. Therefore, the majority of variable values are close to the median values, which is effective in handling outliers [16].

After data normalization, the data samples are converted into chunks based on the timestamps and feature sizes of network traffic datasets. For the Car-Hacking dataset, as it has 9 important features (CAN ID and DATA[0]-DATA[7]), each chunk of 27 consecutive samples with 9 features ($27 \times 9 = 243$ feature values in total) are transformed into an image of shape $9 \times 9 \times 3$ [15]. Thus, each transformed image is a square color image with three channels (red, green, and blue). Similarly, the CICIDS2017 dataset with 20 important features generated from [3] is transformed to $20 \times 20 \times 3$ color images, so each chunk of this dataset consists of $20 \times 3 = 60$ consecutive data samples. As the images are generated based on the timestamps of the data samples, the time-series correlations of the original network data can be retained.

In the next step, the transformed images are labeled based on the attack patterns in the data chunks. If all the samples in a chunk/image are normal samples, this image is labeled "Normal". On the other hand, if a chunk/image contains attack samples, this image is labeled as the most frequent attack type in this chunk. For example, if a DoS attack occurs in a chunk with the highest proportion, the corresponding image will be labeled "DoS attack".

After the above data pre-processing procedures, the final transformed image set is generated as the input of CNN models. The representative samples for each type of attack in the Car-Hacking dataset and the CICIDS2017 dataset are shown in Fig. 5.3 and Fig. 5.4. For the Car-Hacking dataset, it can be seen from Fig. 5.3 that there are large differences in the feature patterns between the normal samples and different types of attacks. The feature patterns of fuzzy attack images are more random than normal images, while DoS attack samples are high-frequency empty messages, causing pure black patterns. Gear and RPM spoofing attacks are launched by injecting messages with certain CAN IDs and packets to masquerade as legitimate users, so their images also have fixed feature patterns [12]. Similarly, the attack patterns of CICIDS2017 can be obviously distinguished according to the feature patterns shown in Fig. 5.4.

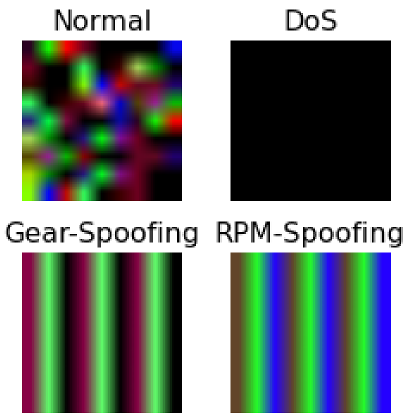

Figure 5.3: The representative sample images of each class in the Car-Hacking dataset.

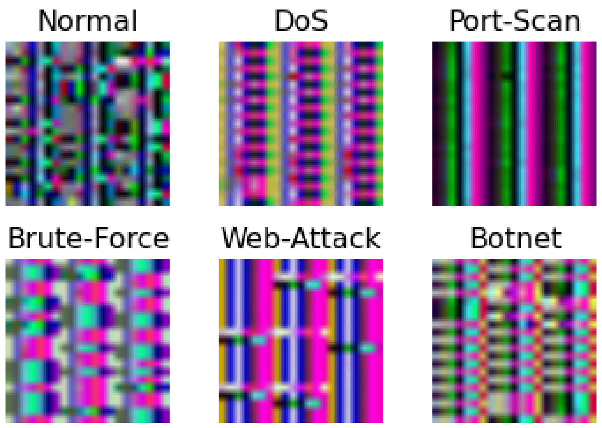

Figure 5.4: The representative sample images of each class in the CICIDS2017 dataset.

5.3.3 CNN and Transfer Learning

CNN is a common DL model that is widely used in image classification and recognition problems [8]. The images can be directly inputted into CNN models without additional feature extraction and data reconstruction processes. A typical CNN comprises three types of layers: convolutional layers, pooling layers, and fully-connected layers [8]. In convolutional layers, the feature patterns of images can be automatically extracted by convolution operations. In pooling layers, the data complexity can be reduced without losing important information through local correlations to avoid over-fitting. Fully-connected layers serve as a conduit to connect all features and generate the output.

For DL models, Transfer Learning (TL) is the process of transferring the weights of a Deep Neural Network (DNN) model trained on one dataset to another dataset [17]. The TL technique has been successfully applied to many image processing tasks. This is because the feature patterns learned by the bottom layers of CNN models are usually general patterns that are applicable to many different tasks, and only the features learned by the top layers are specific features for a particular dataset [17]. Therefore, the bottom layers of CNN models can be directly transferred to different tasks. To improve the effectiveness of TL, fine-tuning can be used in the TL process of DL models. In fine-tuning, most of the layers of the pre-trained model are frozen (*i.e.*, their weights are retained), while a few of the top layers are unfrozen to re-train the model on a new dataset. Fine-tuning enables the learning model to update the higher-order features in the pre-trained model to better fit the target task or dataset [17].

In the proposed framework, we have selected VGG16, VGG19, Xception, Inception, and InceptionResnet as the base CNN models due to their success in most image classification problems [10]. These CNN models are pre-trained on the ImageNet dataset and have demonstrated great performance on general image classification tasks. The ImageNet dataset is a benchmark image processing dataset that has more than one million images of 1,000 classes [10].

The VGG16 models with 16 layers (VGG16) and with 19 layers (VGG19) proposed in [18] have achieved a reduced error rate of 7.3% on the ImageNet Challenge. The VGG16 architecture comprises five blocks of convolutional layers and three fully connected layers, while the VGG19 architecture has three more convolutional layers. The Inception network introduced in [19] uses convolutional feature extractors that combine different contexts to obtain different types of feature patterns, which reduces the computational cost through dimensionality reduction. Xception [20] is an extension of the Inception network that uses depthwise separable convolutions to replace the standard network convolutions. The memory requirement of Xception is slightly smaller than Inception. InceptionResnet is another extension of Inception that incorporates the residual connections from Resnet into the Inception network [10]. InceptionResnet outperforms Inception models on image classification challenges, but it requires twice the computational operations and memory than Inception.

After using transfer learning and fine-tuning to train five state-of-the-art CNN models on the vehicle network datasets, the top-3 best performing CNN models are selected as the base learners to construct the ensemble models that are introduced in the next subsection.

5.3.4 Proposed Ensemble Learning Models

Ensemble learning is a technique that integrates multiple base learning models to construct an ensemble model with improved performance. Ensemble learning is widely used in data analytics problems because an ensemble of multiple learners usually performs better than single learners [3].

Confidence averaging is an ensemble learning approach that combines the classification probability values of base learners to find the class with the highest confidence value [21]. In DL models, softmax layers can output a posterior probability list that contains the classification confidence of each class. The confidence averaging method calculates the average classification probability of base learners for each class, and then returns the class label with the highest average confidence value as the final classification result. The confidence value of each class is calculated using the softmax function [21]:

$$\text{Softmax}(\mathbf{z})_i = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{C} e^{\mathbf{z}_j}} \tag{5.1}$$

Where $\mathbf{z}$ is the input vector, C is the number of classes in the dataset, $e^{\mathbf{z}_i}$ and $e^{\mathbf{z}_j}$ are the standard exponential functions for the input and output vectors, respectively.

The predicted class label obtained by the confidence averaging method can be denoted by:

$$\hat{y} = \underset{i \in \{1, \cdots, c\}}{\text{argmax}} \frac{\sum_{j=1}^{k} p_j \left(y = i \mid B_j, \boldsymbol{x}\right)}{k} \tag{5.2}$$

Where B_j is the j_{th} base learner, k is the number of selected base CNN learners, and $k = 3$ in the proposed IDS; $p_j(y = i \mid B_j, \boldsymbol{x})$ indicates the prediction confidence of a class value i in a data sample $\boldsymbol{x}$ using B_j.

Unlike the conventional voting method that only considers the class labels, confidence averaging enables the ensemble model to detect uncertain classification results and correct the misclassified samples through the use of classification confidence. The computational complexity of an entire ensemble model depends on the complexity of base learners, while the time complexity for the confidence averaging method itself is only $O(NKC)$, where N is the number of instances, K is the number of base CNN models, and C is the number of classes [22]. As

K and C are usually small, the execution speed of the confidence averaging method is usually high.

Concatenation [23] is another ensemble strategy for DL models. A concatenated CNN aims to extract the highest order features generated from the top dense layer of base CNN models and use concatenate operations to integrate all the features into a new concatenated layer that contains all the features. The concatenated layer is followed by a drop-out layer to reduce redundant features and a softmax layer to construct a new CNN model. The advantage of concatenation is that it can combine the highest order features to construct a comprehensive new model. However, as the new model needs to be re-trained on the entire dataset, it introduces additional model training time. The computational complexity of the concatenation method is $O(NF)$, where N is the number of data samples, and F is the total number of features extracted from the dense layers of the base CNN models.

5.3.5 Hyper-Parameter Optimization (HPO)

To better fit the base models to the selected datasets and further improve the models' performance, the hyper-parameters of CNN models need to be tuned and optimized

Similar to other DL models, CNN models have a large number of hyper-parameters that need tuning. These hyper-parameters can be classified as model-design hyper-parameters and model-training hyper-parameters [11]. Model-design hyper-parameters are the hyper-parameters that should be set in the model design process. In the proposed TL framework, the model-design hyper-parameters include the number or percentage of frozen layers, the learning rate, and the drop-out rate. On the other hand, model-training hyper-parameters are used to balance the training speed and model performance, involving the batch size, the number of epochs, and early stop patience. The above hyper-parameters have a direct impact on the structure, effectiveness, and efficiency of CNN models.

HPO is an automated process of tuning hyper-parameters of ML or DL models using optimization techniques [11]. Among the optimization techniques used for HPO problems, PSO is a widely-used metaheuristic optimization method that identifies optimal hyper-parameter values via the information sharing and cooperation among the particles in a swarm [11]. At the initial stage of PSO, each individual in the group is initialized with a position $\overrightarrow{x_i}$ and velocity $\overrightarrow{v_i}$. After each iteration, the velocity of each particle are updated based on its own current best position $\overrightarrow{p_i}$ and the current global optimal position $\overrightarrow{p}$ shared by other individuals:

$$\overrightarrow{v_i} := \overrightarrow{v_i} + U(0, \varphi_1)(\overrightarrow{p_i} - \overrightarrow{x_i}) + U(0, \varphi_2)(\overrightarrow{p} - \overrightarrow{x_i}), \tag{5.3}$$

where $U(0, \varphi)$ is the continuous uniform distribution calculated by the acceleration constants φ_1 and φ_2.

Finally, the particles can gradually move towards the promising regions to identify the global optimum. PSO is chosen in the proposed framework due to its support to different types of hyper-parameters and its low time complexity of $O(NlogN)$ [11].

5.4 Performance Evaluation

5.4.1 Experimental Setup

The experiments were conducted using Scikit-learn and Keras libraries in Python. In the experiments, the proposed DL models were trained on a Dell Precision 3630 machine with an i7-8700 processor and 16 GB of memory and tested on a Raspberry Pi 3 machine with a BCM2837B0 64-bit CPU and 1 GB of memory, representing an IoV central server machine and a vehicle-level local machine, respectively.

The proposed framework is evaluated on two benchmark vehicle network security datasets, Car-Hacking [12] and CICIDS2017 [13] datasets, as described in Section 5.3.2. Five-fold cross-validation is used to evaluate the proposed model, which can avoid over-fitting and biased results. On the other hand, as network traffic data is usually highly imbalanced data that only has a small percentage of attack samples, four different metrics, including accuracy, precision, recall, and F1-scores, are used for performance evaluation. Furthermore, to evaluate the efficiency of the proposed method, model training time on the server-level machine and model testing time on the vehicle-level machine are also monitored and compared.

Table 5.1: Hyper-parameter Configuration of CNN models

Hyper-Parameter	**Model**	**Search Range**	**Optimal Value**
Number of epochs	General (All CNN models)	[5, 50]	20
Batch size		[32, 128]	128
Early stop patience		[2, 5]	3
Learning rate		(0.001, 0.1)	0.003
Drop-out rate		(0.2, 0.8)	0.5
Number of frozen layers	Xception	[60,125]	121
	Vgg16	[8, 16]	15
	Vgg19	[10, 19]	19
	Inception	[80, 159]	148
	InceptionResnet	[300, 572]	522

Table 5.2: Performance Evaluation of Models on Car-hacking Dataset

Method	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Training Time (s)**	**Test Time Per Packet (ms)**
P-LeNet [2]	98.10	98.14	98.04	97.83	-	-
1D-CNN [6]	99.96	99.94	99.63	99.80	-	-
DCNN [8]	99.93	99.84	99.84	99.91	-	-
VGG16-PSO	99.97	99.97	99.97	99.97	384.9	0.2
VGG19-PSO	100.0	100.0	100.0	100.0	417.9	0.2
Xception-PSO	100.0	100.0	100.0	100.0	529.2	0.3
Inception-PSO	100.0	100.0	100.0	100.0	733.6	0.6
InceptionResnet -PSO	100.0	100.0	100.0	100.0	970.4	1.3
Concatenation (Proposed)	**100.0**	**100.0**	**100.0**	**100.0**	**2490.5**	**3.2**
Confidence Averaging (Proposed)	**100.0**	**100.0**	**100.0**	**100.0**	**1680.7**	**2.7**

5.4.2 Experimental Results and Discussion

To construct optimal models, the major hyper-parameters of all the base CNN models in the proposed framework were optimized using PSO. As CNN models with default hyper-parameter values can already achieve near 100% accuracy on the Car-Hacking dataset, the HPO process was only implemented for the CICIDS2017 dataset. Table 5.1 illustrates the initial search range and the optimal values of the hyper-parameters. After HPO, the optimized CNN models were used as base learners to construct the proposed ensemble models.

The results of evaluating the optimized CNN models and the proposed ensemble models on the Car-Hacking and CICIDS2017 datasets are shown in Tables 5.2 & 5.3, respectively. As shown in Table 5.2, all optimized base CNN models except VGG16 achieve 100% accuracy

Table 5.3: Performance Evaluation of Models on CICIDS2017 Dataset

Method	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Training Time (s)**	**Test Time Per Packet (ms)**
KNN [13]	96.3	96.2	93.7	96.3	15243.6	0.2
RF [13]	98.82	98.8	99.955	98.8	1848.3	0.3
MLP [5]	99.46	99.52	99.40	99.46	-	1.1
VGG16-PSO	99.724	99.625	99.724	99.674	436.5	0.1
VGG19-PSO	99.849	99.850	99.849	99.850	688.1	0.1
Xception-PSO	99.699	99.700	99.699	99.697	655.5	0.2
Inception-PSO	99.750	99.725	99.750	99.729	782.8	0.3
InceptionResnet -PSO	99.849	99.850	99.849	99.850	1187.2	0.7
Concatenation (Proposed)	**99.899**	**99.900**	**99.899**	**99.898**	**3598.7**	**1.8**
Confidence Averaging (Proposed)	**99.925**	**99.925**	**99.924**	**99.925**	**2658.1**	**1.5**

and F1-scores. This is mainly because the normal and attack patterns in the Car-Hacking dataset can be obviously distinguished through the transformed images shown in Fig. 3. The two ensemble techniques, concatenation and confidence averaging methods, can also achieve 100% F1-scores, while the total training time of confidence averaging is much lower than concatenation (1680.7 s versus 2490.5 s). Therefore, the confidence averaging method is more efficient. Moreover, the performance of the proposed models is compared with other state-of-the-art methods [2] [6] [8]. As shown in Table 5.2, most of the compared approaches achieve high accuracy due to the simplicity of the Car-Hacking dataset. Among the models shown in Table 5.2, the proposed ensemble methods show the best performance by achieving at least 0.09% F1-score improvements.

For the CICIDS2017 dataset, the optimized base CNN models achieve high F1-scores of 99.674% to 99.850% after implementing data transformation and PSO, as shown in Table 5.3. The proposed confidence averaging ensemble model also achieves the highest F1-score of 99.925%, which is slightly higher than the F1-score of the concatenation model (99.899%). The two ensemble models also outperform other recent methods in the literature [5] [13]. Additionally, the total training time of the confidence averaging is also much lower than the concatenation approach.

The higher performance of the proposed models when compared with other state-of-the-art IDSs supports the reasons for using CNN, TL, and HPO techniques. Furthermore, the average test/prediction time of the proposed ensemble models for each packet on the Raspberry Pi machine is at a low level, from 1.5 ms to 3.2 ms, as shown in Tables 5.2 & 5.3. As the real-time requirement of vehicle anomaly detection systems is usually 10 ms for the analysis of each packet [24], the low prediction time of the proposed models indicates the feasibility of applying the proposed IDS to real-time IoV systems.

5.5 Conclusion

As modern vehicles are increasingly connected, the cyber-threats to IoV systems are also increasing significantly. To protect connected vehicles from being breached by cyber-attacks, this work proposed a transfer learning and ensemble learning-based IDS framework that uses optimized CNN models to identify various types of attacks in IoV systems. Additionally, a chunk-based data transformation method is proposed to transform vehicle network traffic data to image data used as the input of CNN models. The proposed IDS is evaluated on the Car-Hacking and CICIDS2017 dataset, representing intra-vehicle and external network data, respectively. The experimental results show that the proposed IDS framework can effectively identify various types of attacks with higher F1-scores of 100% and 99.925% than other compared state-of-the-art methods on the two benchmark datasets. Moreover, the model testing results on a vehicle-level machine show the feasibility of the proposed IDS in real-time vehicle networks. In future work, this framework will be extended to develop an online adaptive model that can achieve online learning and address concept drift in time-series vehicle network data.

Bibliography

[1] L. Yang and A. Shami, “A Transfer Learning and Optimized CNN Based Intrusion Detection System for Internet of Vehicles,” in *2022 IEEE International Conference on Communications (ICC)*, Seoul, South Korea, 2022, pp. 1-6.

[2] S. T. Mehedi, A. Anwar, Z. Rahman, and K. Ahmed, “Deep Transfer Learning Based Intrusion Detection System for Electric Vehicular Networks,” *Sensors*, vol. 21, no. 14, 2021.

[3] L. Yang, A. Moubayed, and A. Shami, “MTH-IDS: A Multi-Tiered Hybrid Intrusion Detection System for Internet of Vehicles,” *IEEE Internet Things J.*, 2021.

[4] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, “Tree-based Intelligent Intrusion Detection System in Internet of Vehicles,” *proc. 2019 IEEE Glob. Commun. Conf.*, pp. 1–6, Hawaii, USA, 2019.

[5] A. Rosay, F. Carlier, and P. Leroux, “Feed-forward neural network for Network Intrusion Detection,” in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, May 2020, pp. 1–6.

[6] M. Delwar Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi, “An Effective In-Vehicle CAN Bus Intrusion Detection System Using CNN Deep Learning Approach,” in *2020 IEEE Glob. Commun. Conf. (GLOBECOM)*, Taipei, Taiwan, 2020.

[7] L. Yang *et al.*, ”Multi-Perspective Content Delivery Networks Security Framework Using Optimized Unsupervised Anomaly Detection,” *IEEE Trans. Netw. Serv. Manag.*, 2021.

[8] H. M. Song, J. Woo, and H. K. Kim, “In-vehicle network intrusion detection using deep convolutional neural network,” *Veh. Commun.*, vol. 21, p. 100198, 2020.

[9] L. Yang, D. M. Manias, and A. Shami, “PWPAE: An Ensemble Framework for Concept Drift Adaptation in IoT Data Streams,” in *2021 IEEE Glob. Commun. Conf. (GLOBECOM)*, Madrid, Spain, Dec. 2021.

[10] D. Petrov and T. M. Hospedales, “Measuring the Transferability of Adversarial Examples,”, *arXiv*, pp. 1–13, 2019.

[11] L. Yang and A. Shami, “On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[12] E. Seo, H. M. Song, and H. K. Kim, "GIDS: GAN based Intrusion Detection System for In-Vehicle Network," *2018 16th Annu. Conf. Privacy, Secur. Trust. PST 2018*, pp. 1–6, 2018.

[13] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *in Proc. Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[14] R. Panigrahi and S. Borah, "A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems," *Int. J. Eng. Technol.*, vol. 7, no. 3.24, pp. 479-482, 2018.

[15] F. Hussain, S. G. Abbas, M. Husnain, U. U. Fayyaz, F. Shahzad, and G. A. Shah, "IoT DoS and DDoS Attack Detection using ResNet," in *2020 IEEE 23rd International Multi-topic Conference (INMIC)*, 2020, pp. 1–6.

[16] S.-F. Lokman, A. T. Othman, M. H. A. Bakar, and S. Musa, "The Impact of Different Feature Scaling Methods on Intrusion Detection for in-Vehicle Controller Area Network (CAN)," in *Advances in Cyber Security*, 2020, pp. 195–205.

[17] M. M. Leonardo, T. J. Carvalho, E. Rezende, R. Zucchi, and F. A. Faria, "Deep Feature-Based Classifiers for Fruit Fly Identification (Diptera: Tephritidae)," in *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, 2018, pp. 41–47.

[18] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv*, pp. 1–14, 2014.

[19] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-December, pp. 2818–2826, 2016.

[20] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 1800–1807, 2017.

[21] J. Large, J. Lines, and A. Bagnall, "A probabilistic classifier ensemble weighting scheme based on cross-validated accuracy estimates," *Data Min. Knowl. Discov.*, vol. 33, no. 6, pp. 1674–1709, 2019.

[22] L. Yang and A. Shami, "A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams," *IEEE Internet Things Mag.*, vol. 4, no. 2, pp. 96–101, 2021.

[23] M. Rahimzadeh and A. Attar, "A modified deep convolutional neural network for detecting COVID-19 and pneumonia from chest X-ray images based on the concatenation of Xception and ResNet50V2," *Informatics Med. Unlocked*, vol. 19, p. 100360, 2020.

[24] A. Moubayed, A. Shami, P. Heidari, A. Larabi, and R. Brunner, "Edge-enabled V2X Service Placement for Intelligent Transportation Systems," *IEEE Trans. Mob. Comput.*, vol. 1233, pp. 1–13, 2020.

Chapter 6

MTH-IDS: A Multi-Tiered Hybrid Intrusion Detection System for Internet of Vehicles

6.1 Introduction

With the increasing research and rapid development of the Internet of Vehicles (IoV) technology, connected vehicles (CVs) and autonomous vehicles (AVs) are becoming increasingly popular in the modern world [2]. IoV serves as a primary vehicular communication framework that enables reliable communications between vehicles and other IoV entities, such as infrastructures, pedestrians, and smart devices. [2].

IoV consists mainly of intra-vehicle networks (IVNs) and external vehicular networks [2]. IVNs involve an increasing number of electronic control units (ECUs) to adopt various functionalities [3]. All ECUs in a vehicle are connected by a controller area network (CAN) bus to transmit messages and perform actions [4]. On the other hand, external networks connect modern vehicles to the outer environment by vehicle-to-everything (V2X) technologies. V2X technology allows modern vehicles to communicate with other vehicles, roadside infrastructures, and road users [5] [6].

A version of this chapter has been published in IEEE Internet of Things Journal [1].

However, with the increasing level of connectivity and complexity of modern vehicles, their security risks have become a significant concern. Cyber threats may decrease the stability and robustness of IoV, as well as cause vehicle unavailability or traffic accidents. A real-life example can be found in [7]: two attackers compromised and fooled a jeep car into performing dangerous actions, including turning the steering wheel and activating the parking brake at highway speeds, causing severe accidents. In IVNs, CANs are mainly vulnerable to message injection attacks due to their broadcast communication strategy and the lack of authentication [5]. In external networks of IoV, vehicle systems are exposed to various common cyber-attacks, like denial-of-service (DoS), sniffing, and global positioning system (GPS) spoofing attacks [8]. This is because, in large external vehicular networks comprising various types of networks and entities, every node is a potential entry point for cyber-attacks.

Many traditional security mechanisms, like certain authentication and cryptographic techniques, are unsuitable for intra-vehicle networks because they are not supported in CANs or may violate timing constraints of CAN communications [9]. Thus, intrusion detection systems (IDSs) have become an essential component in modern IoV to identify malicious threats on vehicular networks [10]. IDSs are often incorporated into external networks as an essential component of the defense system to identify malicious attacks that can breach firewalls and authentication mechanisms. Although many previous works have made some success developing IDSs, intrusion detection is still a challenging problem due to the high volume of network traffic data, numerous available network features, and various cyber-attack patterns [8].

Machine learning (ML) and data mining algorithms have been recognized as effective models to design IDSs [11]. In this chapter, a multi-tiered hybrid intrusion detection system (MTH-IDS) is proposed to efficiently identify known and zero-day cyber-attacks on both intra-vehicle and external networks using multiple ML algorithms. This work is an extension of Chapter 3. The proposed MTH-IDS framework consists of two traditional ML stages (data pre-processing and feature engineering) and four tiers of learning models:

1. Four tree-based supervised learners — decision tree (DT), random forest (RF), extra trees (ET), and extreme gradient boosting (XGBoost) — used as multi-class classifiers for known attack detection;
2. A stacking ensemble model and a Bayesian optimization with tree Parzen estimator (BO-

TPE) method for supervised learner optimization;

3. A cluster labeling (CL) k-means used as an unsupervised learner for zero-day attack detection;
4. Two biased classifiers and a Bayesian optimization with Gaussian process (BO-GP) method for unsupervised learner optimization.

Therefore, tiers 1 and 3 of the MTH-IDS are designed for basic known and unknown attack detection functionalities, respectively, while tiers 2 and 4 are designed to optimize the base learners in tiers 1 and 3 for model performance enhancement. A comprehensive and robust IDS with both known and unknown attack detection functionalities can be obtained after the model learning and optimization procedures. Additionally, the quality of the used datasets can be improved by data pre-processing and feature engineering procedures to achieve more accurate attack detection.

The performance of the proposed MTH-IDS is evaluated on two public network datasets, the CAN-intrusion-dataset [10] and the CICIDS2017 dataset [12], representing the intra-vehicle and external network traffic data, respectively. The model's feasibility, effectiveness, and efficiency are evaluated using various metrics, including accuracy, detection rates, false alarm rates, F1-scores, and model execution time.

To our knowledge, no previous work proposed such a hybrid IDS that optimizes learning models to accurately detect existing and zero-day attack patterns on both intra-vehicle and external vehicular networks.

The main contributions of this chapter are as follows:

1. It proposes a novel multi-tiered hybrid IDS that can accurately detect the various surveyed types of cyber-attacks launched on both intra-vehicle and external vehicular networks;
2. It proposes a novel feature engineering model based on information gain (IG), fast correlation-based filter (FCBF), and kernel principal component analysis (KPCA) algorithms;
3. It proposes a novel anomaly-based IDS based on CL-k-means and biased classifiers to detect zero-day attacks;
4. It discusses the use of Bayesian optimization techniques to automatically tune the pa-

rameters of each tier in the proposed IDS for model optimization;

5. It evaluates the performance and overall efficiency of the proposed model on two state-of-the-art datasets, CAN-intrusion-dataset and CICIDS2017, and discusses its feasibility in real-world IoV devices.

The remainder of this chapter is organized as follows: Section 6.2 discusses the related work. Section 6.3 presents the vulnerabilities of intra-vehicle and external vehicular networks, as well as the attack scenarios and IDS deployment. In Section 6.4, all the tiers and algorithms in the proposed MTH-IDS are discussed in detail. Section 6.5 presents and discusses the experimental results. Section 6.6 concludes the chapter.

6.2 Related Work

6.2.1 CAN Intrusion Detection

The research on IDS development for IoV and connected vehicles has been considered critical in recent years. Many research works have a focus on detecting attacks on CAN-based intra-vehicle networks. Alshammari *et al.* [13] proposed an intrusion classification model to identify CAN intrusions on in-vehicle networks utilizing support vector machine (SVM) and k-nearest neighbors (KNN) algorithms. Barletta *et al.* [14] proposed a distance-based IDS for CAN intrusion detection using a X–Y fused Kohonen network with the k-means algorithm (XYF-K). The proposed method shows high accuracy on the CAN-intrusion dataset, but its main limitation is the high computational complexity. Olufowobi *et al.* [15] proposed a specification-based real-time IDS named SAIDuCANT to detect the in-vehicle network attacks. The effectiveness of SAIDuCANT is evaluated on a synthetic dataset and the CAN-intrusion dataset. Olufowobi *et al.* [16] proposed an anomaly-based IDS for CAN attack detection using the adaptive cumulative sum (CUSUM) algorithm. This technique can effectively detect intrusions with low delay based on statistical changes. Lee *et al.* [17] proposed an Offset Ratio and Time Interval based IDS (OTIDS) to detect CAN attacks in in-vehicle networks. They also created a CAN dataset by simulating DoS, fuzzy, and impersonation attacks for IDS evaluation.

Deep learning (DL) methods are also widely used for intra-vehicle network IDS development. Lokman *et al.* [18] proposed an unsupervised DL-based anomaly detection model named stacked sparse autoencoders (SSAEs) to discover anomalies in CAN-bus data for intra-vehicle network security enhancement. Song *et al.* [19] proposed a deep convolutional neural network (DCNN) method named Reduced Inception-ResNet to detect intra-vehicle attacks and achieve high detection performance on the CAN-intrusion dataset. Ashraf *et al.* [20] proposed a DL-based IDS for IoV using a long-short term memory (LSTM) autoencoder algorithm. The effectiveness of the proposed model is evaluated on the CAN-intrusion-dataset and UNSW-NB15 dataset, representing in-vehicle network and external network datasets, respectively. DL methods can often achieve high accuracy, but they are computationally expensive due to high model complexity.

6.2.2 External Network Intrusion Detection

Intrusion detection in IoV or external vehicular networks has also attracted significant attention. Alheeti *et al.* [21] proposed an intelligent IDS using back-propagation neural networks to detect DoS attacks in external vehicular networks using the Kyoto 2006+ dataset, but did not consider other attack types. Rosay *et al.* [22] proposed a multi-layer perceptron (MLP) based network IDS for cyber-attack detection in IoT and connected vehicles. The proposed model has been implemented on an automotive microprocessor, and its performance is evaluated on the two variants of the CICIDS2017 dataset. Aswal *et al.* [23] analyzed the applicability of six classical ML algorithms for Bot attack detection on IoV. They used the Bot attack files in the CICIDS2017 dataset to represent the Botnets in vehicular networks, but they did not consider other attacks. Aloqaily *et al.* [24] proposed a network IDS for IoV and connected vehicles using deep belief network (DBN) and decision tree (DT) algorithms. This method shows high accuracy on the NSL-KDD dataset. Gao *et al.* [25] proposed a distributed network IDS for distributed DoS (DDoS) attack detection in vehicular networks and V2X systems. Two general network benchmark datasets, the NSL-KDD and UNSW-NB15 datasets, are used to present the vehicular network datasets and evaluate the IDS. Schmidt *et al.* [26] proposed a spline-based IDS for vehicular networks using the knot flow classification (KFC) method and used the NSL-KDD dataset to represent vehicle networks for model evaluation.

Several other research works also pay attention to the IDS development of general networks and use benchmark datasets for method evaluation. Min *et al.* [27] proposed a semi-supervised learning model, named SU-IDS, by combining the auto-encoder algorithm with k-means to detect cyber-attacks using the NSL-KDD and CICIDS2017 datasets. Yao *et al.* [28] proposed a DL model named spatial-temporal deep learning on communication graphs (STDeepGraph) by combining the convolutional neural network (CNN) and long short-term memory (LSTM) methods. The performance of STDeepGraph is evaluated on the UNSW-NB15 and CICIDS2017 datasets. Injadat *et al.* [29] proposed a novel multi-stage optimized ML-based IDS for network attack detection and evaluated the model's performance on the CICIDS2017 and UNSW-NB15 datasets. The ML models used in this chapter are optimized by hyper-parameter optimization (HPO) methods.

6.2.3 Literature Comparison

Although various studies about vehicular network IDS development have been published, most of them are only designed for known attack detection on either intra-vehicle ([13]- [19]) or external networks ([21]- [27]). Additionally, several papers [21] [23] [25] only consider a specific type of attack, like Botnets or DoS attacks. However, in real-world applications, both intra-vehicle and external networks are vulnerable to various types of attacks with both existing and new patterns. The IDS proposed in [20] is the only research that considers both CAN bus and external networks, and the IDS proposed in [21] is the only technique that can detect both known and unknown attacks. Thus, there still should an IDS designed for the detection of both known and zero-day attacks on both intra-vehicle and external vehicular networks. Our proposed IDS aims to achieve this.

On the other hand, for the deployment of IDSs in real-world vehicle systems, vehicle-level model testing and real-time analysis should be performed to validate the feasibility of the IDSs. However, only five papers [15] [17] [19] [22] [24] did vehicle-level testing or real-time analysis, so the feasibility of other techniques in real-world IoVs is not proven. Therefore, our proposed IDS has been evaluated in a vehicle-level machine to verify whether it meets the real-time requirements of vehicular networks.

Table 6.1: Comparison of Recent Intrusion Detection Techniques for IoVs

Paper	**In-vehicle Network Attack Detection**	**External Network Attack Detection**	**Multiple Types of Known Attack Detection**	**Zero-Day Attack Detection**	**Vehicle-Level Model Testing or Real-time Analysis**	**Data Pre-Processing and Sampling**	**Feature Engineering**	**Model Optimization**
Alshammari *et al.* [13]	✓		✓				✓	
Barletta *et al.* [14]	✓		✓					
Olufowobi *et al.* [15]	✓		✓		✓			
Olufowobi *et al.* [16]	✓		✓					
Lee *et al.* [17]	✓		✓		✓			
Lokman *et al.* [18]	✓		✓					
Song *et al.* [19]	✓		✓		✓			
Ashraf *et al.* [20]	✓	✓	✓				✓	
Alheeti *et al.* [21]		✓		✓			✓	
Rosay *et al.* [22]		✓	✓		✓		✓	
Aswal *et al.* [23]		✓					✓	
Aloqaily *et al.* [24]		✓	✓		✓			
Gao *et al.* [25]		✓						
Schmidt *et al.* [26]		✓	✓					
Min *et al.* [27]		✓	✓					
Yao *et al.* [28]		✓	✓					
Injadat *et al.* [29]		✓	✓				✓	✓
Proposed MTH-IDS	✓	✓	✓	✓	✓	✓	✓	✓

An effective IDS should achieve a high detection rate and a low false alarm rate. Moreover, to meet the real-time requirements of IoV, an IDS should have low computational complexity and high efficiency. Thus, three important procedures, including data sampling, feature engineering, and model optimization, are implemented in our proposed MTH-IDS to improve the efficiency and accuracy of IoV attack detection. The details of how these three procedures can improve the model's performance are provided in Sections 6.4.2 to 6.4.4. However, only six existing techniques [13] [20]- [23] [29] performed feature engineering and only one research work [29] implemented model optimization. The use of efficiency and accuracy enhancement techniques enables our proposed MTH-IDS to outperform most existing research works in terms of detection rate, false alarm rate, and execution speed.

To summarize, the main functionalities and components of the proposed MTH-IDS are listed in Table 6.1. Table 6.1 also clearly compares 17 existing literature with the proposed MTH-IDS based on the important functionalities that an effective and efficient vehicular network IDS should have.

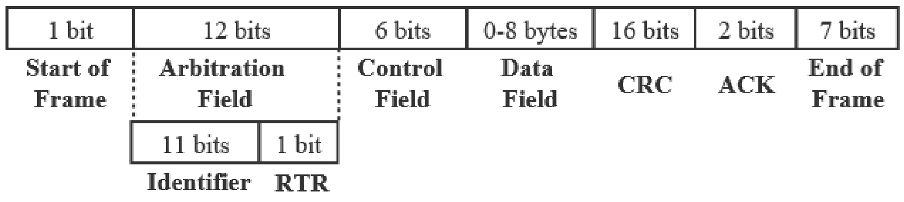

Figure 6.1: CAN packet structure.

6.3 Vehicular Networks, Vulnerabilities, and IDS Deployment

6.3.1 Vulnerabilities of Intra-vehicle Networks

Modern vehicles often contain 70-100 ECUs that are in-vehicle components used to enable various functionalities [3]. CAN [10] is a bus communication protocol that defines an international standard for efficient and reliable intra-vehicle communications among ECUs. A CAN-bus is built based on differential signaling and comprises a pair of channels, CAN-High and CAN-Low, representing the two signals, 1 and 0, respectively [30]. CAN is the most common type of IVN due to its low cost and complexity, high reliability, noise-resistance, and fault-tolerance properties [8] [10]. However, CAN is vulnerable to various cyber threats due to its broadcast transmission strategy, lack of authentication and encryption, and unsecured priority scheme [4].

CAN messages, or packets, are transmitted via CAN-bus. The data frame is the most important type of CAN packet used to transmit user data [31]. Fig. 6.1 shows the structure of a CAN packet, which consists of seven fields [5]: start of frame, arbitration field, control field, data field, CRC (cyclic redundancy code) field, acknowledge (ACK) field, and end of frame. Among all fields, the data field with the size of 0-8 bytes is the most important and vulnerable one, since it contains the actual transmitted data that determines the node actions [10]. An attacker can intrude or take control of a vehicle by injecting malicious messages into the data field of CAN packets, resulting in compromised nodes or vehicles; so-called message injection attacks.

Message injection attacks are the primary type of intra-vehicle attack and can be further classified as DoS attacks, fuzzy attacks, and spoofing attacks by their objectives [10]. In DoS attacks, a CAN is flooded with massive high-priority messages to cause latencies or unavailability of other legitimate messages. Similarly, fuzzy attacks can be launched by injecting

Table 6.2: Common Attack Types on External Vehicular Networks

Attack Type	Description and IoV Scenarios
DoS [33]	Send a large number of requests to exhaust the compromised nodes' resources, causing vehicle unavailability or accidents.
GPS Spoofing [33]	Masquerade as authorized IoV users to provide a node with false information, like false geographic information, therefore causing fake evidence, event delay, or property losses.
Jamming [33]	Jam signals to prevent legitimate IoV devices from communicating with connected vehicles.
Sniffing [33]	Capture vehicular network packets to steal confidential or sensitive information of vehicles, users, or enterprises.
Brute-force [12]	Crack passwords in vehicle systems to take control of vehicles or machines and perform malicious actions.
Botnets [12]	Infect multiple connected vehicles and IoV devices with Bot viruses to breach them and launch other attacks.
Infiltration [12]	Traverse the compromised vehicle systems and create a backdoor for future attacks.
Web Attack [12]	Hack IoV servers or web interfaces of connected vehicles to gain confidential information or perform malicious actions.

arbitrary messages with randomly spoofed identifiers or packets, causing compromised vehicles to exhibit unintended behaviors, like sudden braking or gear shift changes. Spoofing or impersonation attacks, such as gear spoofing and revolutions per minute (RPM) spoofing attacks, are launched by injecting messages with certain CAN identifiers (IDs) to masquerade as legitimate users and take control of the vehicles.

6.3.2 Vulnerabilities of External Vehicular Networks

In a similar fashion, V2X technology enables interactions and communications between vehicles and other IoV entities, including pedestrians, infrastructures, smart devices, and network systems [5] [32]. With the increasing connectivity of modern IoV, external vehicular networks are becoming large networks that involve various other networks and devices. Thus, external vehicular networks are vulnerable to various general cyber threats because each vehicle or device is a potential entry point for intrusions. Typical attacks in IoV include DoS, GPS spoofing, jamming, sniffing, brute-force, Botnets, infiltration, and web attacks [12] [33]. The description and IoV scenarios of these common external vehicular network attacks are summarized in Table 6.2.

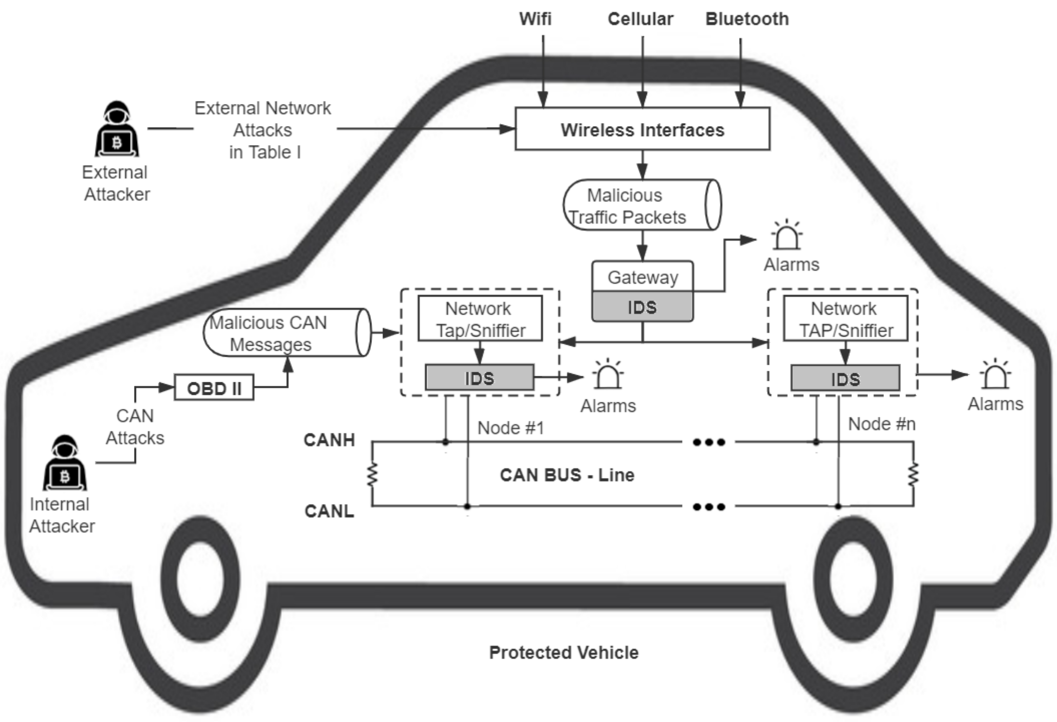

Figure 6.2: The proposed IDS-protected vehicle architecture.

6.3.3 Attack Scenarios and IDS Deployment

The attack scenarios and general architecture of a vehicle protected by IDS are shown in Fig. 6.2. As the physical interface for ECU communications, the on-board diagnostics II (OBD-II) interface is often exploited by internal attacks to inject malicious CAN messages into the vehicle systems, therefore taking control of the CAN nodes to perform malicious actions, like sudden braking. On the other hand, the external attackers can launch the cyber-attacks listed in Table 6.2 through various wireless interfaces, including WiFi, cellular network, and Bluetooth [20].

To secure IoV, the proposed IDS can be placed in different locations in vehicular networks. In intra-vehicle networks, the IDS can be deployed on top of the CAN-bus to detect malicious messages [34]. Since every message is broadcasted to all nodes, it will also be transmitted through the IDS when its signal changes from CAN-High to CAN-Low. The IDS will monitor the CAN packets and identify potential intrusions. If an attack is detected, alarms will be triggered on every node.

On the other hand, the central gateways are common network devices to incorporate IDSs, so the proposed IDS can be placed inside the gateway to monitor the external network traffic [35]. Thus, abnormal network traffic can be detected when it is passed through the gateways in external vehicular networks. The potential deployment of the proposed IDS is also shown in Fig. 6.2.

To protect the vehicle from being breached, all packets or messages transmitted in the protected vehicular network are captured by packet taps or sniffers (*e.g.*, NetFlow), and then analyzed by the proposed IDS before being forwarded to the protected vehicle [36]. For example, if an attacker is launching a DoS attack by sending a large volume of malicious traffic to a vehicle system, the malicious traffic will be detected by the proposed IDS by processing the traffic data captured by the sniffer; alarms will then be generated, and the attacker's access will be denied, therefore protecting the vehicle from being breached [37]- [39].

6.3.4 Real-Time Requirements of Vehicle Systems

The standardized vehicular communications specify the performance requirements of vehicle safety services based on two primary metrics: packet-delivery-ratio (PDR) and latency [40]. For IDS development, the latency metric should be considered to meet the real-time requirements. Latency indicates the time needed to transmit a packet from its source to its destination. According to the United States (US) department of transportation, the highest priority vehicle safety services, like collision and attack warnings, should have a latency of 10 to 100 ms at the utmost [40]. On the other hand, for autonomous or cooperative driving, the V2X traffic safety applications require a stringent latency requirement of 10 to 20 ms [39] [41]. Thus, for a vehicle-level IDS, the time needed to process each network packet is required to be less than 10ms to meet the real-time or latency requirements.

6.4 Proposed MTH-IDS Framework

6.4.1 System Architecture

The purpose of this work is to develop an IDS that can protect both intra-vehicle and external networks from being breached by the various common attacks presented in Sections 6.3.1 and

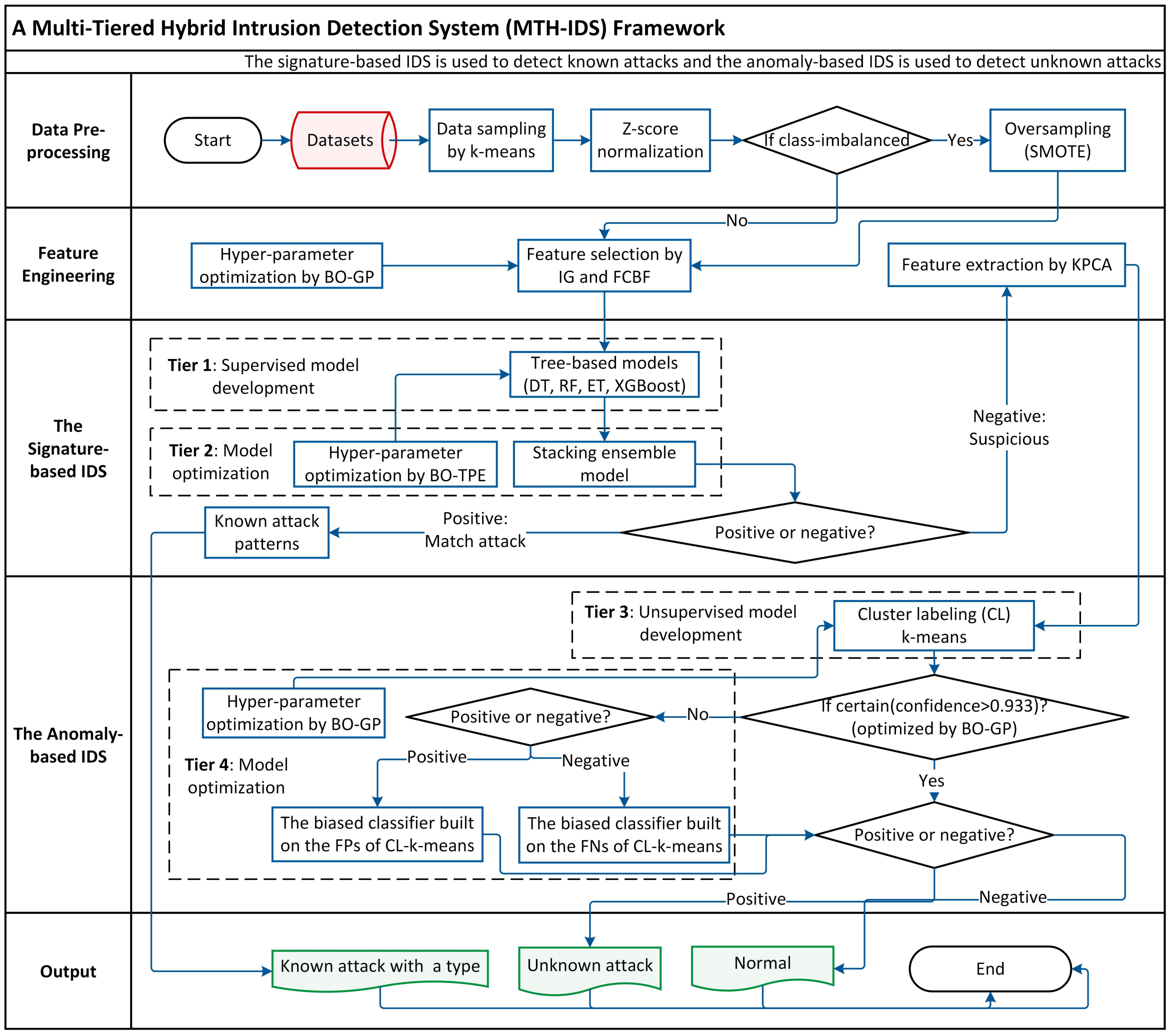
A Multi-Tiered Hybrid Intrusion Detection System (MTH-IDS) Framework
The signature-based IDS is used to detect known attacks and the anomaly-based IDS is used to detect unknown attacks
Data Pre-processing
Start
Datasets
Data sampling by k-means
Z-score normalization
If class-imbalanced
Yes
Oversampling (SMOTE)
No
Feature Engineering
Hyper-parameter optimization by BO-GP
Feature selection by IG and FCBF
Feature extraction by KPCA
The Signature-based IDS
Tier 1: Supervised model development
Tree-based models (DT, RF, ET, XGBoost)
Tier 2: Model optimization
Hyper-parameter optimization by BO-TPE
Stacking ensemble model
Negative: Suspicious
Known attack patterns
Positive: Match attack
Positive or negative?
The Anomaly-based IDS
Tier 3: Unsupervised model development
Cluster labeling (CL) k-means
Hyper-parameter optimization by BO-GP
Positive or negative?
No
If certain(confidence>0.933)? (optimized by BO-GP)
Tier 4: Model optimization
Positive
Negative
Yes
The biased classifier built on the FPs of CL-k-means
The biased classifier built on the FNs of CL-k-means
Positive or negative?
Positive
Negative
Output
Known attack with a type
Unknown attack
Normal
End

Figure 6.3: The framework of the proposed MTH-IDS.

6.3.2. In this chapter, a novel multi-tiered hybrid IDS is proposed to detect both known and unknown cyber-attacks on vehicular networks with optimal performance. Fig. 6.3 demonstrates the architecture of the proposed system, comprising four main stages: data pre-processing, feature engineering, a signature-based IDS, and an anomaly-based IDS.

Firstly, intra-vehicle and external network traffic datasets are collected for the purpose of system performance evaluation on both types of vehicular networks. Data pre-processing consists of a k-means-based cluster sampling method used to generate a highly-representative subset, and a SMOTE method used to avoid class-imbalance. In the feature engineering process, the datasets are processed by information-gain-based and correlation-based feature selection methods to remove irrelevant and redundant features, and then passed to the KPCA model to further reduce dimensionality and noisy features. The proposed data pre-processing and feature engineering procedures can greatly improve the quality of the network data for more accurate model learning. The signature-based IDS is then developed to detect known attacks by training four tree-based machine learners as the first tier of the proposed MTH-IDS: DT, RF, ET, XGBoost. In the second tier, a stacking ensemble model and the BO-TPE method are used to further improve the intrusion detection accuracy by combining the output of the four base learners from the first tier and optimizing the learners. In the next stage, an anomaly-based IDS is constructed to detect unknown attacks. In the anomaly-based IDS, the suspicious instances are passed to a cluster-labeling (CL) k-means model as the third tier to effectively separate attack samples from normal samples. The fourth tier of the MTH-IDS comprises the BO-GP method and two biased classifiers used to optimize the model and reduce the classification errors of the CL-k-means. Ultimately, the detection result of each test sample is returned, which could be a known attack with its type, an unknown attack, or a normal packet. To summarize the rationale behind the algorithms used in the proposed IDS, the brief description and performance impact of each algorithm are presented in Table 6.3. A detailed description is provided in Sections 6.3.2 to 6.3.4.

6.4.2 Data Pre-Processing

Data Sampling by K-means Clustering

In real life, training ML models on massive amounts of network traffic data is unrealistic and may cost a massive amount of time, especially in the hyper-parameter tuning process that needs to train a ML model multiple times. For model training efficiency improvement purposes, data sampling is a common technique that can generate a subset of the original data to reduce the training complexity of a model [42].

In the proposed system, to obtain a highly-representative subset, a k-means-based cluster sampling method is utilized. Cluster sampling is a common data sampling method by which the original data points are grouped into multiple clusters; then, a proportion of data is sampled from each cluster to form a representative subset [42]. Unlike random sampling, which randomly selects every data sample with an equal probability, cluster sampling can generate a

Table 6.3: Rationale and Performance Impact of Each Component of the MTH-IDS

Stage	Algorithm	Rationale and Description	Performance Impact
Data pre-processing	K-means cluster sampling	Network traffic data is often large, while IoV devices often have limited computational power and resources. The k-means sampling method can generate highly representative subsets for more efficient training because the removed data is mostly redundant data.	Improve model training efficiency.
	SMOTE	Network traffic data is often imbalanced data because most data samples are collected under normal conditions in real-world vehicle systems. SMOTE can create high-quality samples for minority classes to avoid class-imbalance and ineffective classifiers.	Improve detection rate.
	Z-score	Different features often have different ranges, which can bias the model training. The Z-score method can normalize features to a similar scale and handle outliers.	Improve model accuracy and training efficiency.
Feature engineering	IG	For certain tasks like intrusion detection, many collected features can be irrelevant, causing additional training time. The IG method can remove those unimportant features.	Improve model training efficiency.
	FCBF	Certain features are redundant because they contain very similar information. FCBF can remove redundant features by calculating the correlation between each pair of features.	Improve model accuracy and training efficiency.
	KPCA	The anomaly-based IDSs are sensitive to the quality of features. KPCA can further extract the most relevant features to reduce dimensionality and noisy information.	Improve model accuracy and training efficiency.
The signature-based IDS	DT, RF, ET, and XGBoost	Tree-based ML algorithms often perform better than other ML algorithms on complex tabular data to which IoV data belong. Four tree-based supervised algorithms are used to train base classifiers for known intrusion detection.	Detect various types of known attacks.
	BO-TPE	The default hyper-parameters of ML algorithms often cannot return the best model. BO-TPE can optimize the models' hyper-parameters to obtain the optimized base classifiers.	Improve accuracy of known attack detection.
	Stacking	Ensemble models can often achieve higher accuracy than any single model. Stacking ensemble can combine the base classifiers to obtain a meta-learner with better performance.	Improve accuracy of known attack detection.
The anomaly-based IDS	CL-k-means	For unknown attack detection, CL-k-means can generate a sufficient number of normal and attack clusters to identify zero-day attacks from the newly arriving data.	Detect unknown attacks.
	BO-GP	CL-k-means has an important hyper-parameter, the number of clusters, k. BO-GP is an effective HPO method to optimize k and obtain the optimized CL-k-means model.	Improve accuracy of unknown attack detection.
	Two biased classifiers	CL-k-means may return many errors when detecting complex unknown attacks. Two biased classifiers are trained on the FPs and FNs of CL-k-means to reduce the errors.	Improve accuracy of unknown attack detection.

highly-representative subset because the discarded data points are mostly redundant data.

Among all clustering algorithms, k-means is the most common one for data sampling due to its simple implementation and low computational complexity [43]. K-means clustering algorithms are used to divide the data points into k clusters based on their Euclidean, Manhattan, or Mahalanobis distances [44] [45]. The data samples in the same group can be considered similar samples, so sampling from each group can greatly reduce the size of data without losing important information. K-means aims to minimize the sum of squares of distances between all the data points and the corresponding centroid of the cluster, denoted by [43]:

$$\sum_{i=0}^{n_k} \min_{u_j \in C_k} \left(\mathbf{x}_i - u_j \right)^2 , \tag{6.1}$$

where $(\mathbf{x}_1, \cdots, \mathbf{x}_n)$ is the data matrix; u_j, also called the centroid of a cluster C_k, is the mean of all the samples in C_k; and n_k is the total number of sample points in the cluster C_k. K-means has a linear time complexity of $O(nkt)$, where n is the data size, k is the number of clusters, and t is the number of iterations [43].

After implementing the k-means to cluster the original data samples into k clusters, random sampling is then applied to each cluster to select 10% of the data as the sampled subset. The percentage of data sampling can vary, depending on the data scale and resource limitations. Additionally, the main hyper-parameter of k-means, the number of clusters [46], k, is tuned by Bayesian Optimization (BO) to improve the quality of the subset.

Hyper-parameter optimization (HPO) is the process of building an optimized ML model for a specific problem or dataset using an optimization algorithm [47]. BO algorithms are an efficient group of HPO algorithms that determine the next hyper-parameter value based on the previous evaluation results [48]. In BO, a surrogate model is used to fit all the currently tested data points into the objective function; an acquisition function is then used to locate the next point. Two common surrogate models for BO are the Gaussian process (GP) and the tree Parzen Estimator (TPE) [46].

In GP surrogate models, the predictions follow a Gaussian distribution [46]:

$$p(y|x, D) = N\left(y|\hat{\mu}, \hat{\sigma}^2\right), \tag{6.2}$$

where D is the hyper-parameter configuration space, $y = f(x)$ is the value of the objective function for each hyper-parameter configuration with its mean as μ and covariance as σ. The BO method with GP (BO-GP) exhibits great performance on optimizing a small number of continuous or discrete hyper-parameters due to its fast convergence speed, but is inefficient for conditional hyper-parameters since it treats each hyper-parameter independently [46]. BO-GP has a time complexity of $O(n^3)$ and space complexity of $O(n^2)$ [46].

Since k-means methods generally only have one discrete hyper-parameter, k, that needs to be tuned, BO-GP serves as an effective HPO method for k-means because of its fast convergence speed. The silhouette coefficient, a common distance-based metric that can effectively evaluate clustering performance, is chosen to be the metric of the k-means model and used as the objective function of BO-GP. It measures how similar a data point is to other data points within the same cluster and how different the data point is from the data points in other clusters [44].

Reduce Class-Imbalance by Oversampling

Class-imbalance issues often occur in network traffic data, since the percentage of normal samples is often much larger than the percentage of attack samples in real-world network data, resulting in biased models and low detection rate [49].

Class-imbalance problems are mainly solved by resampling methods, including random sampling and synthetic minority oversampling techniques (SMOTE), which can create new instances for the minority classes to balance the dataset [49]. Unlike random sampling, which simply replicates the instances and may cause over-fitting, SMOTE [50] can synthesize high-quality instances based on the concept of KNN; thus, SMOTE is chosen in the proposed IDS to solve class-imbalance. For each instance X in the minority class, assuming X_i is a sample randomly selected from the k nearest neighbors of X, a new synthetic instance X_n can be denoted by [29]:

$$X_n = X + rand(0, 1) * (X_i - X), i = 1, 2, \cdots, k, \tag{6.3}$$

where $rand(0, 1)$ represents a random number in the range of $(0, 1)$. Thus, SMOTE is utilized in the proposed IDS to create high-quality instances for minority classes.

Data Normalization

After implementing the k-means and SMOTE methods to obtain a representative and balanced dataset, several additional data pre-processing steps are completed for the next steps. Firstly, the network traffic datasets are encoded with a label encoder used to transform categorical features into numerical features to support the inputs of ML algorithms, because many ML algorithms cannot support string features directly. After that, the network datasets are normalized by the Z-score algorithm since the collected features in network traffic data often have largely different ranges, and ML models often perform better on normalized datasets [21]. An unnormalized dataset with largely different feature scales may result in a biased ML model that only lays emphasis on large-scale features. Through the Z-score method, the features can be normalized to have a mean of 0 and a standard deviation of 1. By implementing the Z-score method, each normalized feature value, x_n, is denoted by:

$$x_n = \frac{x - \mu}{\sigma}, \tag{6.4}$$

where x is the original feature value, μ and σ are the mean and standard deviation of the feature values, respectively.

6.4.3 Feature Engineering

A high-quality and highly representative dataset can be generated after data pre-processing. On the other hand, obtaining an optimal feature list by appropriate feature engineering can also improve the quality of datasets for more accurate and efficient model learning. A comprehensive feature engineering method that consists of IG, FCBF, and KPCA, is implemented before ML model training to remove irrelevant, redundant, and noisy features while retaining the important features [51].

Feature Selection by Information Gain

As a common feature selection (FS) method, the information gain (IG) method is used to select important features. IG, the amount of information gained or the changes in entropy, can be used to measure how much information a feature can bring to the targeted variable [52]. IG is chosen in the proposed system since it can obtain an importance score for each feature at a fast speed due to its low computational complexity of $O(n)$ [52]. The importance score of each feature enables us to select the most relevant features for the task. Assuming T is the target variable, for each feature denoted by a random variable X, the IG value of using the feature X, is denoted by [52]:

$$IG(T|X) = H(T) - H(T|X), \tag{6.5}$$

where $H(T)$ is the entropy of the target variable T, and $H(T|X)$ is the conditional entropy of T over X.

Therefore, the feature importance of a feature X, or the correlation between X and the target T, can be represented by the IG value of T over X, $IG(T|X)$. A feature X is considered more important to the target T than another feature Y if $IG(T|X) > IG(T|Y)$ [52].

To implement the IG-based FS method, the importance of each feature is calculated based on eq. (6.5) and normalized to have a sum of 1.0, denoting the relative importance. The features are then ranked by their importance and are selected from top to bottom until the total importance of selected features reaches the correlation threshold, α. The remaining features, with the total feature importance less than $1 - \alpha$, are discarded. To obtain an appropriate correlation threshold, α is optimized by BO-GP that uses the validation accuracy as the objective function for HPO. After this process, the irrelevant features are eliminated, and a reduced number of informative features with high importance are obtained for the next step.

Fast Correlation Based Filter (FCBF)

Although the IG-based FS method eliminates the unimportant features to reduce time complexity, many redundant features still exist. Feature redundancy may increase time and space complexity, and degrade model performance by increasing the probability of being misled by noisy data, as well as increasing the risk of over-fitting [51]. Thus, removing redundant features by calculating the correlations of input features is beneficial for model performance and efficiency.

Among the correlation-based FS algorithms, the fast correlation-based filter (FCBF) [53] algorithm is selected since it has shown great performance on high dimensional datasets by effectively removing redundant features while retaining informative features, and has a low time complexity of $O(nlogn)$ [54]. In FCBF, the symmetrical uncertainty (SU) is calculated to measure the correlations between features by normalizing the IG values [53]:

$$SU(X,Y) = 2\left[\frac{\mathrm{IG}(X|Y)}{H(X)+H(Y)}\right]. \tag{6.6}$$

$SU(X,Y)$ is in the range $[0,1]$ with the value 1 indicating a perfect correlation between the two features X and Y, while the value 0 indicates the two features are fully independent.

The FCBF method searches the features in the feature space based on their SU values until the entire feature space has been explored. The highly correlated features are regarded as redundant features, and only one of them will be retained.

In the proposed FS approach, the SU value of each pair of features is calculated as their correlations. The correlation threshold, α, is also optimized by BO-GP. When the correlation value between two features is larger than α, the one with higher feature importance is retained while the other is discarded. The correlation calculation and feature deletion procedures are repeated until each pair of features in the feature list are not highly correlated ($SU <= \alpha$). The FS model that combines the IG method and the FCBF algorithm is named IG-FCBF.

Kernel Principal Component Analysis (KPCA)

Although utilizing IG-FCBF can return a better feature set than only using IG, FCBF has a major limitation that it only calculates the correlation between pairs of features, but does not consider correlations among three or more different features, resulting in undiscovered noisy features [54]. On the other hand, the unsupervised learning models in the anomaly-based IDS are more sensitive to appropriate features than supervised learning models since they rely on the changes of feature values instead of the ground truth labels to process data [44] [51]. Hence, kernel principal component analysis (KPCA) is utilized after implementing the IG-FCBF method for the anomaly-based IDS.

PCA [55] is a feature extraction algorithm that uses orthogonal transformations to transform a set of correlated features onto a smaller subset of uncorrelated features, named principal components. KPCA is an improved version of PCA that uses the kernel trick to learn a non-linear function or decision boundary to reduce the dimensionality of non-linear data [56]. KPCA is selected due to its adaptability to non-linear data, as well as its capacity to reduce computational complexity, the risk of over-fitting, and distracting noise [51].

Additionally, the two essential hyper-parameters in KPCA, the number of extracted features and the kernel type, are optimized by the BO-GP method using validation accuracy as the objective function to improve the model performance. It is efficient for BO-GP to optimize these discrete and categorical hyper-parameters. KPCA is used with IG-FCBF to construct the IG-FCBF-KPCA method to obtain an optimal dataset with extracted features as the input of the anomaly-based IDS.

6.4.4 The Proposed Hybrid IDS

IDSs are mainly classified as signature-based IDSs and anomaly-based IDSs. Signature-based IDSs are designed to detect the known attack patterns by training supervised ML models on labeled datasets. However, they often lack the capacity to detect new attack patterns that are not previously stored in the databases [57]. On the other hand, anomaly-based IDSs can distinguish unknown attack data from normal data by unsupervised learning algorithms based on the assumption that new attack data are more statistically similar to the known attack data than normal data, but they often return many false alarms [58]. Thus, a hybrid IDS that consists of a signature-based IDS and an anomaly-based IDS is proposed in this chapter to effectively detect both known and zero-day attacks.

The Signature-based IDS

After the data pre-processing and feature engineering procedures, the obtained labeled datasets are trained by an ensemble learning model to develop a signature-based IDS. In the proposed signature-based IDS, four tree-based ML algorithms — decision tree (DT), random forest (RF), extra trees (ET), and extreme gradient boosting (XGBoost) — are selected as the base learners.

DT [59] is a common ML algorithm that uses a tree-structure to fit data and make predictions. DT algorithms have multiple hyper-parameters that require tuning, including the tree depth, minimum sample split, minimum sample leaf, maximum sample nodes, and minimum weight fraction leaf, etc. [46]. RF [60] is an ensemble learning model that uses the majority voting rule to combine multiple decision tree classifiers, while ET [61] combines a collection of randomized decision trees built on different subsets of a dataset. XGBoost [62] is a gradient-boosted decision tree (GBDT) based algorithm designed for speed and performance improvement. For RF, ET, and XGBoost, the hyper-parameters of DT are also the important hyper-parameters for them because they are all constructed by integrating multiple DTs. Additionally, they have an essential hyper-parameter that needs to be tuned, being the number of base DTs to be built for each model, "n_estimators", which has a direct impact on model performance. XGBoost has another hyper-parameter, the learning rate, which determines the convergence speed [46].

Assuming the number of instances is n, the number of features is f, and the number of DTs in ensemble models is t, the time complexity of DT, RF, ET, and XGBoost is $O(n^2 f)$, $O(n^2 \sqrt{f} t)$, $O(nft)$, and $O(nft)$, respectively [8].

The algorithm choosing reasons are as follows [8] [63]:

1. RF, ET, and XGBoost are all ensemble models that combine multiple DTs and can effectively work on non-linear and complex data to which network traffic data belongs; hence, they often perform better than other ML algorithms, like naïve Bayes (NB) and KNN, which often do not perform well on complex datasets.
2. They enable parallel execution, which significantly reduces model training time and improves efficiency.
3. They calculate feature importance during the model training process, which is beneficial for feature engineering procedures.
4. The tree-based algorithms have randomness in their construction process, which enables us to build a robust ensemble model that has better generalizability than using other ML algorithms.

After obtaining the four tree-based ML models, they are combined using stacking, an ensemble learning method, to improve model performance because the generalizability of a combination of multiple base learners is usually better than that of a single model [64]. Stacking is a standard ensemble learning technique that uses the output labels estimated by four base learners (DT, RF, ET, and XGBoost) as the input features to train a strong meta-learner that makes the final prediction [64]. Using stacking can learn the information from all four base learners to reduce the errors of single learners and obtain a more reliable and robust meta-classifier. In the proposed system, the best-performing one among the four base models is chosen as the algorithm to build the meta-learner because it is most likely to achieve the best performance.

The important hyper-parameters of the four tree-based ML algorithms are optimized by a HPO method, BO with tree-Parzen estimator (BO-TPE). BO-TPE creates two density functions, $l(x)$ and $g(x)$, to act as the generative models for variables. With a pre-specified threshold y^* to separate the relatively good and poor results, the objective function of TPE is modeled by the Parzen windows [46]:

$$p(x|y, D) = \begin{cases} l(x), & \text{if } y < y^* \\ g(x), & \text{if } y > y^* \end{cases}, \tag{6.7}$$

where $l(x)$ and $g(x)$ indicate the probability of detecting the next hyper-parameter value in the well-performing regions and in the poor-performing regions, respectively. BO-TPE detects the optimal hyper-parameter values by maximizing the ratio $l(x)/g(x)$. The Parzen estimators are organized in a tree structure, so the specified conditional dependencies of hyper-parameters can be retained. Additionally, BO-TPE can optimize all types of hyper-parameters effectively [46]. Therefore, BO-TPE is used to optimize the hyper-parameters of the tree-based ML models that have many hyper-parameters.

The Anomaly-based IDS

The proposed signature-based IDS can detect multiple types of known attacks effectively. However, attackers can still carry out zero-day attacks that are not included in the known attack patterns and can be misclassified as normal states. Therefore, the instances labeled "normal" by the signature-based IDS will be considered suspicious instances because some of them can be unknown attack samples. A novel anomaly-based IDS architecture is then developed to identify zero-day attacks by processing the suspicious instances.

After feature engineering, the optimized dataset obtained from the output of the IG-FCBF-KPCA method is used to train the anomaly-based IDS. The first tier of the proposed anomaly-based IDS comprises the cluster labeling (CL) k-means developed by improving the k-means model introduced in Section 6.4.2. Since there are millions of instances collected under many different situations in network traffic datasets, a sufficient number of clusters should be used to distinguish between normal and attack data. The main procedures of the proposed CL-k-means method are as follows:

1. Split the dataset into a sufficient number of clusters using k-means.
2. Label each cluster by the majority label of data samples. In each cluster, the class label to which most of the instances belong, “normal” or “attack”, is assigned to this cluster.
3. Label each sample in the test set as “normal” or “attack” based on the label of the cluster that this instance is classified into by k-means.
4. For each test sample i that is classified into a cluster, calculate the percentage of majority class samples in this cluster as the confidence or clustering probability, p_i.
5. Optimize the number of clusters (k) and distance metric as the major hyper-parameters of k-means, by the BO-GP algorithm to obtain the optimal CL-k-means model. The validation accuracy on the test set is used as the objective function for BO-GP.

K-means is selected to distinguish between attack and normal data mainly due to the real-time requirements of vehicle-level systems. K-means is computationally faster than most other clustering algorithms because it has a linear time complexity of $O(nkt)$, where n is the data size, k is the number of clusters, and t is the number of iterations [43]. The model training time is further reduced by using mini-batch k-means, which uses randomly sampled subsets as mini-batches in each training iteration [65]. Additionally, k-means guarantees convergence and easily adapts to new samples.

To increase the detection rate and reduce the false alarm rate of the CL-k-means method, the second tier of the proposed anomaly-based IDS uses two biased classifiers to reduce the false negatives (FNs) and false positives (FPs), respectively. Biased classifiers are developed by the following procedures:

1. Collect the FNs and FPs obtained from the training set using the proposed CL-k-means method.
2. Select the best-performing singular supervised learning model in the signature-based IDS (*e.g.*, RF) as the algorithm to construct biased classifiers.
3. Train the first biased classifier, B_1, on all the FNs along with the same amount of randomly sampled normal data to build a model that aims to reduce FNs.

4. Train the second biased classifier, B_2, on all the FPs along with the same amount of randomly sampled attack data to build a model that aims to reduce FPs.

After implementing the proposed CL-k-means model, each data sample whose clustering probability (p_i) is less than a threshold p_i^*, is regarded as an uncertain instance. The threshold p_i^* is a continuous variable and has been optimized to be 0.933 by BO-GP that uses the validation accuracy as the objective function. After obtaining the two trained classifiers, the uncertain instances will be passed to B_1 (if labeled "normal" by the CL-k-means) or B_2 (if labeled "attack" by the CL-k-means) to obtain its final classification result.

The proposed anomaly-based IDS is constructed under the assumption that new attack patterns are unknown and future incoming data samples are unlabeled; hence, only the FNs and FPs obtained during the training phase are used to build biased classifiers. This enables the proposed IDS to detect new attack patterns without additional procedures that are difficult to perform, like constant data labeling and model updates.

Compared to other unsupervised anomaly detection algorithms like isolation forest (iForest) and one-class SVM (OC-SVM) [66], the proposed CL-k-means method with biased classifiers has the following advantages to achieving high accuracy and efficiency:

1. The proposed CL-k-means model has the capacity of using a sufficient number of clusters to model the data samples with various attack and normal patterns. This makes the proposed method have better generalizability and data pattern modeling capability than other outlier detection methods, like iForest and OC-SVM, which are mostly binary models.
2. The number of clusters in CL-k-means, k, is automatically optimized by BO-GP. This enables the proposed model to automatically fit different datasets and tasks according to the complexity of data patterns.
3. The main difficulty of unknown attack detection is that unsupervised learning models often return more misclassified samples than supervised learning models. Thus, using the biased classifiers can effectively reduce the FPs and FNs because they can learn the patterns of misclassified data samples that are difficult to be identified by the CL-k-means.

4. The use of the cluster probability, p_i, can greatly improve the model efficiency. This is because the new samples that are very similar to existing attack or normal patterns (with high confidence) can be labeled directly, and only the uncertain samples (with relatively low probability) are passed to the biased classifiers for further identification.
5. The use of mini-batch k-means can significantly reduce the execution time of the MTH-IDS to meet the real-time requirements of IoV.

However, certain attack patterns are very similar to normal patterns, which makes it difficult to distinguish them. Additionally, the samples collected under certain legitimate network events, like crowd events, may still be misclassified as attack samples because the new patterns they have are largely different from existing normal patterns. Additionally, although k-means is more suitable for IoV IDS development than other unsupervised learning algorithms due to its low complexity, the data distribution modeling limitation of k-means is another potential issue to be better addressed [43]. If time and budget permits, k-means can be replaced by other clustering algorithms with the same cluster labeling technique based on the specific data shape and distribution to further improve the system performance. Online learning techniques that can keep updating learning models based on the new attack patterns may also improve the generalizability and accuracy of the IDS, which will be our future work.

6.4.5 Runtime Complexity

The training process of the proposed MTH-IDS can be done at a server machine with high computational speed, while the testing process should be implemented in vehicle systems. Developing models with low runtime complexity enables the proposed IDS to reduce the latency of vehicle systems and meet real-time requirements. In the implementations, each test sample will be passed through the stacking model constructed with four tree-based algorithms, the CL-k-means method, and one of the biased classifiers. Since the runtime complexity of DT is $O(df)$ and the runtime complexity of RF, ET, XGBoost is $O(dft)$, where d is the maximum depth of the trees, f is the number of features and t is the number of trees, the maximum runtime complexity of the signature-based IDS is only $O(dft)$ [67]. For the proposed CL-k-means method in the anomaly-based IDS, its runtime complexity is $O(fk)$, where k is the number of clusters [68]. The biased classifier in the anomaly-based IDS is also the best-performing tree-based algorithm, so its maximum runtime complexity is also $O(dft)$.

Therefore, the maximum overall runtime complexity of the proposed IDS is at a low-level, only $O(2dft + fk)$, since the values of d, f, t, and k, are a few dozens at most. The model test time will be calculated in the experiments to evaluate the feasibility of the proposed IDS in vehicle systems.

6.4.6 Validation Metrics

To evaluate the generalizability of the proposed framework and avoid over-fitting, both cross-validation and hold-out methods are used in the known attack detection experiments. Specifically, the train-test-validation split and model evaluation procedures for each dataset are as follows:

1. Use 70%-30% train-test split to generate a training set with 70% of data samples and a test set with 30% of data samples. The test set will remain untouched before the final hold-out validation.
2. Implement 10-fold cross-validation on the training set to evaluate the proposed model on different regions of the dataset. In each iteration/fold of the 10-fold cross-validation, 90% of the original training set is used for model training, and 10% of the original training set is used as the validation set for model testing.
3. Test the trained model obtained from Step 2 on the untouched test set to evaluate the model performance on a new dataset.

70%-30% train-test split and 10-fold cross-validation were chosen because they are standard and sufficient numbers to construct powerful validation methods to avoid over-fitting and concept drift issues [69]. If the proposed method can accurately detect intrusions in both cross-validation and hold-out validation, there are no underfitting or overfitting issues [70].

On the other hand, hold-out validation is used to evaluate the proposed system for unknown attack detection. For each attack type as a zero-day attack, a validation set consisting of all the instances of this attack type and the same amount of randomly sampled normal data is generated; all the other samples are used as the training set. Therefore, the validation results can be used to evaluate whether the proposed system has the capacity to detect the patterns of each type of unknown attack. This validation process is based on the assumption that new attack data is more statistically similar to certain other known attack data than normal data [58].

Several metrics, including accuracy (Acc), detection rate (DR), false alarm rate (FAR), and F1-score, are used to comprehensively evaluate the performance of the proposed IDS [29]. By calculating the true positives (TPs), true negatives (TNs), false positives (FPs), and false negatives (FNs) of the proposed model, the used metrics are calculated by the following equations [29]:

$$Acc = \frac{TP+TN}{TP+TN+FP+FN} \tag{6.8}$$

$$DR = \frac{TP}{TP+FN} \tag{6.9}$$

$$FAR = \frac{FP}{TN+FP} \tag{6.10}$$

$$F1 = \frac{2\times TP}{2\times TP+FP+FN} \tag{6.11}$$

Model execution time, calculated by the average of the model training and validation time in the 10-fold cross-validation or hold-out validation, is used to evaluate model efficiency. An efficient IDS should be able to achieve a high F1-score and low execution time simultaneously.

6.5 Performance Evaluation

6.5.1 Experimental Setup

To develop the proposed MTH-IDS, the feature engineering and ML algorithms were implemented using the Pandas [71], Scikit-learn [72], Xgboost [73] libraries in Python, while the HPO methods were implemented by extending the Skopt [74] and Hyperopt [75] libraries. The experiments were carried out on a Dell Precision 3630 Tower machine with an i7-8700 central processing unit (CPU) (6-Core, 3.20 GHz) and 16 Gigabytes (GB) of memory, and a Raspberry Pi 3 machine with a BCM2837B0 64-bit CPU and 1 GB of memory, representing a server machine for model training and a vehicle-level machine for model testing, respectively.

Code for the major modules is available at: https://github.com/Western-OC2-Lab/Intrusion-Detection-System-Using-Machine-Learning

The experiments are divided into three parts, one for the known intrusion detection by evaluating the signature-based IDS component on the labeled datasets, one for unknown intrusion detection by evaluating the anomaly-based IDS component on the unlabeled datasets, and one for the entire model evaluation by analyzing the CPU resource usage on a vehicle-level machine.

6.5.2 Data Description

For the purpose of intra-vehicle network IDS development, the first used dataset is the CAN-intrusion-dataset proposed in 2018 [10]. The dataset is generated by logging CAN traffic via the OBD-II port of a vehicle when CAN attacks are launched. The features of this dataset include timestamp, CAN ID, data length code (DLC), and the 8-bit data field of CAN packets (DATA[0]-DATA[7]). Since the feature "timestamp" has a strong correlation with cyber-attack simulation periods and can lead to biased models and results, this feature was removed from the feature space. As the authors in [10] created the CAN-intrusion-dataset by injecting attacks into random CAN IDs, this CAN ID feature is retained. Based on the CAN packet structure shown in Fig. 6.1, ten features extracted from the identifier field and data field of CAN packets, including CAN ID, DLC, and DATA[0]-DATA[7], were preliminarily selected for IDS development. The attack types of the CAN-intrusion-dataset are shown in Table 6.4. Moreover, the label distributions of the original dataset, the training set, and the test set are shown in Table 6.4. As the minority classes have large numbers of samples (at least 491,847 samples), SMOTE is not required for balancing the CAN-intrusion-dataset.

For external network IDS development, there is a shortage of public IoV benchmark datasets due to popularization, privacy, and commercialization issues [22] [76] [77]. On the other hand, wireless local area networks (WLANs) and cellular networks are common communication strategies for IoV and connected vehicles, so the attacks launched on conventional computer networks can be considered similar to the intrusions carried out on external vehicular networks [22] [78]. Therefore, many research projects and papers [20] - [26] develop external vehicular network IDSs using general network security datasets, including KDD-99, NSL-KDD, Kyoto 2006+, UNSW-NB15, and CICIDS2017 [78]. Among these cyber-security datasets, CI-

Table 6.4: Class Label and Size of The CAN-Intrusion-Dataset

Class Label	Original Number of Samples	Number of Training Set Samples	Number of Test Set Samples
Normal	14,037,293	9,826,105	4,211,188
DoS	587,521	411,265	176,256
Fuzzy	491,847	344,293	147,554
RPM Spoofing	654,897	458,428	196,469
Gear Spoofing	597,252	418,076	179,176

CIDS2017 [12] is the most representative dataset of current external networks because it is the most state-of-the-art dataset and contains more features, instances, and cyber-attack types than other datasets [22]. Thus, the network traffic flow data in the CICIDS2017 dataset is chosen in the proposed MTH-IDS to represent the complex external vehicular network data. Moreover, to better relate the CICIDS2017 dataset to IoV applications, we have associated each type of attack in the CICIDS2017 dataset with the external vehicular network threats described in Table 6.2 based on the detailed analysis of the CICIDS2017 dataset in [79]. The specifics of the CICIDS2017 dataset and the corresponding external vehicular attack types are shown in Table 6.5. Since the Bot, brute-force, infiltration, and web attack classes are minority classes with small numbers of samples (from 36 to 13,835), the SMOTE method described in Section 6.4.2 was implemented to synthesize more samples to enable the minority classes to have at least 100,000 samples. Addressing class-imbalance can avoid obtaining biased models with low attack detection rates.

6.5.3 Performance Analysis of Known Intrusion Detection

To evaluate the proposed IDS for known intrusion detection, the ML models in the signature-based IDS are trained and tested on the two labeled datasets that represent intra-vehicle and external vehicular network traffic data. The results of 10-fold cross-validation of the proposed models on the CAN-intrusion dataset [10] and the CICIDS2017 dataset [12] are shown in Tables 6.6 and 6.7, respectively.

As shown in Table 6.6, after utilizing the proposed IG-FCBF feature selection method with the optimized correlation threshold ($\alpha = 0.9$) to select the top four significant features (“CAN ID”, “DATA[5]”, “DATA[3]”, and “DATA[1]”), the proposed signature-based IDS can reach

Table 6.5: Class Label, Attack Type, and Size of The CICIDS2017 Dataset

Class Label	**Corresponding Attack Type in Table 6.2 [79]**	**Original Number of Samples**	**Number of Training Set Samples After Balancing**	**Number of Test Set Samples**
BENIGN	-	2,273,097	1,591,168	681,929
Bot	Botnets	1,966	100,000	590
DDoS	DoS	380,699	266,489	114,210
DoS GoldenEye				
DoS Hulk				
DoS Slow-httptest				
DoS Slowloris				
Heartbleed				
Port-Scan	Sniffing	158,930	111,251	47,679
SSH-Patator	Brute-Force	13,835	100,000	4,150
FTP-Patator				
Infiltration	Infiltration	36	100,000	11
Web Attack – Brute Force	Web Attack	2,180	100,000	654
Web Attack – Sql Injection				
Web Attack – XSS				

high accuracy of 99.999% on the CAN-intrusion dataset. The proposed method is compared with recent promising approaches proposed in the literature [13]– [15], [18]– [20], as shown in Table 6.6. As the attack and normal patterns in this dataset can be obviously distinguished, most of the compared methods also achieve high accuracy. The authors in [19] proposed a deep convolutional neural network (DCNN) method that can achieve a high average F1-score of 0.9991, but it requires the model training machines to have graphics processing units (GPUs), making it difficult to be used in vehicle-level systems due to budget constraints. For six other compared approaches proposed in the recent literature, our proposed system achieves at least

Table 6.6: Performance Evaluation of Classifiers on The CAN-Intrusion-Dataset

Method	**Acc (%)**	**DR (%)**	**FAR (%)**	**F1**	**Execution Time (S)**
KNN [13]	97.4	96.3	5.3	0.934	911.6
SVM [13]	96.5	95.7	4.8	0.933	13765.6
XYF-K [14]	99.1	98.39	0.0	0.9879	-
SAIDuCANT [15]	87.21	86.66	1.76	0.92	-
SSAE [18]	-	98.5	2.0	0.98	-
DCNN [19]	99.93	99.84	0.16	0.9991	-
LSTM-Autoencoder [20]	99.0	99.0	0.0	0.99	-
MTH-IDS	**99.999**	**99.999**	**0.0006**	**0.99999**	**365.3**

Table 6.7: Performance Evaluation of Classifiers on The CICIDS2017 Dataset

Method	**Acc (%)**	**DR (%)**	**FAR (%)**	**F1**	**Execution Time (S)**
KNN [12]	96.3	96.2	6.3	0.963	15243.6
RF [12]	98.82	98.8	0.145	0.988	1848.3
SU-IDS [27]	99.13	99.65	1.4	-	-
STDeepGraph [28]	99.4	98.6	1.3	-	-
DBN [80]	98.95	95.82	4.19	0.9581	-
GAN-RF [81]	99.83	98.68	7.24	0.9504	-
DeepCoin [82]	99.811	94.10	0.986	-	-
Multi-SVM [83]	98.55	98.22	0.41	0.983	34896.5
PCA-RF [84]	99.6	99.6	1.0	0.996	-
MTH-IDS (Without FS & HPO)	99.861	99.753	0.110	0.99860	5238.4
MTH-IDS (Multi-Class Model)	**99.879**	**99.818**	**0.101**	**0.99879**	**1563.4**
MTH-IDS (Binary Model)	**99.895**	**99.806**	**0.084**	**0.99895**	**478.2**

0.89% accuracy and F1-score improvement.

The experimental results on the CICIDS2017 dataset are shown in Table 6.7. By implementing the proposed IG-FCBF method with the optimized correlation threshold ($\alpha = 0.9$) to select 20 features from 80 original features and using HPO methods to obtain optimized ML models, the F1-score of the proposed IDS has improved from 99.861% to 99.879%, and the execution time has decreased by 70.2%, as shown in Table 6.7. This justifies the proposed FS method and the BO-TPE method can greatly improve the system efficiency and slightly improve the model accuracy. In addition to the multi-classification results used to evaluate the IDS's capacity to detect various types of attacks, the IDS is also implemented to train a binary classification model that can distinguish between normal and abnormal network traffic data and return one of these two labels. As shown in Table 6.7, the proposed IDS reaches 99.895% accuracy and saves 69.4% of the execution time by training the binary classification model. Binary classifiers and multi-classifiers can be chosen according to the specific needs of users.

Although there are many existing works that evaluate their models on the CICIDS2017 dataset, most of them have different or inexplicit validation configurations. Nevertheless, the proposed model is quantitatively compared with recent promising methods that achieve good performance on the CICIDS2017 dataset and have a similar validation configuration, proposed in the literature [12], [27]- [28], [80]- [84]. The approaches proposed in the literature [80]- [82] have achieved high accuracy (98.95%-99.83%), but a relatively low detection rate (94.10%-

98.68%) and F1-score (0.9504-0.9581) due to the imbalanced dataset. Other methods proposed in the literature [12], [27]- [28], [83]- [84] have also achieved high accuracy or F1-scores, but still slightly lower than our proposed model. To summarize, even though the comparison with existing approaches is not a straightforward process, the proposed system shows better performance by at least achieving a 0.279% higher F1-score than of the same-level validation configuration approaches by quantitative comparison.

Therefore, the experimental results show that the proposed IDS can efficiently separate normal and malicious network traffic data and effectively detect various types of known cyber-attacks in vehicle systems.

6.5.4 Performance Analysis of Unknown Intrusion Detection

At this stage, all the models in the anomaly-based IDS are trained for binary classification by labeling the instances of all attack types as "attack" and normal instances as "normal". In the proposed system, after being evaluated by the signature-based IDS, all data samples of known attack types will be returned, and other normal instances will be labeled "suspicious" and passed to the anomaly-based IDS to determine whether any unknown attacks exist.

For the CAN-intrusion-dataset that represents intra-vehicle network traffic data, each type of CAN attack is regarded as a new attack type in each experiment. The evaluation results of unknown CAN attack detection are shown in Table 6.8. For DoS, gear spoofing, and RPM spoofing attack types, the proposed system can reach 100% detection rates, very low false alarm rates (0.0%-0.449%), and very high F1-scores (more than 0.997). However, the DR and F1 for the fuzzy attack are much lower (73.053% and 0.84389). This is because the feature values of the fuzzy attack packets are random numerical values, and certain random values can be very similar to normal packets, making it difficult for unsupervised learning algorithms to distinguish them. Moreover, the performance of the proposed anomaly-based IDS is compared to the CL-k-means model without biased classifiers. Table 6.8 shows that the F1 score of the proposed MTH-IDS on the CAN-intrusion-dataset can be largely improved from 0.82643 to 0.96307 by implementing the two biased classifiers after the CL-k-means model. To summarize, the proposed system can effectively detect most unknown attacks on intra-vehicle net-

Table 6.8: Performance Evaluation on Each Type of Unknown Attack of The CAN-Intrusion-Dataset

Attack Type	**Validation Instances**	**DR (%)**	**FAR (%)**	**F1**
DoS	1,289,386	100.0	0.0	1.0
Fuzzy	1,193,712	73.053	0.057	0.84389
Gear Spoofing	1,299,117	100.0	0.449	0.99736
RPM Spoofing	1,356,762	100.0	0.003	0.99998
Average (MTH-IDS)	**5,138,977**	**93.740**	**0.128**	**0.96307**
Average (CL-k-means)	5,138,977	79.233	0.261	0.82643

Table 6.9: Performance Evaluation on Each Type of Unknown Attack of The CICIDS2017 Dataset

Attack Type	**Validation Instances**	**DR (%)**	**FAR (%)**	**F1**
Bot	3,932	63.276	21.669	0.68426
DDoS	256,054	62.697	11.698	0.71902
DoS GoldenEye	20,586	83.931	20.461	0.82127
DoS Hulk	462,146	67.440	11.806	0.75248
DoS Slow-httptest	10,998	76.687	19.094	0.78339
DoS Slowloris	11,592	83.834	7.902	0.87447
FTP-Patator	15,876	51.298	12.686	0.62564
Heartbleed	22	100.0	18.182	0.91667
Infiltration	72	72.222	5.556	0.81250
Port-Scan	317,860	98.962	17.849	0.91288
SSH-Patator	11,794	95.828	23.351	0.87443
Web Attack – Brute Force	3,014	89.516	17.319	0.86558
Web Attack – Sql Injection	42	95.238	23.810	0.86957
Web Attack – XSS	1,304	3.681	14.417	0.06233
Average (MTH-IDS)	**1,115,292**	**75.943**	**13.882**	**0.80013**
Average (CL-k-means)	1,115,292	72.682	15.357	0.77305

works except fuzzy attacks.

On the other hand, training on more different types of attack samples enables us to design a more comprehensive IDS that can detect more unknown attack types effectively [85]. Therefore, several experiments were conducted on the CICIDS2017 dataset that contains data samples of 14 different common cyber-attacks types to illustrate potential attacks launched on external vehicular networks. In each experiment of the validation process, each type of attack is regarded as an unknown attack, and the results are shown in Table 6.9.

From Table 6.9, it can be seen that the proposed system exhibits different performances when applied to the experiments of different types of unknown attacks. By implementing the proposed methods, the false alarm rates for most of the attack types are at a low level of

Table 6.10: Performance Evaluation on The Untouched Test Set

Dataset	Acc (%)	DR (%)	FAR (%)	F1
CAN-intrusion-dataset	99.99	100.0	0.00005	0.9999
CICIDS2017	99.88	99.77	0.10	0.9988

Table 6.11: Model Evaluation on A Vehicle-Level System

System Component	Dataset 1: Avg Test Time (ms)	Dataset 2: Avg Test Time (ms)	Dataset 1: Model Space (MB)	Dataset 2: Model Space (MB)
Z-score	0.028	0.031	-	-
KPCA	0.005	0.009	0.002	0.006
Stacking	0.389	0.297	2.02	14.36
CL-k-means	0.145	0.157	0.48	0.78
Biased Classifiers	0.007	0.015	0.11	1.06
Sum	**0.574**	**0.509**	**2.61**	**16.21**

less than 20%. The detection rates for the “Heartbleed”, “Port-Scan”, “SSH-Patator”, “Web Attack – Brute Force”, and “Web Attack – Sql Injection” attacks are high (from 89.516% to 100%), while the detection rates for other types of attacks are relatively lower (from 51.298% to 83.931%). The F1-scores for most of the attack types are larger than 0.80. The only type of attack that the proposed system cannot detect effectively is the “Web Attack – XSS” whose results show a very low F1-score (0.062), because their data distribution is very similar to normal data distributions. The average F1-score of the proposed MTH-IDS on all the attacks is 0.80013, which is higher than the CL-k-means model without biased classifiers (0.77305).

Thus, the proposed IDS can detect most of the previously-unseen types of attacks with a relatively high detection rate and a relatively low false alarm rate on both intra-vehicle and external vehicular networks. Nevertheless, there is still some room for improvement since effectively detecting zero-day attacks is still an unsolved research problem.

6.5.5 Vehicle-Level Model Evaluation and Discussion

The proposed IDS with all trained models are tested on Raspberry Pi 3, a vehicle-level machine, to evaluate its feasibility in vehicular environments. Moreover, implementing the proposed model on the untouched test sets can evaluate its generalizability.

The experimental results on the test sets are shown in Table 6.10. As shown in Table 6.10, the F1-scores of the proposed IDS on the 30% test sets of the CAN-intrusion-dataset and CI-

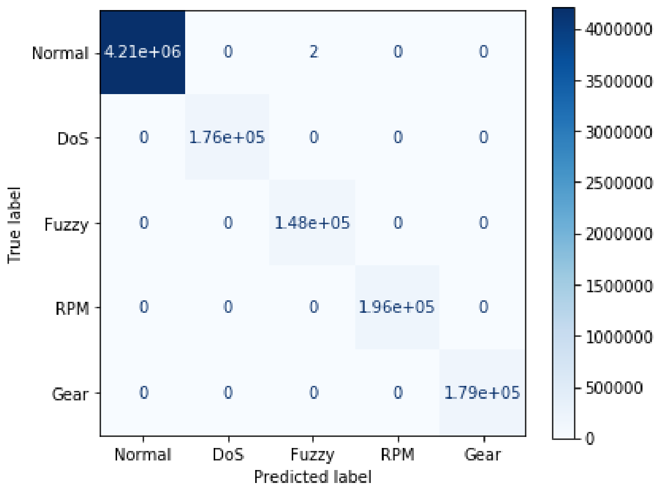

Figure 6.4: Confusion matrix for the test set of CAN-intrusion-dataset.

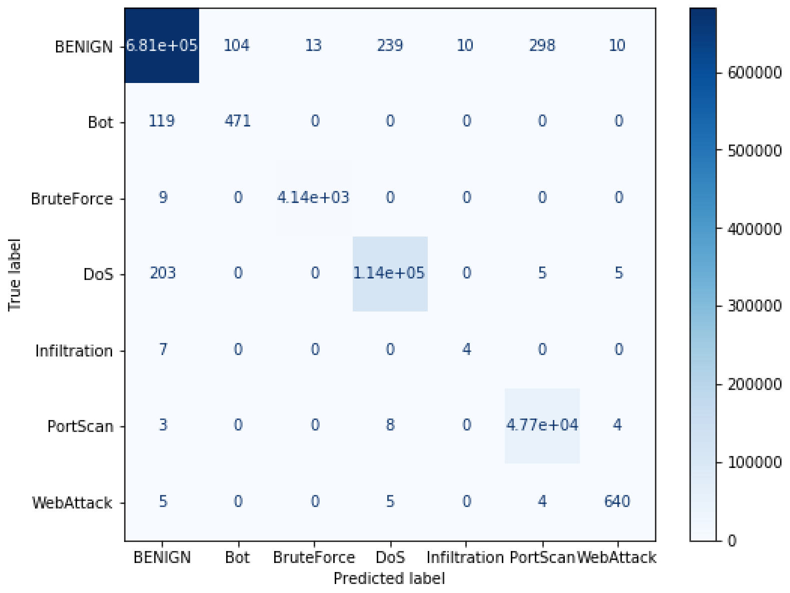

Figure 6.5: Confusion matrix for the test set of CICIDS2017 dataset.

CIDS2017 dataset are 99.99% and 99.88%, respectively. Moreover, the confusion matrices of evaluating the proposed method on the test sets of the CAN-intrusion-dataset and CICIDS2017 dataset are shown in Fig. 6.4 and Fig. 6.5, respectively. For the CAN-intrusion-dataset, as shown in Fig. 6.4, the proposed method can accurately detect all the DoS, RPM spoofing, and gear spoofing attack samples, and only has two false alarms for the fuzzy attack detection. These results are in line with other similar works from the literature that used the same dataset. For example, the authors in [19] also achieved high accuracy of 99.93%. The main reason for achieving high accuracy is that the large difference between the attack and normal patterns in the CAN-intrusion-dataset can be obviously distinguished. For the CICIDS2017 results shown in Table 6.10 and Fig. 6.5, the attack patterns are more difficult to be distinguished than the CAN-intrusion-dataset, but the classification error rate is still at a very low level (0.12%), except for the infiltration attack type that has 7 misclassified samples out of 11 test samples. This is because the number of data samples for the infiltration attack is only 36, which is insufficient to train an effective classifier to accurately identify this attack. Nevertheless, the proposed model can accurately detect other attacks with an overall F1-score of 99.88%. As the test sets were untouched before the hold-out validation, the high performance indicates the strong generalizability of the proposed framework on new datasets.

The proposed model can achieve high performance without over-fitting mainly due to the following reasons [86]:

1. It trains on large-sized datasets, as using more data samples can improve the generalizability of the proposed method.
2. It implements a comprehensive feature engineering method to improve the generalizability by removing irrelevant and misleading features that may cause over-fitting.
3. It uses the stacking ensemble method to combine the results of base learners. Ensemble models often have better generalizability than single models because combining the single learners can reduce the variance of estimation and prevent over-fitting.

On the other hand, to deploy the proposed IDS in real-world vehicle systems, the real-time requirements of vehicle safety services should be met. For each packet transmitted on vehicular networks, the alarm generation time should be less than 10ms to meet the real-time requirements, as described in Section 6.3.4. For each packet passed to the proposed IDS, it will be processed by Z-score normalization and four trained models: KPCA, stacking, CL-k-means, and a biased classifier (either FN-based or FP-based). As shown in Table 6.11, the average of the processing time for each packet in the CAN-intrusion-dataset (Dataset 1) and the CICIDS2017 dataset (Dataset 2) is only 0.574 ms and 0.509 ms, respectively, which is much lower than the vehicular network security latency requirement (10ms). Moreover, since we trained the models on the small-size subsets obtained by the proposed k-means cluster sampling method, the total size of the trained models for the intra-vehicle network IDS and external network IDS is only 2.61 MB and 16.21 MB, respectively, which is much less than the memory limit of vehicle-level machines that often have more than 1 GB RAM, like Raspberry Pi 3. Thus, the experimental results show the feasibility of applying the proposed system to real-time vehicle systems.

6.6 Conclusion

To enhance IoV security, this work proposed a multi-tiered hybrid intrusion detection system (MTH-IDS) model that can detect various types of known and zero-day cyber-attacks on both intra-vehicle and external-vehicular networks for modern vehicles. The proposed MTH-IDS consists of two traditional ML stages (data pre-processing and feature engineering) and four main tiers of learners utilizing multiple machine learning algorithms. Through data pre-processing and feature engineering, the quality of the input data can be significantly improved for more accurate model learning. The first tier of the proposed system consists of four tree-based supervised learners used for known attack detection, while the second tier comprises the BO-TPE and stacking models for supervised base learner optimization to achieve higher accuracy. The third tier consists of a novel CL-k-means unsupervised model used for unknown/zero-day attack detection. Lastly, BO-GP and two biased classifiers are used to construct the fourth tier for unsupervised learner optimization. The four tiers of learning models enable the proposed MTH-IDS to achieve optimal performance for both known and unknown attack detection in vehicular networks. Through the performance evaluation of the proposed IDS on the two public datasets that represent intra-vehicle and external vehicular network data, the proposed system can effectively detect various types of known attacks with accuracies of 99.99% and 99.88% on the CAN-intrusion-dataset and CICIDS2017 dataset, respectively. Moreover, the proposed system can detect various types of unknown attacks with average F1-scores of 0.963 and 0.800 on the CAN-intrusion-dataset and CICIDS2017 dataset, respectively. The experimental results on a vehicle-level machine show the feasibility of the proposed system in real-time environments. In future work, the anomaly-based IDS framework can be further improved by doing research on other unsupervised learning and online learning methods.

Bibliography

[1] L. Yang, A. Moubayed, and A. Shami, "MTH-IDS: A Multi-Tiered Hybrid Intrusion Detection System for Internet of Vehicles," *IEEE Internet Things J.*, 2021.

[2] H. Liang *et al.*, "Network and system level security in connected vehicle applications," *IEEE/ACM Int. Conf. Comput. Des. Dig. Tech. Pap. ICCAD*, pp. 1–7, 2018.

[3] M. Gmiden, M. H. Gmiden, and H. Trabelsi, "An intrusion detection method for securing in-vehicle CAN bus," *2016 17th Int. Conf. Sci. Tech. Autom. Control Comput. Eng. STA 2016 - Proc.*, pp. 176–180, 2017.

[4] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-vehicle network attacks and countermeasures: Challenges and future directions," *IEEE Netw.*, vol. 31, no. 5, pp. 50–58, 2017.

[5] O. Y. Al-Jarrah, C. Maple, M. Dianati, D. Oxtoby, and A. Mouzakitis, "Intrusion Detection Systems for Intra-Vehicle Networks: A Review," *IEEE Access*, vol. 7, pp. 21266–21289, 2019.

[6] L. Yang, "Comprehensive Visibility Indicator Algorithm for Adaptable Speed Limit Control in Intelligent Transportation Systems", M.A.Sc. thesis, University of Guelph, 2018.

[7] J. Golson, "Jeep hackers at it again, this time taking control of steering and braking systems," *The Verge*, Aug. 2016. [Online]. Available: https://www.theverge.com/2016/8/2/12353186/car-hack-jeep-cherokee-vulnerability-miller-valasek. [Accessed: 11-Nov-2020].

[8] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, "Tree-based Intelligent Intrusion Detection System in Internet of Vehicles," *proc. 2019 IEEE Glob. Commun. Conf.*, pp. 1–6, Hawaii, USA, 2019.

[9] Q. Wang, Y. Qian, Z. Lu, Y. Shoukry, and G. Qu, "A delay based plug-in-monitor for Intrusion Detection in Controller Area Network," *Proc. 2018 Asian Hardw. Oriented Secur. Trust Symp. AsianHOST 2018*, pp. 86–91, 2019.

[10] E. Seo, H. M. Song, and H. K. Kim, "GIDS: GAN based Intrusion Detection System for In-Vehicle Network," *2018 16th Annu. Conf. Privacy, Secur. Trust. PST 2018*, pp. 1–6, 2018.

[11] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, "Machine learning towards intelligent systems: applications, challenges, and opportunities," *Artif. Intell. Rev.*, 2021.

[12] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *in Proc. Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[13] A. Alshammari, M. A. Zohdy, D. Debnath, and G. Corser, "Classification Approach for Intrusion Detection in Vehicle Systems," *Wirel. Eng. Technol.*, vol. 09, no. 04, pp. 79–94, 2018.

[14] V. S. Barletta, D. Caivano, A. Nannavecchia, and M. Scalera, "A Kohonen SOM Architecture for Intrusion Detection on In-Vehicle Communication Networks," *Appl. Sci.*, vol. 10, no. 15, 2020.

[15] H. Olufowobi, C. Young, J. Zambreno, and G. Bloom, "SAIDuCANT: Specification-Based Automotive Intrusion Detection Using Controller Area Network (CAN) Timing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 1484–1494, 2020.

[16] H. Olufowobi et al., "Anomaly Detection Approach Using Adaptive Cumulative Sum Algorithm for Controller Area Network," *AutoSec 2019 - Proc. ACM Work. Automot. Cybersecurity, co-located with CODASPY 2019*, pp. 25–30, 2019.

[17] H. Lee, S. H. Jeong, and H. K. Kim, "OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame," *Proc. - 2017 15th Annu. Conf. Privacy, Secur. Trust. PST 2017*, pp. 57–66, 2018.

[18] S. F. Lokman *et al.*, "Stacked Sparse Autoencoders-Based Outlier Discovery for In-Vehicle Controller Area Network (CAN)," *Int. J. Eng. Technol.*, vol. 7, no. 4.33, pp. 375-380, 2018.

[19] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Veh. Commun.*, vol. 21, p. 100198, 2020.

[20] J. Ashraf, A. D. Bakhshi, N. Moustafa, H. Khurshid, A. Javed, and A. Beheshti, "Novel Deep Learning-Enabled LSTM Autoencoder Architecture for Discovering Anomalous Events From Intelligent Transportation Systems," *IEEE Trans. Intell. Transp. Syst.*, pp. 1–12, 2020.

[21] K. M. Ali Alheeti and K. Mc Donald-Maier, "Intelligent intrusion detection in external communication systems for autonomous vehicles," *Syst. Sci. Control Eng.*, vol. 6, no. 1, pp. 48–56, 2018.

[22] A. Rosay, F. Carlier, and P. Leroux, "Feed-forward neural network for Network Intrusion Detection," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, May 2020, pp. 1–6.

[23] K. Aswal, D. C. Dobhal, and H. Pathak, "Comparative analysis of machine learning algorithms for identification of BOT attack on the Internet of Vehicles (IoV)," in *2020 International Conference on Inventive Computation Technologies (ICICT)*, Feb. 2020, pp. 312–317.

[24] M. Aloqaily, S. Otoum, I. Al Ridhawi, and Y. Jararweh, “An intrusion detection system for connected vehicles in smart cities,” *Ad Hoc Networks*, vol. 90, p. 101842, 2019.

[25] Y. Gao, H. Wu, B. Song, Y. Jin, X. Luo, and X. Zeng, “A Distributed Network Intrusion Detection System for Distributed Denial of Service Attacks in Vehicular Ad Hoc Network,” *IEEE Access*, vol. 7, pp. 154560–154571, 2019.

[26] D. A. Schmidt, M. S. Khan, and B. T. Bennett, “Spline-based intrusion detection for VANET utilizing knot flow classification,” *Internet Technol. Lett.*, vol. 3, no. 3, pp. 2–7, 2020.

[27] E. Min, J. Long, Q. Liu, J. Cui, Z. Cai, and J. Ma, “SU-IDS: A semi-supervised and unsupervised framework for network intrusion detection,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11065 LNCS, pp. 322–334, 2018.

[28] Y. Yao, L. Su, Z. Lu, and B. Liu, “STDeepGraph: Spatial-temporal deep learning on communication graphs for long-term network attack detection,” *Proc. - 2019 18th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. IEEE Int. Conf. Big Data Sci. Eng. Trust. 2019*, pp. 120–127, 2019.

[29] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, “Multi-Stage Optimized Machine Learning Framework for Network Intrusion Detection,” *IEEE Trans. Netw. Serv. Manag.*, 2020.

[30] K. Zdeněk and S. Jiří, “Simulation of CAN bus physical layer using SPICE,” *Int. Conf. Appl. Electron.*, pp. 8–11, 2013.

[31] S. F. Lokman, A. T. Othman, and M. H. Abu-Bakar, “Intrusion detection system for automotive Controller Area Network (CAN) bus system: a review,” *Eurasip J. Wirel. Commun. Netw.*, vol. 2019, no. 1, 2019.

[32] L. Yang, R. Muresan, A. Al-Dweik and L. J. Hadjileontiadis, ”Image-Based Visibility Estimation Algorithm for Intelligent Transportation Systems,” in *IEEE Access*, vol. 6, pp. 76728-76740, 2018.

[33] Y. Fraiji, L. Ben Azzouz, W. Trojet, and L. A. Saidane, “Cyber security issues of Internet of electric vehicles,” *IEEE Wirel. Commun. Netw. Conf. WCNC*, vol. 2018-April, pp. 1–6, 2018.

[34] B. Groza and P. S. Murvay, “Efficient Intrusion Detection with Bloom Filtering in Controller Area Networks,” *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 4, pp. 1037–1051, 2019.

[35] U. E. Larson, D. K. Nilsson, and E. Jonsson, “An approach to specification-based attack detection for in-vehicle networks,” *IEEE Intell. Veh. Symp. Proc.*, pp. 220–225, 2008.

[36] D. Zhou, Z. Yan, Y. Fu, and Z. Yao, “A survey on network data collection,” *J. Netw. Comput. Appl.*, vol. 116, no. May, pp. 9–23, 2018.

[37] A. Moubayed, A. Refaey, and A. Shami, "Software-Defined Perimeter (SDP): State of the Art Secure Solution for Modern Networks," *IEEE Netw., vol.* 33, no. 5, pp. 226–233, 2019.

[38] P. Kumar, A. Moubayed, A. Refaey, A. Shami, and J. Koilpillai, "Performance Analysis of SDP For Secure Internal Enterprises," *IEEE Wirel. Commun. Netw. Conf. WCNC*, vol. 2019-April, 2019.

[39] A. Moubayed and A. Shami, "Softwarization, Virtualization, & Machine Learning For Intelligent & Effective V2X Communications," *IEEE Intell. Transp. Syst. Mag.*, pp. 1–14, 2020.

[40] M. Y. Abualhoul, O. Shagdar, and F. Nashashibi, "Visible Light inter-vehicle Communication for platooning of autonomous vehicles," *IEEE Intell. Veh. Symp. Proc.*, vol. 2016-Augus, no. IV, pp. 508–513, 2016.

[41] A. Moubayed, A. Shami, P. Heidari, A. Larabi, and R. Brunner, "Edge-enabled V2X Service Placement for Intelligent Transportation Systems," *IEEE Trans. Mob. Comput.*, vol. 1233, pp. 1–13, 2020.

[42] K. M. Faraoun and A. Boukelif, "Neural Networks Learning Improvement using the K-Means Clustering Algorithm to Detect Network Intrusions," *INFOCOMP J. Comput. Sci.*, vol. 5, no. 3, pp. 28–36, 2006.

[43] N. Shi, X. Liu, and Y. Guan, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," *3rd Int. Symp. Intell. Inf. Technol. Secur. Informatics, IITSI 2010*, pp. 63–67, 2010.

[44] A. Moubayed, M. Injadat, A. Shami, and H. Lutfiyya, "Student Engagement Level in e-Learning Environment: Clustering Using K-means," *Am. J. Distance Educ.*, vol. 34, no. 02, pp. 1–20, 2020.

[45] A. Moubayed, M. Injadat, A. Shami, and H. Lutfiyya, "DNS Typo-Squatting Domain Detection: A Data Analytics & Machine Learning Based Approach," *2018 IEEE Glob. Commun. Conf. GLOBECOM 2018 - Proc.*, 2018.

[46] L. Yang and A. Shami, "On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[47] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, "Multi-split Optimized Bagging Ensemble Model Selection for Multi-class Educational Data Mining," *Springer's Appl. Intell.*, 2020.

[48] M. Injadat, F. Salo, A. B. Nassif, A. Essex, and A. Shami, "Bayesian Optimization with Machine Learning Algorithms Towards Anomaly Detection," *2018 IEEE Glob. Commun. Conf. GLOBECOM 2018 - Proc.*, 2018.

[49] Z. Chen *et al.*, "Machine learning based mobile malware detection using highly imbalanced network traffic," *Inf. Sci. (Ny).*, vol. 433–434, pp. 346–364, 2018.

[50] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer, "SMOTE: Synthetic Minority Over-Sampling Technique," *J. Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002.

[51] F. Salo, A. B. Nassif, and A. Essex, "Dimensionality reduction with IG-PCA and ensemble classifier for network intrusion detection," *Comput. Networks*, vol. 148, pp. 164–175, 2019.

[52] L. Yu and H. Liu, "Efficiently handling feature redundancy in high-dimensional data," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 685–690, 2003.

[53] L. Yu and H. Liu, "Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution," *Proceedings, Twent. Int. Conf. Mach. Learn.*, vol. 2, pp. 856–863, 2003.

[54] S. Egea, A. Rego Manez, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Intelligent IoT traffic classification using novel search strategy for fast-based-correlation feature selection in industrial environments," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1616–1624, 2018.

[55] M. N. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, "Systematic ensemble model selection approach for educational data mining," *Knowledge-Based Syst.*, vol. 200, p. 105992, 2020.

[56] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis," in *Artificial Neural Networks — ICANN'97*, 1997, pp. 583–588.

[57] A. Garg and P. Maheshwari, "Performance analysis of Snort-based Intrusion Detection System," *ICACCS 2016 - 3rd Int. Conf. Adv. Comput. Commun. Syst. Bringing to Table, Futur. Technol. from Arround Globe*, vol. 01, pp. 1–5, 2016.

[58] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters," *Conf. Res. Pract. Inf. Technol. Ser.*, vol. 38, no. January, pp. 333–342, 2005.

[59] S. Sahu and B. M. Mehtre, "Network intrusion detection system using J48 Decision Tree," *2015 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2015*, pp. 2023–2026, 2015.

[60] A. Tesfahun and D. Lalitha Bhaskari, "Intrusion detection using random forests classifier with SMOTE and feature reduction," *Proc. - 2013 Int. Conf. Cloud Ubiquitous Comput. Emerg. Technol. CUBE 2013*, pp. 127–132, 2013.

[61] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, 2006.

[62] Y. Xia, C. Liu, Y. Y. Li, and N. Liu, "A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring," *Expert Syst. Appl.*, vol. 78, pp. 225–241, 2017.

[63] L. Yang and A. Shami, “A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams,” *IEEE Internet Things Mag.*, 2021.

[64] M. Mohammed, H. Mwambi, B. Omolo, and M. K. Elbashir, “Using stacking ensemble for microarray-based cancer classification,” *2018 Int. Conf. Comput. Control. Electr. Electron. Eng. ICCCEEE 2018*, pp. 1–8, 2018.

[65] A. Feizollah, N. B. Anuar, R. Salleh, and F. Amalina, “Comparative study of k-means and mini batch k-means clustering algorithms in android malware detection using network traffic analysis,” *Proc. - 2014 Int. Symp. Biometrics Secur. Technol. ISBAST 2014*, pp. 193–197, 2015.

[66] A. Vikram and Mohana, “Anomaly detection in Network Traffic Using Unsupervised Machine learning Approach,” in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, Jun. 2020, pp. 476–479.

[67] MJ. Kearns, ”The computational complexity of machine learning,” *MIT press*, 1990.

[68] N. Shi, X. Liu, and Y. Guan, “Research on k-means clustering algorithm: An improved k-means clustering algorithm,” *3rd Int. Symp. Intell. Inf. Technol. Secur. Informatics, IITSI 2010*, pp. 63–67, 2010.

[69] D. Kim, Y. Tao, S. Kim, and A. Zeller, “Where should we fix this bug? A two-phase recommendation model,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 11, pp. 1597–1610, 2013.

[70] S. Yadav and S. Shukla, “Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification,” *Proc. - 6th Int. Adv. Comput. Conf. IACC 2016*, no. Cv, pp. 78–83, 2016.

[71] The pandas development team, “pandas-dev/pandas: Pandas.” *Zenodo*, Feb. 2020.

[72] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011, [Online]. Available: http://scikit-learn.sourceforge.net.

[73] T. Chen and T. He, “xgboost: eXtreme Gradient Boosting,” *R Packag*. version 0.4-2, pp. 1–4, 2015.

[74] T. Head, MechCoder, G. Louppe, et al., “scikitoptimize/scikit-optimize: v0.5.2,” 2018.

[75] B. Komer, J. Bergstra, and C. Eliasmith, “Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn,” *Proc. 13th Python Sci. Conf.*, no. Scipy, pp. 32–37, 2014.

[76] N. Kaja, “Artificial Intelligence and Cybersecurity: Building an Automotive Cybersecurity Framework Using Machine Learning Algorithms,” University of Michigan-Dearborn, 2019.

[77] O. Y. Al-Jarrah, C. Maple, M. Dianati, D. Oxtoby, and A. Mouzakitis, “Intrusion Detection Systems for Intra-Vehicle Networks: A Review,” *IEEE Access*, vol. 7, pp. 21266–21289, 2019.

[78] A. Rosay, F. Carlier, and P. Leroux, “MLP4NIDS: An Efficient MLP-Based Network Intrusion Detection for CICIDS2017 Dataset,” in *Machine Learning for Networking*, 2020, pp. 240–254.

[79] R. Panigrahi and S. Borah, “A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems,” *Int. J. Eng. Technol.*, vol. 7, no. 3.24, pp. 479-482, 2018.

[80] W. Elmasry, A. Akbulut, and A. H. Zaim, “Evolving deep learning architectures for network intrusion detection using a double PSO metaheuristic,” *Comput. Networks*, vol. 168, 2020.

[81] J. H. Lee and K. H. Park, “GAN-based imbalanced data intrusion detection system,” *Pers. Ubiquitous Comput.*, 2019.

[82] M. A. Ferrag and L. Maglaras, “DeepCoin: A Novel Deep Learning and Blockchain-Based Energy Exchange Framework for Smart Grids,” *IEEE Trans. Eng. Manag.*, pp. 1–13, 2019.

[83] R. Vijayanand, D. Devaraj, and B. Kannapiran, “Intrusion detection system for wireless mesh network using multiple support vector machine classifiers with genetic-algorithm-based feature selection,” *Comput. Secur.*, vol. 77, pp. 304–314, 2018.

[84] R. Abdulhammed, M. Faezipour, H. Musafer, and A. Abuzneid, “Efficient network intrusion detection using PCA-based dimensionality reduction of features,” *2019 Int. Symp. Networks, Comput. Commun. ISNCC 2019*, 2019.

[85] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, “Survey of intrusion detection systems: techniques, datasets and challenges,” *Cybersecurity*, vol. 2, no. 1, 2019.

[86] B. Ghojogh and M. Crowley, “The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial,” *arXiv*, pp. 1–23, 2019.

Chapter 7

Multi-Perspective Content Delivery Networks Security Framework Using Optimized Unsupervised Anomaly Detection

7.1 Introduction

With the increasing popularity of caching techniques in Internet communications, it is estimated that 71% of Internet traffic will be delivered through content delivery networks (CDNs) by 2021 [3]. A CDN is a geographically distributed network of servers that work together to provide fast communications of Internet contents, including hypertext markup language (HTML) pages, JavaScript files, images, audios, videos, etc. [4]. CDNs are developed to improve the process of content delivery through caching mechanisms and multiple edge servers [5]. As a large-scale CDN service provider, Akamai accounts for approximately 20% of all web traffic [6].

CDNs improve the connectivity and efficiency of global communications, but also introduce vulnerabilities to the connected networks. Caching mechanisms become a major target

Parts of the content in this chapter has been published in IEEE Transactions on Network and Service Management [1] and submitted to the Digital Forensics book, World Scientific [2].

of cyber-attacks since open proxy caches may be exploited by attackers to transmit malicious traffic and perform various harmful activities, which causes network congestion, unavailability, or other severe consequences [7].

Cache pollution attacks (CPAs) and denial of service (DoS) attacks are the two major types of cyber-attacks launched on CDNs to cause service unavailability or to degrade the caching service by reducing the cache hit rate and increasing latency [8] [9]. In high-rate networks, even a moderate degradation of the cache hit rate, or a moderate increase of latency may result in severe network congestion or a massive amount of additional data transmissions [7]. CPAs are launched by polluting the cache space with a large number of unpopular or illegitimate contents; therefore, legitimate clients will get many cache misses for popular files, making the caching mechanism ineffective [8]. Similarly, DoS attacks are launched by sending a sudden burst of requests to exhaust the network resources of certain targeted nodes. However, DoS attacks are not necessarily launched by sending requests for unpopular files [9].

To protect a CDN against DoS and CPAs, an effective method is to explore and analyze network access logs for the purpose of abnormal behavior analysis and attack pattern detection in CDNs [10]. In this chapter, we focus on anomaly detection in CDNs by analyzing 169 gigabytes (GB) of unlabeled real-world CDN access log data provided by a major CDN operator. This work aims to detect DoS attacks and CPAs based on the behaviors of abnormal network entities, including the malicious Internet Protocol (IP) addresses, abnormal contents, and compromised nodes, through the analysis of access logs. The proposed anomaly detection framework consists of a multi-perspective feature engineering, an unsupervised machine learning (ML) model built with optimized isolation forest (iForest) [11] and Gaussian mixture models (GMM) [12], and a multi-perspective result validation method. The proposed work can be considered a labeling technique on unlabeled CDN log data for anomaly detection use cases.

On the other hand, as the data is completely unlabeled, multiple experts from Ericsson Inc. were involved in the learning phase to help construct effective ML models, which is a standard data learning process named human-in-the-loop (HITL). HITL is the process of creating ML models by leveraging the power of both machine and human intelligence [13] [14]. Labeling massive amounts of data usually need HITL to obtain accurate labels for unlabeled data in the unsupervised learning process, as ML models themselves are often unable to determine true labels by themselves [14]. Therefore, the HITL process is included in the proposed framework to ensure accurate anomaly detection on the unlabeled dataset. HITL in the proposed framework mainly includes the attack pattern & feature analysis, ML result analysis, and final result validation.

Detecting potential cyber-attacks and affected abnormal network entities can also trigger other network defense and mitigation mechanisms, such as blacklisting malicious client IPs, isolating compromised nodes, and removing abnormal cached contents out of the cache space [7]. Thus, CDNs can recover from cyber-attacks or be prevented from potential attacks with the help of effective network anomaly detection techniques.

This chapter makes the following contributions:

1. It summarizes the potential patterns and characteristics of DoS and CPA attacks to assist with anomaly detection in CDNs;
2. It proposes a comprehensive network feature engineering model that generates features from multiple perspectives, including content, client IP, service provider, and account-offering perspectives;
3. It proposes an optimized unsupervised anomaly detection model utilizing iForest, GMM, and Bayesian optimization (BO), to detect cyber-attacks and affected network entities effectively;
4. It proposes a multi-perspective result validation technique that can effectively reduce the false alarm rate and improve the detection rate of unsupervised CDN anomaly detection models.

This chapter is organized as follows: Section 7.2 provides an overview of CDNs and potential cyber-attacks. Section 7.3 presents the related works regarding network anomaly detection. Section 7.4 discusses the proposed anomaly detection model in detail, including the model framework, feature engineering, utilized algorithms, and validation procedures. Section 7.5 presents and discusses the experimental results. Section 7.6 discusses the open issues and practical usage of the proposed framework. Section 7.7 concludes the chapter.

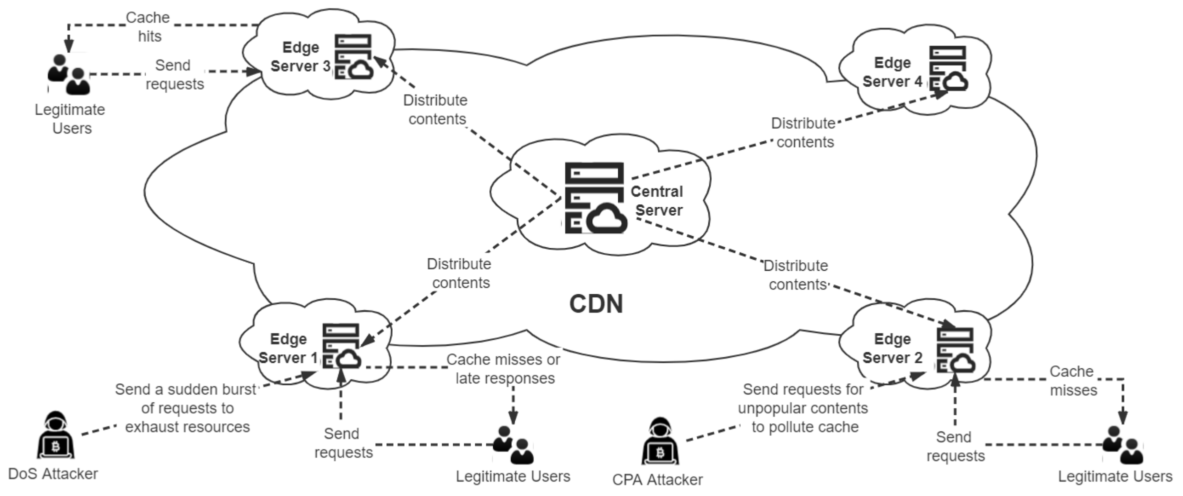

Figure 7.1: An overview of CDN and cyber-attack scenarios.

7.2 Problem Statement

7.2.1 CDN Overview

In traditional Internet, the same contents are required to be transmitted from servers to clients repeatedly. With the rapidly increasing demand for large-scale content distribution, content networks, or caching networks, have been developed to improve content delivery efficiency [15] [16]. In content networks, contents can be cached in servers to serve future requests [17]. As a common type of content network and an effective solution for large-scale content delivery, content delivery networks (CDNs) have been widely deployed in modern networks [17]. As the most important strategy of caching networks and CDNs, caching is the process of storing copies of content in temporary storage locations or named caches, which can largely reduce latency, enabling fast access to websites or applications [3] [16]. Through caching mechanisms, CDNs can cache content in edge servers that are closer to end-users than the central server, making it more efficient to deliver web contents [4]. An overview of CDN is shown in Fig. 7.1.

CDN works in the following procedures [4]:

1. When a CDN receives a request for a content from a client, this request is routed to the closest edge server to the client.
2. The closest edge server will fetch the content from the central server that has this content.
3. The edge server responds to the client with the requested content.

4. A copy of the requested content is stored in the edge server as the caching process for future requests. The cached content will be retained in the cache space if the end-users keep requesting the same content.

If the content requested by an end-user has been saved in the cache space of the edge server, the content will be loaded at a fast speed, so-called a cache hit. In contrast, if the content has not been saved in the cache space, a cache miss will occur, and the edge server will pass the request to the central server to fetch the content and save it in the cache space to reduce the future request processing speed. Thus, CDNs can handle the rapidly growing volumes of network traffic and Internet content with low latency through edge servers and caching mechanisms. Moreover, CDNs can mitigate DDoS attacks because the traffic can be dispersed to multiple edge servers to keep responding to users' requests, making it difficult for cyber-attackers to paralyze the entire network [18].

On the other hand, the research works for CDN anomaly detection are very limited, because CDN operators (*e.g.*, Akamai, Limelight) often keep CDN traffic data private due to liability implications [19]. Most research works of anomaly detection in caching networks are for information-centric networks (ICNs) [20]. Information-Centric Networking (ICN), alternatively known as Named Data Networking (NDN) or Content-Centric Networking (CCN), is a future Internet architecture that can be regarded as an improved version of CDNs to provide large-scale content delivery with less resource footprint and system complexity [17] [21]. Caching is an essential component in both ICNs and CDNs, since they both aim to provide efficient content delivery through caching mechanisms [3] [17]. Thus, ICNs and CDNs are both vulnerable to the cyber-attacks that aim to disrupt caching services, like DoS attacks and CPAs. The main difference between ICNs and CDNs is that ICNs assign a unique name for each content as its identifier to replace IP addresses for content delivery, while IP address information is required in CDNs [17] [22].

7.2.2 Service Targeting Attacks in CDN

However, the use of caching mechanisms introduces several vulnerabilities to caching networks, especially CDNs [16]. Several service targeting attacks can be carried out on CDNs to disrupt their services by exploiting the caching mechanisms of CDNs.

Firstly, if a cache misses occurs, even though the content can be an unpopular or illegitimate file, the edge server close to the end-user will fetch this content from the central server and store it in the cache space. However, due to the memory constraints of server machines, a cache can only save a limited number of contents [23]. Based on this caching strategy, cyber-attackers can send a large or moderate number of requests for unpopular contents to occupy the cache space of certain edge servers. This attack is called a cache pollution attack (CPA) that aims to pollute the cache space of certain edge servers [15]. CPAs are one of the most severe threats on emerging content networks, including ICNs and CDNs, because the caching mechanisms have made it easier for cyber-attackers to corrupt cache by requesting unpopular or invalid contents [8] [21]. The performance and services of caching networks can be significantly degraded by CPAs. After being attacked by CPAs, the cache of compromised nodes is filled with unpopular contents, resulting in a high cache miss rate and latency for popular contents requested by legitimate users [15]. Due to the latency issues caused by CPAs, CPAs are especially damaging for the transmission of latency-sensitive contents, like live-streaming contents [24]. A CPA scenario for CDNs is shown in Fig. 7.1.

CPAs can be classified as locality-disruption attacks (LDAs) and false-locality attacks (FLAs) based on their behaviors [7]. LDAs are launched by sending a moderate number of requests for a large number of low-popularity contents to occupy the cache space that should belong to popular contents, therefore degrading the locality of cache files and cache efficiency. On the other hand, FLAs are launched by repeatedly sending a large number of requests for a few targeted low-popularity contents to constantly refresh these polluted contents and occupy certain areas of the cache space, thereby degrading the cache hit rate of legitimate requests.

Secondly, since CDNs respond to every incoming request with Internet content in chronological order, cyber-attackers can send a sudden burst of requests to certain nodes to overwhelm these edge servers, named denial of service (DoS) attacks [9]. DoS attacks are the most common attack that can stop the entire network from functioning shortly or indefinitely to prevent legitimate clients from accessing content [9].

Unlike ICNs that are vulnerable to a special type of DoS attack, named Interest flooding attacks (IFAs), due to its Pending Interest Table (PIT) strategy for content caching, common

DoS attacks launched in CDNs are conventional Hypertext Transfer Protocol (HTTP) flooding attacks [25]. These DoS attacks intend to exhaust the network and memory resources of CDN nodes, so that the requests from legitimate users may be delayed or their requested contents cannot be cached by these affected nodes, causing cache misses or late responses [18]. A DoS scenario for CDNs is shown in Fig. 7.1.

As a special case of DoS attacks, distributed DoS (DDoS) attacks are carried out by utilizing multiple compromised devices, named bots, to send a sudden burst of requests together to certain nodes, while DoS attacks can be launched by a single machine. It is more difficult to detect DDoS attacks than DoS attacks since they are carried out from multiple locations instead of a single origin [5] [9].

As CPAs and DoS attacks are two common types of service targeting attacks that pose severe threats to CDNs to make caching mechanisms, the core of CDNs, unavailable to legitimate users, the major purpose of this work is to protect CDNs against these two attacks.

7.2.3 Anomaly Detection in CDN

CDNs themselves can mitigate DoS and DDoS attacks, since they use highly distributed edge servers that can disperse the traffic to avoid the entire system breakdown [18]. However, latency will still be increased due to the unavailability of edge nodes caused by attacks. Additionally, if DoS attacks cannot be detected and compromised nodes cannot recover, more CDN edge servers and clients will be affected by the attacks, and the entire system will fail eventually. Certain conventional mechanisms, like firewalls and filtering techniques, can mitigate DoS attacks by limiting the amount of traffic entering the CDN. However, DoS attacks and crowd events have similar patterns, making it difficult for conventional mechanisms to filter only the attacks [5]. Thus, there is a need to develop an anomaly detection system to distinguish them. On the other hand, compared to DoS attacks, CPAs are more flexible, stealthy, and challenging. CPAs can be launched by sending a moderate number of requests for certain unpopular contents to retain them in the cache through the normal content delivery process, making it difficult for conventional mechanisms to defend against CPAs. This emphasizes the importance of anomaly detection system development.

CDNs usually have firewalls and authentication mechanisms as their first layer of defense. Anomaly detection systems can be incorporated into CDNs as the second layer of defense to identify attacks that have breached the first layer of defense. Identifying those attacks is crucial because they may be more malicious than the attacks that are blocked by the first layer of defense [26]. Additionally, anomaly detection systems can adapt to the changing patterns of cyber-attacks by analyzing the continuously-generated CDN log data that reflect the current network states. Thus, the work aims to propose an anomaly detection system to detect service targeting attacks (*i.e.*, CPAs and DoS attacks) and abnormal network entities from multiple perspectives to secure CDNs.

7.3 Related Work

7.3.1 CPA & DoS Attack Detection

Service targeting attacks, including CPAs and DoS attacks, pose a severe threat to caching networks, such as CDNs and ICNs [7] [16]. As the leading CDN service provider, Akamai Technologies reported in 2019 that more than 800 types of DoS attacks were found in financial services industries [27]. These DoS attacks can exploit vulnerabilities towards the websites and disturb the financial services, causing severe financial losses. Due to the lack of serious scrutiny for cached contents, real-world CDNs are vulnerable to multiple caching-related attacks, like CPAs, which have been recorded in the "Vulnerability Notes Database" [28]. Moreover, Nguyen *et al.* [23] proposed a new caching attack, Cache-Poisoned Denial-of-Service (CPDoS), that targets CDNs and other vulnerable caching systems. CPDoS attacks combine the ideas of CPA and DoS attacks by sending a large number of requests with malicious headers to pollute cache space and paralyze victim websites.

Several research works have focused on CPA detection. Conti *et al.* [20] conducted experiments to prove that CPAs are a realistic threat to caching networks and proposed a lightweight detection technique to detect CPAs accurately. However, the experiments were conducted on a simulated network instead of a real-world network. Xie *et al.* [29] proposed a novel method named CacheShield to avoid storing the low-popularity contents (less than the popularity threshold) in servers' caches. However, this strategy can also reject certain legitimate contents, and is inefficient for gradually enhanced attacks. Karami *et al.* [30] proposed an Adaptive Neuro-Fuzzy Inference System (ANFIS) to mitigate CPAs in ICNs. In ANFIS, every content will be given a grade to measure its goodness, and the system will replace low-grade contents in caches with high-grade contents based on a goodness threshold to mitigate cache pollution. However, it has high overhead costs. On the other hand, the above CPA detection methods are all based on threshold mechanisms, but they lack adaptability to complex and changeable network environments.

Many recent research works have considered DoS attack prevention and detection. Rahman *et al.* [9] proposed a distributed virtual honeypot method to mitigate DoS attacks in CDNs. This method can maintain smooth content delivery in CDN edge servers, but cannot protect the main server. Moubayed *et al.* [31] [32] proposed an ensemble learning classifier to effectively detect several types of domain name system (DNS) attacks, including cache poisoning and DoS attacks. The proposed method achieves high accuracy but does not analyze the detection results with related features to summarize attack patterns for future anomaly detection. Kumar *et al.* [33] [34] proposed a security framework based on the software-defined perimeter (SDP) to protect modern networks from being breached by DoS attacks. Certain DoS attacks can be effectively defended or prevented by the proposed method, but it lacks the capacity to detect the malicious attacks that have already breached the networks.

7.3.2 Abnormal IP & Node Detection

Detecting abnormal network entities, including abnormal client IP addresses and nodes, is important for CDN protection. Several recent works have extracted network features from the client IP point of view for anomaly detection. Lee *et al.* [25] proposed CDN request routing and site allocation algorithms to distinguish between the requests from DoS attackers and legitimate users in CDNs, but they did not consider other attack types. Chiba *et al.* [35] proposed a novel method that can effectively extract features from the structures of IP addresses and applied the support vector machine (SVM) model to detect malicious websites. Pinto *et al.* [36] presented and utilized SVM on the network traffic data to identify malicious IP addresses and achieved the cross-validation accuracy of 83% to 95%. Fiadino *et al.* [37] used the HTTP flow data collected from a primary European Internet service provider to detect traffic anomalies in CDNs. However, only detecting numerically abnormal traffic is insufficient to identify cyber-attacks accurately.

Detecting compromised nodes is also a critical process to ensure a quick recovery and reliable functioning of networks. La *et al.* [38] proposed a misbehavior node detection algorithm using a weighted-link method to secure a hierarchical sensor network. Berjab *et al.* [39] proposed a novel framework based on observed spatiotemporal (ST) and multivariate-attribute (MVA) sensor correlations to detect abnormal nodes in wireless sensor networks (WSNs). Pandey *et al.* [40] proposed an innovative method that uses intrusion detection system (IDS) agents to identify compromised nodes in WSNs according to their behavior, and the proposed method shows efficiency in small networks. However, the above methods lack a root cause analysis to find what caused the abnormal nodes for the purpose of future intrusion prevention.

7.3.3 Research Contributions

Many of the research works presented in this section are promising and have achieved good outcomes. However, most of them use software like the network simulator version 3 (NS-3) to build a simulated network for data collection, which may be biased or noisy. In our work, a recent real-world CDN access log dataset collected from a major operator is used to show the effectiveness of applying our proposed anomaly detection model to real-world networks.

On the other hand, most existing research works only detect anomalies from a single perspective, like the client IP or the service provider perspective. This is often insufficient to validate whether the detected abnormal network behavior is due to malicious cyber-attacks or legitimate network events. Thus, many false alarms may be returned. Caltagirone *et al.* [41] proposed a pivoting intrusion analysis model named “diamond” that uses the communication information between compromised nodes and malicious IP addresses to reveal the detail of attackers, but it is only a theoretical model. On the other hand, our proposed method extracts more CDN attributes/features and conducts anomaly detection from several different perspectives, and then performs a multi-perspective analysis to validate anomaly detection results for

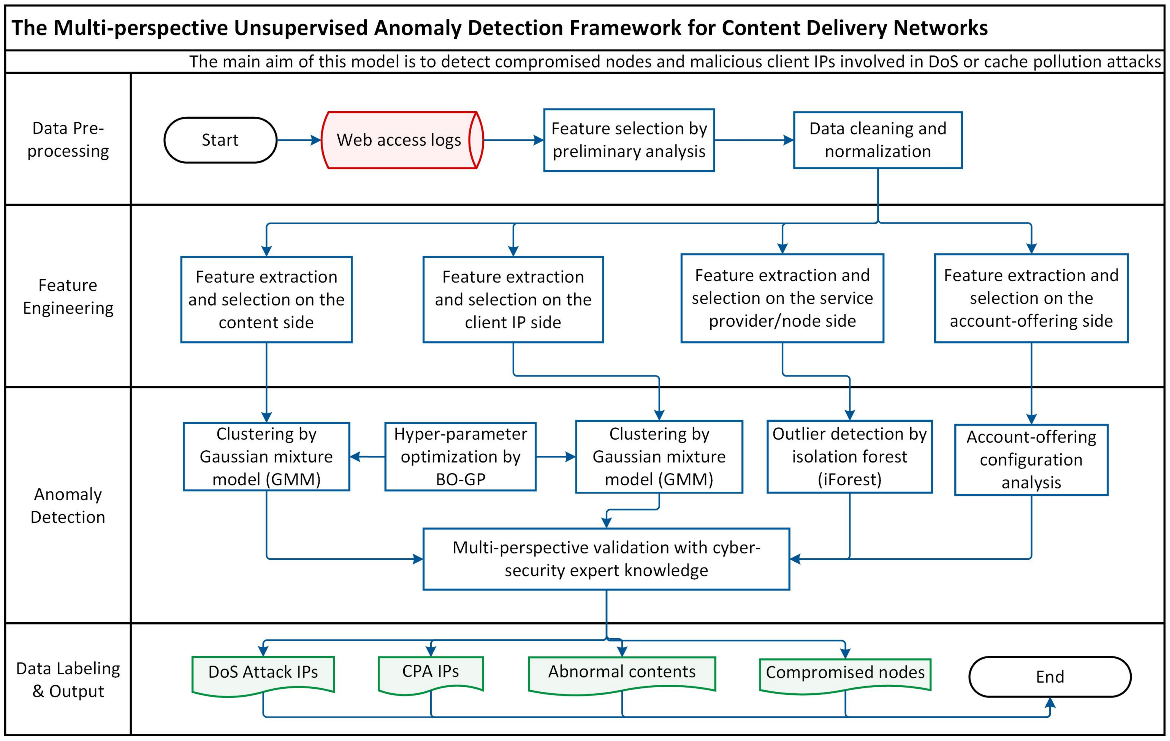
The Multi-perspective Unsupervised Anomaly Detection Framework for Content Delivery Networks
The main aim of this model is to detect compromised nodes and malicious client IPs involved in DoS or cache pollution attacks
Data Pre-processing
Start
Web access logs
Feature selection by preliminary analysis
Data cleaning and normalization
Feature Engineering
Feature extraction and selection on the content side
Feature extraction and selection on the client IP side
Feature extraction and selection on the service provider/node side
Feature extraction and selection on the account-offering side
Anomaly Detection
Clustering by Gaussian mixture model (GMM)
Hyper-parameter optimization by BO-GP
Clustering by Gaussian mixture model (GMM)
Outlier detection by isolation forest (iForest)
Account-offering configuration analysis
Multi-perspective validation with cyber-security expert knowledge
Data Labeling & Output
DoS Attack IPs
CPA IPs
Abnormal contents
Compromised nodes
End

Figure 7.2: The framework of the proposed anomaly detection approach.

the purpose of false alarm reduction.

Moreover, many recent research works treat ML models as black-box methods and do not analyze how the detected anomalies and corresponding features can reflect a specific type of cyber-attack. In our proposed method, the behaviors and characteristics of CPA and DoS attacks from multiple perspectives are summarized and analyzed together with the ML-based anomaly detection results to perform a root cause analysis and find which type of cyber-attack or event causes the anomalies. Ultimately, both the abnormal network entities and their corresponding cyber-attack types will be detected effectively.

7.4 Proposed Anomaly Detection Framework

7.4.1 System Overview and Deployment

The proposed method aims to characterize abnormal events and separate them from normal network events based on the analytics of CDN access log data from different perspectives, including content, client IP, account-offering, and node perspectives. Fig. 7.2 depicts the

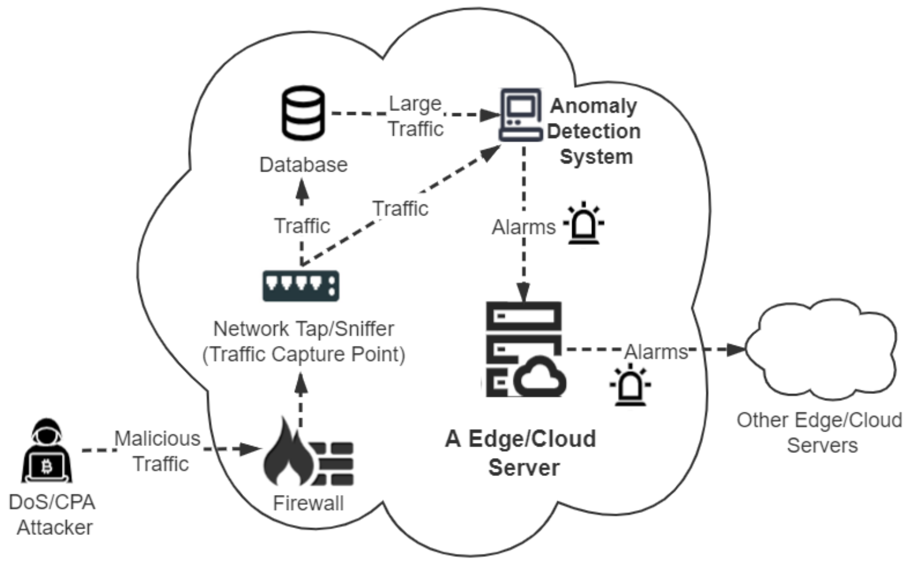

Figure 7.3: The deployment of the proposed anomaly detection system.

framework of the proposed anomaly detection model. The overall architecture of the proposed system is divided into four parts: data pre-processing, feature engineering, anomaly detection, and data labeling. At the first stage, the raw access log data is pre-processed and cleaned to generate a sanitized dataset. A comprehensive feature engineering method is then implemented to extract the datasets that can effectively reflect the behaviors of network attacks from different perspectives. Next, the extracted datasets are trained by an optimized unsupervised anomaly detection model based on GMM, iForest, and BO to discern between abnormal and normal data patterns. At the last stage, a multi-perspective validation analysis is conducted to reduce the errors in the anomaly detection results. Ultimately, abnormal network entities, including malicious IPs, abnormal contents, and compromised nodes, as well as their corresponding cyber-attack types, can be effectively detected to secure CDNs.

For the deployment in CDNs, the proposed anomaly detection system can be placed in both the central server and edge servers, as shown in Fig. 7.3. In edge servers, the proposed system can keep monitoring the network traffic to detect abnormal network entities and send alarms to the central server as soon as an attack occurs; hence, the central server can notice other edge servers and make corresponding countermeasures. When placed in the central server, the proposed system can have a comprehensive view of the operation of the entire network, and can protect the central server when certain edge servers have been exploited by attackers to breach the central server. Specifically, in each edge or cloud server, all the network traffic that is not stopped by the first layer of defense (*e.g.*, firewalls) will be captured by network taps or sniffers, and then analyzed by the proposed anomaly detection system [42]. The large traffic can also be stored in a database for comprehensive analysis by the proposed anomaly detection system. If an attack is detected in an edge or cloud server, all the edge and cloud servers will receive an alarm. As such, the network administrators in the central server and edge servers can make corresponding countermeasures to stop current attacks and prevent future attacks.

7.4.2 Data Acquisition and Preprocessing

For the purpose of network anomaly detection, data acquisition is the first phase of any ML-based model framework. Due to the complexity of modern network configurations, numerous network fields are often recorded in network log data to reflect network states and characteristics. For instance, the authors in [43] provided 248 unique network features that can be collected from the network packets transmitted between clients and servers. However, considering hundreds of network features for anomaly detection is often unrealistic due to limited budgets and resources. Additionally, a large number of features that include many irrelevant features may introduce noises, which have an adverse impact on the performance of ML models.

In this chapter, we focus on anomaly detection in CDN access logs. In network communications, access logs record the information of all requests for the contents that users have requested from web servers. The information includes client IP addresses, timestamps, protocol, user-agent information, uniform resource locator (URL) of content, etc. [44]. Analyzing access logs can help us recognize the states and behaviors of a network in various time periods. Access logs are often recorded in a combined log format that contains multiple network fields for each request, as described in [44]. The following is an example of a record in general access logs:

127.0.0.1 - - [12/Dec/2016:04:54:20 -4000] "GET /support.html HTTP/1.1" 200 11179 - "Mozilla/5.0(compatible;Googlebot/2.1;+http://www.g oogle.com/bot.html)" ...

The access log data used in our chapter is a general network access log acquired from real-world web servers. In this 169 GB access log data provided by the CDN operator, there are more than 0.4 billion requests/samples and 30 fields that characterize the behavior of each request. This CDN log data contains data samples with various characteristics and probability distributions, enabling the detection of different types of cyber-attacks.

Although 30 network fields are available in the CDN access logs, only part of them are useful for anomaly detection, and other fields are irrelevant or for other uses. After preliminary analysis on the CDN dataset and potential abnormal behaviors, 12 fields that might be helpful for anomaly detection are selected, including IP address, timestamp, HTTP method, status code, bytes returned, request delivery time, service type, cache hit indicator, node name, account-offering, content-URL, and content type. These features are also the standard access log features described in [44]. The description of the preliminarily selected network fields is shown in Table 7.1. Other fields that do not have a direct impact on cyber-attack detection, such as protocol, network name, referrer string, are removed from the feature set, because they have either the same values or empty values for almost all the requests. To ensure accurate feature selection, HITL is used for feature validation. Thus, this preliminary feature selection process is validated by multiple cybersecurity experts and industrial partner security network engineers. The experts are from multiple organizations, including third and disinterested parties. All the experts involved in this work can guarantee their professionalism, objectivity, and impartiality in the analysis procedures.

After obtaining the reduced raw data, several steps are completed to clean and pre-process the data.

In the first step, apparent noise or error data, mainly the request samples with many empty fields, are discarded. Then, certain features, like the timestamp, are formatted for easier comparison or calculation. Moreover, string features, like the service category (dynamic or static) and cache hit indicator (hit or miss), are converted to binary or numerical features for easier calculation.

Data cleaning is then followed by data normalization. Data normalization is beneficial when features have different ranges since features with larger ranges are often considered more important than smaller range features in ML model training, which can cause misleading results. Therefore, the datasets are normalized by the min-max normalization method to be in

Table 7.1: Description of Selected Network Log Attributes

Feature	**Description**
IP	Client IP address
Timestamp	Request start time in format: "[dd/mmm/yyyy:hh:mm:ss -zzzz]"
HTTP method	HTTP request method, *e.g.*, GET, POST, etc.
Status code	HTTP status code: 2xx indicates a successful response; 3xx indicates a redirection; 4xx indicates a client error; 5xx indicates a server error
Bytes	Bytes returned over the network without headers after a request
Delivery time	Duration from the beginning until the end of a request and all bytes are delivered, in milliseconds
Service type	The type of service, *e.g.*, static, live streaming, progressive download, etc.
Cache hit indicator	Service cache hit/miss indicator, hit or miss
Node	Node name, representing a service provider
Account offering	The network account offering name (*e.g.*, streaming-1, static-2, etc.), indicating a service accessed from the IP space
Content-URL	The content part of a URL, indicating unique content
Content type	The type of the content in each request, *e.g.*, image, video, audio, text, etc.

the range of 0 to 1 [26]. The normalized value of each feature value, x_n, is denoted by,

$$x_n = \frac{x - min}{max - min}, \tag{7.1}$$

where x is the original feature value, min and max are the minimum and maximum values of each original feature.

Min-max normalization is chosen in the proposed framework due to two main reasons [45]:

1. Min-max scaling can transform all the features to have the same range of [0, 1], making it easier for ML models to process the dataset and for people to compare the results; while many other normalization methods cannot obtain the exact same range of features;
2. Compared to other methods, like standard scaler and power transformer, the min-max scaler is more sensitive to outliers, enabling the ML algorithms to detect anomalies more accurately, as the purpose of this work is to identify anomalies and attacks.

The use of min-max normalization can improve the accuracy of ML models and reduce the difficulty of result analysis. Nevertheless, the performance of using different normalization methods is often very similar [45]. Additionally, as normalization is a small part of ML pipelines, it does not have a significant impact on the final anomaly detection results.

Other procedures, including feature engineering, unsupervised learning model development, and multi-perspective validation, are more important.

7.4.3 Feature Engineering

Although there are 12 features for each request in the initial dataset, it is difficult to identify abnormal network entities and cyber-attacks using only the original request dataset since a single request is often insufficient to reflect network anomalies or attacks. Thus, other features that can effectively reflect abnormal network behaviors should be extracted or generated for the purpose of anomaly detection.

Feature engineering aims to obtain useful features from the log traces collected from CDN deployment based on domain knowledge [5]. In this chapter, a multi-perspective feature engineering method that extracts features from four main perspectives: content, client IP, service provider (node), and account-offering, is proposed to obtain the dedicated datasets for different purposes. The generated datasets extracted from different perspectives can be processed separately and then analyzed together to detect abnormal network events more accurately than any single perspective. In the proposed framework, detecting malicious client IPs and compromised nodes is the main objective; the information collected from the content and account-offering perspectives is the supporting information for more accurate abnormal IP and node detection. To detect CPAs and DoS attacks, their main characteristics reflected from the four considered perspectives are summarized below.

The content perspective is to monitor the properties of the requested and cached contents. From the content perspective, it is beneficial to identify the abnormal contents that might be used by attackers to launch cyber-attacks. The abnormal traffic that characterizes a CPA is a sudden burst or a periodic sending of requests for low popularity contents. Additionally, most CPAs are used to breach a few targeted nodes, so for each abnormal content, the number of requests sent to each targeted node should also be large. On the other hand, if a content gets a large number of requests from many different IPs, there might be a crowd event or a distributed denial of service (DDoS) attack.

The service provider perspective is meant to monitor the edge servers or nodes that receive requests and transmit contents through CDNs. The cache hit rates and data transfer rates of compromised nodes are often degraded due to uncached legitimate contents and network unavailability/congestion caused by CPAs or DoS attacks. The average content popularity of each compromised node is also reduced by CPAs since the node's cache space will be occupied by unpopular files. The affected nodes should be identified and isolated as soon as an intrusion occurs so that other legitimate nodes can avoid communicating with the compromised nodes until recovery.

The client IP perspective is meant to monitor clients that send requests for the contents provided by servers. Considering the client IP perspective enables us to detect potential malicious clients, allowing us to stop or prevent cyber-attacks by blocking the requests from these malicious IP addresses. Detecting abnormal client IPs is crucial because these IP addresses represent the origins of cyber-attackers. Attackers that are launching different types of attacks exhibit different behaviors. For CPA attackers, they send high-frequency requests for low popularity contents, while DoS attackers may send a sudden burst of requests for files with any popularity. Additionally, for the two types of CPA attackers, LDA attackers send requests for a large number of unpopular contents, while FLA attackers only send requests for a few targeted contents, but the number of requests sent for each content (request per content ratio) is often large.

The account-offering (AO) perspective indicates the streaming source that provides a specific type of service, such as static, live streaming, and progressive download content distribution. An AO's configuration also determines the behavior of its serviced client IPs. If certain IPs behave abnormally and do not match the AO configuration patterns, they have a high probability of being anomalies. Thus, the analysis from the AO perspective can help us validate whether the anomaly detection results are real attacks or false alarms.

The extracted features from four perspectives are summarized in Tables 7.2 - 7.5. After attack pattern analysis in Section 7.2.2, we found that only some of the features have a direct impact on the detection of CPAs and DoS attacks. Considering more features may cause misleading results or additional computational time. Thus, the most important features that can directly reflect the attack patterns are selected in the proposed method, based on the potential feature patterns/characteristics of CPAs and DoS attacks summarized in Tables 7.6 and 7.7.

Table 7.2: Description of Extracted Features from the Content Perspective

Perspective	**Feature**	**Description**
Content	Number of requests	The total number of requests for per content
	Popularity	The popularity of per content, represented by the normalized number of IPs that sent requests to per content
	Cache hit rate	The number of cache hits divided by the total number of requests for per content
	Request per IP ratio	The ratio of the total number of requests sent for per content to the total number of IPs which sent requests for per content
	Request per node ratio	The ratio of the total number of requests sent for per content to the total number of nodes which received requests for per content

Table 7.3: Description of Extracted Features from the Node Perspective

Perspective	**Feature**	**Description**
Node	Cache hit rate	The number of cache hits divided by the total number of requests received by per node
	Cache hit rate of legitimate IPs	Average cache hit rate of IPs which only requested for popular contents on per node
	Data transfer rate (MB/s)	The total bytes returned divided by the total delivery time for per node
	Request error rate	The percentage of requests with errors (4xx/5xx status code) received by per node
	Average request popularity	Average content popularity of requests received by per node
	Account-offering request rate	The percentage of requests sent through per account-offering for per node, *e.g.*, “account1: 80%, account2: 20%”

The selected features can reflect most scenarios of a CDN that is under CPA or DoS attacks.

Considering the original features in Table 7.1, except for the HTTP method, all other raw features in Table 7.1 have been used to create the final features in Tables 7.2 - 7.7. Specifically, “IP”, “Node”, and “Content-URL” are used to extract datasets from the IP, node, and content perspectives, “Timestamp” is used to calculate the average request interval of client IPs, “Status code” is used to calculate the request error rate of nodes, “Bytes” and “Delivery time” are used to calculate the data transfer rate, “Cache hit indicator” is used to calculate the cache hit rate of IPs and nodes, “Account-offering”, “Service type”, and “Content type” are used to determine the behaviors of account offerings for result validation purposes. Therefore, the fields selected by the cybersecurity experts do not include any noise. “HTTP method” is removed because DoS and CPAs can be launched using any HTTP method, whether they use GET, POST, or any other HTTP methods; hence, it is irrelevant for DoS & CPA detection. Although the “HTTP

Table 7.4: Description of Extracted Features from the Client IP Perspective

Perspective	Feature	Description
Client IP	Number of requests	The total number of requests sent by per IP
	Average request interval	The average time interval between consecutive requests sent by per IP
	Number of nodes	The total number of unique nodes that received requests from per IP
	Number of contents	The total number of unique contents requested by per IP
	Request per content ratio	The ratio of the total number of requests sent by per IP to the total number of contents requested by per IP
	Request per node ratio	The ratio of the total number of requests sent by per IP to the total number of nodes which received requests from per IP
	Average request popularity	Average content popularity of requests sent by per IP
	Cache hit rate	The number of cache hits divided by the total number of requests sent by per IP
	Request error rate	The percentage of requests with errors (4xx/5xx status code) sent by per IP
	Account-offering request rate	The percentage of requests are sent through per account-offering for per IP, *e.g.*, “account1: 80%, account2: 20%”

Table 7.5: Description of Extracted Features from the Account-Offering Perspective

Perspective	Feature	Description
Account-offering (AO)	Number of requests	The total number of requests through per AO
	Number of nodes	The total number of unique nodes that received requests sent through per AO
	Service type	The type of service provided by per AO, *e.g.*, static, live streaming, progressive download, etc.
	Content type	The type of content provided by per AO, *e.g.*, image, video, audio, text, etc.
	Cache hit rate	The number of cache hits divided by the total number of requests sent through per AO
	Request popularity	Average content popularity of requests sent through per AO

method” is irrelevant for CPA & DoS attack detection, it can still be used for other tasks, like the detection of other types of attacks.

This feature extraction and selection process can be automated as a general feature engineering method by summarizing the potential patterns of certain types of cyber-attacks and selecting the features that can reflect the attack patterns, as discussed in this subsection. Through this process, the proposed multi-perspective feature engineering method can extract and select

Table 7.6: Potential Patterns of CPAs

Attack Type	**Perspective**	**Feature**	**Abnormal Patterns**
CPA	Node	Cache hit rate	Low
		Cache hit rate of legitimate IPs	Low
		Data transfer rate (MB/s)	Low
		Average request popularity	Low
	Client IP	Number of requests	Large
		Average request interval	Short
		Number of nodes	Small
		Request per content ratio	LDA: Low FLA: High
		Average request popularity	Low
	Content	Popularity	Low
		Request per IP ratio	FLA: High LDA: Low
		Request per node ratio	High
	AO	Request popularity	Low

Table 7.7: Potential Patterns of DoS Attacks

Attack Type	**Perspective**	**Feature**	**Abnormal Patterns**
DoS	Node	Cache hit rate	Low
		Cache hit rate of legitimate IPs	Low
		Data transfer rate (MB/s)	Low
		Request error rate	High
	Client IP	Number of requests	Large
		Average request interval	Short
		Number of nodes	Small
		Cache hit rate	Low
		Request error rate	High
	AO	Cache hit rate	Low

the core features to reflect CPA and DoS attacks for the anomaly detection model development presented in the next subsection.

7.4.4 Unsupervised Anomaly Detection

Compromised Node Detection

The extracted node-based dataset contains the information about 50 different nodes. To detect abnormal or compromised nodes, isolation forest (iForest) [11], an unsupervised outlier detection algorithm that aims to separate isolated data samples (anomalies) from normal samples, is utilized in the proposed framework. The proposed abnormal node detection method has two main steps:

1. Use iForest, an outlier detection algorithm, to separate numerically abnormal samples from normal samples.
2. Analyze the behaviors of each numerically abnormal node, and label the nodes that match the summarized patterns/characteristics of different types of attacks.

IForest is an ensemble learning algorithm constructed with multiple binary search trees, named isolation trees (iTrees) [46]. Each iTree is constructed by splitting the samples based on feature values. The number of splittings required to isolate a sample indicates its path length (*i.e.*, the number of edges from an iTree's root node to its leaf node). The path length of anomalies is often shorter than normal samples. This is because normal samples are often the majority and in dense areas, making it unlikely for an iTree to isolate them from each other, while anomalies are the opposite. A score is given to each sample based on the path length, so that the outliers that are sparsely distributed and distant from the dense normal samples can be detected [46].

IForest is chosen for abnormal node detection due to the following reasons [11] [46]:

1. Unlike many other ML algorithms, iForest performs well on small-scale data to which the node-based dataset belong. This is because iForest uses the short path length of data samples to indicate the anomalies, which can be obtained regardless of data size.
2. Unlike clustering algorithms, iForest does not require anomalies to have similar characteristics to detect them since it discerns outliers from normal samples based on data density.
3. IForest is computationally efficient because it has a linear time complexity of $O(N)$, a low memory requirement, and parallel execution support.
4. IForest has good interpretability since it uses a tree-structure to make decisions and split data samples.

To develop an effective ML model for a specific task, hyper-parameter tuning should be implemented to detect the hyper-parameter configuration that can return the optimal architecture of the ML model [47]- [49]. As an important hyper-parameter of iForest, the contamination level determines the proportion of data samples that will be detected as outliers. To build an optimized iForest model, the contamination level is tuned using Bayesian optimization (BO).

BO algorithms are a set of efficient hyper-parameter optimization (HPO) methods that detect the optimal hyper-parameter based on the currently-evaluated results [50]. In BO, a surrogate model is used to fit all the currently-tested samples into the objective function; an acquisition function is then used to locate the next points by considering both the unexplored regions and currently-promising regions in the search space [51].

Gaussian process (GP) is a common surrogate model for BO. In GP surrogate models, any finite combination of the random variables follows a Gaussian distribution [52]:

$$p(y|x, D) = N\left(y|\hat{\mu}, \hat{\sigma}^2\right), \tag{7.2}$$

where D is the hyper-parameter configuration space, $y = f(x)$ is the objective function value for each hyper-parameter configuration with its mean as $\hat{\mu}$ and covariance as $\hat{\sigma}^2$.

The BO method using GP (BO-GP) has a time complexity of $O(N^3)$ and a space complexity of $O(N^2)$ [52]. BO-GP is inefficient for a large hyper-parameter search space but exhibits great performance on optimizing a small number of continuous or discrete hyper-parameters. Thus, BO-GP is used to optimize the contamination level of the iForest model. The silhouette coefficient [53] [54], a distance-based metric that can measure the similarity of normal samples and the difference between numerically normal and anomalous samples, is chosen to be the metric of the iForest model and used as the objective function to be optimized by BO-GP.

After using the optimized iForest model to detect abnormal nodes, many false positives will be returned since many of the detected compromised nodes do not match the cyber-attack patterns. Mainly, the cache hit rate of the compromised nodes and their serviced legitimate client IPs should be reduced due to attacks. The data transfer rate and average popularity of the abnormal nodes should also be low due to network congestions and the filled cache space occupied by unpopular files, respectively. On the other hand, although certain other isolated data points, like the nodes that received a very small number of requests or returned a very high data transfer rate, are numerically different from most normal nodes, they are unlikely to be under CPA or DoS attacks. Therefore, the patterns of the detected anomalous nodes are compared with the behaviors of potentially compromised nodes summarized in Tables 7.6 & 7.7. Ultimately, the affected nodes that match the abnormal patterns are preliminarily labeled abnormal.

Abnormal Content & Client IP Detection

Unlike the node-based dataset that only contains 50 unique nodes, both the content-based datasets and IP-based datasets have more than one million unique contents or unique IPs. For large-scale data, only using a binary outlier detection method (*e.g.*, iForest) is insufficient to identify attack samples, because using only two categories cannot describe various normal and abnormal patterns, and numerical anomalies may not be the true attack samples since cyber-attacks have their own specific patterns or characteristics, as described in Section 7.4.3.

For large-sized data, clustering algorithms would be more effective in identifying abnormal contents and IP addresses since they can obtain multiple numerically abnormal clusters, enabling us to determine the true anomalies by comparing the characteristics of the abnormal clusters with the specific attack patterns. Clustering algorithms are a set of unsupervised learning models that aim to group data points into different clusters. Data samples in the same cluster should have similar patterns or properties, while those in different clusters should have different patterns [55].

The proposed abnormal IP and content detection method has two main steps:

1. Use a clustering algorithm to group the client IPs or contents into a sufficient number of clusters.
2. Analyze the characteristics of each cluster, and label the IPs or contents in this cluster as "normal" or "abnormal" based on the summarized patterns of different types of attacks.

Gaussian mixture model (GMM) is a distribution-based clustering constructed with multiple Gaussian distribution components and a probability density function [12]. GMM models data points by utilizing Gaussian distribution models with parameters estimated by the expectation-maximization (EM) algorithms. In GMM, each Gaussian component can be denoted by a multivariate Gaussian distribution [12]:

$$G(\mathbf{x} \mid \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \tag{7.3}$$

where $\mathbf{x}$ is the data points, μ is the mean or the expectation of the Gaussian distribution, Σ is the covariance, and D is the dimensionality of the dataset.

A GMM with K Gaussian components models the data by the following probability density function [12]:

$$p(\mathbf{x} \mid \theta) = \sum_{i=1}^{K} \pi_i G(\mathbf{x} \mid \mu_i, \Sigma_i) \tag{7.4}$$

where $\theta = \{\pi_i, \mu_i, \Sigma_i\}$ are the parameters of GMM, π_i is the weight of each Gaussian component, and $\sum_{i=1}^{K} \pi_i = 1$.

The GMM parameters are obtained by the EM algorithm that repeats two main steps until convergence: the E-step calculates the expectation of each Gaussian component, and the M-step maximizes the calculated expectations to update the parameters of Gaussian distributions [12]. The time complexity of training a GMM is $O(NKD^2)$ for N data instances, K Gaussian components, and D features or dimensions [56].

The main reasons for choosing GMM for the content-based and IP-based datasets are:

1. Based on the visualization of the probability density functions, most of the extracted features on the content and client IP sides follow Gaussian or near-Gaussian distributions. Additionally, GMM considers feature covariance and can model more flexible cluster shapes than many other clustering methods, like k-means, which can only return globular cluster shapes. Therefore, GMM can fit the datasets effectively.
2. Unlike many other clustering algorithms, like k-means and hierarchical clustering, which can only return a cluster label or identity number, GMM is able to give a confidence value to each test sample, indicating the probability of belonging to each cluster. The probability can be used to find uncertain samples and take further actions to reduce errors.
3. Although k-means has a training time complexity of $O(NKD)$ [57] that is lower than GMM, the training time of GMM is still low because the proposed feature engineering method has effectively reduced the dimensionality of the data. On the other hand, the

run-time complexity of a GMM is also $O(NKD)$, so the execution time of running an trained GMM is low.

For the GMM applied to the CDN datasets, it has a major hyper-parameter that requires tuning, which is K, the number of clusters or Gaussian components [52]. Identifying an optimal value of K is crucial since it determines whether a sufficient number of Gaussian components are constructed to describe and distinguish normal and abnormal data patterns. On the other hand, a too-large K will lead to additional model training time.

Since GMMs only have a discrete hyper-parameter, the number of Gaussian components, that requires tuning in most cases, BO-GP serves as an effective HPO method for GMMs [52]. The silhouette coefficient is also selected as the metric of the GMM model since it measures how similar a data sample is to other data samples within the same cluster and how different a data sample is from the samples in other clusters.

After grouping the contents and IPs into optimized numbers of clusters using GMM and BO-GP, the characteristics of each cluster will be analyzed based on the DoS & CPA patterns, and the clusters that match the abnormal content and IP patterns summarized in Tables 7.6 & 7.7 are deemed to have passed the initial detection. The IPs and contents in these clusters are preliminarily labeled "abnormal" at this stage.

7.4.5 Multi-perspective Result Validation

Using unsupervised machine learning algorithms, including GMM and iForest, enables us to distinguish numerically abnormal content, nodes, and IPs from normal ones. However, certain legitimate events, like misconfigurations and crowd events, may perform similar behaviors as cyber-attacks and be misclassified as anomalies. Therefore, a multi-perspective result validation analysis is performed to eliminate the false alarms and improve the detection rate, so as to identify the real network entities affected by CPAs and DoS attacks.

Time-series Analysis

At the first stage of result validation, time series analysis is conducted by analyzing the changes of certain features in periods (*e.g.*, hourly and daily changes) to find the abnormal events and potential attacks.

The major feature changes to be monitored in general CDN access log data for DoS & CPA detection include follows:

1. The changes in the hourly number of requests can help us to find potential crowd events and DoS attacks when there is a sudden burst of requests in certain time periods;
2. The changes in the hourly cache hit rate can help us to detect potential DoS attacks when there is a sudden burst of error requests that aim to exhaust network resources;
3. The changes in the hourly request popularity can be used to identify potential CPAs when certain IPs start to send a large number of requests for unpopular contents, or certain nodes get many requests for unpopular contents.

The general process of time-series validation is shown in Fig. 7.4. After we find the time periods in which certain features change abnormally, they will be compared with the information about the known legitimate events provided by the CDN operator. If known legitimate events did not occur in these abnormal periods, there is a high probability that cyber-attack occurred. Thus, the active IPs, nodes, and contents in these abnormal periods will be analyzed using the proposed optimized iForest or GMM methods to detect the abnormal network entities affected in potential cyber-attacks. In most real-world applications, legitimate event information should be recorded for network maintenance purposes. In case of no information about legitimate events, expert intervention can be involved in the validation process as a HITL procedure to help determine the real attacks. Moreover, network administrators can still use authentication mechanisms to confirm the identities of all these suspicious entities to identify real attacks.

The cross-perspective analysis is also utilized in the time-series validation process to help validate the real abnormal network entities. To perform the cross-perspective analysis, the results from each perspective are used to validate the results from the other two perspectives. To be specific, the information about the detected compromised nodes can help us to validate the potential malicious IPs which try to attack these nodes and the potential abnormal contents which are used to pollute these nodes. For the abnormal IPs and contents detected from their own perspectives, these results can be used to validate which nodes are targeted by the attackers

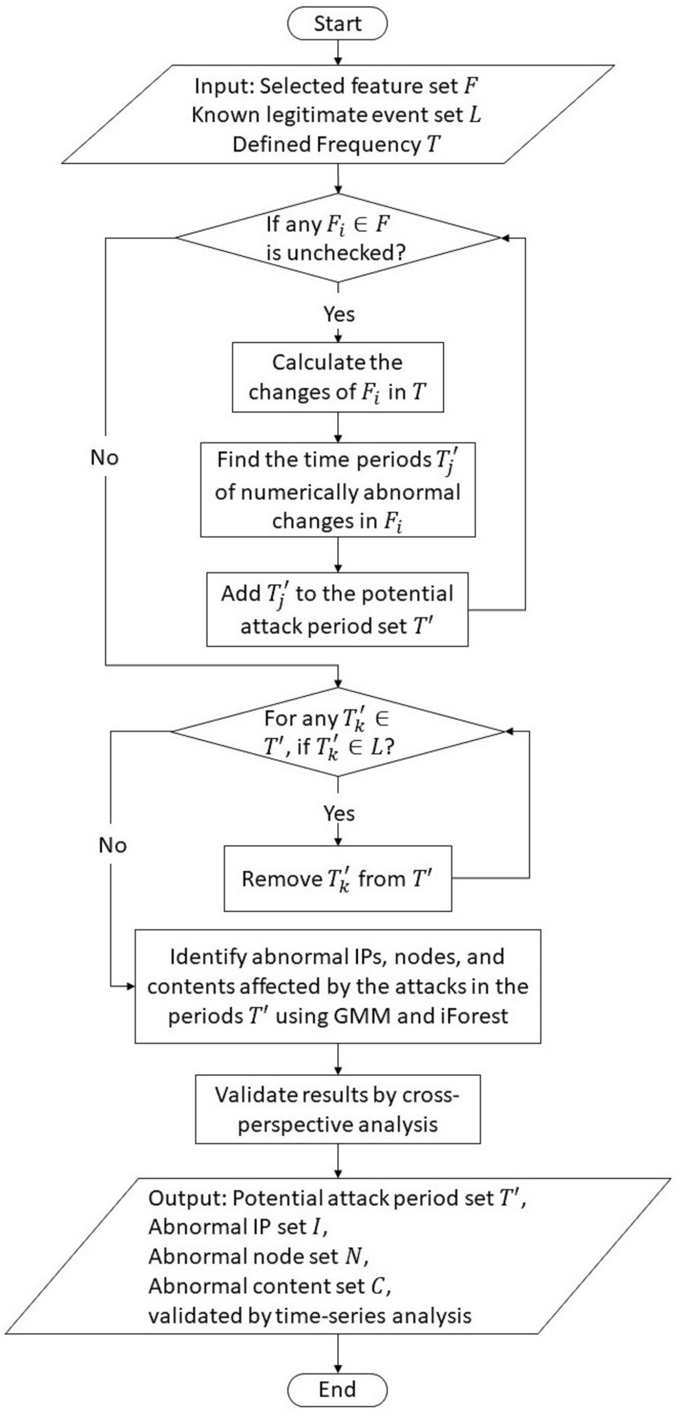
Start
Input: Selected feature set F
Known legitimate event set L
Defined Frequency T
If any $F_i \in F$ is unchecked?
Yes
No
Calculate the changes of F_i in T
Find the time periods T_j' of numerically abnormal changes in F_i
Add T_j' to the potential attack period set T'
For any $T_k' \in T'$, if $T_k' \in L$?
Yes
No
Remove T_k' from T'
Identify abnormal IPs, nodes, and contents affected by the attacks in the periods T' using GMM and iForest
Validate results by cross-perspective analysis
Output: Potential attack period set T',
Abnormal IP set I,
Abnormal node set N,
Abnormal content set C,
validated by time-series analysis
End

Figure 7.4: The flow chart of time-series analysis

who utilized these abnormal IPs and contents. Through this process, the true abnormal entities that are affected by attacks can be identified effectively.

In conclusion, time series analysis enables us to locate the specific days or time periods

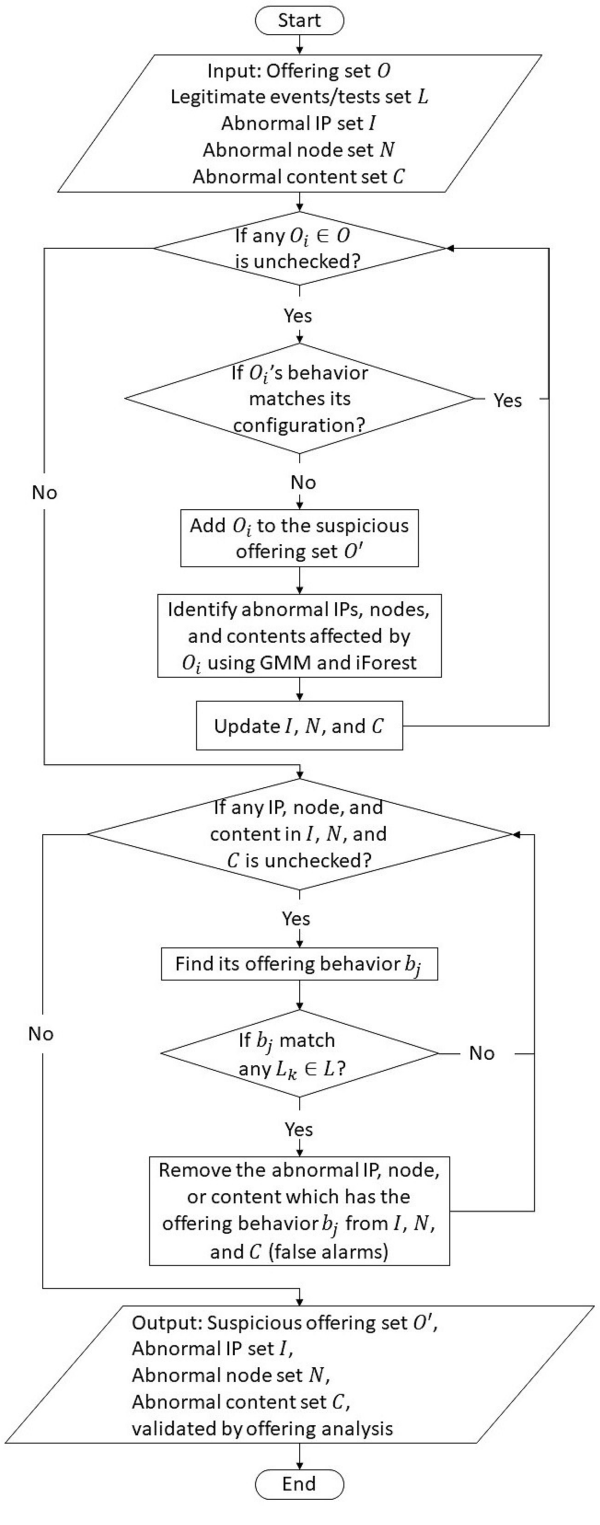
Start
Input: Offering set O
Legitimate events/tests set L
Abnormal IP set I
Abnormal node set N
Abnormal content set C
If any $O_i \in O$ is unchecked?
Yes
No
If O_i's behavior matches its configuration?
Yes
No
Add O_i to the suspicious offering set O'
Identify abnormal IPs, nodes, and contents affected by O_i using GMM and iForest
Update I, N, and C
If any IP, node, and content in I, N, and C is unchecked?
Yes
No
Find its offering behavior b_j
If b_j match any $L_k \in L$?
No
Yes
Remove the abnormal IP, node, or content which has the offering behavior b_j from I, N, and C (false alarms)
Output: Suspicious offering set O',
Abnormal IP set I,
Abnormal node set N,
Abnormal content set C,
validated by offering analysis
End

Figure 7.5: The flow chart of account offering analysis

of potential attacks and legitimate events (*e.g.*, crowd events), thus validating the results by analyzing the nodes, IPs, and contents affected during these periods.

Account-Offering Analysis

The account-offering (AO) analysis is to analyze anomaly detection results based on the AO configurations and behaviors to distinguish true attacks from legitimate outliers. The potential behavior of an AO can often be estimated based on its configuration. If cyber-attacks occur, the characteristics of the affected AOs can change to abnormal, which can help us to validate the abnormal network entities serviced by these AOs. For example, if an AO that is configured to be the major stream of static content services gets numerous requests for non-existent live streaming contents, DoS attacks might be launched through this AO to overwhelm certain nodes. On the other hand, if an AO is used for the legitimate tests of old progressive download videos, it may be misclassified as anomalies by the proposed unsupervised models because, similar to CPAs, there will also be a large number of requests for low-popularity contents; thus, the affected nodes, IPs, and contents can be false alarms. AO analysis requires the configuration information of AOs to determine whether the AOs' behaviors match their configurations. The general process of account-offering analysis is shown in Fig. 7.5. Through this process, the false alarms from legitimate events will be removed to improve the anomaly detection accuracy.

After implementing the optimized unsupervised anomaly detection method and performing the multi-perspective result analysis, the abnormal contents, compromised nodes, and malicious IPs with their potential attack types can be identified effectively.

7.5 Experimental Results & Discussion

7.5.1 Experiments Setup

The experiments are conducted on a machine with a 6 Core i7-8700 processor and 16 GB of memory. The dataset used for experiments is 169 GB of 8 days web access logs collected by a major CDN operator from December 12th to 19th, 2016, including 452,264,816 unique requests/samples. Through the proposed multi-perspective feature engineering, we have obtained the IP-based dataset (1,268,027 unique IPs) and the node-based dataset (50 unique nodes) for abnormal IP and node detection. Additionally, the content-based dataset (1,867,584 unique contents) and the account-offering dataset (70 unique AOs) are also generated to support anomaly detection result validation. On the other hand, the dates of crowd events are provided by the CDN operator, including 12th, 14th, and 15th, December 2016 (days 1, 3, and 4). Additionally, the CDN operator has used many conventional security mechanisms as the first layer of defense, including firewalls with filtering mechanisms, authentication, hashing, and load balancers, to mitigate cyber-attacks. Our proposed anomaly detection system serves as the second layer of defense to detect anomalies that are not stopped by the first layer of defense.

At the first stage of the experiments, the anomaly detection model based on iForest and GMM is implemented on the obtained content-based, client IP-based, and node-based datasets individually to preliminarily detect abnormal contents, malicious IPs, and compromised nodes. Once the numerically abnormal network entities are separated from normal entities by the proposed optimized unsupervised learning model, a multi-perspective analysis is then performed to validate the results as the second stage of the experiments. The validation process includes the time-series analysis based on the known legitimate events periods, and the account-offering analysis based on the account-offering configuration information and their practical behaviors. As such, false alarms can be reduced, and real abnormal network entities can be identified effectively.

7.5.2 Unsupervised Anomaly Detection Results

Compromised Node Detection Results

Based on the extracted node-based datasets, there are 50 unique nodes that have received requests in the 8 days dataset. Through the optimized iForest method that returns the contamination level of 0.22, as well as the comparison between node behaviors and cyber-attack patterns, 11 nodes that have behaved abnormally at least on one day are preliminarily identified as compromised nodes, including the nodes Number (No.) 0, 3, 4, 5, 7, 8, 9, 25, 36, 39, and 47. Among the detected abnormal nodes, nodes No. 0, 5, 7, and 25 had a low cache hit rate (less

Table 7.8: Mean Values of Each Feature of Normal and Abnormal Nodes in Preliminary Anomaly Detection Using IForest

Node Labels	**Cache hit rate**	**Legitimate IP cache hit rate**	**Data transfer rate (MB/s)**	**Request error rate**	**Request popularity**
Normal	0.886	0.928	0.696	0.003	0.925
Abnormal	0.286	0.299	0.374	0.052	0.961

than 0.5), while other abnormal nodes had a relatively high request error rate.

The mean value of each feature for normal and abnormal nodes is shown in Table 7.8. It is shown that the compromised nodes have lower cache hit rates and data transfer rates while having higher request error rates than the normal nodes, which is due to potential network congestion and unavailability caused by the attacks. Additionally, the cache hit rates of the legitimate IPs serviced by the abnormal nodes are much lower than the IPs serviced by the normal nodes. A multi-perspective validation will be conducted at the next stage to evaluate the detected abnormal nodes and reduce errors.

Abnormal Content Detection Results

1,867,584 unique contents have been requested in the 8 days dataset. A GMM optimized by BO-GP is trained on the content-based dataset to detect potential abnormal contents that might be used by attackers to launch attacks. As the major hyper-parameter of GMM, the optimal number of Gaussian components is found to be 28, which returns the highest silhouette score of 0.96. As shown in Table 7.9, the 169 low-popularity contents in cluster No. 28 have got a large number of requests on a couple of target nodes. Therefore, these unpopular contents might have been requested many times by CPA attackers to occupy the cache space, making the legitimate and popular contents get cache misses. Therefore, the 169 contents in cluster No. 28 are preliminarily identified as potential abnormal contents. The clusters No. 1-27 are classified as normal clusters based on the comparison with cyber-attack patterns, and the behaviors of the clusters No. 3-27 are omitted in Table 7.9.

Malicious Client IP Detection Results

There are 1,268,027 unique IP addresses in the extracted IP-based datasets. For abnormal IP detection, two GMMs are trained on two different feature sets separately to detect CPAs and

Table 7.9: Average Feature Values of Content Clusters in Preliminary Anomaly Detection Using GMM

Content cluster No.	Avg number of requests	Avg request per node ratio	Avg request per IP ratio	Avg cache hit rate	Avg popularity
1	1.0	1.0	1.0	0.134	0.0
2	225.7	8.2	1.2	0.857	1.0
…	…	…	…	…	…
28	1021.4	601.2	580.8	0.835	0.254

Table 7.10: Average Feature Values of IP Clusters in Preliminary DoS Attack Detection Using GMM

IP cluster No.	Avg number of requests	Avg number of nodes	Avg request interval (s)	Avg cache hit rate	Avg request error rate	Avg request popularity
1	4.2	1.422	1.44	0.925	0.0	0.976
2	116.4	4.12	365.8	0.908	0.011	0.957
…	…	…	…	…	…	…
47	405607.1	15.6	0.673	0.868	0.124	0.998

DoS attacks, respectively.

For DoS attack detection, the major considered features are the number of requests, requests per node ratio, and average request interval. By using BO-GP to optimize the GMM, the optimal number of clusters is found to be 47 with the highest silhouette score (0.55). As shown in Table 7.10, the 310 IPs in cluster No. 47 are detected as potential DoS attack IPs, since they have sent a large number of requests to several target nodes at a very high frequency. Their request popularity is very high (99.8%), so they are unlikely to be CPA IPs.

On the other hand, for CPA detection, the major considered features are the number of requests, request per content ratio, and request popularity. 34 clusters are found to be able to effectively distinguish potential CPA IPs, returning an optimized GMM with the silhouette score of 0.47. The behavior of each cluster is compared with the FLA and LDA patterns summarized in Section 7.4.3. As shown in Table 7.11, the 21 IPs in cluster No. 33 are detected as the potential FLA IPs since they have sent a large number of requests to a couple of target nodes for several unpopular contents. Similarly, the 33 IPs in cluster No. 34 are detected as the potential LDA IPs since they have sent a large number of total requests to many unpopular contents to occupy the cache space. Thus, 54 unique IPs are preliminarily identified as potential CPA IPs.

Table 7.11: Average Feature Values of IP Clusters in Preliminary CPA Detection Using GMM

IP cluster No.	Avg number of requests	Avg number of nodes	Avg request per content ratio	Avg cache hit rate	Avg request error rate	Avg request popularity
1	1.3	1.14	1.0	0.057	0.019	0.500
2	200.4	28.4	1.0	0.699	0.007	0.627
...	...	...	...	...	...	...
33	4426.6	1.90	769.8	0.856	0.040	0.186
34	16028.4	5.18	1.1	0.077	0.019	0.359

7.5.3 Multi-perspective Results Validation

Although ML algorithms can identify the numerically abnormal IPs and nodes, many false alarms may occur, and some real abnormal IPs and nodes may also be ignored. Thus, multi-perspective validation is done based on the in-depth analysis of the datasets, including time series analysis and account-offering analysis.

Time Series Analysis

Since only the date information about the crowd events is provided by the CDN operator, the hourly number of requests is calculated to find the specific crowd event time periods, as shown in Fig. 7.6. According to the hourly number of requests changes, the legitimate event time periods are found to be 00:00-05:00 on day 1, 00:00-02:00 on day 3, and 00:00-03:00 on day 4. By extracting and analyzing the IP-based dataset in these time periods, 250 IPs that have only sent requests in the crowd events have been detected. The daily behavior of these 250 IPs is shown in Table 7.12. Although they have sent a large number of requests at a very high frequency, since they have only sent requests in the crowd events, and other legitimate IPs in the crowd events got more than 99% cache hit rates, these 250 IPs should belong to legitimate IPs. After checking the 310 IPs that are preliminarily detected as DoS attack IPs by the GMM and described in the last subsection, these 250 legitimate event IPs are removed from the abnormal IP list and labeled normal. Thus, 250 false alarms are removed, and the number of potential DoS attack IPs is reduced to 60 after the first stage of time-series analysis.

At the next stage, in order to find the potential cyber-attack time periods, the hourly cache hit rate changes are calculated since the reduced cache hit rate is the most representative behavior of a CDN node that is under a CPA or DoS attack. As shown in Fig. 7.7, the hourly cache hit

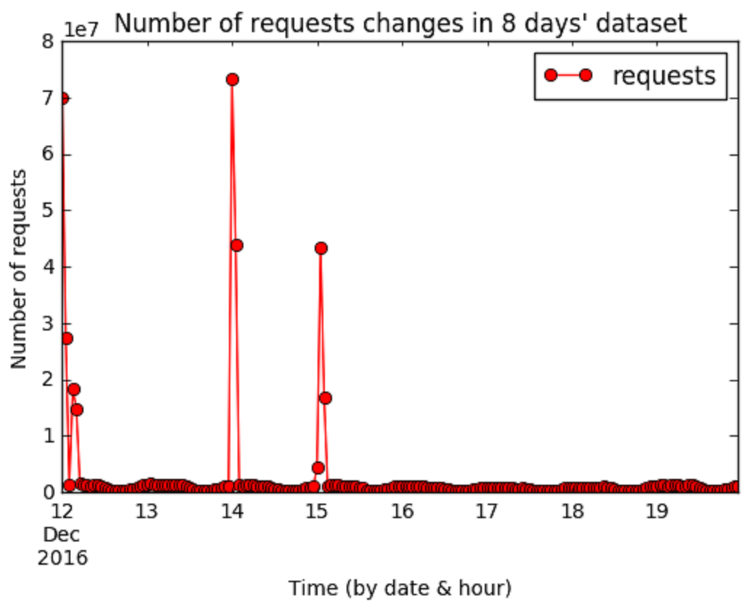

Figure 7.6: The hourly number of requests change in 8 days dataset.

Table 7.12: Main Feature Values of 250 Crowd Event IPs

Day	Avg number of requests	Avg number of nodes	Avg request interval (s)	Avg cache hit rate	Avg request popularity
1	501600.6	21.5	0.021	0.997	1.0
3	459155.2	10.5	0.061	0.996	1.0
4	491830.9	26.0	0.014	0.997	1.0

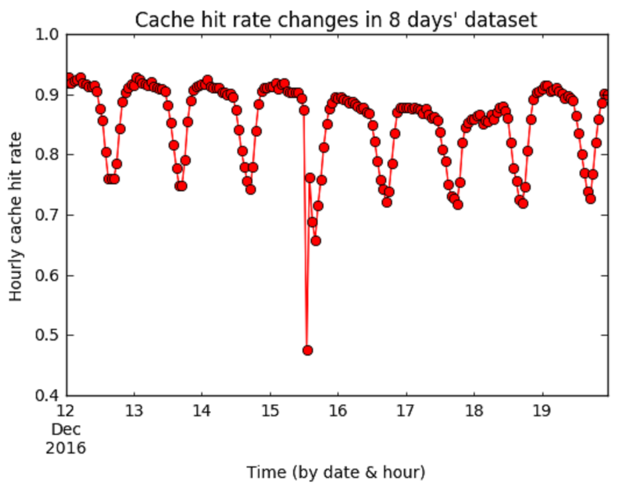

Figure 7.7: The hourly cache hit rate change in 8 days dataset.

rate remains stationary during the eight days except for 13:00-15:00, day 4. An abnormal event might occur in this time period, so this time period is labeled as an abnormal event period.

Table 7.13: Main Feature Values For Compromised Nodes Under Potential DDoS Attacks

Node No.	Cache hit rate on day 4	Avg cache hit rate on other days	Request error rate on day 4	Avg request error rate on other days	Number of requests on day 4
3	0.65	0.70	0.11	0.0009	13,922
4	0.64	0.74	0.02	0.0006	50,947
8	0.62	0.78	0.10	0.0008	16,054
9	0.73	0.79	0.03	0.0009	15,057
25	0.21	-	0.61	-	531,044
28	0.76	0.89	0.08	0.0014	52,690
33	0.78	0.86	0.04	0.0013	59,415
36	0.54	0.75	0.13	0.0014	10,873
39	0.70	0.75	0.04	0.0016	11,893
42	0.76	0.87	0.09	0.0023	46,475
47	0.62	0.85	0.19	0.0023	93,482

To determine the root cause of the significant cache hit rate drop, we have analyzed the requests, nodes, and client IPs in this abnormal period. Firstly, most of the requests in this abnormal period have got the status code 404, indicating resource-not-found errors. Thus, attackers have sent many requests for non-existing contents to cause many cache misses. A large number of 404-error requests will exhaust the network resources and cause network unavailability or congestion, so a DoS attack might occur during this abnormal period.

In the next step, we have explored the compromised nodes in this abnormal period. It is found that most of the 404-error requests have been sent to node No. 25, so it is the major target of the attack. Additionally, 10 other nodes are found to be the secondary targets since they also got many 404-error requests during the abnormal period, but not as many as node No. 25. As shown in Table 7.13, node No. 25 had a very low cache hit rate (0.21) and a very high request error rate (0.61) on day 4. The other 10 compromised nodes' cache hit rates are also slightly reduced on day 4, and their request error rates on day 4 are much higher than their error rates on other days. By comparing the compromised nodes in this abnormal event period with the abnormal nodes detected by iForest in Section 7.5.2, three more abnormal nodes missed by the iForest but detected by the time-series validation process, nodes No. 28, 33, and 42, are added to the detected abnormal node list. Although these three nodes' cache hit rates are not very low (more than 76%), they have also been affected by the potential cyber-attacks on day 4. Thus, 14 unique nodes are labeled as abnormal nodes at this stage.

Table 7.14: Average Feature Values of IP Clusters in DDoS Attack Detection using GMM

Cluster No.	Avg number of requests	Avg number of nodes	Avg request interval (s)	Avg cache hit rate	Avg request error rate	Avg request popularity
1	1.09	1.08	24.9	0.850	0.0	0.941
2	354.6	3.78	6.76	0.579	0.0001	0.810
...	...	...	...	...	...	...
10	3039.9	2.39	0.6	0.026	0.972	0.947

After that, the 11,834 client IPs that have sent requests in the abnormal event period are also analyzed. An optimized GMM is utilized to cluster these 11,834 IPs into 10 clusters, which has the highest silhouette score of 0.73. As shown in Table 7.14, among the 10 clusters, the 103 IPs in the cluster No. 10 are found to be the malicious IPs since they have sent a large number of 404-error requests to an average of 2.39 target nodes at a high frequency. Thus, these 103 IPs are likely to have launched a distributed DoS (DDoS) attack in the abnormal period 13:00-15:00, day 4. By comparing these 103 IPs with the 60 potential DoS attack IPs detected by the optimized GMM and crowd event analysis, 41 IPs are the overlaps, and the total number of DoS attack IPs is increased to 122.

Lastly, it can be seen that this DDoS attack cannot be found by only considering the number of requests shown in Fig. 7.6. This is because this DDoS attack mainly targeted node No. 25. Although node No. 25 received a large number of malicious requests in this DDoS attack period, this number of requests is still small compared to the total number of requests received by all nodes or in the crowd events. This emphasizes the reasons for monitoring the changes in different features instead of only one feature for time-series analysis; otherwise, real anomalies may be missed.

Account-offering (AO) Analysis

In the final stage of result validation, account-offering analysis is performed on the current anomaly detection results to remove false alarms.

There are 70 unique AOs in the 8 days dataset. To analyze each AO, its number of requests, number of nodes, cache hit rate, service type, and request popularity are calculated and analyzed together with their configuration information provided by the CDN operator. The information about the suspicious AOs is shown in Table 7.15. Firstly, the AOs No. 1-4 are the

Table 7.15: The Behaviors of Suspicious Account-offerings

AO No.	**Number of requests**	**Number of nodes**	**Cache hit rate**	**Service type**	**Request popularity**
1	44,985,524	27	0.999	Static	1.0
2	78,505,897	28	0.999	Static	1.0
3	57,276,415	34	0.997	Static	1.0
4	63,844,994	25	0.997	Static	1.0
5	184,294	34	0.809	Progressive download	0.262
6	1,487,0894	47	0.747	Live streaming	0.762
7	7,557,320	48	0.763	Live streaming	0.751
8	127,665,987	49	0.907	Static	0.985
Other static AOs	6,501,377	5.06	0.992	Static	0.999

major AOs that provide services during crowd events since they have received a large number of requests for high-popularity contents with more than 99% cache hit rate during the crowd event periods. The 250 detected crowd event IPs mainly used these 4 AOs, which further proves that they are the legitimate IPs instead of DoS attack IPs. Thus, these 250 IPs are the false positives of the proposed unsupervised model.

For the detected abnormal IPs and nodes, the AOs No. 5-8 shown in Table 7.15 are the four suspicious AOs since almost all the requests of the detected potential abnormal IPs and nodes were sent through these 4 AOs. It is found in the configuration information that the node No. 5 is the major progressive download content source, and certain unpopular contents might have been requested by an IP many times for being progressively downloaded. The AOs No. 6 & 7 are the major live streaming content sources and have relatively high cache hit rates and request popularity, about 75%. Thus, certain IPs that have used any of these two AOs but got a very low cache hit rate or request popularity are likely to be abnormal. Moreover, the AO No. 8 is the major static content source. It can be seen in the last row of Table 7.15 that other static AOs have got a more than 99% cache hit rate and request popularity, much higher than the AO No. 8, so the AO No. 8 might be used by certain IPs to launch attacks.

Further result validation is done through the AO analysis. Firstly, among the 14 detected abnormal nodes, the 11 compromised nodes under potential DDoS attacks listed in Table 7.13 are validated to be the real compromised nodes since their major AOs belong to the abnormal AOs No. 6-8. For the other three detected abnormal nodes (nodes No. 0, 5, 7) shown in Table

Table 7.16: Mean Values of Each Feature on All Days for Three Abnormal Nodes

Node No.	**Avg cache hit rate**	**Avg legitimate IP cache hit rate**	**Avg data transfer rate (MB/s)**	**Avg request error rate**	**Avg number of requests**	**Avg request popularity**
0	0.49	0.0	0.69	0.0	344	0.935
5	0.06	0.08	0.13	0.0	2383.8	0.905
7	0.0002	0.0	0.0009	0.0002	531.6	0.881

Table 7.17: Mean Values of Each Feature For Normal and Detected Abnormal Nodes

Node label	**Cache hit rate**	**Legitimate IP cache hit rate**	**Data transfer rate (MB/s)**	**Request error rate**	**Request popularity**
Normal	0.936	0.928	0.805	0.004	0.978
Abnormal	0.405	0.416	0.506	0.049	0.936

7.16, since their AOs are legitimate AOs and they have not received many requests (less than 2,500), so they might have been breached before day 1, so any new legitimate requests were unable to get cache hits. Additionally, nodes No. 5 & 7 were attacked by CPAs since they have much lower request popularities than normal nodes (0.905 & 0.881 versus 0.978). Therefore, a total of 14 compromised nodes, including 3 nodes that have already been attacked before day 1, and 11 nodes that were under attack on day 4, have been identified and illustrated in Tables 7.13 and 7.16, and their average feature values are shown in Table 7.17.

According to the configuration information provided by the CDN operator, it is known that as a major source for progressive download videos, the AO No. 5 was used for the tests of old videos, most of which have low popularity. Thus, AO No. 5 has some similar characteristics to CPAs since it is also used to request for low popularity files many times, but it was used by legitimate users to do tests. Therefore, for the detected abnormal contents and CPA IPs, those using the AO No.5 as the major AO are found to be legitimate entities and removed from the abnormal content and IP list. After removing the false positives (113 contents and 21 IPs), the behaviors of the detected 56 real abnormal contents and 33 real CPA IPs are shown in Tables 7.18 and 7.19, respectively. For the two types of CPAs, 30 IPs are labeled as LDA IPs, and 3 other IPs are labeled as FLA IPs.

Lastly, through AO analysis, all the detected DoS IPs have used the AOs No. 6-8 to make requests, and they are labeled real DoS IPs. Among these 122 DoS IPs, 103 of them are the DDoS attack IPs that have launched a DDoS attack together on day 4, from 13:00 to 15:00.

Table 7.18: Mean Values of Each Feature for Normal and Detected Abnormal Contents

Content label	Avg number of requests	Avg request per node ratio	Avg request per IP ratio	Avg cache hit rate	Avg popularity
Normal	95.6	7.7	1.5	0.21	0.19
Abnormal	1049.7	228.9	284.3	0.852	0.333

Table 7.19: Mean Values of Each Feature For Normal and Detected Potential CPA IPs

IP label	Avg number of requests	Avg number of nodes	Avg request per content ratio	Avg cache hit rate	Avg request error rate	Avg request popularity
Normal	161.3	2.1	1.56	0.925	0.006	0.977
LDA	9461.4	3.76	1.13	0.111	0.0006	0.316
FLA	9418.8	5.5	84.74	0.691	0.006	0.478
CPA	9458.8	3.87	6.36	0.147	0.001	0.326

Table 7.20: Mean Values of Each Feature For Normal and Detected Potential DoS Attack IPs

IP label	Avg number of requests	Avg number of nodes	Avg request interval (s)	Avg cache hit rate	Avg request error rate	Avg request popularity
Normal	161.3	2.1	1177.6	0.925	0.006	0.977
DDoS	3039.9	2.39	0.6	0.026	0.972	0.947
Other DoS	59789.5	4.63	1.97	0.708	0.264	0.983
All DoS	14465.3	2.98	1.44	0.16	0.821	0.943

The other 19 abnormal IPs may have tried to launch a DoS attack individually during different time periods. Their behaviors are shown in Table 7.20.

7.5.4 Results Summary

In summary, 14 compromised nodes, 56 abnormal contents, 33 CPA IPs, and 122 DoS attack IPs were detected by the proposed anomaly detection model, as shown in Table 7.21. Among the detected abnormal network entities, 12 nodes and 122 IPs were affected by DoS attacks, while 2 nodes, 33 IPs, and 56 contents were affected by CPAs. Additionally, 384 false positives (FPs) and 65 false negatives (FNs) were removed through the multi-perspective validation process. As the utilized datasets are completely unlabeled, all these anomaly detection results have been analyzed and verified to be 100% accurate by multiple cybersecurity experts and industrial partner security network engineers. This verification process is a necessary HITL procedure in real-world applications, as the experts and engineers have in-depth knowledge

Table 7.21: Summary of Detected Anomalies

Label	Number of nodes	Number of IPs	Number of contents
Normal	36	1,267,872	1,867,528
DoS	12	122	-
CPA	2	33	56
Removed FPs	0	271	113
Removed FNs	3	62	0

of the dataset, legitimate events, and cyber-attack patterns. This process also validates the effectiveness of the proposed framework.

Lastly, the performance of the proposed framework is compared with several data analytics and anomaly detection techniques. As discussed in Section 7.4.4, binary outlier detection algorithms are more suitable for the node-based dataset that only has 50 samples, so two standard binary outlier detection algorithms, iForest [11] and one-class support vector machine (OC-SVM) [58], are used for the comparison of abnormal node detection. On the other hand, clustering algorithms are suitable for IP & content-based datasets that have more than 1 million samples, so two common clustering algorithms, k-means [59] and GMM [12], are used for the comparison of abnormal IP & content detection. For the performance metrics, since the datasets are highly imbalanced, precision (Pre), Recall (Rec), and F1-score are used with accuracy (Acc) for model evaluation. By calculating the harmonic mean of the precision and recall, F1-score is a reliable metric to measure the classification performance on imbalanced datasets [26].

The model performance comparison is shown in Table 7.22. The accuracy of most methods are larger than 99%, but this is mainly due to the imbalanced dataset (more than 99% of data samples are normal data). The iForest and GMM models used for the result comparison are only themselves without the proposed validation procedures; hence, many FPs and FNs occurred, which reduced the F1-scores of iForest and GMM to 88.0%, 35.8%, and 49.8% on the node, IP, and content-based datasets, respectively. This emphasizes the importance of implementing the proposed multi-perspective validation method. Moreover, the F1-scores of OC-SVM and k-means algorithms are lower than the iForest and GMM models from all three perspectives, which justifies the rationale for choosing iForest and GMM in our proposed framework.

Table 7.22: Performance Comparison with Regular Anomaly Detection Techniques

Method	**Detection perspective**	**Acc (%)**	**Pre (%)**	**Rec (%)**	**F1 (%)**
Proposed	All	100.0	100.0	100.0	100.0
IForest [11]	Node	94.0	100.0	78.6	88.0
OC-SVM [58]		86.0	73.3	78.6	75.9
GMM [12]	IP	99.97	25.6	60.0	35.8
K-means [59]		99.96	21.1	69.7	32.3
GMM [12]	Content	99.99	33.1	100.0	49.8
K-means [59]		99.99	25.8	100.0	41.0

As the CDN requests/traffic are continuously generated and we aim to detect abnormal network entities (*i.e.*, node, IP, content) and corresponding attacks, the node, IP, and content-based datasets need to be continuously updated by implementing the proposed feature engineering method on the new CDN traffic data. Then, the proposed unsupervised learning models continuously detect anomalies on the updated datasets. These are the major procedures that take time. The reason for the reaction or detection time is because the proposed system needs to process a number of requests to update the behaviors of nodes, IPs, and contents; thus, the anomalies can be identified. To protect CDNs against DoS and CPA, the anomalies should be detected in time. Hence, assuming the traffic is continuously generated, we have evaluated the detection speed of the proposed system by measuring the total execution time from the time each abnormal entity is affected by an attack to the time the proposed method detects this anomaly. This anomaly detection time has been further divided into the feature engineering time and the unsupervised model detection time. Moreover, as different anomalies/attacks show different patterns and need different detection time, we have measured the average (Avg) and maximum (Max) execution time of all abnormal nodes, IPs, and contents in seconds.

The anomaly detection time of the proposed method and the two compared methods (OC-SVM and k-means) is shown in Table 7.23. From the node perspective, the proposed method can detect the abnormal nodes in the average time of 9.9s and the maximum time of 29.9s, which is slighter faster than OC-SVM. The unsupervised detection time for abnormal node detection is low, because the small number of nodes (50) has made it faster for the proposed method to detect the abnormal nodes than the abnormal IPs or contents. From the IP and content perspectives, the proposed method can detect abnormal IPs and contents in the average time of 19.2s and 10.3s, and in the maximum time of 58.5s and 22.9s, respectively. The

Table 7.23: Anomaly Detection Time Comparison

Detection Perspective	**Method**	**Detection Time of Each Anomaly (s)**					
		Feature Engineering		**Unsupervised Detection**		**Total**	
		Avg	**Max**	**Avg**	**Max**	**Avg**	**Max**
Node	Proposed	9.8	29.6	0.1	0.3	9.9	29.9
	OC-SVM [58]			0.5	1.6	10.3	31.2
IP	Proposed	12.4	32.9	6.8	25.6	19.2	58.5
	K-means [59]			3.1	15.3	15.5	48.2
Content	Proposed	5.8	12.5	4.5	10.4	10.3	22.9
	K-means [59]			2.3	7.7	8.1	20.2

detection time of the abnormal IPs is higher than the time of the abnormal contents, because detecting certain DoS IPs needs to analyze a large number of requests. Nevertheless, the maximum abnormal IP detection time is still at a low level (58.5s). This shows that the proposed solution can detect anomalies at the early stages of attacks, which can help CDNs stop current attacks in time and prevent future attacks. Although the anomaly detection time of k-means is lower than the proposed method, the proposed method can achieve much higher performance than k-means, as shown in Table 7.22. Therefore, our proposed method still performs the best among the comparisons by considering both detection accuracy and speed.

7.6 Practical Usage and Open Issues

Given that the proposed framework is working with unlabeled data, the framework can be used as the first level of the anomaly detection process. Unlike traditional anomaly detection processes for unlabeled data, which requires massive manual analysis and expert knowledge, the proposed framework can reduce much overhead using the automatically-tuned ML algorithms and a systematic multi-perspective result validation process proposed in Section 7.4. Additionally, the high performance of the proposed framework has been verified by multiple security experts.

One open issue with this work is that it only considers two common service targeting attacks, *i.e.*, CPAs and DoS attacks. Nevertheless, the proposed framework can be extended to new attacks based on the same procedures:

1. Collect sufficient network log data that contains the new attack samples;

2. Analyze the new attack, summarize the attack patterns, and select appropriate features that can reflect the new attack patterns;
3. Utilize the proposed unsupervised anomaly detection model to detect suspicious activities and compare them with the summarized characteristics of new attacks to determine the real attacks;
4. Utilize the proposed multi-perspective result validation method to remove false alarms and false negatives.

Through this process, any new attacks that can be reflected from CDN log data can be effectively detected by the proposed method.

On the other hand, although the proposed framework can reduce much manual analysis and labeling process, certain expert knowledge and legitimate event information are still required in the proposed framework to achieve accurate anomaly detection. This is because certain numerical anomalies identified by unsupervised ML algorithms are not true anomalies and require further analysis to distinguish between certain legitimate abnormal events and cyber-attacks.

Lastly, several procedures in addition to our work should be implemented for real-world CDN applications. Since this research work can be considered a pseudo labeling process on an unlabeled raw CDN log dataset, the labeled content, client IP, and node-based datasets can be used to train reliable supervised classifiers. Through this procedure, human efforts will not be required in the future anomaly detection process, and the system can automatically detect anomalies by continuously processing the incoming data. Thus, supervised model development is one direction for future work.

Moreover, corresponding countermeasures should be made along with attack detection to stop or prevent cyber-attacks. The proposed multi-perspective anomaly detection framework enables the implementation of countermeasures on both attacker and victim sides.

Firstly, on the client or attacker side, traffic filtering and blacklisting are potential response mechanisms to prevent the detected malicious clients from sending requests to CDN servers [60]. Detecting abnormal IPs enables network administrators to locate the origins of cyber-attacks [61]. Additionally, the domains and specific geographic locations can be found based on the detected malicious client IP addresses. To make countermeasures, the detected malicious

IPs, as well as the IPs from the same domains and locations, can be added to the blacklist to block or limit the traffic from these IPs until the identities of these suspicious clients are verified. Hence, the current attacks can be stopped, and future attacks can be prevented by blacklisting the detected malicious IPs [9].

Secondly, on the victim side, countermeasures can be made from both content and node perspectives. Blacklisting can also be used for detected abnormal contents [60]. The contents in the blacklist will be deleted from the cache space of any edge servers. Additionally, the edge servers can reject the requests sent for these abnormal contents or never cache them. Through this process, CPAs can be stopped because the detected abnormal contents cannot be used by CPA attackers to pollute the cache space of CDN servers. On the other hand, the compromised nodes can be isolated until they recover. Specifically, the compromised node information will be notified to other nodes, so that other nodes can avoid communicating with the affected nodes that are under attack [60]. The nodes can recover by limiting the requests sent to them or implementing the blacklisting strategies from the client IP and content perspectives. Once recovered, the nodes can be removed from the isolation list and continue communications.

In conclusion, as the proposed anomaly detection can continuously detect anomalies, any new malicious or compromised network entities can be identified, and corresponding countermeasures can be made to defend against cyber-attacks when they occur. Moreover, the multi-perspective countermeasures enhance the defense capabilities of CDNs. For example, an attacker cannot continue an attack by simply changing IP addresses, because the countermeasures from other perspectives, like isolating nodes and blacklisting contents, can still stop the attack.

Since the development of supervised models and countermeasures is outside the scope of this work, it will be our future work.

7.7 Conclusion

CDNs have become a major content distribution technology in modern networks. However, their caching mechanism introduces additional vulnerabilities. In this chapter, we proposed a multi-perspective anomaly detection approach based on a real-world general CDN access log dataset to identify abnormal network entities and cyber-attacks. To detect DoS and cache pollution attacks, we first summarized their patterns and then extracted features from four main perspectives: content, client IP, account-offering, and node perspectives. After obtaining the extracted datasets from multiple perspectives, the anomalies were identified using the optimized unsupervised learning model constructed with the optimized isolation forest and Gaussian mixture models. A comprehensive validation method, including multi-perspective analysis, time-series analysis, and account-offering analysis, was implemented to validate the detected abnormal network entities and the corresponding cyber-attacks. Thus, detection errors can be effectively reduced. Ultimately, the abnormal contents, compromised nodes, and malicious IPs were detected and labeled. In future work, the labeled anomaly detection results can be used for classifier development so that an automated process can be developed to detect new attacks and abnormal network entities effectively. Certain security mechanisms, such as isolating and blacklisting the detected abnormal network entities, can be utilized after anomaly detection to secure CDNs.

Bibliography

[1] L. Yang *et al.*, ”Multi-Perspective Content Delivery Networks Security Framework Using Optimized Unsupervised Anomaly Detection,” *IEEE Trans. Netw. Serv. Manag.*, 2021.

[2] L. Yang, A. Moubayed, A. Shami, A. Boukhtouta, P. Heidari, S. Preda, R. Brunner, D. Migault, and A. Larabi, “Forensic Data Analytics for Anomaly Detection in Evolving Networks,” Submitted to *Digital Forensics*, World Scientific, 2022.

[3] J. Zhao, P. Liang, W. Liufu, and Z. Fan, “Recent Developments in Content Delivery Network: A Survey,” in *Parallel Architectures, Algorithms and Programming*, 2020, pp. 98–106.

[4] M. Z. Shafiq, A. R. Khakpour, and A. X. Liu, “Characterizing caching workload of a large commercial Content Delivery Network,” *Proc. - IEEE INFOCOM*, vol. 2016-July, pp. 1–9, 2016.

[5] A. Boukhtouta, M. Pourzandi, R. Brunner, and S. Dault, “Fingerprinting Crowd Events in Content Delivery Networks: A Semi-supervised Methodology,” in *IFIP Annual Conference on Data and Applications Security and Privacy*, 2018, pp. 312–329.

[6] E. Nygren, R. K. Sitaraman, and J. Sun, “The Akamai network: A platform for high-performance Internet applications,” *Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, 2010.

[7] L. Deng, Y. Gao, Y. Chen, and A. Kuzmanovic, “Pollution attacks and defenses for Internet caching systems,” *Comput. Networks*, vol. 52, no. 5, pp. 935–956, 2008.

[8] R. Aliyev, D. Seo and H. Lee, ”DROP-FAST: Defending against DDoS Attacks using Cloud Technology”, *Int. Conf. on Security and Management*, 2013.

[9] M. M. Rahman, S. Roy, and M. A. Yousuf, “DDoS Mitigation and Intrusion Prevention in Content Delivery Networks using Distributed Virtual Honeypots,” *1st Int. Conf. Adv. Sci. Eng. Robot. Technol. 2019, ICASERT 2019*, vol. 2019, no. Icasert, 2019.

[10] K. L. Moore, T. J. Bihl, K. W. Bauer, and T. E. Dube, “Feature extraction and feature selection for classifying cyber traffic threats,” *J. Def. Model. Simul.*, vol. 14, no. 3, pp. 217–231, 2017.

[11] L. Sun, S. Versteeg, and A. Rao, “Detecting Anomalous User Behavior Using an Extended Isolation Forest Algorithm: An Enterprise Case Study.” *arXiv Prepr. arXiv1609.06676*, pp. 1–13, 2016, [Online]. Available: https://arxiv.org/abs/1609.06676.

[12] M. Bitaab and S. Hashemi, "Hybrid Intrusion Detection: Combining Decision Tree and Gaussian Mixture Model," in *2017 14th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*, 2017, pp. 8–12.

[13] A. Holzinger, "Interactive machine learning for health informatics: when do we need the human-in-the-loop?," *Brain Informatics*, vol. 3, no. 2, pp. 119–131, 2016.

[14] D. R. Honeycutt, M. Nourani, and E. D. Ragan, "Soliciting human-in-the-loop user feedback for interactive machine learning reduces user trust and impressions of model accuracy," *arXiv Prepr. arXiv2008.12735*, pp. 1–10, 2020, [Online]. Available: https://arxiv.org/abs/2008.12735.

[15] H. Park, I. Widjaja, and H. Lee, "Detection of cache pollution attacks using randomness checks," *IEEE Int. Conf. Commun.*, pp. 1096–1100, 2012.

[16] J. B. Gouge, "A targeted denial of service attack on data caching networks", *UNF Graduate Theses and Dissertations*, 2015.

[17] C. Ghasemi, H. Yousefi, and B. Zhang, "Far Cry: Will CDNs Hear NDN's Call?," in Proceedings of the 7th ACM Conference on Information-Centric Networking, 2020, pp. 89–98.

[18] S. Triukose, Z. Al-Qudah, and M. Rabinovich, "Content Delivery Networks: Protection or Threat?," in *Computer Security – ESORICS 2009*, 2009, pp. 371–389.

[19] B. Zolfaghari *et al.*, "Content Delivery Networks: State of the Art, Trends, and Future Roadmap," ACM Comput. Surv., vol. 53, no. 2, Apr. 2020.

[20] M. Conti, P. Gasti, and M. Teoli, "A lightweight mechanism for detection of cache pollution attacks in Named Data Networking," *Comput. Networks*, vol. 57, no. 16, pp. 3178–3191, 2013.

[21] J. Chen, H. Xu, S. Penugonde, Y. Zhang, and D. Raychaudhuri, "Exploiting ICN for efficient content dissemination in CDNs," *Proc. - 4th IEEE Work. Hot Top. Web Syst. Technol. HotWeb 2016*, pp. 14–19, 2016.

[22] C. Ghasemi, H. Yousefi, and B. Zhang, "ICDN: An NDN-Based CDN," in *Proceedings of the 7th ACM Conference on Information-Centric Networking*, 2020, pp. 99–105.

[23] H. V. Nguyen, L. Lo Iacono, and H. Federrath, "Your cache has fallen: Cache-poisoned denial-of-service attack," *Proc. ACM Conf. Comput. Commun. Secur.*, pp. 1915–1930, 2019.

[24] H. Wang, X. Chen, W. Wang, and M. Y. Chan, "Content pollution propagation in the overlay network of peer-to-peer live streaming systems: Modelling and analysis," *IET Commun.*, vol. 12, no. 17, pp. 2119–2131, 2018.

[25] K.-W. Lee, S. Chari, A. Shaikh, S. Sahu, and P.-C. Cheng, "Improving the resilience of content distribution networks to large scale distributed denial of service attacks," *Comput. Networks*, vol. 51, no. 10, pp. 2753–2770, 2007.

[26] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, “Tree-based Intelligent Intrusion Detection System in Internet of Vehicles,” *2019 IEEE Glob. Commun. Conf.*, no. Ml, pp. 1–6, 2019.

[27] E. Shuster, L. LaSeur, O. Katz, and S. Ragan, “Financial Services Attack Economy,” *Akamai Technol.*, vol. 5, no. 4, pp. 1–40, 2019.

[28] “Vulnerability Note VU# 335217: Content Delivery Networks handle HTTP headers in different and unexpected ways”, *Technical report US CERT Vulnerability Notes Database*, Jan 2020, [online] Available: http://www.kb.cert.org/vuls/id/836068.

[29] M. Xie, I. Widjaja, and H. Wang, “Enhancing cache robustness for content-centric networking,” *Proc. - IEEE INFOCOM*, pp. 2426–2434, 2012.

[30] A. Karami and M. Guerrero-Zapata, “An ANFIS-based cache replacement method for mitigating cache pollution attacks in Named Data Networking,” *Comput. Networks*, vol. 80, pp. 51–65, 2015.

[31] A. Moubayed, M. Injadat, A. Shami, and H. Lutfiyya, “DNS Typo-Squatting Domain Detection: A Data Analytics & Machine Learning Based Approach,” *2018 IEEE Glob. Commun. Conf. GLOBECOM 2018 - Proc.*, 2018.

[32] A. Moubayed, E. Aqeeli, and A. Shami, “Ensemble-based Feature Selection and Classification Model for DNS Typo-squatting Detection,” in *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2020.

[33] P. Kumar, A. Moubayed, A. Refaey, A. Shami, and J. Koilpillai, “Performance Analysis of SDP For Secure Internal Enterprises,” *IEEE Wirel. Commun. Netw. Conf. WCNC*, vol. 2019-April, 2019.

[34] A. Moubayed, A. Refaey, and A. Shami, “Software-Defined Perimeter (SDP): State of the Art Secure Solution for Modern Networks,” *IEEE Netw.*, vol. 33, no. 5, pp. 226–233, 2019.

[35] D. Chiba, K. Tobe, T. Mori, and S. Goto, “Detecting malicious websites by learning IP address features,” *Proc. - 2012 IEEE/IPSJ 12th Int. Symp. Appl. Internet*, SAINT, 2012, no. July, pp. 29–39, 2012.

[36] A. Pinto, “Defending Networks with Incomplete Information: A Machine Learning Approach,” *Black Hat USA*, pp. 1–10, 2013, [Online]. Available: https://www.blackhat.com/us-13/briefings.html#Pinto.

[37] P. Fiadino, A. D’Alconzo, A. Bär, A. Finamore, and P. Casas, “On the detection of network traffic anomalies in content delivery network services,” *2014 26th Int. Teletraffic Congr. ITC 2014*, 2014.

[38] V. H. La and A. R. Cavalli, “A Misbehavior Node Detection Algorithm for 6LoWPAN Wireless Sensor Networks,” *Proc. - 2016 IEEE 36th Int. Conf. Distrib. Comput. Syst. Work. ICDCSW 2016*, pp. 49–54, 2016.

[39] N. Berjab, H. H. Le, C. M. Yu, S. Y. Kuo, and H. Yokota, “Abnormal-Node detection based on spatio-Temporal and Multivariate-Attribute correlation in wireless sensor networks,” *Proc. - IEEE 16th Int. Conf. Dependable, Auton. Secur. Comput.*, pp. 568–575, 2018.

[40] S. K. Pandey, P. Kumar, J. P. Singh, and M. P. Singh, “Intrusion detection system using anomaly technique in wireless sensor network,” *Proceeding - IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2016*, pp. 611–615, 2017.

[41] S. Caltagirone, A. Pendergast, and C. Betz, “The Diamond Model of Intrusion Analysis,” *DTIC Doc. Tech. Rep.*, 2013.

[42] L. Yang, A. Moubayed, and A. Shami, “MTH-IDS: A Multi-Tiered Hybrid Intrusion Detection System for Internet of Vehicles,” *IEEE Internet Things J.*, 2021.

[43] D. Z. and M. C. Andrew Moore, “Discriminators for use in flow-based classification,” Dept. Comput. Sci., Queen Mary Univ., London, U.K., Tech. Rep. RR-05-13, 2005.

[44] A. Juvonen, T. Sipola, and T. Hämäläinen, “Online anomaly detection using dimensionality reduction techniques for HTTP log analysis,” *Comput. Networks*, vol. 91, pp. 46–56, 2015.

[45] T. Zehelein, S. Schuck, and M. Lienkamp, “Automotive Damper Defect Detection Using Novelty Detection Methods,” in *Proceedings of the ASME 2019 Dynamic Systems and Control Conference*, 2019.

[46] G. A. Susto, A. Beghi, and S. McLoone, “Anomaly detection through on-line isolation Forest: An application to plasma etching,” in *2017 28th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, May 2017, pp. 89–94.

[47] M. N. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, “Systematic Ensemble Model Selection Approach for Educational Data Mining,” *Knowledge-Based Syst.*, vol. 200, p. 105992, 2020.

[48] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, “Multi-split Optimized Bagging Ensemble Model Selection for Multi-class Educational Data Mining,” *Appl. Intell.*, 2020.

[49] L. Yang and A. Shami, “A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams,” *IEEE Internet Things Mag.*, 2021.

[50] M. Injadat, F. Salo, A. B. Nassif, A. Essex, and A. Shami, “Bayesian Optimization with Machine Learning Algorithms Towards Anomaly Detection,” *2018 IEEE Glob. Commun. Conf. GLOBECOM 2018 - Proc.*, 2018.

[51] M. Injadat, A. Moubayed, A. B. Nassif and A. Shami, “Multi-Stage Optimized Machine Learning Framework for Network Intrusion Detection,” *IEEE Trans. Netw. Serv. Manag.*, 2020.

[52] L. Yang and A. Shami, “On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[53] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011, [Online]. Available: http://scikit-learn.sourceforge.net.

[54] A. Moubayed, M. Injadat, A. Shami, and H. Lutfiyya, “Student Engagement Level in e-Learning Environment: Clustering Using K-means,” *Am. J. Distance Educ.*, vol. 34, no. 02, pp. 1–20, 2020.

[55] F. Salo, M. N. Injadat, A. Moubayed, A. B. Nassif, and A. Essex, “Clustering Enabled Classification using Ensemble Feature Selection for Intrusion Detection,” *2019 Int. Conf. Comput. Netw. Commun. ICNC 2019*, pp. 276–281, 2019.

[56] R. C. Pinto and P. M. Engel, “Scalable and Incremental Learning of Gaussian Mixture Models.” *arXiv Prepr. arXiv1701.03940*, pp. 1–13, 2017, [Online]. Available: http://arxiv.org/abs/1701.03940.

[57] N. Shi, X. Liu, and Y. Guan, “Research on k-means clustering algorithm: An improved k-means clustering algorithm,” *3rd Int. Symp. Intell. Inf. Technol. Secur. Informatics, IITSI 2010*, pp. 63–67, 2010.

[58] C. She, W. Wen, Z. Lin, and K. Zheng, “Application-Layer DDOS Detection Based on a One-Class Support Vector Machine,” *Int. J. Netw. Secur. Its Appl.*, vol. 9, no. 1, pp. 13–24, 2017.

[59] A. Gupta and P. Nahar, “Detection of Cache Pollution Attacks in a Secure Information-Centric Network,” in *Data Analytics and Management*, 2021, pp. 377–397.

[60] A. Fiandrotti, R. Gaeta, and M. Grangetto, “Simple countermeasures to mitigate the effect of pollution attack in network coding based peer-to-peer live streaming,” *IEEE Trans. Multimed.*, vol. 17, no. 4, pp. 562–573, 2017.

[61] A. Carlin, M. Hammoudeh, and O. Aldabbas, “Intrusion Detection and Countermeasure of Virtual Cloud Systems - State of the Art and Current Challenges,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 6, no. 6, pp. 1–15, 2015.

Chapter 8

A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams

8.1 Introduction

As Internet and mobile device use grows rapidly, the number of Internet of Things (IoT) devices and the produced IoT data continue to increase significantly. As presented in the Cisco report, more than 30 billion IoT devices are estimated to be connected in 2021, and around five quintillion bytes of IoT data are produced every day [2]. IoT data can be collected from different sources and domains, like IoT sensors and smart devices, and can then be transmitted to the central servers for analytics through various communication strategies, including WiFi, Bluetooth, ZigBee, etc. Meaningful insights can be extracted from IoT data streams using data analytics methods to support various IoT applications, like anomaly detection [3] [4]. Fig. 8.1 illustrates the overall architecture of IoT data analytics.

However, the analytics of tremendous volumes of ever-increasing IoT streaming data and the use of ML models in IoT systems still face many challenges [2]. Time and resource constraints are the main issues with IoT data analytics because of the low-power and low-cost

A version of this chapter has been published in IEEE Internet of Things Magazine [1].

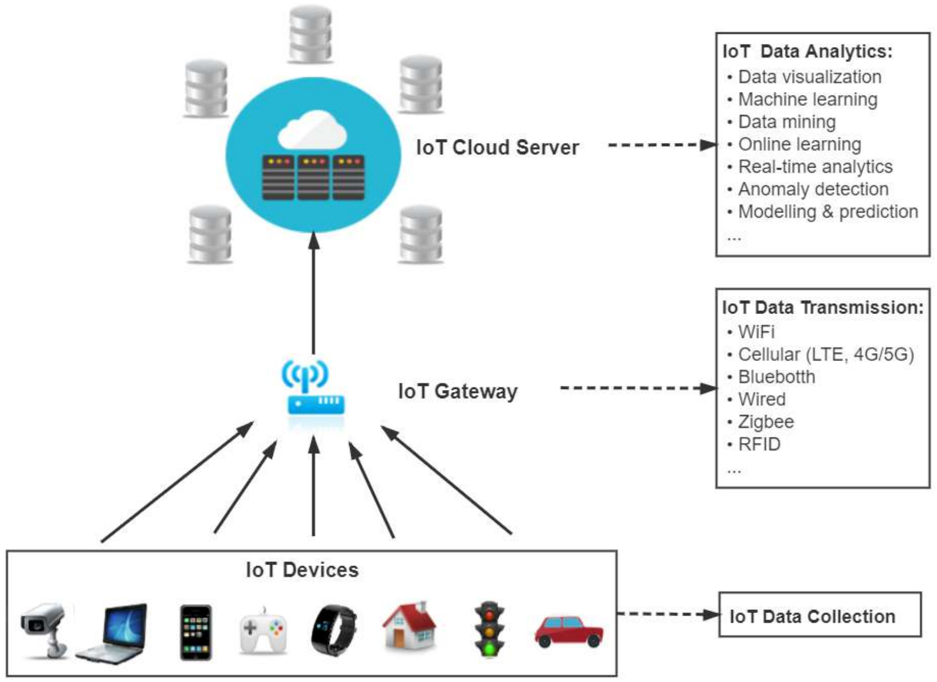

Figure 8.1: The architecture of IoT data analytics.

requirements of IoT devices [5]. On the other hand, IoT data is usually dynamic and non-stationary streaming data due to the ever-changing nature of IoT systems and applications. In real-world IoT systems, the physical events monitored by IoT sensors can change over time, and unpredictable abnormal events can occur occasionally. Additionally, IoT device components will age and need update or replacement periodically [6]. These factors can cause inevitable changes in the statistical distributions of IoT data streams, known as concept drift [5]. The occurrence of concept drift can cause IoT system failure or performance degradation. Traditional offline machine learning (ML) models cannot deal with concept drift, making it necessary to develop online adaptive analytics models that can adapt to the predictable and unpredictable changes in the IoT data [7].

Therefore, this chapter proposes a drift adaptive ML-based framework for IoT streaming data analytics. The framework consists of a light gradient boosting machine (LightGBM) for IoT data learning, a particle swarm optimization (PSO) method for model optimization, and a proposed novel method named optimized adaptive and sliding windowing (OASW) for concept drift adaptation. The effectiveness and efficiency of the proposed adaptive framework are evaluated using two public IoT cyber-security datasets: IoTID20 [8] and NSL-KDD [9], to evaluate the proposed framework in intrusion detection use cases as an example for IoT data stream analytics.

The main contributions of this chapter can be summarized as follows:

1. We discuss the challenges and potential solutions for IoT streaming data analytics.
2. We propose a novel drift adaptation method named OASW to address the concept drift issue. Its performance was evaluated through comparison with other state-of-the-art approaches.
3. We propose an optimized adaptive framework for IoT anomaly detection use cases with offline and online learning functionalities based on LightGBM, PSO, and OASW.

The remainder of this chapter is organized as follows: Section 8.2 discusses the challenges of IoT data stream analytics. Section 8.3 describes the proposed adaptive framework and the novel OASW algorithm. Section 8.4 presents and discusses the experimental results. Section 8.5 concludes the chapter.

8.2 IoT Data Analytics Challenges and Potential Solutions

Due to the high velocity, volume, and variability characteristics of IoT streaming data, there are two major challenges related to IoT data analytics: 1) time & memory constraints and 2) concept drift [6]. In this section, the challenges and potential solutions for IoT streaming data analytics are discussed. Table 8.1 summarizes them.

8.2.1 Time and Memory Constraints

"High velocity" and "high volume" indicate the high generation speed and large scale of IoT streaming data. This requires that IoT data streams should be processed and analyzed as soon as they reach the learning model. However, in IoT systems, most IoT devices are low-power and low-cost devices with limited computational resources, which limit their data analytics speeds [5]. The memory constraints of IoT devices also limit their capabilities to process and

Code is available at: https://github.com/Western-OC2-Lab/OASW-Concept-Drift-Detection-and-Adaptation

Table 8.1: IoT Data Analytics Challenges and Solutions

IoT Data Characteristic	**Challenge**	**Description**	**Potential Solutions**
High Velocity and Volume	Time & Memory Constraints	Large volumes of IoT data are continuously produced at a high rate, making it difficult to process and store all the data due to the time and memory constraints of low-cost IoT devices. This requires that the average IoT data analytics speed is higher than the average data generation/collection time to meet the real-time processing requirements; otherwise, it could cause IoT service unavailability or system failure.	Online learning methods with low computational complexity and forgetting mechanisms are potential solutions to achieve real-time processing of IoT data streams: 1) Sliding window methods: They use a sliding window to retain and process only the most recent data samples and discard old samples to save learning time and storage space. 2) Incremental learning methods: They can process every incoming new data sample by partially updating the learning model. The data and model complexity can be reduced by discarding the historical data samples and model components to address the execution time and memory constraints of IoT data analytics.
High Variability	Concept Drift Detection	Due to the non-stationary IoT data and dynamic IoT environments, concept drift issues often occur in IoT data, causing analytics model degradation. Drift detection faces two main challenges: many causing factors and multiple types of drifts in IoT systems.	Concept drift can be detected using the following techniques: 1) Window-based methods (e.g., ADWIN): They use fixed-sized sliding windows or adaptive windows as data memories for different concepts to detect the occurrence of concept drift. 2) Performance-based methods (e.g., DDM & EDDM): They monitor the model performance degradation rate to detect concept drift.
	Concept Drift Adaptation	After drift detection, the observed drift should be effectively handled so that the learning model can adapt to the new data patterns.	Concept drift can be handled using the following techniques: 1) Adaptive algorithms (e.g., SAM-KNN): They handle concept drift by fully retraining or altering the learning model on an updated dataset after detecting a drift. 2) Incremental learning methods (e.g., HATT): They partially updated the learning model when new samples arrive or drift is detected. 3) Ensemble learning methods (e.g., ARF & SRP): They combine multiple base learners trained on data streams of different concepts.

store large volumes of IoT data and high complexity learning models. Thus, it is essential to develop low computational complexity analytics models.

Online learning methods that enable real-time analytics are able to satisfy the time and memory constraints of IoT systems. Unlike batch learning techniques that often train a learning model on the entire training set, online learning methods can keep updating the learning model as each new data sample arrives within a short execution time. Sliding window (SW) and incremental learning methods are two potential solutions for IoT online learning [5]. SW methods retain a limited number of recent data samples and discard the old data samples using sliding windows. Thus, they have forgetting mechanisms that can reduce the storage requirements of IoT devices. Incremental learning is an online learning technique that uses every incoming data sample in the model training and updating process. It can retain the historical patterns and trends of the entire data in the learning model without storing all the data, and adapt to the new data patterns by partially updating the learning model (*e.g.*, replacing the nodes of Hoeffding trees).

8.2.2 Concept Drift Detection

"High variability" implies the concept drift issue associated with the non-stationary IoT data and the dynamic IoT environments. IoT streaming data is prone to many types of data distribution changes due to the dynamic IoT environments. For example, the physical events monitored by IoT sensors can change over time, and the sensing components age or need updates periodically. The corresponding IoT data distribution changes are named concept drift [6] [7].

The occurrence of concept drift could degrade the decision-making capabilities of IoT data analytics models, causing severe consequences in IoT systems. For example, the misleading decision-making process of IoT anomaly detection framework could significantly degrade their detection accuracy, making the IoT system vulnerable to various malicious cyber-attacks. To address concept drift, effective methods should be able to detect concept drift and adapt to the changes accordingly. Therefore, concept drift detection and adaptation are the two main challenges related to the high variability issue of IoT streaming data [7].

The first challenge of drift detection is the multiple types of concept drift, including gradual, sudden, and recurring drifts. The second challenge of drift detection is the various factors that can cause concept drift in IoT systems, including the IoT event factors (*e.g.*, system updates, IoT device replacement, and abnormal network events) and time-series factors (*e.g.*, seasonality and trends).

Window-based methods and performance-based methods are two potential solutions for drift detection [7]. Adaptive Windowing (ADWIN) is a common window-based method that uses adaptive sliding windows to detect concept drift based on the statistical difference between two adjacent subwindows [7]. Windowing methods are often fast and easy to implement, but they may lose certain useful historical information. On the other hand, Drift Detection Method (DDM) and Early Drift Detection Method (EDDM) are two popular performance-based drift detection methods that determine the occurrence of concept drift by monitoring the degree of model performance degradation [7]. Performance-based methods can effectively detect the drifts that cause model degradation, but they require the availability of ground-truth labels.

8.2.3 Concept Drift Adaptation

After drift detection, the observed drift should be effectively handled so that the learning model can adapt to the new data patterns. The concept drift adaptation challenge can be addressed using three potential solutions: adaptive algorithms, incremental learning, and ensemble learning methods [7].

Adaptive algorithms handle concept drift by fully retraining or altering the learning model on an altered dataset after detecting a drift. They are often the combinations of a ML model and a drift detection technique. For example, Losing *et al.* [10] proposed the self-adjusting memory with k-nearest neighbor (SAM-KNN) algorithm that uses KNN to train a learner and a dual-memory method to store both new and old useful data to fit current and previous concepts.

In incremental learning methods, the learning model is often partially updated when new samples arrive or drift is detected. Manapragada *et al.* [11] proposed an incremental learning method named Hoeffding Anytime Tree (HATT) that selects and splits nodes as soon as the confidence level is reached instead of identifying the best split in Hoeffding trees. This strategy makes HATT more efficient and accurate to adapt to concept drift.

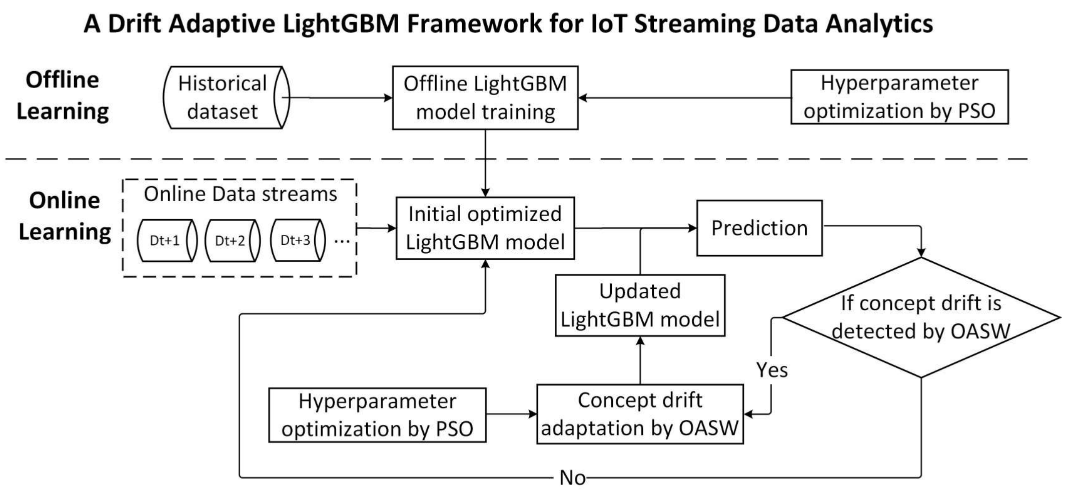
A Drift Adaptive LightGBM Framework for IoT Streaming Data Analytics
Offline Learning
Historical dataset
Offline LightGBM model training
Hyperparameter optimization by PSO
Online Learning
Online Data streams
Dt+1
Dt+2
Dt+3
...
Initial optimized LightGBM model
Prediction
Updated LightGBM model
If concept drift is detected by OASW
Yes
Hyperparameter optimization by PSO
Concept drift adaptation by OASW
No

Figure 8.2: The framework of the proposed drift adaptive IoT data analytics model

Ensemble learning is the technique of combining multiple base learners to construct an ensemble model with better generalization ability. Gomes *et al.* proposed Adaptive Random Forest (ARF) [12] and Streaming Random Patches (SRP) [13] ensemble algorithms that both use Hoeffding trees as the base learners and ADWIN as the drift detector. Ensemble models can retain historical and new data patterns in different base learners to address concept drift adaptation, but they often require high execution time.

8.3 Proposed System Framework

8.3.1 System Overview

The purpose of this work is to develop an adaptive IoT streaming data analytics framework that can address the time & memory constraints, as well as the concept drift issues described in Section 8.2. Fig. 8.2 demonstrates the overall architecture of the proposed adaptive LightGBM model. It comprises two stages: offline learning to obtain an initial trained model, and online training to detect IoT attacks in online data streams.

At the offline training stage, current IoT traffic data is collected to create a historical dataset. The historical dataset is then used to train an initial LightGBM model. Moreover, the hyper-parameters of the LightGBM model are tuned by PSO, a hyperparameter optimization (HPO) method, to construct the optimized LightGBM model.

At the online stage, the proposed system will process the data streams that are continuously generated over time. At the beginning of this stage, the initial LightGBM model obtained from the offline learning is used to process the data streams. If concept drift is detected in the new data streams by the proposed OASW method, the LightGBM model will then be retrained on the new concept data samples collected by the adaptive window of OASW to fit the current concept of the new IoT traffic data. Thus, the proposed system can adapt to the ever-changing new IoT traffic data patterns and maintain accurate cyber-attack detection.

8.3.2 Optimized LightGBM

LightGBM is a fast and high-performance ML model based on the ensemble of multiple decision trees [14]. Unlike many other ML algorithms, LightGBM is still an efficient model on large-sized and high-dimensional data, mainly due to its two utilized methods: gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB). GOSS is a down-sampling method that only retains the data samples with large gradients and removes the samples with small gradients during model training, which greatly reduces the time and memory usage. EFB method is used to reduce the feature size by bundling mutually exclusive features, which significantly reduces the training speed without losing important information.

The reasons for choosing LightGBM as the base model for IoT attack detection are as follows:

1. LightGBM is an ensemble model that has better generalizability and robustness than many other ML algorithms when working on non-linear and high-dimensional data.
2. By introducing GOSS and EFB, the time and space complexity of LightGBM has greatly reduced from $O(NF)$ to $O(N'B)$, where N and N' are the original and the reduced number of instances, respectively, and F and B are the original and bundled number of features, respectively.
3. LightGBM has built-in support for categorical data processing and feature selection, which simplifies the data pre-processing and feature engineering procedures.
4. LightGBM supports parallel execution by multi-threading, which substantially improves

the model efficiency.

To summarize, LightGBM can achieve both high accuracy and efficiency for data analytics, which is suitable for IoT systems with time and resource constraints.

To build an effective model with high prediction accuracy, the hyperparameters of the LightGBM model are tuned by PSO, a HPO technique. HPO is the process of automatically detecting the optimal hyperparameter values of a ML model to improve model performance [15].

Among HPO methods, PSO is a popular population-based optimization algorithm that detects the optimal values through communication and cooperation among individuals in a group [15]. In PSO, each particle in the swarm will continuously update its own position and velocity based on its current individual best position and the current global best position shared by other particles. Thus, the particles will move towards the candidate global optimal positions to detect the optimal solution.

The reasons for choosing PSO to tune hyperparameters are as follows [15]:

1. PSO is easy to implement and has a fast convergence speed.
2. PSO is faster than most other HPO methods, since it has a low computational complexity of $O(N \log N)$ and supports parallel execution.
3. PSO is effective for different types of hyperparameters and large hyperparameter space, like the hyperparameter space of LightGBM.

Five main hyperparameters, the number of leaves (*num_leaves*), the maximum tree depth (*max_depth*), the minimum number of data samples in one leaf (*min_data_in_leaf*), the learning rate(*learning_rate*), and the number of base learners (*n_estimators*), are tuned by the PSO method. By detecting the optimal values of these hyperparameters, an optimized LightGBM model can be obtained for accurate IoT data analytics.

8.3.3 OASW: Proposed Drift Adaptation Algorithm

The OASW method is proposed in this chapter to detect concept drift and adapt to the ever-changing IoT data streams for accurate analytics. OASW is designed based on the combination of ideas in sliding and adaptive window-based methods, as well as in performance-based methods. The complete OASW method is given by Algorithm 1. It has two main functions named

"*DriftAdaptation*" and "*HPO*". The "*DriftAdaptation*" function aims to detect concept drift in the streaming data and update the LightGBM model with the new concept samples for drift adaptation using the given hyperparameter values. The "*HPO*" function is used to tune and optimize the hyperparameters of the "*DriftAdaptation*" function using PSO.

In OASW, there are two types of windows: a sliding window used to detect concept drift and an adaptive window used to store new concept samples. The size of the sliding window is t, and the maximum size of the adaptive window is t'_{max}. Additionally, two thresholds, α and β, are used to indicate the warning level and the actual drift level for concept drift detection.

The main procedures of OASW are as follows. For each incoming data sample i in the new stream, its sliding window, W_i, contains $(i-t)_{th}$ to i_{th} samples. The accuracies of W_i and W_{i-t} (indicating the current and last complete windows, respectively) are calculated and compared. If the accuracy of the sliding window drops α percent from timestamp $i-t$ to i, the warning level is reached, and the adaptive window starts to collect incoming data samples as new concept samples (lines 5-11). After that, if the sliding window accuracy keeps dropping β percent to the drift level, a drift alarm will be triggered, and the old learner will be updated by retraining on the new concept samples collected in the adaptive window (lines 12-17).

Moreover, to obtain a robust and stable learner, the adaptive window will keep collecting new samples until one of the following two conditions are met: 1) The new concept accuracy drops to the warning level α when compared to the drift starting point, indicating the current learner is incapable of processing the new concept and requires updating. 2) The size of the adaptive window reaches t'_{max}, which ensures that the memory and real-time requirements are met.

Then, the learner will be updated again on the samples in the adaptive window to become a more robust learner, and the system will change to the normal state for the next potential drift detection (lines 25-34).

On the other hand, if in the warning state, the sliding window accuracy stops dropping, or even increases to the normal level, it will be seen as a false alarm. The adaptive window will then be released, and the system will change back to the normal state to monitor potential new drift (lines 18-23).

In the OASW algorithm, four parameters, α, β, t, and t'_{max}, are the critical hyperparameters

Algorithm 4: Optimized Adaptive and Sliding Windowing (OASW)

```
Input:
    $Stream$: a data stream,
    $\alpha, \beta$: the warning and drift thresholds,
    $t$: the fixed size of a sliding window,
    $t'_{max}$: the maximum size of the adaptive window,
    $Classifier$: a classifier trained on offline dataset,
    $Space$: hyperparameter configuration space,
    $MaxTime$: the maximum hyperparameter search times.
Output:
    $HP_{opt}$: the detected optimal hyperparameter values,
    $MaxAcc$: the average overall accuracy.
1  Function DriftAdaptation($Stream, Classifier, \alpha, \beta, t, t'_{max}$):
2      $W' \leftarrow \emptyset$;                                // Initialize the adaptive window
3      $State \leftarrow 0$;                // An indicator of normal, drift, and warning states
4      for all samples $x_i \in Stream$ do
5          $W_i \leftarrow$ a sliding window of the last $t$ samples of $x_i$;
6          $AccWin_i \leftarrow accuracy(W_i)$;                          // Current window accuracy
7          $AccWin_{i-t} \leftarrow accuracy(W_{i-t})$;                     // Last window accuracy
8          if $(Indicator == 0)\&\&(AccWin_i < \alpha * AccWin_{i-t})$ then   // New window accuracy drops from the normal to warning
9              $W' \leftarrow W' \cup \{x_i\}$                      // $W'$ starts to collect new samples
10             $State \leftarrow 1$;                                        // Warning occurs
11         end
12         if $State == 1$ then                                           // In a warning state
13             $t' \leftarrow Size(W')$;
14             if $AccWin_i < \beta * AccWin_{i-t}$ then          // New window accuracy drops to drift level
15                 $State \leftarrow 2$;                                      // Drift occurs
16                 $f \leftarrow i$;            // Obtain the first new concept window accuracy as a baseline
17                 $Classifier' \leftarrow$ Retrain $Classifier$ on $W'$;  // Retrain the classifier on new concept samples
18             else if $(AccWin_i \geq \alpha * AccWin_{i-t}) || t' == t'_{max})$ then   // False alarm (the warning state changes back to normal or
                 stay constant)
19                 $W' \leftarrow \emptyset$;                              // Release the adaptive window
20                 $State \leftarrow 0$                                  // Change to a normal state
21             else                                                  // Still in the warning state
22                 $W' \leftarrow W' \cup \{x_i\}$                    // $W'$ keeps collecting new samples
23             end
24         end
25         if $State == 2$ then                                             // In a drift state
26             $t' \leftarrow Size(W')$;
27             if $(AccWin_i < \alpha * AccWin_{f+t}) || t' == t'_{max})$ then   // When new concept accuracy drops to the warning level or
                 sufficient new concept samples are collected
28                 $Classifier' \leftarrow$ Retrain $Classifier'$ on $W'$;          // Construct a robust classifier
29                 $W' \leftarrow \emptyset$;                           // Release the new concept window
30                 $State \leftarrow 0$                                  // Change to a normal state
31             else
32                 $W' \leftarrow W' \cup \{x_i\}$                    // $W'$ keeps collecting new samples
33             end
34         end
35     end
36     return $AvgAcc$;                                                  // The average accuracy
37 Function HPO($Stream, Space, MaxTime$):
38     $MaxAcc \leftarrow 0$;
39     for $j \leftarrow 1$ to $MaxTime$ do
40         $\alpha, \beta, t, t'_{max} \leftarrow SelectConfiguration(Space)$;      // Search optimal HP values by PSO
41         $Acc \leftarrow DriftAdaptation(Stream, Classifier, \alpha, \beta, t, t'_{max})$ ;   // Evaluate the current HP configuration
42         if $MaxAcc < Acc$ then
43             $MaxAcc \leftarrow Acc$ ;
44             $HP_{opt} \leftarrow \alpha, \beta, t, t'_{max}$ ;      // Update accuracy and optimal hyperparameter values
45         end
46     end
47     return $MaxAcc, HP_{opt}$;                              // The best accuracy & hyperparameters
```

that have a direct impact on the performance of the OASW model. Therefore, PSO is used to tune these four hyperparameters to obtain the optimized adaptive learner, since PSO is efficient for both continuous and discrete hyperparameters to which the hyperparameters of OASW belong.

To implement OASW, the "*HPO*" function is given the configuration space of the four hyperparameters. PSO will then detect the optimal hyperparameter combination that returns the highest overall accuracy (lines 37-47). The detected optimal hyperparameters will then be given to the "*DriftAdaptation*" function to construct the optimized model for accurate IoT data analytics.

OASW has a training complexity of $O(NM)$, and a low run-time and space complexity of $O(N)$, where N is the number of instances, and M is the maximum hyperparameter search times in PSO.

Compared to other concept drift handling methods, the proposed OASW method has the following advantages:

1. Unlike many other methods that focus on either drift detection or drift adaptation, OASW has both functionalities because it uses a sliding window for drift detection and an adaptive window for drift adaptation.
2. OASW has better generalization capability and adaptability than most other approaches when applied to different datasets or tasks since it can automatically tune the hyperparameters to fit specific datasets by using PSO.
3. OASW detects concept drift and updates the learning model mainly based on model performance degradation, which ensures that the learner is only updated when necessary.
4. OASW makes a trade-off between the model accuracy and computational complexity by using the adaptive window to collect sufficient new concept samples while removing previous concept samples.

8.4 Performance Evaluation

8.4.1 Experimental Setup

The experiments were implemented in Python by extending the Scikit-multiflow [16] framework on a Raspberry Pi 3 machine with a BCM2837B0 64-bit CPU and 1 GB of memory, representing a real IoT device. Two IoT anomaly detection datasets, IoTID20 [8] and NSL-KDD [9], are used to evaluate the proposed framework as examples for IoT classification problems. The proposed framework can be applied to other IoT classification problems using the same procedures. Additionally, the proposed framework can also be applied to general IoT regression problems by only changing the performance metric to a regression metric (*e.g.*, negative mean squared error or negative mean absolute error).

IoTID20 is a novel IoT traffic dataset with unbalanced data samples (94% normal samples versus 6% abnormal samples) for abnormal IoT device detection, while NSL-KDD is a balanced benchmark dataset for concept drift and network intrusion detection. They are both widely used in many research projects to validate anomaly detection models in IoT environments. Using these two datasets enables the model evaluation on both balanced and unbalanced datasets. For the purpose of this work, a reduced IoTID20 dataset that has 62,578 records and a reduced NSL-KDD dataset that has 35,140 records are used. The IoTID20 dataset was randomly sampled based on the timestamps (1 data point per 10 timestamps). For the NSL-KDD dataset, it is known that there is a sudden drift from the training set to the test set, but there is no drift in the training set [12]; hence, for a clear comparison between the two concepts, the last 10% of the training set and the entire test set is used for model evaluation.

The proposed method aims to distinguish attack samples from normal samples in IoT systems, so the datasets are utilized as binary classification datasets. Hold-out and prequential validation methods are used for model evaluation. For hold-out validation, the first 10% of the data samples in each dataset are used as the training set for offline/initial model training, and the last 90% are used as the test set for online learning. Prequential validation, or named test-and-train validation, is only used in online learning, where each input instance is first used to test the model and then used for model updating. To evaluate the performance of the proposed framework, multiple metrics, including accuracy, precision, recall, and f1-score, are used in

Table 8.2: Hyperparameter Configuration of LightGBM and OASW

Model	Hyper-parameter	Search Range	Optimal Value (IoTID20)	Optimal Value (NSL-KDD)
LightGBM	*n_estimators*	[50, 500]	300	300
	max_depth	[5, 50]	40	42
	learning_rate	(0, 1)	0.56	0.81
	num_leaves	[100, 2000]	200	100
	min_data_in_leaf	[10, 50]	35	45
OASW	α	(0.95, 1)	0.999	0.978
	β	(0.90, 1)	0.990	0.954
	t	[100, 1000]	300	350
	t'_{max}	[500, 5000]	1000	3100

the experiments.

For model comparison, four methods introduced in Section 8.2.3, including SAM-KNN [10], HATT [11], ARF [12], and SRP [13], are also evaluated on the two considered datasets. These four methods are state-of-the-art drift adaptation approaches that have proven effective in many drift datasets and applications. All the drift adaptation methods were implemented using the default parameter values set in Scikit-multiflow or the original papers.

8.4.2 Experimental Results and Discussion

To obtain optimized LightGBM and OASW models, their hyperparameters are automatically tuned by PSO. The initial hyperparameter search range and detected hyperparameter values of the LightGBM and OASW models on the two considered datasets are shown in Table 8.2. After using PSO, the optimal hyperparameter values were assigned to the proposed models to construct optimized models for IoT attack detection.

Figures 8.3 & 8.4 and Tables 8.3 & 8.4 show the accuracy comparison of the proposed OASW & LightGBM model against the state-of-the-art drift adaptive methods introduced in Section 8.2.3. It can be seen in Tables 8.3 & 8.4 that the proposed adaptive model outperforms all other methods in terms of accuracy on the two datasets. On the IoTID20, as shown in Fig. 8.3 and Table 8.3, the proposed method can achieve the highest accuracy of 99.92% among all implemented models by adapting a slight concept drift detected at point 13408. Without drift adaptation, the offline LightGBM model has a slightly lower accuracy of 99.78%. The accuracies of the other four state-of-the-art methods are also lower than the accuracy of our

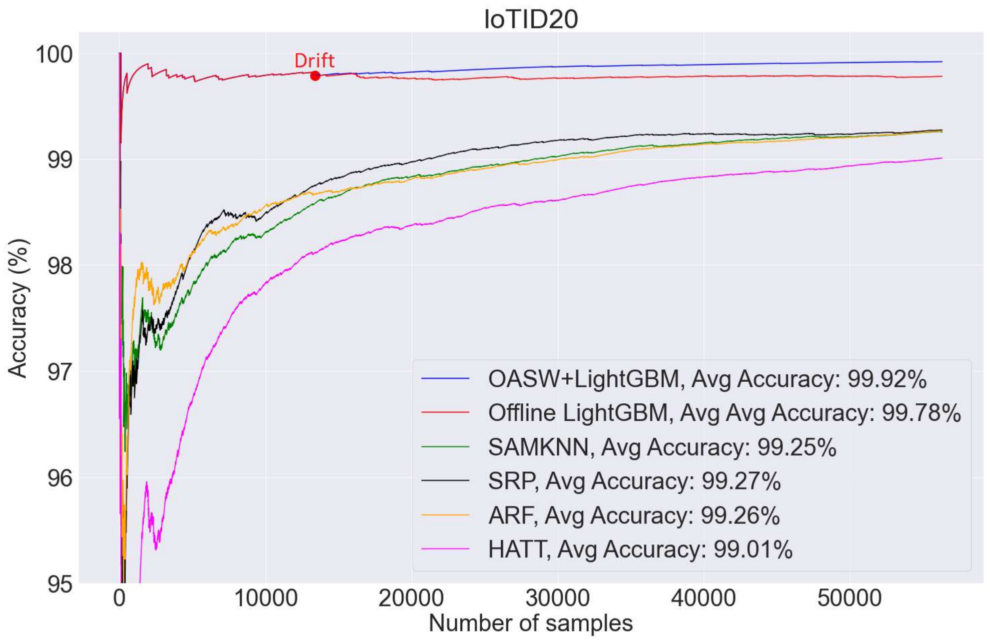

Figure 8.3: Accuracy comparison of different drift adaptation methods on the IoTID20 dataset.

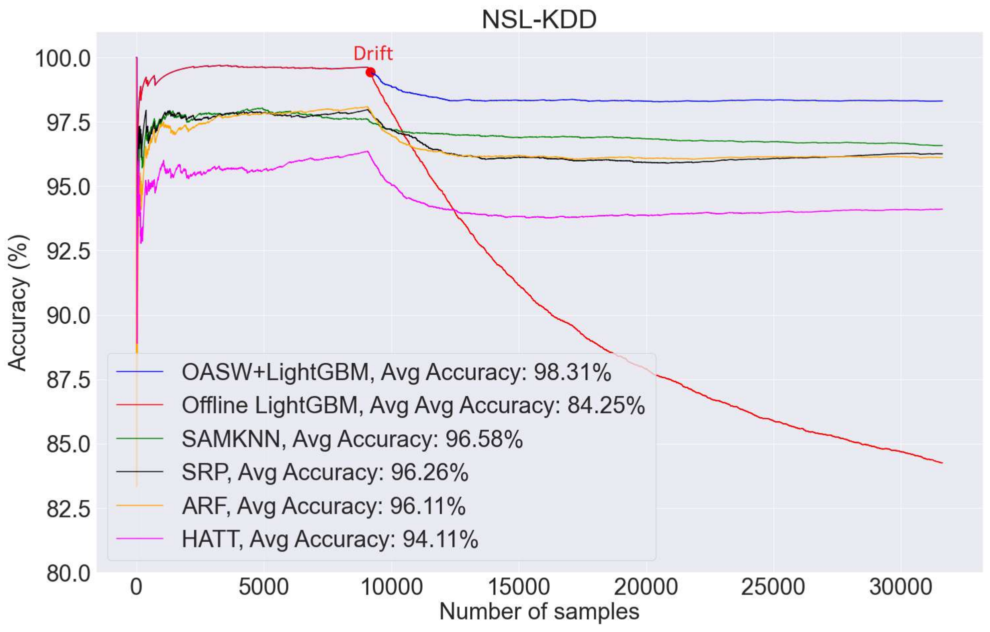

Figure 8.4: Accuracy comparison of different drift adaptation methods on the NSL-KDD dataset.

proposed method (99.01% - 99.27%).

For the NSL-KDD dataset, there is a severe drift at the beginning of the test set [12]. As shown in Fig. 8.4 and Table 8.4, by adapting to the sudden drift detected at point 9183, the proposed method can achieve the highest accuracy of 98.31%, while the offline LightGBM model's

Table 8.3: Performance Comparison of Drift Adaptation Methods on The IoTID20 Dataset

Method	**IoTID20 Dataset**					
	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Avg Test Time (ms)**	**Memory Usage (MB)**
SAM-KNN [10]	99.25	99.40	99.80	99.60	43.4	179.7
HATT [11]	99.01	99.21	99.74	99.47	4.8	0.8
ARF [12]	99.26	99.37	99.85	99.61	5.9	0.9
SRP [13]	99.27	99.35	99.88	99.61	67.8	3.8
Offline LightGBM	99.78	99.82	99.95	99.88	0.6	1.9
Proposed OASW & LightGBM	99.92	99.93	99.98	99.96	7.8	0.4

Table 8.4: Performance Comparison of Drift Adaptation Methods on The NSL-KDD Dataset

Method	**NSL-KDD Dataset**					
	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Avg Test Time (ms)**	**Memory Usage (MB)**
SAM-KNN [10]	96.58	96.13	97.6	96.86	21.6	137.2
HATT [11]	94.11	94.26	94.86	94.56	3.5	7.3
ARF [12]	96.11	95.81	97.05	96.42	5.6	6.8
SRP [13]	96.26	96.21	96.88	96.55	35.7	15.3
Offline LightGBM	84.25	97.75	72.51	83.26	0.3	3.7
Proposed OASW & LightGBM	98.31	98.57	98.30	98.43	9.1	1.8

accuracy drops significantly to only 84.25% without drift adaptation. This emphasizes the development of drift adaptation methods. The other four compared methods, SAM-KNN, SRP, ARF, and HATT, have much lower accuracy than the proposed method (94.11% - 96.58%).

The average online prediction time for each instance and the total memory usage are also calculated to evaluate the proposed method for real-time online learning considering the time and memory limitations of IoT devices, as shown in Tables 8.3 and 8.4. Among the six methods implemented, the memory usage of the proposed method is the smallest on both datasets, because the proposed model updates itself on a relatively small subset obtained by OASW instead of on the entire streaming data. The average prediction time of the proposed model for each instance on the Raspberry Pi 3 machine is only 7.8 ms and 9.1 ms on the two used datasets, much shorter than SAM-KNN and SRP. This is mainly due to the sliding window strategy and the efficiency of LightGBM. The prediction time of ARF and HATT for each instance is shorter than the proposed model, but their accuracy is much lower than the proposed method. Therefore, the proposed method still performs the best among the drift methods in terms of the trade-off between accuracy and efficiency. Moreover, the average execution time of the proposed framework for each sample on a desktop machine with an i7-8700 processor & 16 GB of memory and a powerful Google Colaboratory Cloud machine with a Xeon processor is only 1.3 ms and 0.9 ms, respectively. The high data processing speed on powerful cloud machines shows the feasibility of implementing the proposed framework in real-time environments.

In conclusion, the experimental results show the effectiveness and robustness of the proposed adaptive LightGBM model for IoT streaming data analytics.

8.5 Conclusion

The increasing popularity of IoT systems has brought great convenience to humans, but it also increases the difficulty to collect and process large volumes of IoT data collected from various sensors in IoT environments. Compared to conventional static data, IoT data is often big streaming data under non-stationary and rapidly-changing environments. Adaptive ML methods are appropriate solutions since they have the capacity to process constantly evolving IoT data streams by adapting to potential concept drifts. In this chapter, we proposed the adaptive LightGBM model for IoT data analytics with high accuracy and low time and memory usage. Through the integration of our proposed novel drift-handling algorithm (*i.e.*, OASW), an ensemble ML algorithm (*i.e.*, LightGBM), and a hyperparameter method (*i.e.*, PSO), the proposed model has the capacity to automatically adapt to the ever-changing data streams of dynamic IoT systems. The proposed method is evaluated and discussed by conducting experiments on two public IoT anomaly detection datasets, IoTID20 and NSL-KDD. Based on the comparison with several state-of-the-art drift adaptation methods, the proposed system is able to detect IoT attacks and adapt to concept drift with higher accuracies of 99.92% and 98.31% than the other methods on the IoTID20 and NSL-KDD datasets, respectively.

Bibliography

[1] L. Yang and A. Shami, “A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams,” *IEEE Internet Things Mag.*, vol. 4, no. 2, pp. 96–101, 2021.

[2] T. Stack, “Iot Data Continues To Explode Exponentially. Who Is Using That Data And How?,” *Cisco Blogs*, 2020.

[3] M. Injadat, A. Moubayed, and A. Shami, “Detecting Botnet Attacks in IoT Environments: An Optimized Machine Learning Approach,” in *IEEE-ICM2020*, 2020, pp. 1–4.

[4] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, “Tree-based Intelligent Intrusion Detection System in Internet of Vehicles,” *proc. 2019 IEEE Glob. Commun. Conf.*, pp. 1–6, Hawaii, USA, 2019.

[5] A. A. Cook, G. Misirli, and Z. Fan, “Anomaly Detection for IoT Time-Series Data: A Survey,” *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6481–6494, 2020, doi: 10.1109/JIOT.2019.2958185.

[6] N. L. A. Ghani, I. A. Aziz, and M. Mehat, “Concept Drift Detection on Unlabeled Data Streams: A Systematic Literature Review,” in *2020 IEEE Conference on Big Data and Analytics (ICBDA)*, 2020, pp. 61–65, doi: 10.1109/icbda50157.2020.9289802.

[7] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under Concept Drift: A Review,” *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 12, pp. 2346–2363, 2019, doi: 10.1109/TKDE.2018.2876857.

[8] I. Ullah and Q. H. Mahmoud, “A Scheme for Generating a Dataset for Anomalous Activity Detection in IoT Networks,” in *Advances in Artificial Intelligence*, 2020, pp. 508–520.

[9] J. Liu, B. Kantarci, and C. Adams, “Machine learning-driven intrusion detection for Contiki-NG-based IoT networks exposed to NSL-KDD dataset,” *WiseML 2020 - Proc. 2nd ACM Work. Wirel. Secur. Mach. Learn.*, pp. 25–30, 2020, doi: 10.1145/3395352.3402621.

[10] V. Losing, B. Hammer, and H. Wersing, “KNN classifier with self adjusting memory for heterogeneous concept drift,” *Proc. - IEEE Int. Conf. Data Mining, ICDM*, vol. 1, pp. 291–300, 2017, doi: 10.1109/ICDM.2016.141.

[11] C. Manapragada, G. I. Webb, and M. Salehi, “Extremely fast decision tree,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 1953–1962, 2018, doi: 10.1145/3219819.3220005.

[12] H. M. Gomes *et al.* , “Adaptive random forests for evolving data stream classification,” *Mach. Learn.*, vol. 106, no. 9–10, pp. 1469–1495, 2017, doi: 10.1007/s10994-017-5642-8.

[13] H. M. Gomes, J. Read, and A. Bifet, “Streaming random patches for evolving data stream classification,” *Proc. - IEEE Int. Conf. Data Mining, ICDM*, vol. 2019-November, no. Icdm, pp. 240–249, 2019, doi: 10.1109/ICDM.2019.00034.

[14] D. Jin, Y. Lu, J. Qin, Z. Cheng, and Z. Mao, “SwiftIDS: Real-time intrusion detection system based on LightGBM and parallel intrusion detection mechanism,” *Comput. Secur.*, vol. 97, p. 101984, 2020, doi: 10.1016/j.cose.2020.101984.

[15] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020, doi: https://doi.org/10.1016/j.neucom.2020.07.061.

[16] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, “Scikit-multiflow: A Multi-output Streaming Framework,” *J. Mach. Learn. Res.*, vol. 19, pp. 1–5, 2018, doi: 10.5555/3291125.3309634.

Chapter 9

PWPAE: An Ensemble Framework for Concept Drift Adaptation in IoT Data Streams

9.1 Introduction

With the rapid development of the Internet of Things (IoT), IoT devices have provided numerous new capabilities and services to people. The IoT has been applied to many areas of daily life, including smart cities, smart homes, smart cars, intelligent transportation systems (ITS), smart healthcare, smart agriculture, and so on [2] - [4]. It is estimated that one trillion devices or IP addresses will be connected to IoT networks by 2022 [2].

Despite the various new functionalities provided by IoT devices, the deployment of IoT devices has led to many security risks [5]. Current IoT systems are vulnerable to most existing cyber-threats, since many IoT device manufacturers prioritize low-cost and technical functionalities over security mechanisms. In 2018, cyber-attacks against IoT devices increased by 215.7% [5]. Various types of common network attacks, like distributed denial of service (DDoS), botnets, and phishing attacks, can be launched in IoT systems [6] - [8]. These threats affect both IoT devices and the entire internet ecosystem, since a single cyber-attack may cause

A version of this chapter has been published in IEEE GlobeCom 2021 [1].

a large-scale IoT system failure.

IoT traffic analysis has been widely used in IoT systems to detect compromised IoT devices and malicious cyber-attacks [5]. The behaviors of IoT devices and can be classified as normal or abnormal by supervised machine learning (ML) algorithms based on the characteristics of IoT traffic data [9]. However, real-world IoT traffic data is usually dynamic data streams that are generated continuously in non-stationary IoT environments. Moreover, the changing behaviors of IoT attacks make it difficult to adapt to the ever-changing environments. Hence, the underlying distribution of the IoT traffic data often changes unpredictably over time, known as concept drift [11] [12]. Traditional static ML models are often incapable of reacting to data distribution changes. These changes can have a direct impact on the model prediction performance since the patterns in the trained ML models will become invalid in future predictions. Effective ML-based IoT anomaly detection systems should have the adaptability to self-calibrate to the new concepts and cyber-attack patterns to ensure robustness [11].

In this work, a drift adaptive framework, named Performance Weighted Probability Averaging Ensemble (PWPAE) framework, is proposed for effective IoT anomaly detection. This framework can be deployed on IoT cloud servers to process the big data streams transmitted from the IoT end devices through wireless communication strategies, as shown in Fig. 9.1. The proposed framework is an ensemble learning framework that uses the combinations of two popular drift detection methods, adaptive windowing (ADWIN) [13] and drift detection method (DDM) [14], and two state-of-the-art drift adaptation methods, adaptive random forest (ARF) [15] and streaming random patches (SRP) [16], to construct base learners. The base learners are weighted according to their real-time performance and integrated to construct a robust anomaly detection ensemble model with improved drift adaptation performance.

The main contributions of this chapter are as follows:

1. It investigates concept drift adaptation methods.
2. It proposes a novel drift adaptation method named PWPAE to address the performance limitations of current concept drift methods.
3. It evaluates the proposed PWPAE framework on two public IoT cyber-security datasets,

Code is available at: https://github.com/Western-OC2-Lab/PWPAE-Concept-Drift-Detection-and-Adaptation

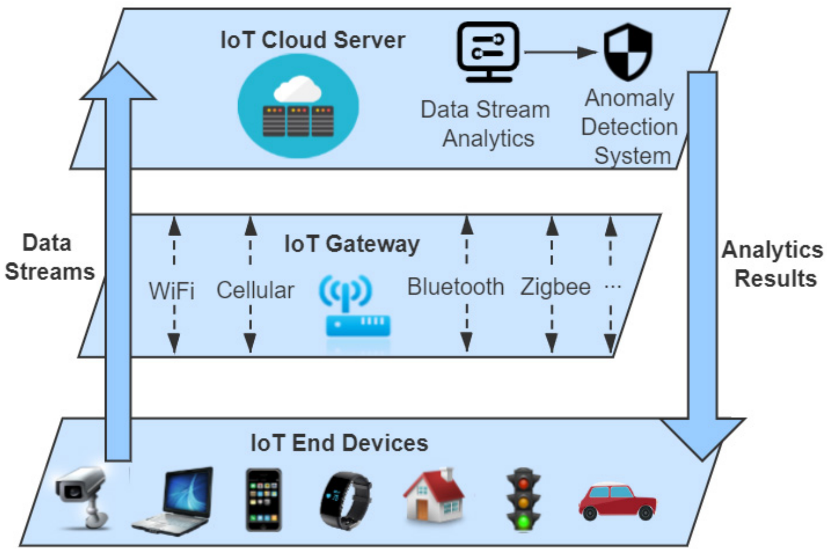

Figure 9.1: The architecture of IoT data stream analytics.

IoTID20 [6] and CICIDS2017 [17], for IoT anomaly detection use cases.

The remainder of this chapter is organized as follows: Section 9.2 provides a literature review of state-of-the-art IoT anomaly detection and concept drift adaptation methods. Section 9.3 describes the proposed PWPAE drift adaptation framework. Section 9.4 presents and discusses the experimental results. Section 9.5 concludes the chapter.

9.2 Related Work

In this section, the existing works on IoT traffic analysis and concept drift adaptation are reviewed.

9.2.1 IoT Traffic Analysis

Several existing works focused on IoT anomaly detection using traffic data. Injadat *et al.* [5] proposed an optimized ML approach that uses decision tree and Bayesian optimization with Gaussian Process (BO-GP) algorithms to detect botnet attacks in IoT systems. The proposed algorithm achieves 99.99% accuracy on Bot-IoT-2018 dataset. Ullah *et al.* [6] proposed a novel botnet IoT dataset for IoT network anomaly detection and evaluated the performance of seven basic ML models on this dataset. Among the ML algorithms, the random forest and ensemble models achieve better performance. Yang *et al.* [7] proposed a tree-based stacking algorithm for network traffic analysis on the Internet of Vehicles (IoV) environments. The proposed stacking method achieves high detection accuracy on the IoV and CICIDS2017 datasets.

The above methods achieve high accuracy for cyber-attack detection in IoT systems. However, they are static ML models designed for offline learning and do not have the adaptability to online changes of IoT data, making them ineffective to be deployed in real-world IoT systems.

9.2.2 Concept Drift Methods

Due to the non-stationary IoT environments, IoT streaming data analysis often faces concept drift challenges that the data distributions change over time. The occurrence of concept drift issues often degrades the performance of IoT anomaly detection models, causing severe security issues. Concept drifts can be classified as sudden and gradual drifts, according to the data distribution changing speed [18]. To handle concept drift, an effective anomaly detection model should accurately detect the drifts and quickly adapt to the detected drifts to maintain high prediction accuracy [19].

Concept Drift Detection

Drift detection is the first procedure to handle concept drift. ADWIN and DDM are the two most common concept drift detection techniques. ADWIN [13] is a distribution-based method that uses an adaptive sliding window to detect concept drift based on data distribution changes. ADWIN identifies concept drift by calculating and analyzing the average of certain statistics over the two sub-windows of the adaptive window. The occurrence of concept drift is indicated by a large difference between the averages of the two sub-windows. Once a drift point is detected, all the old data samples before that drift time point are discarded [19].

ADWIN can effectively detect gradual drifts since the sliding window can be extended to a large-sized window to identify long-term changes. However, the mean value is not always an effective measure to characterize changes.

Drift Detection Method (DDM) is a popular model performance-based method that defines two thresholds, a warning level and a drift level, to monitor model's error rate and standard deviation changes for drift detection [14]. In DDM, the occurrence of concept drift is indicated by a significant increase of the sum of model's error rate and standard deviation. DDM is easy to implement and can avoid unnecessary model updates since a learner will only be updated when its performance degrades significantly. DDM can effectively identify sudden drift, but its response time is often slow for gradual drifts. This is because a large number of data samples need to be stored to reach the drift level of a long gradual drift, causing memory overflows [20].

Concept Drift Adaptation

After drift detection, an appropriate drift adaptation algorithm should be implemented to deal with the detected drifts and maintain high learning performance. Current drift adaptation methods can be classified into two main categories: incremental learning methods and ensemble methods.

Incremental learning is to learn samples one by one in chronological order to partially update the learning model. The Hoeffding tree (HT) is a type of decision tree (DT) that uses the Hoeffding bound to incrementally adapt to data streams [20]. Compared to a DT that chooses the best split, the HT uses the Hoeffding bound to calculate the number of necessary samples to select the split node. Thus, the HT can update its node to adapt to newly incoming samples. However, the HT does not have mechanisms to address specific types of drift. The Extremely Fast Decision Tree (EFDT) [22], also named Hoeffding Anytime Tree (HATT), is an improved version of the HT that splits nodes as soon as it reaches the confidence level instead of detecting the best split in the HT. This splitting strategy makes the EFDT adapt to concept drifts more accurately than the HT, but its performance still needs improvement.

To achieve better concept drift adaptation, ensemble learning methods have been proposed to construct robust learners for data stream analytics. Ensemble methods can be further classified as block-based ensembles and online ensembles [21]. Block-based ensembles split the data streams into fixed-size blocks and train a base learner on each block. When a new block arrives, the base learners will be evaluated and updated. Block-based ensembles have accurate reactions to gradual drifts, but often delay reacting to sudden drifts. Another difficulty of block-based ensemble methods is to choose an appropriate block size to achieve a trade-off between the drift reaction speed and base learners' learning performance [21].

Streaming Ensemble Algorithm (SEA), Accuracy Weighted Ensemble (AWE), and Accuracy Updated Ensemble (AUE) are three common block-based ensembles [19] [23]. Among the block-based ensembles, AUE is usually the best performing method. In AUE, all base learners are incrementally updated with a portion of samples from each new chunk. Moreover, AUE assigns weights to base learners using non-linear error functions for performance enhancement. Experimental studies showed that the performance of AUE constructed with HTs is better than other chunk-based ensembles, like AWE and SEA [23].

Online ensembles aim to integrate multiple incremental learning models, like HTs, to further improve the learning performance. Gomes *et al.* [15] proposed the adaptive random forest (ARF) algorithm that uses HTs as base learners and ADWIN as the drift detector for each tree. Through the drift detection process, the poor-performing base trees are replaced by new trees to fit the new concept. ARF often performs better than many other methods since the random forest is also a well-performing ML algorithm. Additionally, ARF has an effective resampling technique and the adaptability to different types of drifts. Gomes *et al.* [16] also proposed a novel adaptive ensemble method named Streaming Random Patches (SRP) for streaming data analytics. SRP combines the random subspace and online bagging method to make predictions. SRP uses the similar technology of ARF, but it uses the global subspace randomization strategy, instead of the local subspace randomization technique used by ARF. The global subspace randomization is a more flexible method that improves the diversity of base learners. The prediction accuracy of SRP is often slightly better than ARF, but the execution time is often longer. Leverage bagging (LB) [24] is another popular online ensemble that uses bootstrap samples to construct base learners. It uses Poisson distribution to increase the data diversity and leverage the bagging performance. LB is easy to implement, but often performs worse than SRP and ARF.

Although there are many existing concept drift adaptation methods, they have performance limitations in terms of prediction accuracy and drift reaction speed. Incremental learning methods are often underperforming due to their low model complexity and limited drift adaptability, while the drift reaction speed and block size determination are two major challenges for block-based ensembles. Online ensembles, like ARF and SRP, often perform better than incremental

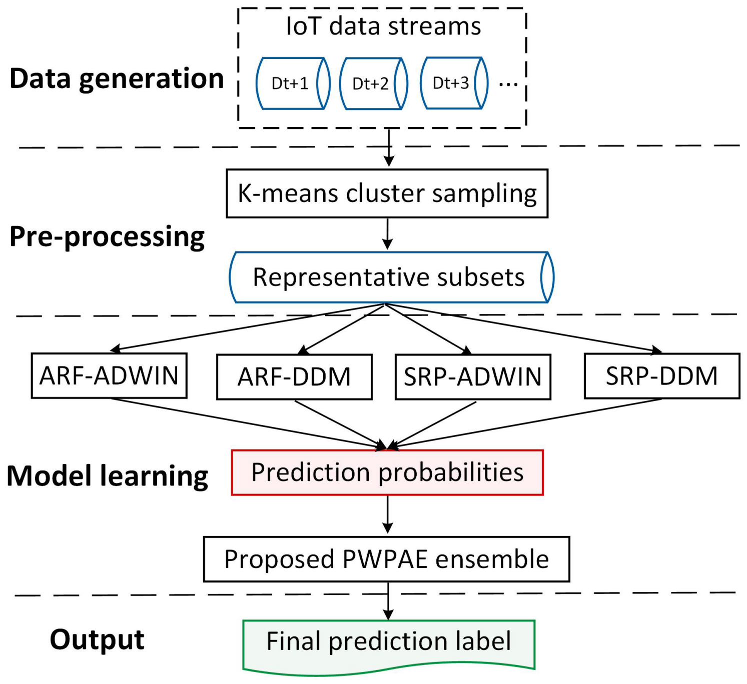

Figure 9.2: The proposed drift adaptive IoT anomaly detection framework.

learning and block-based ensemble methods, but they introduce additional randomness in their model construction process due to their randomization strategies, causing unstable learning models. Thus, this chapter aims to propose a stable and robust online ensemble model with improved drift adaptation performance.

9.3 Proposed Framework

9.3.1 System Overview

Figure 9.2 provides an overview of the proposed IoT anomaly detection framework. The main procedures are as follows. Firstly, incoming IoT data streams are sampled to generate a highly-representative subset using the k-means cluster sampling method. Secondly, four concept drift adaptation methods (ARF-ADWIN, ARF-DDM, SRP-ADWIN, and SRP-DDM) are constructed as the base learners for initial anomaly detection and drift adaptation. After that, an ensemble model is constructed by integrating the prediction probabilities of the four base learners based on the proposed PWPAE framework. Lastly, the ensemble model is deployed and can effectively detect cyber-attacks and adapt to concept drifts.

9.3.2 Data Pre-Processing

As IoT data streams are usually large amounts of data that are continuously generated, using all the data samples for learning model development is often infeasible and unnecessary. Thus, an effective data sampling method should be implemented to select highly-representative data samples.

In the proposed system, the k-means-based cluster sampling method is used to obtain a representative subset [8]. The k-means sampling method can group the data samples into multiple clusters and select a proportion of samples from each cluster, as the data samples in the same cluster have similar characteristics. Due to the large size of IoT traffic datasets, 1% of the original data samples are selected by the k-means cluster sampling method to evaluate the proposed framework. The size of the generated subset can vary, depending on the IoT data generation speed and the computational power of server machines. Compared to other sampling methods, k-means cluster sampling can generate a high-quality and highly-representative subset because the discarded data points are mostly redundant data.

9.3.3 Drift Adaptation Base Learner Selection

To construct a robust ensemble model, the combination of two basic drift detection methods (ADWIN & DDM) and two state-of-the-art drift adaptation methods (ARF & SRP) described in Section 9.2.2 are used for base learners' construction. Thus, the four base learners are ARF-ADWIN, ARF-DDM, SRP-ADWIN, SRP-DDM. The reasons for choosing them as base learners are as follows [19] [23]:

1. They are all online ensemble models with strong adaptability to concept drift. All of them are constructed with multiple HTs, an effective incremental learning base model. Thus, each base learner already has a stronger data stream analysis capability than most existing drift adaptation methods.
2. ARF and SRP are both state-of-the-art drift adaptation methods whose performance has been proven to be better than other existing drift adaptation methods by experimental

studies in [15] [16].

3. Unlike block-based ensembles (*e.g.*, SEA, AWE, and AUE), ARF and SRP are both online ensembles that do not require the tuning of data chunk sizes. Choosing an inappropriate chunk size often results in additional execution time or drift detection delays.
4. As described in Section 9.2.2, DDM works well on sudden drift detection, while ADWIN has a stronger ability to detect gradual drift. Using both drift detection methods enables the proposed ensemble model to detect both drift types effectively.
5. An effective ensemble model should ensure a high diversity of base learners to increase the chance of learning performance improvement; otherwise, a low diversity ensemble often shows very similar performance to certain base learners. Although ARF and SRP both use Hoeffding trees as base learners, their construction methods are largely different (local subspace randomization versus global subspace randomization), which increases the randomness and diversity in the model construction process. Hence, a more robust ensemble model with a high diversity can be obtained.

9.3.4 Drift Adaptation Ensemble Framework: PWPAE

In this chapter, a novel ensemble strategy, named Performance Weighted Probability Averaging Ensemble (PWPAE), is proposed to integrate the base learners for IoT data stream analytics. Unlike the pre-defined or static weights used by many existing ensemble techniques, PWPAE assigns dynamic weights to base learners according to their real-time performance. Assuming a data stream $D = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_n}, y_n)\}$, and the target variable has c different classes, $y \in 1, \ldots, c$, for each input data $\boldsymbol{x}$, the target class estimated by PWPAE can be denoted by:

$$\hat{y} = \underset{i \in \{1,\cdots,c\}}{\operatorname{argmax}} \frac{\sum_{j=1}^{k} w_j p_j \left(y = i \mid L_j, \boldsymbol{x}\right)}{k} \tag{9.1}$$

where L_j is a base learner, k is the number of base learners, and $k = 4$ in the proposed framework; $p_j(y = i|L_j, \boldsymbol{x})$ indicates the prediction probability of a class value i on the data sample $\boldsymbol{x}$ using the j_{th} base learner L_j; w_j is the weight of each base learner L_j.

After each data sample is processed, the current real-time error rate is calculated by dividing the total number of misclassified samples by the total number of processed samples. The weight of each base learner, w_j, is calculated using the reciprocal of its real-time error rate, which can be represented by:

$$w_j = \frac{1}{Error_{rt} + \epsilon} \tag{9.2}$$

where ϵ is a small constant value to avoid the denominator being equal to 0. If $Error_{rt} \rightarrow 0$ for a base learner, an extremely large weight of $1/\epsilon$ is given to this base learner, as it can already make predictions perfectly.

The weighting function can be considered an improved version of the weighting function in the AUE algorithm described in Section 9.2.2. In the proposed PWPAE model, the real-time error rates, instead of the mean square error rates of data blocks used in AUE, are used to calculate the weights of base learners for more accurate drift adaptation. Using the real-time error rate on all processed samples enables the ensemble model to consider the overall performance of each base learner on a specific task. The time complexity of the ensemble model mainly depends on the complexity of based learners, while the time complexity of PWPAE itself is only $O(nck)$, where n is the number of samples, c is the number of class values in the target variable, k is the number of base learners, as c and k are usually small numbers.

Compared to other ensemble methods, the proposed PWPAE technique has the following advantages:

1. The reciprocal-based weighting function chosen in the proposed PWPAE is an improved version of the weighting function in AUE, which has been proven to outperform other existing ensemble methods in the experimental studies in [23], as this weight function can amplify the weights of well-performing base learners, while also considering other base learners.
2. Compared to the static weights used in many existing ensemble methods, the dynamic weights calculated by the real-time error rates can be used to adjust the importance of base learners based on their real-time performance, which ensures that the current well-performing base learners are given higher weights.
3. Compared to the hard majority voting of class labels used in many existing ensembles, the prediction probability ensemble used in the proposed PWPAE is more flexible and robust, as it takes each learner's uncertainty into account to avoid arbitrary decisions.

Through the proposed PWPAE strategy that integrates the four base learners (ARF-DDM, ARF-ADWIN, SRP-DDM, and SRP-ADWIN), a strong and robust ensemble model can be obtained for effective drift detection and adaptation in IoT data streams analysis.

Furthermore, as ARF and SRP are the state-of-the-art well-performing drift adaptation method, they are used to construct the base learners in the proposed framework. In the future, if new and better-performing drift adaptation methods are proposed, ARF and SRP can be replaced by the new methods to construct a better-performing ensemble model using the same PWPAE strategy.

9.4 Performance Evaluation

9.4.1 Experimental Setup

The proposed framework was implemented using Python 3.6 by extending the Scikit-Multiflow [25] framework on a machine with an i7-8700 processor and 16 GB of memory, representing an IoT central server machine for big data analytics purposes.

Two datasets are used to evaluate the proposed framework. The first dataset is the IoTID20 dataset [6] created by using the normal and attacker IoT devices for IoT network traffic data generation. The 83 different network features were derived from flow-based and packet features for cyber-attack detection, such as flow duration, the total forward and backward packets, active and idle time, etc. The second dataset is the CICIDS2017 dataset [17] that involves the most updated cyber-attack scenarios. As different types of attacks were launched in different time periods in the data creating process, the attack patterns in the dataset change over time, causing multiple concept drifts in the CICIDS2017 dataset.

After using the k-means clustering sampling method, a representative IoTID20 subset that has 6,252 records and a representative CICIDS2017 subset that has 28,303 records are used for model evaluation. They are both highly-imbalanced datasets with a normal/abnormal ratio of 94%/6% and 80%/20%, respectively. This enables the model evaluation on unbalanced datasets.

As the major purpose of anomaly detection systems is to distinguish cyber-attacks from

Table 9.1: Performance Comparison of Drift Adaptation Methods on The IoTID20 Dataset

Method	IoTID20 Dataset				
	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Avg Test Time (ms)**
HT [20]	95.45	95.91	99.42	97.63	0.3
EFDT [22]	97.19	97.53	99.55	98.53	0.3
LB [24]	97.46	97.97	99.36	98.66	4.3
ARF-ADWIN [15]	97.85	98.59	99.13	98.86	0.9
ARF-DDM [15]	98.99	99.05	99.89	99.47	0.6
SRP-ADWIN [16]	98.93	99.05	99.83	99.44	3.4
SRP-DDM [16]	98.79	98.86	99.87	99.36	3.1
Proposed PWPAE	**99.16**	**99.16**	**99.96**	**99.56**	8.3

Table 9.2: Performance Comparison of Drift Adaptation Methods on The CICIDS2017 Dataset

Method	CICIDS2017 Dataset				
	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Avg Test Time (ms)**
HT [20]	91.61	73.03	81.11	76.86	0.2
EFDT [22]	95.71	87.54	87.5	87.52	0.2
LB [24]	98.01	94.67	93.67	94.17	3.0
ARF-ADWIN [15]	98.68	96.76	95.54	96.15	0.8
ARF-DDM [15]	98.77	96.61	96.23	96.42	0.5
SRP-ADWIN [16]	98.82	97.15	95.96	96.55	2.8
SRP-DDM [16]	98.70	96.06	96.39	96.23	2.2
Proposed PWPAE	**99.20**	**98.24**	**97.10**	**97.67**	6.4

normal states, the used datasets are treated as binary datasets with two labels, “normal” or “abnormal”. Hold-out and prequential validations are used to evaluate the proposed framework. For hold-out evaluation, the first 10% of the data is used for initial model training, and the last 90% of the data is used for online testing. In prequential validation, also called test-and-train validation, each input sample in the online test set is first used to test the learning model and then used for model training/updating. As the two used datasets are both unbalanced datasets, five metrics, including accuracy, precision, recall, and f1-score, and execution time, are used to evaluate the anomaly detection performance of the proposed framework.

9.4.2 Experimental Results and Discussion

Tables 9.1 & 9.2 and Figs. 9.3 & 9.4 show the performance comparison of the proposed PWPAE framework with other state-of-the-art drift adaptive approaches introduced in Section 9.2.2, including ARF [15], SRP [16], HT [20], EFDT [22], and LB [24]. As shown in Tables 9.1 & 9.2, the proposed PWPAE method outperforms all other compared models in terms of accu-

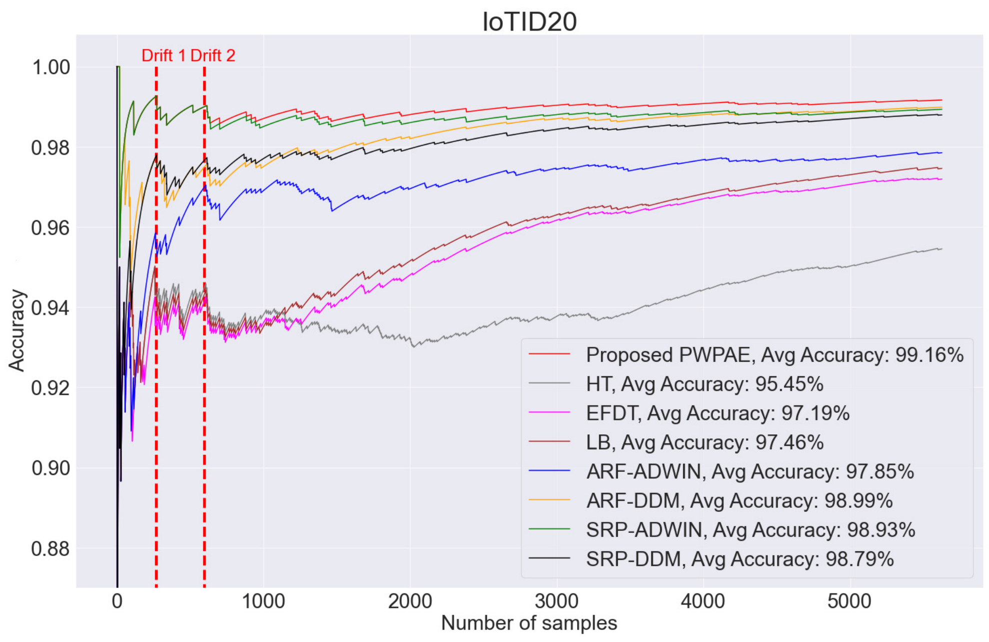

Figure 9.3: Accuracy comparison of different drift adaptation methods on the IoTID20 dataset.

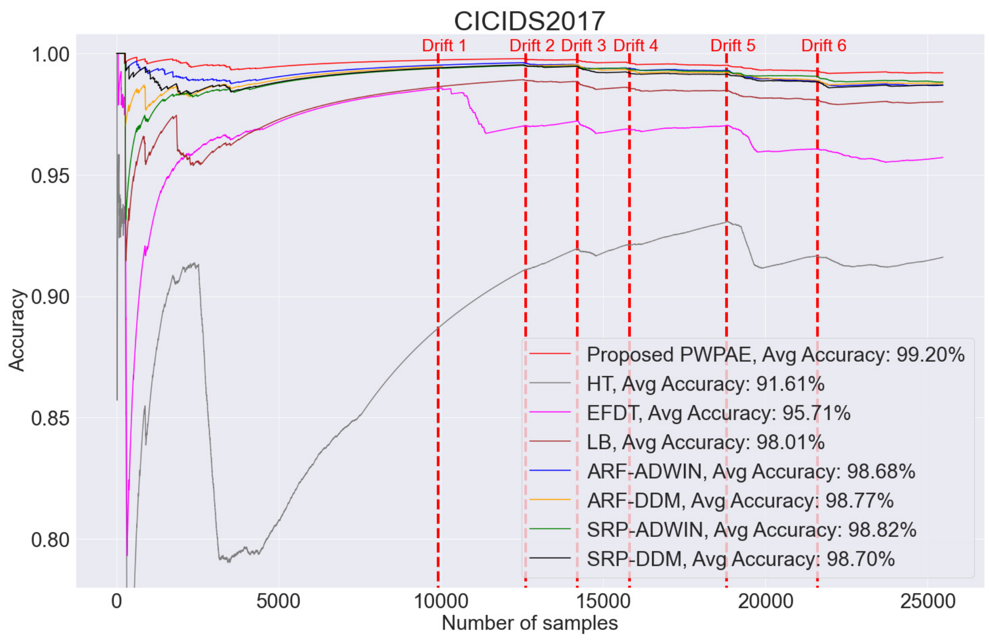

Figure 9.4: Accuracy comparison of different drift adaptation methods on the CICIDS2017 dataset.

racy, precision, recall, and F1-score on the two datasets. As shown in Fig. 9.3, on the IoTID20 dataset, two small drifts occurred at the early stage of the experiment, and all the implemented methods can quickly adapt to the drifts, although their adaptabilities are different. As shown in Table 9.1 Among all the methods, the proposed PWPAE framework achieves the highest accuracy of 99.16%, while the accuracies of the four base learners (ARF-ADWIN, ARF-DDM, SRP-ADWIN, and SRP-DDM) are slightly lower than PWPAE (97.85% -98.99%). The other three drift adaptation methods, HT, EFDT, and LB, have lower accuracies and F1-scores.

For the CICIDS2017 dataset, as shown in Fig. 9.4, there are six concept drifts that occurred in the experiments, including both sudden drifts (drifts 1, 3, 5) and gradual drifts (drifts 2, 4, 6). Similar to the results on the IoTID20 dataset, the proposed PWPAE and four base learners (ARF-ADWIN, ARF-DDM, SRP-ADWIN, and SRP-DDM) can quickly adapt to the drifts and maintain high accuracies, and PWPAE achieves the highest accuracy of 99.20%, as shown in Table 9.2. On the other hand, HT, EFDT, and LB have much lower accuracies of 91.61% to 98.01%. The higher performance of four base learners when compared with other state-of-the-art drift adaptation methods also supports the reasons for selecting them as base learners.

For the average online prediction time for each instance shown in Tables 9.1 and 9.2, although the proposed PWPAE method requires higher time (8.3 ms and 6.4 ms on the IoTID20 and CICIDS2017 datasets, respectively) than certain other compared methods, the average execution time is still at a low level (less than 10 ms). On the other hand, despite the high generation speed of IoT data streams, the use of k-means cluster sampling enables the proposed PWPAE method to maintain high accuracy on a sampled subset, so as to increase the model learning speed and achieve real-time data analytics. Additionally, the current PWPAE method can achieve the best accuracy and F1-score among the compared state-of-the-art drift adaptation methods for IoT anomaly detection.

9.5 Conclusion

The rapidly developing IoT systems have brought great convenience to human beings, but also increase the risk of being targeted by malicious cyber-attackers. To address this challenge, IoT anomaly detection systems have been developed to protect IoT systems from cyber-attacks based on the analytics of IoT data streams. However, IoT data is often dynamic data under non-stationary and rapidly-changing environments, causing concept drift issues. In this chapter, we propose a drift adaptive IoT anomaly detection framework, named PWPAE, based on the ensemble of state-of-the-art drift adaptation methods. According to the performance evaluation on the IoTID20 and CICIDS2017 datasets that represent the IoT traffic data streams, the proposed framework can effectively detect IoT attacks with concept drift adaptation by achieving high accuracies of 99.16% and 99.20% on the two datasets, much higher than other state-of-the-art approaches. In future work, the proposed framework can be extended by integrating other drift adaptation methods with better performance, diversity, and speed.

Bibliography

[1] L. Yang, D. M. Manias, and A. Shami, “PWPAE: An Ensemble Framework for Concept Drift Adaptation in IoT Data Streams,” in *IEEE Global Communications Conference (GlobeCom)*, 2021, pp. 1–6.

[2] M. F. Elrawy, A. I. Awad, and H. F. A. Hamed, “Intrusion detection systems for IoT-based smart environments: a survey,” *J. Cloud Comput.*, vol. 7, no. 1, p. 21, 2018.

[3] H. Kim, J. Ben-Othman, L. Mokdad, and K. Lim, “CONTVERB: Continuous Virtual Emotion Recognition Using Replaceable Barriers for Intelligent Emotion-Based IoT Services and Applications,” *IEEE Netw.*, vol. 34, no. 5, pp. 269–275, 2020.

[4] H. Kim and J. Ben-Othman, “A virtual emotion detection system with maximum cumulative accuracy in two-way enabled multi domain IoT environment,” *IEEE Commun. Lett.*, vol. 25, no. 6, pp. 2073–2076, 2021.

[5] M. Injadat, A. Moubayed, and A. Shami, “Detecting Botnet Attacks in IoT Environments: An Optimized Machine Learning Approach,” in *Proc. 32th Int. Conf. Microelectron. (ICM)*, 2020, pp. 1–4.

[6] I. Ullah and Q. H. Mahmoud, “A Scheme for Generating a Dataset for Anomalous Activity Detection in IoT Networks,” in *Advances in Artificial Intelligence*, 2020, pp. 508–520.

[7] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, “Tree-based Intelligent Intrusion Detection System in Internet of Vehicles,” *proc. 2019 IEEE Glob. Commun. Conf.*, pp. 1–6, Hawaii, USA, 2019.

[8] L. Yang, A. Moubayed, and A. Shami, “MTH-IDS: A Multi-Tiered Hybrid Intrusion Detection System for Internet of Vehicles,” *IEEE Internet Things J.*, 2021.

[9] L. Yang *et al.*, ”Multi-Perspective Content Delivery Networks Security Framework Using Optimized Unsupervised Anomaly Detection,” *IEEE Trans. Netw. Serv. Manag.*, 2021.

[10] M. Marjani *et al.*, “Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges,” *IEEE Access,* vol. 5, pp. 5247–5261, 2017.

[11] L. Yang and A. Shami, “A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams,” *IEEE Internet Things Mag.*, 2021.

[12] D. M. Manias and A. Shami, “Making a Case for Federated Learning in the Internet of Vehicles and Intelligent Transportation Systems,” *IEEE Network,* 2021.

[13] A. Bifet and R. Gavaldà, “Learning from time-changing data with adaptive windowing,” *Proc. 7th SIAM Int. Conf. Data Min.*, pp. 443–448, 2007.

[14] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, “Learning with drift detection,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3171, pp. 286–295, 2004.

[15] H. M. Gomes *et al.* , “Adaptive random forests for evolving data stream classification,” *Mach. Learn.*, vol. 106, no. 9–10, pp. 1469–1495, 2017.

[16] H. M. Gomes, J. Read, and A. Bifet, “Streaming random patches for evolving data stream classification,” *Proc. - IEEE Int. Conf. Data Mining, ICDM*, vol. 2019-November, no. Icdm, pp. 240–249, 2019.

[17] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in *Proc. Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[18] D. M. Manias, I. Shaer, L. Yang, and A. Shami, ”Concept Drift Detection in Federated Networked Systems,” in *2021 IEEE Glob. Commun. Conf. (GLOBECOM)*, Madrid, Spain, Dec. 2021.

[19] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under Concept Drift: A Review,” *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 12, pp. 2346–2363, 2019.

[20] S. Wares, J. Isaacs, and E. Elyan, “Data stream mining: methods and challenges for handling concept drift,” *SN Appl. Sci.*, vol. 1, no. 11, p. 1412, 2019.

[21] Y. Sun, Z. Wang, H. Liu, C. Du, and J. Yuan, “Online Ensemble Using Adaptive Windowing for Data Streams with Concept Drift,” *Int. J. Distrib. Sens. Networks*, vol. 12, no. 5, p. 4218973, 2016.

[22] C. Manapragada, G. I. Webb, and M. Salehi, “Extremely fast decision tree,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov.Data Min.*, pp. 1953–1962, 2018.

[23] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, “Ensemble learning for data stream analysis: A survey,” *Inf. Fusion*, vol. 37, pp. 132–156, 2017.

[24] A. Bifet, G. Holmes, and B. Pfahringer, “Leveraging bagging for evolving data streams,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6321 LNAI, no. PART 1, pp. 135–150, 2010.

[25] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, “Scikit-multiflow: A Multi-output Streaming Framework,” *J. Mach. Learn. Res.*, vol. 19, pp. 1–5, 2018.

Chapter 10

A Multi-Stage Automated Online Network Data Stream Analytics Framework for IIoT Systems

10.1 Introduction

The development of Industry 4.0 has fundamentally improved the design and manufacturing technologies. Industry 4.0 enables smart manufacturing via the application of various technologies, such as the Internet of Things (IoT), Artificial Intelligence (AI), big data analytics, cloud computing, and edge computing, robotics, and cybersecurity [2]. Industry 4.0 also achieves technological advancements that increase the level of automation in manufacturing facilities and warehouses [3]. Although Industry 4.0 has reduced production costs, it has omitted the human costs. Industry 5.0 is then proposed to address this problem by increasing human engagement.

Industry 5.0 is a human-centered design solution for the next evolutionary state, in which collaborative robots (cobots) and machines work collaboratively with human resources to enable customizable autonomous production through business social networks [2]. This allows humans and machines to work and cooperate together. In Industry 5.0, humans can devote

A version of this chapter has been submitted to IEEE Transactions on Industrial Informatics [1].

their creativity to responsible activities, while computers take over repetitive and monotonous duties, hence improving production quality and efficiency. Additionally, Industry 5.0 intends to increase the agility, efficiency, and scalability of production facilities and industries. It will enhance human-machine interaction via improved interfaces and automation systems programmed by human inventiveness, resulting in a multiple-fold increase in productivity [3].

Network automation technologies are essential components in Industry 4.0 and 5.0, as well as 5G networks. Network automation refers to the process of automating the design, implementation, operation, and optimization of networks and related services. Network automation solutions are designed to repeatably and reliably complete tasks at different stages of the network lifecycle [4]. Network automation can increase operational efficiency, reduce system errors, increase network service availability, and improve customer experience.

Network data analytics is a critical component of network automation systems. Automated data analytics driven by AI and Machine Learning (ML) algorithms provide insight into the present and future network activities. ML-driven network data analytics models can infer the purposes of network behaviors, conduct predictive analysis, and make decisions or recommendations. Thus, ML approaches provide promising solutions for network automation and 5G networks. For example, the 3rd Generation Partnership Project (3GPP) organization has proposed a new network function, named Network Data Analytics Function (NWDAF). NWDAF generates analytics results using ML and data analytics algorithms, and then distributes them to other network functions (NFs) [5].

Industry 4.0 and Industry 5.0 both rely heavily on IoT systems. IoT is a network of machines, devices, sensors, and other technologies that connect to or interact with one another over the Internet. The Industrial Internet of Things (IIoT) is a subcategory of IoT that refers to the deployment of IoT technology in industrial applications, such as manufacturing, transportation, healthcare, agriculture, etc. [3]. In manufacturing environments, IIoT incorporates a variety of technologies, such as AI, ML, big data analytics, sensor data collection, and automation, to improve productivity and minimize reliance on human labor. The IIoT's basic premise is that intelligent machines are often more effective and efficient than humans in properly capturing and analyzing data [6].

The primary differences between IoT and IIoT systems include the amount of generated data, the scalability, and the data management approaches [6]. Due to the data generation speed of IIoT devices, IIoT data are usually more continuous, sensitive, and precise than in other IoT systems. A potential delay in IIoT systems, like nuclear power plants, may have severe consequences. Thus, data analytics models used in IIoT systems should be advanced online learning and ML approaches capable of processing constantly-generated IoT data streams efficiently [6].

On the other hand, IIoT data is usually non-stationary data streams generated in ever-changing IIoT systems due to their dynamic nature [7]. Thus, in real-world applications, IIoT data analytics often suffers from concept drift issues when IIoT data distributions change over time. The occurrence of concept drift poses considerable challenges in developing ML models, since their learning performance may progressively degrade owing to data distribution changes [8]. Thus, advanced online adaptive learning models should be developed to detect and react to concept drift that occurs in IIoT data streams. As its main purpose is to maintain model performance by updating the learning model, the drift adaptation procedure is also referred to as automated model updates in the network data analytics automation process.

In this work, a novel Multi-Stage Automated Network Analytics (MSANA) framework is proposed for IIoT data stream analytics. It consists of four stages: dynamic data pre-processing, drift-based dynamic feature selection, base model learning and selection, and online ensemble model development. This work is an extension of Chapter 9. As a representative application of IIoT data analytics, the proposed framework is evaluated on two benchmark IoT anomaly detection datasets, IoTID20 [9] and CICIDS2017 datasets [10], to solve IIoT security problems.

The chapter makes the following contributions:

1. It proposes MSANA, a novel and comprehensive framework for automated data stream analytics in IIoT systems, which includes typical data analytics procedures.
2. It proposes the Window-based Performance Weighted Probability Averaging Ensemble (W-PWPAE) method, a novel ensemble drift adaptation strategy for online learning on dynamic data streams.
3. It proposes a novel dynamic feature selection method for data stream analytics with concept drift issues.

4. It evaluates the proposed framework on two public IoT security datasets as a case study, and compares it with various state-of-the-art online learning approaches.

To the best of our knowledge, no previous study has proposed such a complete pipeline for automated data stream analytics in dynamic IIoT systems.

The remainder of the chapter is organized as follows. Section 10.2 presents the related work about concept drift detection and adaptation. Section 10.3 describes the proposed multi-stage framework for automated data stream analytics. The experimental results are presented and discussed in Section 10.4. Finally, Section 10.5 summarizes the chapter.

10.2 Related Work

Due to the dynamic nature of IIoT systems, network data analytics tasks often encounter concept drift issues when data distributions change over time, causing model learning performance degradation. As traditional static ML algorithms is incapable of addressing concept drift issues in data streams, data stream analytics models must be capable of detecting the occurrence of concept drift and then updating themselves to adapt to the detected concept drift. This section introduces and discusses existing methods for concept drift detection and adaptation.

10.2.1 Concept Drift Detection

There are two broad types of concept drift: sudden and gradual drifts [11]. A sudden drift is a rapid change in the data distribution over a short period of time, while a gradual drift occurs when a new data distribution gradually replaces a historical concept. Different drift detection methods have been designed to detect different types of drift.

Distribution-based methods and performance-based methods are two common types of drift detection methods [11]. Distribution-based methods detect concept drift by comparing the distributions of historical and new data over time windows, while performance-based methods identify concept drift by monitoring the degree of model performance degradation.

ADaptive WINdowing (ADWIN) [12] is a popular distribution-based approach that utilizes an adaptable sliding window to detect concept drift. ADWIN identifies data distribution changes by computing and comparing the characteristic values of the old and new distributions, such as the mean and variance values [13]. A significant change in the characteristic values over time indicates that a drift has occurred.

Through the use of an adaptable sliding window, ADWIN works well with gradual drifts and long-term changes. Additionally, ADWIN uses the characteristic values of data streams instead of their true labels to detect drift, which enables its usage in unlabeled datasets. However, changes in the statistics of data windows are sometimes virtual concept drift, resulting in unnecessary model updates.

Drift Detection Method (DDM) [14] and Early Drift Detection Method (EDDM) [15] are two widely-used performance-based methods that monitor the error rate of learning models to detect drift. DDM tracks model performance changes based on the model's error rate and standard deviation [14]. A drift threshold and a warning threshold can be derived using the error rate and standard deviation to identify the occurrence of concept drift. The warning threshold is used to trigger the collection of potential new concept samples, while the drift threshold is used to trigger model updates. DDM can detect all real drifts that degrade model performance, and is effective in detecting sudden drift. DDM, on the other hand, is often slow to react to gradual drifts until the error rate rises significantly.

EDDM [15] is an extension of DDM that detects concept drift using the same drift and warning levels as DDM. EDDM monitors the change rate of the model's error rate to detect drift, instead of the error rate itself used by DDM. Thus, EDDM is able to increase the detection accuracy of gradual drifts while maintaining superior performance for sudden drifts. However, EDDM is still inferior to distribution-based methods for gradual drift detection.

10.2.2 Concept Drift Adaptation

After identifying a concept drift, learning models should be able to adapt to it to enhance model performance. Existing drift-adaptive learning techniques fall into two primary categories: incremental learning and ensemble learning.

Unlike traditional ML algorithms, which are trained on batches of data, incremental learning is the process of learning each incoming data sample in chronological order and partially updating the learner. Hoeffding Trees (HTs) is a basic incremental learning method designed based on Decision Trees (DTs) [11]. HTs employ the Hoeffding inequality to determine the minimum number of data samples necessary for each split node, thus updating nodes to adapt to new samples. While HT is simple to create, it lacks concept drift detection capabilities.

The Extremely Fast Decision Tree (EFDT) [16] is a state-of-the-art incremental learning approach based on HTs. It selects and deploys each node split as soon as it reaches the confidence value, indicating a useful split. The split will be replaced if a more advantageous split is discovered. Through this strategy, EFDT is able to adapt to concept drift more precisely and efficiently than HTs. However, the performance of EFDT is still often inferior to that of ensemble approaches.

Online Passive-Aggressive (OPA) [17] is another incremental learning algorithm for data stream analytics. The principle of OPA is to passively react to correct classifications and aggressively respond to any misclassifications. In online learning tasks, OPA often outperforms many other ML approaches, such as Gaussian naïve Bayes and online perception.

K-Nearest Neighbors with ADWIN drift detector (KNN-ADWIN) [18] is an improved version of the traditional KNN model for online learning problems. The primary difference between KNN-ADWIN and the traditional KNN is that KNN-ADWIN is capable of detecting and adapting to concept drift. It uses an ADWIN drift detector and a dynamic window to determine which samples to retain for model updating. KNN-ADWIN outperforms many ML algorithms in terms of data stream analytics via the use of concept drift detectors.

Self-Adjusting Memory with KNN (SAM-KNN) [18] is another KNN-based online learning method. SAM-KNN detects concept drift via the use of two memory modules: Short-Term Memory (STM) and Long-Term Memory (LTM). STM is used for the current concept, while LTM can memorize historical concepts. Through the use of STM and LTM, SAM-KNN can fit new concepts while keeping the necessary old concepts.

Incremental learning methods usually have a fast learning speed due to partial model updating, but their drift adaptability is often limited. To improve the effectiveness of concept drift adaptation, numerous ensemble learning strategies based on the integration of multiple base models have been developed.

Leverage bagging (LB) [19] is an ensemble technique that constructs and combines multiple base learners (e.g., HTs) using bootstrap samples. It takes advantage of the Poisson distribution to conduct data resampling for data diversity improvement. While LB is simple to construct, its learning speed is often slow due to its ensemble strategy. Additionally, LB is sensitive to noisy data samples.

Adaptive Random Forest (ARF) [20] is an advanced ensemble online learning method that trains multiple HTs as base models and employs a drift detector (e.g., ADWIN) for each HT to address concept drift. ARF uses the local subspace randomization strategy to construct trees, which randomly generates feature subsets to construct each leaf for node splits. ARF is a strong online learner by inducing diversity through resampling techniques and reacting to drift through drift detectors. Thus, it often outperforms a variety of online learning strategies. Additionally, its sampling approach increases the model learning efficiency.

Streaming Random Patches (SRP) [21] is another online bagging ensemble method for data stream analytics. It is a variant of ARF that uses a similar strategy for detecting and adapting to concept drift. Unlike ARF, which utilizes local subspace randomization, SRP uses global subspace randomization to generate random feature subsets for model learning. Global subspace randomization improves the learning performance of SRP but increases the model complexity and learning time.

Ensemble online learning models are effective drift-adaptive learning methods for IIoT data stream analytics, as they integrate the output of multiple base learners for performance improvement. On the other hand, constructing multiple base models introduces additional execution time.

Thus, despite the existence of many promising drift-adaptive learning methods, there is still much room for improvement. The purpose of this study is to propose an ensemble framework capable of balancing model performance and learning speed. Additionally, existing approaches focus only on model learning but ignore other necessary data analytics procedures, such as data pre-processing and feature engineering. Thus, this chapter proposes a comprehensive data analytics framework that includes other typical data analytics procedures. Moreover, we explored the automation of various data analytics procedures for the aim of network automation.

10.3 Proposed Framework

10.3.1 System Overview

The overview of the proposed Multi-Stage Automated Network Analytics (MSANA) framework for automated data analytics in IIoT systems is shown in Fig. 10.1. It consists of four primary stages: dynamic data pre-processing, drift-based dynamic feature selection, base model learning and selection, and online ensemble model development. At the first stage, to improve data quality, the incoming IIoT data streams are pre-processed using data balancing and normalization methods. Secondly, feature selection methods are utilized to select the most relevant features of data streams, and the features will be re-selected when a concept drift is detected. After that, the base online learners are trained using the cleaned data streams. The two leader models and the top two best-performing follower models are then selected during the dynamic model selection procedure. If concept drift occurs, the dynamic model selection module will re-select the base models according to new data distributions, and then update these learning models on the new concept samples in a time window. The ensemble strategy, W-PWPAE, is then used to integrate the output of the selected base learners based on their prediction probabilities and real-time error rates. Lastly, the ensemble learner's prediction results will be returned to make decisions.

10.3.2 Dynamic Data Pre-Processing

Data pre-processing is mostly used to enhance the quality of data streams in order to increase model learning performance. In IIoT data streams, class imbalance and feature range difference are two potential data quality issues, which can be solved by data balancing and normalization methods, respectively. On the other hand, in dynamic IIoT systems, the class distributions and feature ranges are all dynamic variables that may change significantly over time. Thus, in contrast to the static process in traditional ML pipelines, data pre-processing in online data stream analytics should be dynamic procedures that must be updated on a continual basis.

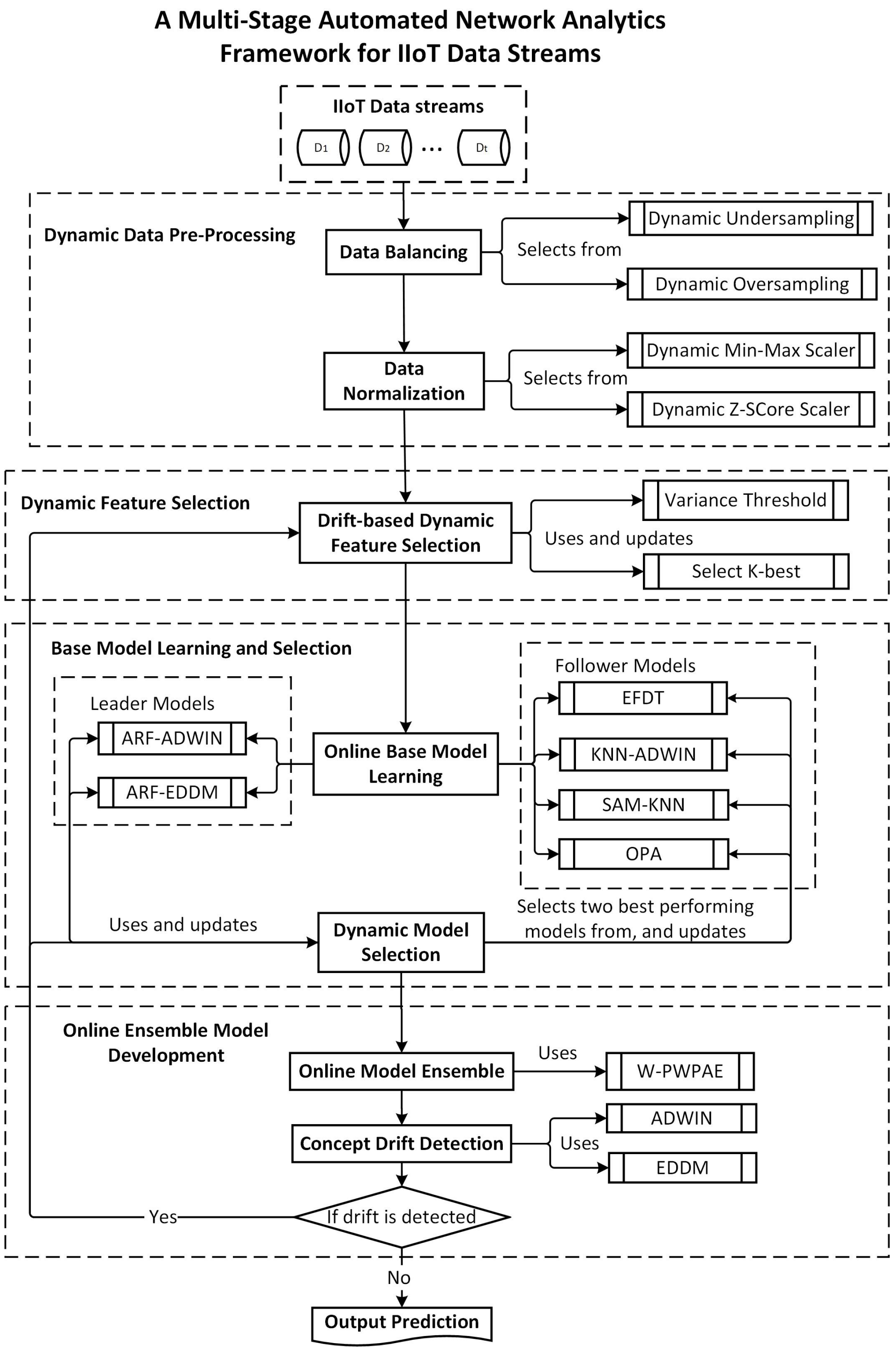

Figure 10.1: The framework of the proposed MSANA system.

Data Balancing

Due to the dynamic nature of IIoT data streams, it is often difficult to maintain balanced distributions of all classes for classification problems, resulting in class imbalance issues. Training on imbalanced datasets may result in biased models with degraded performance. Resampling methods, including under-sampling and over-sampling methods, can be utilized to resolve class imbalance issues [22]. The Dynamic Random Under-Sampling (DRUS) approach randomly discards data samples from majority classes to balance data. On the other hand, the Dynamic Random Over-Sampling (DROS) method balances data by continuously generating more samples for minority classes to increase their proportion. The dynamic implementation of DRUS and DROS enables real-time updating of data distributions in response to class changes in dynamic data streams, thereby ensuring that the current data follows a balanced distribution.

As the DRUS method can reduce the size of data streams, it often improves the efficiency of model learning. However, reducing the majority class samples may result in the loss of crucial information contained in these classes. Conversely, although using DROS may reduce model learning speed due to the increased data size, it often outperforms DRUS due to its ability to balance data without sacrificing any critical information. Thus, DRUS is more suitable for IIoT systems that prioritize efficiency, while DROS is more appropriate for IIoT systems that prioritize performance. As IIoT anomaly detection datasets are often highly imbalanced data with a tiny percentage of anomalies, the proposed system uses DROS to avoid discarding a significant proportion of majority class samples and omitting important information.

Data Normalization

ML and data analytics methods, especially those using distance-based computations (e.g., KNN and k-means), often prioritize features with higher values. If the feature scales of a dataset are largely different, biased models may be created. Hence, data scaling or normalization methods that can normalize the features in a dataset to a comparable scale are beneficial to the learning performance. Z-score and min-max normalization are two commonly-used scaling techniques for data analytics problems.

The Z-score normalization method scales the feature value of each data sample x to a normalized value x_n [23]:

$$x_n = \frac{x - \tilde{\mu}}{\tilde{\sigma}} \tag{10.1}$$

where $\tilde{\mu}$ and $\tilde{\sigma}$ the real-time mean and standard deviation of all processed data samples. Unlike traditional normalization methods, $\tilde{\mu}$ and $\tilde{\sigma}$ are dynamic variables that are recalculated in real-time as new data samples arrive. When concept drift occurs, or the data distribution changes, its real-time mean and standard deviation may also change significantly.

In min-max normalization, the feature value of each data sample x is scaled to [23]:

$$x_n = \frac{x - \widetilde{\min}}{\widetilde{\max} - \widetilde{\min}} \tag{10.2}$$

Similar to Z-score normalization, $\widetilde{\min}$ and $\widetilde{\max}$ are the real-time minimum and maximum values of all processed data samples. Each time a new data sample is processed, $\widetilde{\min}$ and $\widetilde{\max}$ are updated based on the new sample's value. Thus, normalized values for the processed samples can also be updated in response to data distribution changes. Min-max scaler can normalize all features to the same scale of 0-1.

Min-max normalization is better suitable for anomaly detection issues due to its ability to retain outliers (e.g., extremely large or small values) in datasets. On the other hand, Z-score normalization is robust to outliers, so it often performs well for other non-outlier-related data analytics problems. Thus, the normalization method can be determined according to specific problems. As the proposed system aims to solve IIoT anomaly detection problems as a use case, min-max normalization is selected for the proposed framework.

10.3.3 Drift-based Dynamic Feature Selection

As the original features are usually not the optimal features to train an effective ML model, the primary objective of feature selection is to return the updated data with optimal input features for learning performance improvement. Feature selection can also improve the model learning efficiency by discarding irrelevant and noisy features. The proposed Drift-based Dynamic Feature Selection (DD-FS) method used in this work consists of two feature selection techniques: variance threshold and select-k-best. Variance threshold is a feature selection method that aims to eliminate all low-variance features. The variance σ^2 of each feature can be denoted by [24]:

$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n} \tag{10.3}$$

where n is the number of processed samples, x is an input feature, and $\bar{x}$ denotes the mean value of x. Using the variance threshold approach can remove the features whose variance is lower than a given threshold. A low variance indicates that the corresponding feature is often uninformative as it has the same values across the majority of data samples. By removing low-variance features, the learning efficiency of models can be increased. The variance threshold method can also be used for unsupervised learning models, as it can select features without any ground-truth labels.

Select-k-best is a popular feature selection method in which the correlations between each input feature and the target variable are calculated as feature importance scores, and then the features with the k highest importance scores are selected [25]. The feature importance scores can be computed using the Pearson correlation coefficient, a commonly-used metric to measure the correlations between two variables. It can be denoted by [25]:

$$Corr_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 (y_i - \bar{y})^2}} \tag{10.4}$$

where x is an input feature, y is the target variable, $\bar{x}$ and $\bar{y}$ are the mean values of an input feature and the target variable, respectively.

The Pearson correlation coefficient has a range of -1.0 to 1.0, with -1.0 indicating a perfect negative relationship, 1.0 indicating a perfect positive relationship, and 0 indicating that the two variables are fully uncorrelated. Thus, it can be used to quantify the strength of a relationship between each feature and the target variable, making it easy to assess and compare the importance of different features.

Algorithm 5 illustrates the primary procedures of DD-FS. At the first stage, the initial training set S_{train} is learned by the variance threshold and select-k-best methods to generate the initial optimal feature set F'. The initial F' is then utilized to update the feature list of each incoming data sample x_i from the online test set S_{test}. Two drift detectors, ADWIN and EDDM, are used together for drift-based dynamic feature re-selection. If a drift is detected by the drift detectors, the variance threshold and select-k-best feature selection methods will re-learn the recent data samples in a time window s as the new concept samples to generate an updated optimal feature set F'. To minimize unnecessary feature re-selections, the feature set will be

Algorithm 5: Drift-based Dynamic Feature Selection (DD-FS)

```
Input:
  $F$: the original feature set.
Output:
  $F'$: the updated feature set.
$MF_1, F' \leftarrow$ VarianceThreshold_Learning($S_{train}, F$);
$MF_2, F' \leftarrow$ SelectKBest_Learning($S_{train}, F'$);        // Apply FS methods on the training set
$S'_{train} \leftarrow S_{train}$.Update($F'$);                      // Update the feature list
$Drift = 0$;                                                         // Drift indicator
$M \leftarrow$Model_LearningBatch($M, S'_{train}$);                  // Initial learner
for each data instance $x_i \in S_{test}$ do
    $x'_i \leftarrow MF_1$.Transform($x_i, F'$);
    $x'_i \leftarrow MF_1$.Transform($x'_i, F'$);
    $y_{pred_i} \leftarrow$ Model_Prediction($M, x'_i$);             // Predict each new sample
    $Drift_1 \leftarrow$ADWIN.Update($y_{pred_i}, y_{true_i}$);
    $Drift_2 \leftarrow$EDDM.Update($y_{pred_i}, y_{true_i}$);
    if ($Drift_1 == 1$)&&($Drift_2 == 1$) then                       // If both detectors have detected drifts
        $S_{new} \leftarrow S_{test}[i - s, i]$;                      // Recent window samples
        $MF_1, F' \leftarrow$ VarianceThreshold_Learning($S_{new}, F'$);
        $MF_2, F' \leftarrow$ SelectKBest_Learning($S_{new}, F'$);
        $S'_{new} \leftarrow S_{new}$.Update($F'$);                   // Re-select features
        $M \leftarrow$Model_LearningBatch($M, S'_{new}$);             // Update the learner on new features
    else
        $M \leftarrow$Model_LearningOne($M, x'_i$);
    end
end
return $F'$;                                                         // Return the updated feature set
```

updated when both drift detectors have identified the occurrence of concept drift. During the entire online data stream analytics process, this drift-based feature re-selection procedure is automatically repeated each time a concept drift is detected. The drift adaptation functionality of DD-FS is based on the assumption that when concept drift occurs and data distribution changes, the best suitable feature set will change as well.

10.3.4 Model Learning

The proposed model learning framework consists of two stages: base model learning & selection and online model ensemble. The base model learned in the first stage will be selected to construct an ensemble model in the second stage.

Base Model Learning and Dynamic Selection

For base model learning, the lightweight online learning methods introduced in Section 10.2.2 are used to learn the data streams. At the initial stage, the learning methods will process a small-size training set to generate initial base learners. The learners will then learn and predict each incoming sample from the online test set and update themselves if concept drift occurs.

Appropriate base learners should be selected in the proposed dynamic model selection process. Model selection is the process of selecting appropriate base models to construct a robust ensemble model. As the proposed framework is designed for real-time IIoT systems, it should strike a balance between learning performance and efficiency. Thus, relatively lightweight models are selected as the base models in the proposed framework. Two leader models and two follower models are selected in the proposed system.

Firstly, ARF with two different drift detectors, ARF-ADWIN [20] and ARF-EDDM [20], are selected as the two leader base models, since ARF has been proven to be both efficient and effective for a variety of data stream analytics problems [26]. Additionally, the two drift detectors, ADWIN and EDDM, have a strong capacity of dealing with gradual and sudden drifts, respectively; hence, using them with ARF enables the proposed ensemble model to adapt effectively to both gradual and sudden drifts. Moreover, due to the high effectiveness of ARF, using two ARF models with different drift detectors as leader models enables the proposed ensemble model to retain high accuracy even when the follower models do not perform as well as the leader models.

Next, the two follower models are chosen from the other four lightweight and state-of-the-art online learning methods introduced in Section 10.2.2: EFDT [16], KNN-ADWIN [18], SAM-KNN [18], and OPA [17], as shown in Fig. 10.1. Although these four models are not as effective as ARF, they are all well-performing and fast adaptive online learning models that can address concept drift efficiently. The other three models introduced in Section 10.2.2, LB [19], SRP [21], and PWPAE [26], are not selected in the proposed ensemble framework due to their high computational complexity. They are used as comparison models in the experiments.

If concept drift occurs, the follower models will be re-selected from the four candidate models based on their real-time performance in the sliding window of data. This process is called dynamic model selection. The selected top two performing follower models on the data of the new concept are then combined with the ARF-ADWIN and ARF-EDDM leader models to construct a new ensemble model. Additionally, after detecting a concept drift using ADWIN and EDDM, all the four base models will learn the most recent sliding window of data to construct the updated base models that can adapt to the new concept. This dynamic model selection and learning process is beneficial to concept drift adaptation because it enables more effective model updates when the data distribution changes.

Online Ensemble Model Development

After obtaining the base learning models, their prediction outputs are then integrated to construct an ensemble model with improved performance using a novel ensemble strategy, named Window-based Weighted Probability Averaging Ensemble (W-PWPAE). It is extended from the ensemble method PWPAE proposed in Chapter 9 [26].

W-PWPAE integrates base models by assigning dynamic weights to the prediction probabilities of the base models, and then averaging the weighted probabilities. The class with the highest mean probability value, indicating the most confident result, is then selected as the final prediction result. Assuming a data stream $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, and there are c different classes for the target variable, $y \in 1, \ldots, c$, the predicted target class for each input data x can be denoted by:

$$\hat{y} = \underset{i \in \{1, \cdots, c\}}{\operatorname{argmax}} \frac{\sum_{j=1}^{b} w_j p_j \left(y_j = i \mid L_j, x_j \right)}{b} \tag{10.5}$$

where L_j represents the j_{th} base model, $p_j(y_j = i | L_j, x_j)$ indicates the prediction probability of a class value i on the data sample x using the j_{th} base learner L_j; b is the number of base models, where $b = 4$ for the proposed ensemble model, and w_j represents the weight of each base model L_j.

The weight w_j of each base model L_j is computed based on the reciprocal of the model's real-time error rate in the latest window s:

$$w_j = \frac{1}{Error_{s,j} + \epsilon} \tag{10.6}$$

where ϵ is a small constant used to avoid a denominator of 0. Thus, higher weights will be assigned to the base models with lower error rates and better performance.

The window error rate for each base model L_j can be calculated by:

$$Error_{s,j} = \frac{1}{s} \sum_{k=1}^{s} \delta\left(L_j\left(x_k\right), y_k\right) \tag{10.7}$$

where s is the sliding window size, and $\delta\left(L_j\left(x_k\right), y_k\right)$ is the loss function calculated based on the predicted value $L_j(x_k)$ and the ground-truth value y_k.

The window size s is determined according to the detected drift information:

$$\begin{cases} s = DriftArr[-1], \text{ if } DriftArr \text{ is not empty;} \\ s = \alpha * N, \text{ if } DriftArr \text{ is empty.} \end{cases} \tag{10.8}$$

where $DriftArr$ is an array used to record the index of all drift points, α is a ratio, and N represents the total number of processed samples. If any concept drift is detected, the window will include all the data samples with index from the last drift point $DriftArr[-1]$ to the current data point, indicating the new concept data. Otherwise, the window size is determined by a proportion of processed data samples, $\alpha * N$. For example, if α is set to 0.1, the window contains the most recent 10% of the processed data samples. The computational complexity of the W-PWPAE ensemble model is primarily determined by the complexity of the selected based models, whereas the W-PWPAE method itself has a low computational complexity of $O(sck)$, where the window size s, the number of distinct classes c, and the number of base learners k, all usually have small values.

In comparison to other existing drift-adaptive online learning methods, the proposed W-PWPAE approach has the following advantages:

1. Unlike many other existing ensemble learning methods that use the hard majority voting strategy, the proposed framework uses the confidence probability of each base classifier for each class, a more robust and flexible strategy. It considers each base classifier's uncertainty for each data sample to prevent arbitrary decisions.
2. Using the window and performance-based dynamic weighting strategy enables the pro-

posed framework to focus on the model performance on the new concept data, leading to a more effective concept drift adaptation.

3. The selection of lightweight base learning models enables the construction of an efficient ensemble model, as the primary drawback of many existing ensemble models is their high complexity.

10.4 Performance Evaluation

10.4.1 Experimental Setup

The proposed system was implemented in Python 3.7 by extending the River [24] library on a computer equipped with an i7-8700 CPU and 16 GB of memory, representing an IIoT cloud server machine for large data stream analytics.

The proposed approach is evaluated on two public IIoT security datasets: IoTID20 [9] and CICIDS2017 [10]. IoTID20 is a relatively new IoT dataset that was created by generating IoT network traffic data from both legitimate and malicious IoT devices, including 83 different network features. CICIDS2017 is a public network security dataset that was contributed by the Canadian Institute of Cybersecurity and contained state-of-the-art cyberattack scenarios. Due to the fact that the CICIDS2017 dataset was created by launching a variety of different attack types in different time periods, the attack patterns in the dataset have changed over time, resulting in six concept drifts, as shown in Figure 10.2. For the purpose of this work, A representative IoTID20 subset with 6,252 data and a sampled CICIDS2017 subset with 28,303 records are utilized for the model evaluation.

The IIoT data analytics use case solved by the proposed system is anomaly detection, which can be regarded as a binary classification problem by labeling each data sample as a normal sample or an attack sample. The proposed framework is evaluated using the combination of hold-out and prequential validations. For hold-out validation, the first 10% of data is utilized for training the initial base models, and the remaining 90% is used for online testing of dynamic data streams. Prequential validation, also known as test-and-train validation, is utilized to evaluate the proposed model for online learning. In prequential validation, each input sample

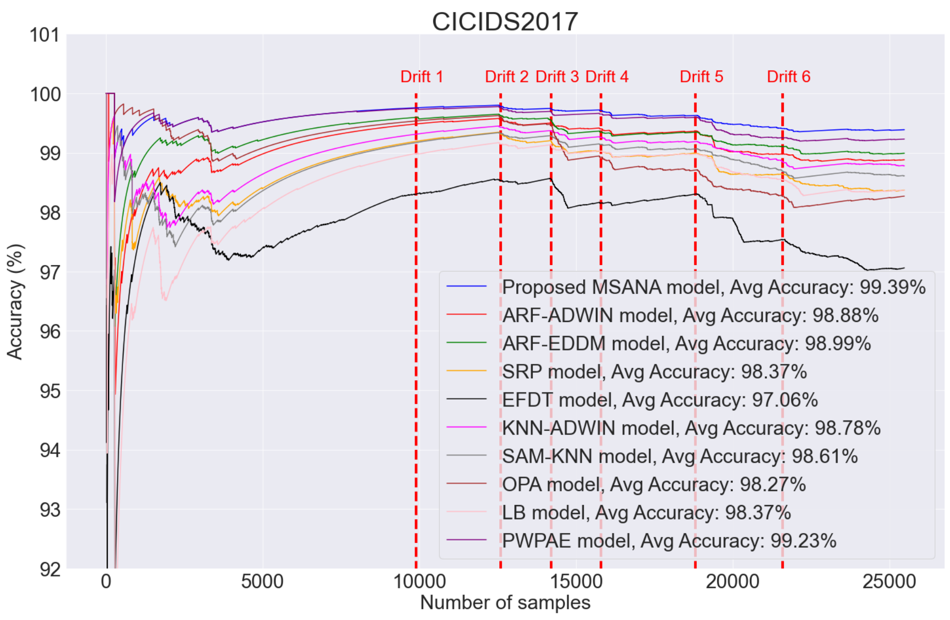

Figure 10.2: Accuracy comparison of state-of-the-art drift adaptation methods on the CICIDS2017 dataset.

in the online test set is firstly test by the learners to monitor their real-time performance, and then learned by the learning model for potential model updates [7].

To provide a comprehensive analysis of the experimental results, five performance metrics, including accuracy, precision, recall, f1-score, and average test time per packet, are used to assess the proposed framework's learning performance.

10.4.2 Experimental Results and Discussion

Figures 10.2 & 10.3 and Tables 10.1 & 10.2 illustrate the performance comparison of the proposed MSANA method against other state-of-the-art online adaptive learning methods presented in Section 10.2.2, including ARF-ADWIN [20], ARF-EDDM [20], SRP [21], EFDT [16], KNN-ADWIN [18], SAM-KNN [18], OPA [17], LB [19], and PWPAE [26].

As shown in Fig. 10.2 and Table 10.1, on the CICIDS2017 dataset, the two leader models, ARF-ADWIN and ARF-EDDM, achieve the greatest accuracy of 98.88% and 98.99% among all the base learners. This demonstrates why they were chosen as leader learners. Among the four follower models (EFDT, KNN-ADWIN, SAM-KNN, and OPA), SAM-KNN and KNN-ADWIN are the top two performing models on the CICIDS2017 dataset, so they are selected

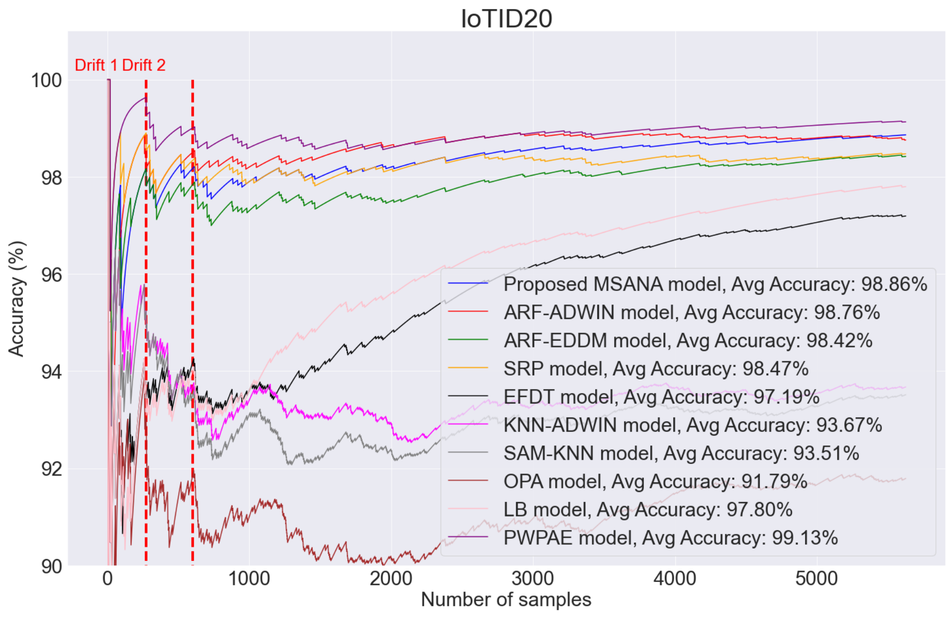

Figure 10.3: Accuracy comparison of state-of-the-art drift adaptation methods on the IoTID20 dataset.

Table 10.1: Performance Comparison of State-of-The-Art Drift Adaptive Online Learning Methods on The CICIDS2017 Dataset

Method	**CICIDS2017 Dataset**				
	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Avg Test Time (ms)**
ARF-ADWIN [20]	98.88	97.27	95.96	96.61	1.1
ARF-EDDM [20]	98.99	97.72	96.17	96.94	1.0
SRP [21]	98.37	95.34	94.82	95.08	3.7
EFDT [16]	97.06	91.31	90.94	91.13	0.5
KNN-ADWIN [18]	98.78	95.52	97.27	96.36	0.7
SAM-KNN [18]	98.61	94.88	96.86	95.86	1.1
OPA [17]	98.37	95.06	95.11	95.08	0.3
LBLB [19]	98.73	96.28	96.07	96.18	4.3
PWPAE [26]	99.23	98.42	96.93	97.66	7.4
Proposed MSANA	**99.39**	**98.69**	**97.61**	**98.15**	**3.5**

as the two final follower models to construct the initial ensemble model. After using the two leader models and two follower models to build an ensemble learner, the proposed MSANA achieves the highest accuracy of 99.39% and F1-score of 98.15% among all the evaluated online learning models. Additionally, the average test time of MSANA is only 3.5 ms per packet, which is less than for other ensemble techniques (SRP, LB, and PWPAE).

The evaluation results for the online learning models on the IoTID dataset are shown in Fig. 10.3 and Table 10.2. Similarly, the two leader models, ARF-ADWIN and ARF-EDDM, achieve

Table 10.2: Performance Comparison of State-of-The-Art Drift Adaptive Online Learning Methods on The IoTID20 Dataset

Method	**IoTID20 Dataset**				
	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Avg Test Time (ms)**
ARF-ADWIN [20]	98.76	98.97	99.72	99.34	1.1
ARF-EDDM [20]	98.42	98.57	99.77	99.17	0.9
SRP [21]	98.47	99.10	99.28	99.19	3.7
EFDT [16]	97.19	97.53	99.55	98.53	0.4
KNN-ADWIN [18]	93.67	95.30	98.13	96.69	0.5
SAM-KNN [18]	93.51	94.93	98.38	96.62	1.4
OPA [17]	91.79	95.99	95.27	95.63	0.2
LBLB [19]	97.80	98.05	99.64	98.94	4.6
PWPAE [26]	99.13	99.08	100.0	99.54	9.0
Proposed MSANA	**98.86**	**98.84**	**99.96**	**99.40**	**2.8**

high F1-scores of 99.34% and 99.17%. For the follower models, EFDT and KNN-ADWIN are selected to build the initial ensemble model due to their better performance when compared with SAM-KNN and OPA. The proposed MSANA method achieves the second-highest accuracy of 98.86% and the second-highest F1-score of 99.40%. Although the proposed MSANA method has a slightly lower accuracy than the existing PWPAE method (98.86% versus 99.13%), it has a much shorter average test time than PWPAE (2.8 ms versus 9.0 ms). This is because, in comparison to the PWPAE method, the proposed MSANA method uses a window strategy and selects lightweight base learners with greater computational speeds to build the ensemble model. Thus, the proposed MSANA method is still the best model in terms of balancing model performance and efficiency among the evaluated models.

10.5 Conclusion

Network automation technologies have drawn much attention for the development of IIoT applications in Industry 4.0 and 5.0. AI and ML algorithms are critical techniques for the automation of network data analytics, which is a key component of IIoT network automation. However, owing to the dynamic nature of IIoT environments, IIoT data is usually large data streams that are continuously generated and changed. As a consequence, concept drift issues often occur as a result of IIoT data distribution changes, causing learning model deterioration. In this chapter, we proposed a comprehensive automated data analytics framework that is capable of processing continuously evolving IIoT data streams while dynamically adapting to concept drifts. The proposed framework consists of dynamic data pre-processing, drift-based dynamic feature selection, dynamic model selection, and online model ensemble using a novel W-PWPAE approach. According to the model performance evaluations on two benchmark IIoT streaming datasets, IoTID20 and CICIDS2017, the proposed framework is capable of effectively processing dynamic IIoT streams with higher accuracy of 98.86% and 99.39%, respectively, than other state-of-the-art methods. The proposed framework may be extended in the future by introducing more advanced data learning techniques to further improve learning performance and speed.

Bibliography

[1] L. Yang and A. Shami, “A Multi-Stage Automated Online Network Data Stream Analytics Framework for IIoT Systems,” Submitted to *IEEE Transactions on Industrial Informatics*, 2022.

[2] P. K. R. Maddikunta et al., “Industry 5.0: A survey on enabling technologies and potential applications,” *J. Ind. Inf. Integr.*, vol. 26, p. 100257, 2022.

[3] Z. Fatima et al., “Production Plant and Warehouse Automation with IoT and Industry 5.0,” *Appl. Sci.*, vol. 12, no. 4, 2022.

[4] D. Rafique and L. Velasco, “Machine Learning for Network Automation: Overview, Architecture, and Applications,” *J. Opt. Commun. Netw.*, vol. 10, no. 10, pp. D126–D143, Oct. 2018.

[5] S. Sevgican, M. Turan, K. Gökarslan, H. B. Yilmaz, and T. Tugcu, “Intelligent network data analytics function in 5G cellular networks using machine learning,” *J. Commun. Networks*, vol. 22, no. 3, pp. 269–280, 2020.

[6] A. Sari, A. Lekidis, and I. Butun, “Industrial Networks and IIoT: Now and Future Trends,” in Industrial IoT: Challenges, Design Principles, Applications, and Security, *I. Butun, Ed. Cham: Springer International Publishing*, 2020, pp. 3–55.

[7] L. Yang and A. Shami, “A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams,” *IEEE Internet Things Mag.*, vol. 4, no. 2, pp. 96–101, 2021

[8] D. M. Manias, I. Shaer, L. Yang, and A. Shami, “Concept Drift Detection in Federated Networked Systems,” in *IEEE Global Communications Conference (GlobeCom)*, 2021, pp. 1–6.

[9] I. Ullah and Q. H. Mahmoud, “A Scheme for Generating a Dataset for Anomalous Activity Detection in IoT Networks,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12109 LNAI, pp. 508–520, 2020.

[10] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in *Proc. Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[11] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under Concept Drift: A Review,” *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 12, pp. 2346–2363, 2019.

[12] A. Bifet and R. Gavaldà, “Learning from time-changing data with adaptive windowing,” *Proc. 7th SIAM Int. Conf. Data Min.*, pp. 443–448, 2007.

[13] E. Uchiteleva, S. Primak, M. Luccini, A. Refaey, and A. Shami, “The TriLS Approach for Drift-Aware Time-Series Prediction in IIoT Environment,” *IEEE Trans. Ind. Informatics*, pp. 1–11, 2021.

[14] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, “Learning with drift detection,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3171, pp. 286–295, 2004.

[15] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, “Early Drift Detection Method,” *4th ECML PKDD Int. Work. Knowl. Discov. from Data Streams*, vol. 6, pp. 77–86, 2006.

[16] C. Manapragada, G. I. Webb, and M. Salehi, “Extremely fast decision tree,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 1953–1962, 2018.

[17] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, “Online Passive-Aggressive Algorithms,” *J. Mach. Learn. Res.*, vol. 7, pp. 551–585, 2006.

[18] V. Losing, B. Hammer, and H. Wersing, “KNN classifier with self adjusting memory for heterogeneous concept drift,” *Proc. - IEEE Int. Conf. Data Mining, ICDM*, vol. 1, pp. 291–300.

[19] A. Bifet, G. Holmes, and B. Pfahringer, “Leveraging bagging for evolving data streams,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6321 LNAI, no. PART 1, pp. 135–150, 2010.

[20] H. M. Gomes et al., “Adaptive random forests for evolving data stream classification,” *Mach. Learn.*, vol. 106, no. 9–10, pp. 1469–1495, 2017.

[21] H. M. Gomes, J. Read, and A. Bifet, “Streaming random patches for evolving data stream classification,” *Proc. - IEEE Int. Conf. Data Mining, ICDM*, vol. 2019-Novem, no. Icdm, pp. 240–249, 2019

[22] P. Kaur and A. Gosain, “Comparing the Behavior of Oversampling and Undersampling Approach of Class Imbalance Learning by Combining Class Imbalance Problem with Noise,” in *ICT Based Innovations*, 2018, pp. 23–30.

[23] A. Pandey and A. Jain, “Comparative Analysis of KNN Algorithm using Various Normalization Techniques,” *Int. J. Comput. Netw. Inf. Secur.*, vol. 9, no. 11, pp. 36–42, 2017.

[24] J. Montiel et al., “River: Machine learning for streaming data in python,” *J. Mach. Learn. Res.*, vol. 22, pp. 1–8, 2021.

[25] S. Pande, A. Khamparia, and D. Gupta, “Feature selection and comparison of classification algorithms for wireless sensor networks,” *J. Ambient Intell. Humaniz. Comput.*, 2021.

[26] L. Yang, D. M. Manias, and A. Shami, “PWPAE: An Ensemble Framework for Concept Drift Adaptation in IoT Data Streams,” in *IEEE Global Communications Conference (GlobeCom)*, 2021, pp. 1–6.

Chapter 11

Conclusion and Future Work

11.1 Conclusion

With the evolving communication and computing technologies, the Internet of Things (IoT) systems have been widely used in a variety of applications, including smart city, smart home, smart healthcare, intelligent transportation systems, etc. The rapidly growing number of IoT devices and correspondingly generated data enables a growing number of IoT functionalities and services, but also brings serious challenges to IoT systems. One challenge is making precise and informed decisions from vast amounts of continuously generated IoT data. A second challenge is automating the data analytics process and improving its performance. Protecting IoT systems and devices from various cyber threats and attacks is the third difficulty.

This thesis aims to solve the technical difficulties associated with IoT data analytics in an effort to improve overall system performance and security. Machine Learning (ML) approaches have shown their capacity for IoT data analytics. The first part of this thesis comprehensively reviews the existing ML and optimization methods for the application of Automated ML (AutoML) technology. The second part of the thesis focuses on the cyber-security issues by proposing four intelligent Intrusion Detection Systems (IDSs) using ML algorithms for the Internet of Vehicles (IoV), as a representative IoT application. The third part of the thesis proposes a comprehensive unsupervised anomaly detection framework for Content Delivery Networks (CDNs), as the primary Internet traffic delivery network. The fourth part of the thesis addresses the dynamic IoT data stream analytics problems by proposing three concept drift adaptive online learning methods. All the proposed methods have been evaluated on benchmark or real-world network datasets and compared with other state-of-the-art methods to illustrate their performance improvement.

The remaining subsections of this chapter provide a summary of the contributions made by each chapter in this thesis. Future research directions are discussed in this chapter as well.

11.2 Summary of Contributions

The summary of each chapter is described as follows.

A comprehensive study of applying AutoML and HPO methods to IoT data analytics has been conducted in Chapter 2. ML and DL algorithms have achieved great success in various data analytics tasks. However, developing effective ML models for specific tasks requires a high level of human expertise, which limits their applicability. Thus, AutoML and HPO methods have been developed to automate and optimize ML models to achieve better learning performance. The advantages and limitations of the state-of-the-art methods for each AutoML procedure are summarized to identify the optimal solutions for applying ML algorithms to specific IoT data analytics tasks. Additionally, a case study of IoT anomaly detection is conducted in Chapter 2 to demonstrate the procedures of AutoML applications. Experimental results have shown the benefits of using AutoML frameworks in IoT data analytics problems.

In Chapter 3, an intelligent intrusion detection system (IDS) is proposed to protect the Internet of Vehicles (IoV) against cyber-attacks based on tree-structure machine learning models, including Decision Tree (DT), Random Forest (RF), Extra Trees (ET), and Extreme Gradient Boosting (XGBoost). The stacking ensemble strategy is also used to combine the results of the four tree-based algorithms for detection performance improvement. To evaluate the proposed IDS, it was tested on two datasets for both intra-vehicle and external networks. The results on both data sets show that the proposed system has 2-3% higher accuracy, detection rate, F1 score, and lower false alarm rate than other existing methods in the recent literature.

In Chapter 4, a novel ensemble method, namely Leader Class and Confidence Decision Ensemble (LCCDE), is proposed to detect various types of cyber-attacks in IoV systems. As ML models often perform differently for different types of attack detection, the proposed LC-

CDE method identifies the best-performing ML models for each type of attack detection as the leader class models to construct a robust ensemble model. Additionally, the prediction confidence information is utilized to help determine the final prediction classes. Three advanced gradient-boosting ML algorithms, XGBoost, LightGBM, and CatBoost, are utilized to construct the proposed LCCDE ensemble model due to their high effectiveness and efficiency. Through the experiments, the proposed IDS framework achieves high F1-scores of 99.9997% and 99.811% on the Car-Hacking and CICIDS2017 datasets, representing intra-vehicle and external vehicular network data, respectively. Moreover, the proposed model's F1-scores are higher than other compared ML methods for detecting every type of attack. This illustrates the benefits of the proposed leader class-based strategy.

In Chapter 5, a transfer learning and ensemble learning-based IDS framework that uses optimized Convolutional Neural Network (CNN) models is proposed to identify various types of attacks in IoV systems. Additionally, a chunk-based data transformation method is proposed to transform vehicle network traffic data into image data used as the input of CNN models. The proposed IDS is evaluated on the Car-Hacking and CICIDS2017 dataset, representing intra-vehicle and external network data, respectively. The experimental results show that the proposed IDS framework can effectively identify various types of attacks with higher F1-scores of 100% and 99.925% than other compared state-of-the-art methods on the two benchmark datasets. Moreover, the model testing results on a vehicle-level machine show the feasibility of the proposed IDS in real-time vehicle networks.

In Chapter 6, a multi-tiered hybrid intrusion detection system (MTH-IDS) model that can detect various types of known and zero-day cyber-attacks on both intra-vehicle and external-vehicular networks is proposed to enhance IoV security. The proposed MTH-IDS consists of two traditional ML stages (data pre-processing and feature engineering) and four main tiers of learners utilizing multiple machine learning algorithms. Through data pre-processing and feature engineering, the quality of the input data can be significantly improved for more accurate model learning. The first tier of the proposed system consists of four tree-based supervised learners used for known attack detection, while the second tier comprises the BO-TPE and stacking models for supervised base learner optimization to achieve higher accuracy. The third tier consists of a novel CL-k-means unsupervised model used for unknown/zero-day attack detection. Lastly, BO-GP and two biased classifiers are used to construct the fourth tier for unsupervised learner optimization. The four tiers of learning models enable the proposed MTH-IDS to achieve optimal performance for both known and unknown attack detection in vehicular networks. Through the performance evaluation of the proposed IDS on the two public datasets that represent intra-vehicle and external vehicular network data, the proposed system can effectively detect various types of known attacks with accuracies of 99.99% and 99.88% on the CAN-intrusion-dataset and CICIDS2017 dataset, respectively. Moreover, the proposed system can detect various types of unknown attacks with average F1-scores of 0.963 and 0.800 on the CAN-intrusion-dataset and CICIDS2017 dataset, respectively. The experimental results on a vehicle-level machine also show the feasibility of the proposed system in real-time environments.

In Chapter 7, a multi-perspective anomaly detection approach based on a real-world general CDN access log dataset is proposed to identify abnormal network entities and cyber-attacks. To detect DoS and cache pollution attacks, their patterns are firstly summarized to extract features from four main perspectives: content, client IP, account-offering, and node perspectives. After obtaining the extracted datasets from multiple perspectives, the anomalies were identified using the optimized unsupervised learning model constructed with the optimized isolation forest and Gaussian mixture models. A comprehensive validation method, including multi-perspective analysis, time-series analysis, and account-offering analysis, was implemented to validate the detected abnormal network entities and the corresponding cyber-attacks. Thus, detection errors can be effectively reduced. Ultimately, the abnormal contents, compromised nodes, and malicious IPs were detected and labeled.

In Chapter 8, an adaptive LightGBM model for IoT data analytics with high accuracy and low time and memory usage is proposed to address concept drift issues in dynamic IoT data analytics. Compared to conventional static data, IoT data is often big streaming data under non-stationary and rapidly-changing environments. Adaptive ML methods are appropriate solutions since they have the capacity to process constantly evolving IoT data streams by adapting to potential concept drifts. Through the integration of our proposed novel drift-handling algorithm (*i.e.*, OASW), an ensemble ML algorithm (*i.e.*, LightGBM), and a hyperparameter method (*i.e.*, PSO), the proposed model has the capacity to automatically adapt to the ever-changing data streams of dynamic IoT systems. The proposed method is evaluated and discussed by conducting experiments on two public IoT anomaly detection datasets, IoTID20 and NSL-KDD. Based on the comparison with several state-of-the-art drift adaptation methods, the proposed system is able to detect IoT attacks and adapt to concept drift with higher accuracies of 99.92% and 98.31% than the other methods on the IoTID20 and NSL-KDD datasets, respectively.

In Chapter 9, a drift adaptive IoT anomaly detection framework, named PWPAE, is proposed based on the ensemble of state-of-the-art drift adaptation methods. It is an advanced ensemble framework that extends the learning model proposed in Chapter 8. According to the performance evaluation on the IoTID20 and CICIDS2017 datasets that represent the IoT traffic data streams, the proposed framework can effectively detect IoT attacks with concept drift adaptation by achieving high accuracies of 99.16% and 99.20% on the two datasets, much higher than other state-of-the-art approaches.

In Chapter 10, a comprehensive automated data analytics framework that is capable of processing continuously evolving IoT data streams while dynamically adapting to concept drifts is proposed for the automation of IoT data analytics. The proposed framework consists of dynamic data pre-processing, drift-based dynamic feature selection, dynamic model selection, and online model ensemble using a novel W-PWPAE approach. According to the model performance evaluations on two benchmark IIoT streaming datasets, IoTID20 and CICIDS2017, the proposed framework is capable of effectively processing dynamic IIoT streams with higher accuracy of 98.86% and 99.39%, respectively, than other state-of-the-art methods.

Lastly, the proposed methods and frameworks in this thesis could be generalized to many other application fields. Specifically,

1. The AutoML and optimization methods proposed in Chapter 2 of this thesis can be generalized to any data analytics applications that use ML algorithms to analyze data and aim to achieve optimal learning performance.
2. The ML models proposed in Chapters 3 - 6 of this thesis can be generalized to any IDS or classification applications.
3. The unsupervised frameworks and strategies proposed in Chapter 7 of this thesis can be generalized to any unlabeled anomaly detection applications.
4. The drift-adaptive online learning methods proposed in Chapters 8 - 10 of this thesis

can be generalized to any dynamic data streams that have concept drift issues or require model updating.

11.3 Future Research Directions

This thesis studies and explores the application of ML and optimization methods in IoT data analytics and cyber-security tasks. Despite the novel approaches and frameworks proposed in this thesis that can tackle many data analytics challenges, there are still numerous future research directions worth investigating. In order to develop more efficient and effective data analytics methods, the following subsections discuss some of these potential future research directions in the areas of AutoML, cyber-security, and IoT data analytics.

11.3.1 Research Directions Towards The Use of AutoML Technology

The AutoML technology introduced in Chapter 2 can be extended in many different directions. One potential research direction is exploring the automated selection and design of ML models. Due to the fact that selecting and designing appropriate ML models still require human expertise, most methods proposed in this thesis are designed manually. In future work, human expertise and machines can collaborate to make this procedure more effectively, as a Human-In-The-Loop (HITL) procedure. For example, human experts can provide a set of potential candidate ML models or architectures based on their experience and human intelligence, and then machines can utilize optimization methods introduced in Chapter 2.5 to identify the most suitable ML models for specific data analytics tasks. Additionally, Neural Architecture Search (NAS) is a novel technology that aims to automatically design deep learning models layer by layer, and it is another research direction worth exploring. Similarly, automated data pre-processing and feature engineering still require further research, because they have a significant effect on model performance and usually require human intervention to improve data quality without losing important information or patterns.

Additionally, as various AutoML methods have distinct benefits and drawbacks, a set of common benchmarks that allow a fair comparison of different techniques can be developed. It would be easier for people to choose appropriate AutoML methods for practical applications if a fair platform or benchmark exists for AutoML applications.

Lastly, the efficiency of AutoML methods is worth exploring, as optimizing ML models usually requires large amounts of time. Distributed ML and transfer learning techniques are potential research areas that can improve the efficiency of AutoML methods.

11.3.2 Research Directions Towards Cyber-Security and IDS Development

The work presented in Chapters 3 - 7 can be extended in various directions. One potential research direction is developing unsupervised IDS systems that can detect unknown or zero-day attack patterns. Although supervised ML algorithms have had great success in developing signature-based IDSs for existing/known attack detection, real-world network data is usually unlabeled and cyber-attackers are always exploring new vulnerabilities to launch new attacks; thus, only developing signature-based IDS is usually insufficient to detect various types of zero-day/new attacks. In Chapter 6, an anomaly-based IDS that is capable of detecting zero-day attacks is proposed, but there is still much room for improvement in terms of detection accuracy.

To detect zero-day attacks, there are several research directions worth exploring. Firstly, different existing and advanced unsupervised anomaly detection algorithms can be investigated, such as Autoencoder and Extreme Gradient Boosting Outlier Detection (XGBOD), as they have the potential to improve the performance of existing IDSs for unknown/zero-day attack detection. Secondly, online learning and adaptive methods are potential solutions for zero-day attack detection, because they can learn new attack patterns over time to detect any new attacks and maintain high detection accuracy.

On the other hand, lightweight IDSs with high detection efficiency can be developed to enable fast response speed and real-time analytics. This is particularly important for deep learning models, as they are usually computationally expensive and require machines with GPUs. In Chapter 5, transfer learning techniques are utilized to reduce model training time and improve efficiency. However, as the IDSs proposed in this thesis prioritize learning performance over efficiency by adopting complex techniques, like ensemble learning and optimization techniques, their efficiency should be explored for better feasibility in real-world networks and systems.

11.3.3 Research Directions Towards IoT Data Stream Analytics and Concept Drift Adaptation

The work presented in Chapters 8 - 10 can be extended in multiple directions. The first research direction is exploring unsupervised online learning methods that can directly detect concept drift using data distribution information, as most existing methods are designed with data labels available and the methods proposed in this thesis are also supervised online learning models. Distribution-based concept drift detection methods, such as ADWIN, IE, and KL divergence, can be investigated to detect and adapt to data distribution changes without immediate labels, as they can use distance metrics to compute the distance between the distributions of data chunks in different time periods.

Another research direction is developing better ensemble drift adaptation methods with improved performance, diversity, and speed. The performance and accuracy of ensemble online learning methods are usually higher than that of single drift adaptation methods, but they usually consume more time and resources, such as the PWPAE methods proposed in Chapter 9. Lightweight methods, like the adaptive LightGBM model introduced in Chapter 8, are worth exploring for ensemble model construction. Thus, to achieve real-time online data analytics, effective online learning methods that can strike a balance between model accuracy and learning efficiency are worth exploring.

Curriculum Vitae

Name:	Li Yang
Post-Secondary Education and Degrees:	2018 - 2022, Ph.D. Electrical and Computer Engineering - Software Engineering Department of Electrical and Computer Engineering Western University London, ON, Canada
	2016 - 2018, MASc. Engineering Systems and Computing School of Engineering University of Guelph Guelph, ON, Canada
	2012 - 2016, B.Eng. Computer Science and Technology School of Computer Science and Technology Wuhan University of Science and Technology Wuhan, Hubei, China
Honours and Awards:	Graduate Student Award for Excellence in Research, ECE UWO, 2022 Award for Outstanding Presentation in Graduate Symposium, ECE UWO, 2022 Chinese Government Award for Outstanding Self-financed Students Abroad, 2022 OC2 Lab Research Excellence Award, OC2 UWO, 2020 OC2 Lab Industrial Research Excellence Award, OC2 UWO, 2020
Related Work Experience:	Teaching & Research Assistant Western University 2018 - 2022
	Teaching & Research Assistant University of Guelph 2016 - 2018

Publications:

Journal Publications

[1] **L. Yang** and A. Shami, “A Multi-Stage Automated Online Network Data Stream Analytics Framework for IIoT Systems,” Submitted to *IEEE Transactions on Industrial Informatics*, 2022.

[2] **L. Yang** and A. Shami, “IoT Data Analytics in Dynamic Environments: From An Automated Machine Learning Perspective,” To appear in *Engineering Applications of Artificial Intelligence*, 2022.

[3] **L. Yang**, A. Moubayed, A. Shami, A. Boukhtouta, P. Heidari, S. Preda, R. Brunner, D. Migault, and A. Larabi, “Forensic Data Analytics for Anomaly Detection in Evolving Networks,” To appear in *Digital Forensics*, World Scientific, 2022.

[4] **L. Yang**, A. Moubayed, A. Shami, P. Heidari, A. Boukhtouta, A. Larabi, R. Brunner, S. Preda, and D. Migault, “Multi-Perspective Content Delivery Networks Security Framework Using Optimized Unsupervised Anomaly Detection,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 686-705, March 2022.

[5] **L. Yang**, A. Moubayed and A. Shami, “MTH-IDS: A Multitiered Hybrid Intrusion Detection System for Internet of Vehicles,” *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 616-632, Jan.1, 2022.

[6] **L. Yang** and A. Shami, “A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams,” *IEEE Internet of Things Magazine*, vol. 4, no. 2, pp. 96-101, June 2021.

[7] **L. Yang** and A. Shami, “On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[8] **L. Yang**, R. Muresan, A. Al-Dweik and L. Hadjileontiadis, ”Image-Based Visibility Estimation Algorithm for Intelligent Transportation Systems,” *IEEE Access*, vol. 6, pp. 76728-76740, 2018.

Conference Publications

[1] **L. Yang**, A. Shami, G. Stevens, and S. DeRusett, "LCCDE: A Decision-Based Ensemble Framework for Intrusion Detection in The Internet of Vehicles," Accepted in *2022 IEEE Global Communications Conference (GLOBECOM)*, 2022, pp. 1-6.

[2] **L. Yang** and A. Shami, "A Transfer Learning and Optimized CNN Based Intrusion Detection System for Internet of Vehicles," in *2022 IEEE International Conference on Communications (ICC)*, 2022, pp. 1-6.

[3] **L. Yang**, D. M. Manias, and A. Shami, "PWPAE: An Ensemble Framework for Concept Drift Adaptation in IoT Data Streams, " in *2021 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2021, pp. 1-6.

[4] D. M. Manias, I. Shaer, **L. Yang**, and A. Shami, "Concept Drift Detection in Federated Networked Systems," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1-6.

[5] **L. Yang**, A. Moubayed, I. Hamieh, and A. Shami, "Tree-based Intelligent Intrusion Detection System in Internet of Vehicles," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1-6.