PARALLEL & DISTRIBUTED DATA ANALYTICS FOR TIME-SENSITIVE APPLICATIONS ON FOG/EDGE ARCHITECTURES

A Thesis Submitted

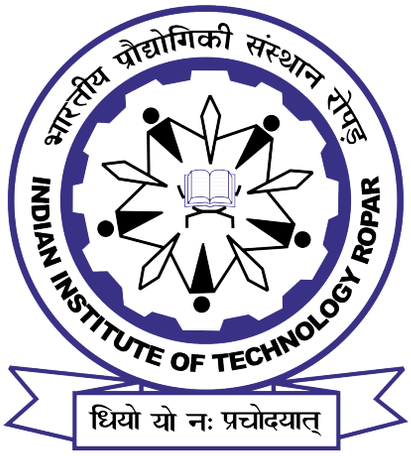
in Partial Fulfilment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

by

Mansi Sahi

(2017csz0010)



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROPAR

June, 2024

Dedicated to my family

Declaration of Originality

I hereby declare that the work which is being presented in the thesis entitled **Parallel & Distributed Data Analytics for Time sensitive Applications on Fog/Edge Architectures** has been solely authored by me. It presents the result of my own independent investigation/research conducted during the time period from January 2017 to February 2024 under the supervision of Dr. Nitin Auluck, Associate Professor, Department of Computer Science and Engineering Indian Institute of Technology Ropar, Punjab. To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted or accepted elsewhere, in part or in full, for the award of any degree, diploma, fellowship, associateship, or similar title of any university or institution. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgments, in line with established ethical norms and practices. I also declare that any idea/data/fact/source stated in my thesis has not been fabricated/ falsified/ misrepresented. All the principles of academic honesty and integrity have been followed. I fully understand that if the thesis is found to be unoriginal, fabricated, or plagiarized, the Institute reserves the right to withdraw the thesis from its archive and revoke the associated Degree conferred. Additionally, the Institute also reserves the right to appraise all concerned sections of society of the matter for their information and necessary action (if any). If accepted, I hereby consent for my thesis to be available online in the Institute's Open Access repository, inter-library loan, and the title & abstract to be made available to outside organizations.

Signature

Name: Mansi Sahi
Entry Number: 2017CSZ0010
Program: PhD
Department: Computer Science and Engineering
Indian Institute of Technology Ropar
Rupnagar, Punjab 140001

Date: 14^{th} June 2024

Acknowledgement

I would like to thank my supervisor, Dr. Nitin Auluck, for his support, encouragement, and insightful feedback. I am deeply grateful for his patience in guiding me through the complexities of the research process, and the constructive discussions that have enriched the quality of this dissertation. I consider myself incredibly fortunate to have a supervisor who demonstrated such genuine concern for my work and consistently provided swift responses to my questions and inquiries.

I am also grateful to my doctoral committee members: Dr. Apurva Mudgal, Dr. Balwinder Sodhi, Dr. M. Prabhakar, and Dr. Anshu Jayal for evaluating my research work periodically and for their support. I am indebted to IIT Ropar for providing me with all the necessary facilities for this research.

Last but not the least, I express my special gratitude to my family & friends for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without all of you. Thank you.

Mansi Sahi
Indian Institute of Technology Ropar

Certificate

This is to certify that the thesis entitled **Parallel & Distributed Data Analytics for Time-sensitive Applications on Fog/Edge Architectures**, submitted by **Mansi Sahi(2017csz0010)** for the award of the degree of **Doctor of Philosophy** of Indian Institute of Technology Ropar, is a record of bonafide research work carried out under my guidance and supervision. To the best of my knowledge and belief, the work presented in this thesis is original and has not been submitted, either in part or full, for the award of any other degree, diploma, fellowship, associateship or similar title of any university or institution.

In my opinion, the thesis has reached the standard fulfilling the requirements of the regulations relating to the Degree.

Signature of the Supervisor
Dr. Nitin Auluck
Department of Computer Science and Engineering
Indian Institute of Technology Ropar
Rupnagar, Punjab 140001
Date: 14^{th} June 2024

Lay Summary

Substantial delays in data transmission between users and remote cloud servers can impede the timely execution of real-time tasks with deadlines. Employing fog nodes to handle such tasks can mitigate this delay. Fog computing entails deploying multiple nodes or micro data centers in close proximity to users and data sources, aiming to reduce propagation delays between these sources and the central cloud data center. This approach addresses performance gaps in the traditional cloud-to-thing architecture by bringing computing capabilities closer to the data source. The architecture of fog nodes may adopt a hierarchical structure to accommodate the varied execution requirements of real-time applications. The primary objectives and contributions of this research are outlined as follows:

- We propose a framework for efficiently partitioning machine learning model splits for online training on edge networks, considering their safety constraints and requirements. The framework aims to minimize training time and communication latency, ensuring that ML models are reliably trained and updated on edge devices without compromising safety, performance, or resource utilization.

- The proposed FCAFE-BNET approach improves the multi-class IDS performance by exploiting various pre-processing steps that help in identifying various attack patterns correctly. The proposed FCAFE-BNET algorithm takes into account dynamic network conditions before allocating the tasks to different fog layers i.e. Cloud/cluster/fog device. Moreover, the use of the early-exit mechanism in the local fog device speed up the inference by reducing the number of computations, without adversely affecting the performance.

- The proposed $D^2 - TONE$ algorithm utilizes data-driven task offloading in order to predict the offloading cost on heterogeneous multi-edge networks. The proposed framework significantly reduces the training time of DNN model by updating model parameters in a parallel and distributed manner on various edge devices without sacrificing the performance of the trained model.

- ML models were trained on the specific non-linear time series dataset with HR and IBP parameters to predict patients' survival outcome, ICU stay, and hospital stay, determining risks following cardiac surgery. The results demonstrated that a combination of ensemble ML models and refined feature engineering could accurately predict patient mortality.

Abstract

Fog computing expands upon the conventional cloud computing model, typically integrating fog nodes at the network edge for computing and storage purposes. By situating these edge devices close to users, it enhances application response times and alleviates the burden on the central cloud server. Additionally, fog computing offers computational, storage, and networking services bridging the gap between users and traditional cloud computing data centers.

This thesis proposes four different frameworks for ML model partitioning on fog architecture, Intrusion Detection System on Fog Architecture, Data-Driven Deep Neural Network Task Offloading on Edge Networks, real-time outcomes prediction in cardiac surgery.

In the first work, we propose a framework that intelligently partitions ML models into smaller sub-models that can be safely executed across multiple edge devices, leveraging their parallel computing capabilities. Further, to enhance the safety and reliability of the online model training process, our approach incorporates the Triple Modular Redundancy (TMR) technique for trusted computation.

The second work proposes a lightweight distributed Intrusion Detection System (IDS) framework, called FCAFE-BNET (Fog based Context Aware Feature Extraction using BranchyNET). The proposed FCAFE-BNET approach considers versatile network conditions, such as varying bandwidth and data load before allocating inference tasks on Cloud/Edge resources. Early exit DNN is used to obtain faster inference generation at the edge. The proposed FCAFE-BNET framework works for both Network-based and Host-based IDS.

In the third work, we propose a D^2–$TONE$ (Data-driven Deep Neural Network Task Offloading on the Network Edge), an approach that employs Machine Learning algorithms for accurately estimating offloading delays, such as computational and transmission delays. D^2–$TONE$ holistically adapts to dynamic network situations and provides optimal/near-optimal offloading solutions in real-time. In addition, the proposed algorithm employs distributed execution of DNN tasks on edge devices/cloud data centers.

The fourth work aims to develop artificial intelligence models based on non-linear time-series data of blood pressure and heart rate to predict the ICU stay, hospital stay, and survival outcome of cardiac surgical patients. Specifically, we aim to construct an end-to-end real-time data analysis pipeline that incorporates artifact removal, non-linear noise reduction, and features engineering. We have performed model predictions on an edge device, so that the alerts to the doctor can be transmitted in real-time.

This thesis also provides a detailed description of the fog computing paradigm. It summarises the state-of-the-art work in the field of ML model partitioning on fog architecture, Intrusion Detection System on Fog Architecture, Data-Driven Deep Neural Network Task Offloading on Edge Networks, and real-time outcomes prediction in cardiac

surgery. Finally, this thesis discusses future research directions in this field.

Keywords: Optimized scheduling; Edge computing; Fog computing; cloud computing; real-time scheduling; Distributed Machine Learning; Network Intrusion Detection Systems; Host Intrusion Detection Systems

List of Publications

Journal

[1] S. Konar, N. Auluck, R. Ganesan, A. Goyal, T. Kaur, M. Sahi, T. Samra, S. Thingnam, and G. D. Puri, "A non-linear time series based artificial intelligence model to predict outcome in cardiac surgery", Health and Technology, Springer, 2022.

[2] M. Sahi, N. Auluck, A. Azim, M. Maruf, "Dynamic Hierarchical Intrusion Detection task offloading in IoT Edge Networks", Journal of Software: Practice and Experience, Wiley, 2024.

[3] M. Maruf, A. Azim, N. Auluck, and M. Sahi, "Pipeline Dnn Model Parallelism for Improving Performance of Embedded Applications.", Journal of Parallel and Distributed Computing, Elsevier, 2024.

[4] N. Auluck, R. Ganesan, T. Kaur, A. Mittal, M. Sahi, S. Konar, T. Samra, G. Puri, and S. Thingnum. "Application of concept Drift Detection and Adaptive Framework for Non-Linear Time Series Data from Cardiac Surgery.", Computational Intelligence, Wiley, 2024.

[5] M. Sahi, N. Auluck, A. Azim, M. Maruf, "Data-Driven Deep Neural Network Task Offloading on Edge Networks", IEEE Transactions on Network and Service Management, 2023. (Under review)

[6] T. Kaur, R. Ganesan, M. Sahi, G. Puri, S. Konar, A. Mittal, T. Samra, S. Thingnum, S. Maheshwar, and N. Auluck, "Neural network-based non-linear time series forecasting for cardiac surgery applications.", Journal of Medical Engineering & Technology, 2024. (Submitted)

Conference Proceedings

[1] M. Sahi and N. Auluck, "An IoT-based Intelligent Irrigation Management System", The Twenty-Sixth International Conference on Advanced Computing and Communications (ADCOM), December 16-18, Silchar, Assam, 2020.

[2] M. Sahi, M. Maruf, A. Azim and N. Auluck, "A framework for partitioning support vector machine models on edge architectures", The Fourth International IEEE Workshop on Deep Learning on Edge for Smart Health and Wellbeing Applications (EDGE-DL), Irvine, CA, USA, August 23, 2021.

[3] M. Sahi, M. Soni, and N. Auluck, "Intrusion detection system on fog architecture", The Fourth International Workshop on Smart Living with IoT, CLoud and Edge Computing (SLICE 2021), Denver, USA, October 4-7, 2021.

[4] Md. Al-Maruf, Mansi Sahi, Nitin Auluck, and Akramul Azim, "Towards Safe Online Machine Learning Model Training and Inference on Edge Networks", The 22nd IEEE International Conference on Machine Learning and Applications (ICMLA), Jacksonville, USA, December, 2023.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Internet of Things (IoT) devices are producing ever-growing volumes of data that require analytics within designated time constraints, as emphasized in [1, 2]. The data produced by these IoT applications is typically analyzed using remote Cloud servers. According to CISCO's reports [3], the number of connected devices is projected to reach 29.3 billion by 2024, a significant increase from the 18.4 billion recorded in 2018. The number of active IoT devices is expected to double by 2030. With the rapid proliferation of IoT applications, there's a projection that the amount of generated data will surpass the current network bandwidth capabilities [4]. Also, there are inherent network delays between IoT devices and the remote Cloud. Due to the above reasons, offloading latency-sensitive tasks to a Cloud server may adversely affect the Quality of Service (QoS) [5, 6] of the applications. In these applications, data analysis needs to be accomplished within a predefined time-frame, to satisfy user expectations. Instances of such systems encompass real-time gaming, control systems, internet streaming of audio/video content, smart vehicles, etc. In the context of smart vehicles, for instance, even a single instance of missing a deadline could lead to a system failure.

The Internet of Things (IoT) is widely recognized as a primary contributor to the generation of Big Data, given its ability to link numerous smart devices that consistently transmit their status [7]. While the IoT concept emphasizes the connectivity and interaction of physical objects, its genuine potential lies not in the objects themselves, but in the extraction of valuable insights from the data they produce. Essentially, the Internet of Things revolves around data, rather than the physical entities. In this regard, Machine Learning (ML) serves as a valuable tool for processing the generated data and converting it into information, knowledge, predictions, insights, and automated decisions [8]. The incorporation of ML techniques in the IoT introduces various challenges, particularly in terms of their computational demands. Earlier, these ML tasks were sent to remote cloud servers for processing, due to the limited computing capacities of these available day-to-day devices. However, the computational power and storage capabilities of contemporary devices have seen substantial growth in the past few years. These improvements in our day-to-day gadget hardware have given rise to the concept of "Edge/Fog Computing" [6].

1.1 Fog Computing

Fog computing is a new and emerging computing model that offers computing resources situated between end-user devices and cloud servers. This approach offers numerous benefits to end-users and cloud computing servers. Fog computing is a distributed computing system designed to enhance cloud computing functionalities, particularly for the Internet of Things (IoT) environment. The objective is to move intelligence, storage, and processing closer to the edge of the network, facilitating faster and more localized computing services for the interconnected smart devices constituting the IoT. It minimizes the storage and computing load on cloud servers by analyzing incoming data and applications in closer proximity to IoT devices. "Edge computing" is a term closely related to fog computing. To be more precise, in edge computing, the networking infrastructure, computing, and storage are consistently within one step away from the data source. In contrast, fog computing [9] represents a computing paradigm where the networking infrastructure, computing, and storage can be positioned anywhere between IoT devices and cloud data centers, not necessarily limited to being just one step away from the IoT devices [6, 10].

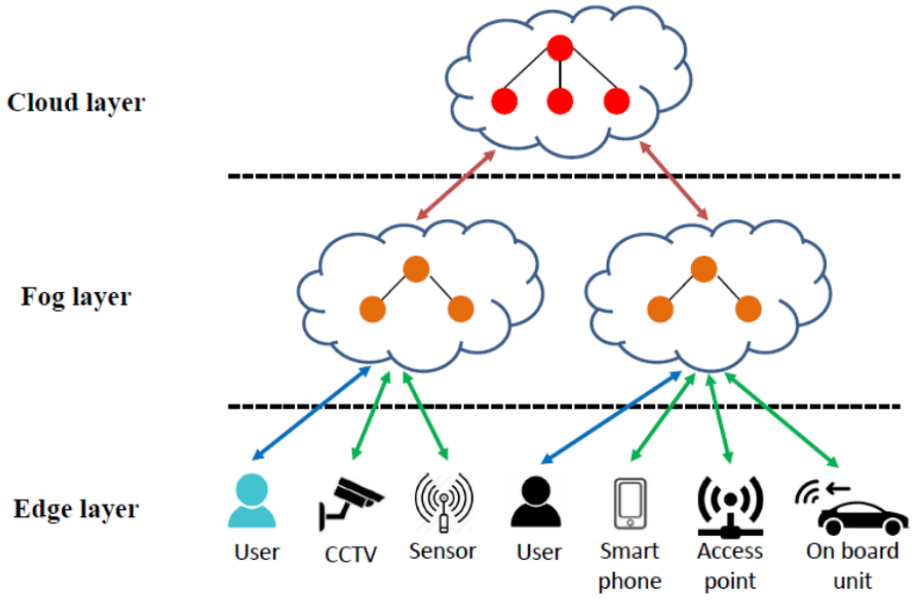


Figure 1.1: Fog Computing

1.1.1 Fog Computing Architecture

The architecture of Fog computing comprises both physical and logical elements, represented by hardware and software, to establish an Internet of Things (IoT) network [11]. As illustrated in Figure 1.1, it encompasses IoT devices, edge nodes, fog nodes, and remote cloud servers. Now, we delve into the components of the fog computing architecture:

- **IoT devices:** These devices are linked within the IoT network, through a variety

of wireless and wired technologies. These devices consistently generate substantial amounts of data. Multiple wireless technologies, such as Zigbee, RFID, Bluetooth, etc., are employed in IoT, along with protocols like IPv4, IPv6, MQTT, and others.

- **Edge nodes:** The IoT devices are directly connected to edge nodes, where the generated data is collected and pre-processed. Such edge nodes are typically employed in applications with lower resource demands, because the devices responsible for data collection and processing have restricted capabilities [12]. An example of this is predictive maintenance, where edge computers, embodied as sensors, assist manufacturers in analyzing the condition of plant equipment, and identifying alterations before a breakdown occurs. Industrial Internet of Things (IIoT) sensors continually observe the health of equipment and employ analytics to provide alerts about upcoming maintenance requirements.

- **Fog nodes:** A fog node is any device equipped with computing, storage, and network connectivity. Multiple fog nodes are distributed across extensive regions to offer assistance to end devices. Their installation occurs at diverse locations, based on distinct applications [13]. Fog nodes are commonly utilized in time-critical applications demanding extensive, resource-intensive data processing derived from a widely distributed network of devices. For instance, the efficient management of smart grids necessitates the processing of substantial volumes of real-time data. The multitude of sensors and other edge devices employed in these applications are both numerous and widely distributed. Consequently, fog nodes are employed to simultaneously process data without compromising response times. Fog nodes encompass devices such as switches, controllers, cameras, routers, etc. These fog nodes handle the processing of highly sensitive data.

- **Cloud server:** The cloud is interconnected with all aggregated fog nodes. Data that is not time-sensitive or of lower sensitivity undergoes processing, analysis, and storage in the cloud. Data Analysis that can afford to wait for an extended duration, whether it be hours, days, or weeks, can be processed at the cloud. Such data is transmitted to the cloud for storage and future analysis.

1.1.2 Necessity for Fog Computing

Cloud computing is an innovative technology that brings numerous advantages to several applications, such as the ability to scale resources, flexibility in managing IT infrastructure, and cost-effective pay-as-you-go models. However, there are also various drawbacks associated with cloud computing that present challenges to real-time applications [14]. These disadvantages encompass:

- **Internet Connectivity:** Applications solely rely on internet connectivity to access cloud computing services. However, if there is an internet outage or weak connectivity, it can lead to service interruptions and increased delays. Consequently,

one of the primary criticisms of cloud computing is its significant reliance on the availability and quality of internet connectivity.

- **Security and Privacy:** Transmitting sensitive operational data from the edge to the cloud poses a threat to both the data and to the edge devices. To safeguard this information in an IoT system, it is crucial to implement various layers of security to guarantee the secure transfer of data to cloud storage systems. Conducting data processing at the edge serves as a preventive measure against data breaches, and facilitates quicker responses.

- **Latency:** Transferring an application's entire device data to the cloud for processing and analytics can span from a brief few minutes to several days [15]. In time-sensitive applications like Industrial IoT, immediate processing of device data is crucial for prompt corrective actions. The fog computing model, in contrast to the cloud computing model, can significantly reduce latency and facilitate rapid decision-making.

- **Data-Transfer and Bandwidth Cost:** Sending substantial amounts of data from the network edge to a cloud server can incur exorbitant expenses [16]. Additionally, the ongoing daily cost of transferring such data may result in unsustainable communication expenses over time.

1.1.3 Applications

Fog computing can be a viable paradigm for applications with real-time demands. Below are examples of applications where Fog computing can prove advantageous:

- **Heathcare:** Cloud computing optimizes and distributes resources efficiently. It operates independent of location, allowing users to access cloud services from any place and device with an internet connection [17]. The vast and varied data generated by IoT can be efficiently accessed through cloud computing. The amalgamation of cloud and IoT minimizes costs and facilitates the aggregation of substantial data. In the healthcare sector, cloud computing serves as a means to monitor patients, maintain records, and effectively manage illnesses by analyzing the accumulated data. Nevertheless, the cloud may not be well-suited for time-critical applications due to various challenges associated with high bandwidth demands, concerns related to safety and security, and intermittent delays. Real-time monitoring, essential for healthcare applications, may not be effectively addressed by the cloud, as it may not meet the immediate response requirements. The transfer of data to the cloud and its subsequent return to the application introduces delays. In healthcare, where timely and accurate responses are crucial for saving lives, these issues become particularly critical. Fog Computing can prove to be instrumental in addressing numerous challenges within the healthcare system. It can efficiently manage tasks at the network edge, allowing for the delegation of certain functions to cloud

data centers. This capability extends to facilitating big data analytics. Moreover, crucial operations can be carried out at the network edge, ensuring that vital and sensitive data can be accessed within milliseconds, when needed. The system is equipped to promptly alert the hospital or emergency services in case of any detected abnormalities, ensuring swift response to serious issues.

- **Privacy and security:** Fog computing enables real-time analysis and response, accelerating the identification and resolution of security risks. It achieves this while safeguarding user privacy through the containment of sensitive data within secure environments. In this thesis, we have explored two applications in this domain:

  - **Intrusion detection in IoT network:** Cloud computing can be utilized for attack monitoring; however, the high latency associated with the cloud-based processing poses challenges in achieving real-time network monitoring [18]. Additionally, the continuous deployment and operation of a model on the cloud can incur substantial costs. To address these issues and enable real-time monitoring of attacks in IoT networks, the fog computing paradigm is introduced. In this approach, data generated by end devices, such as sensors, is processed by fog and edge nodes situated nearby. This significantly reduces latency, ensuring a real-time experience for the user. The tasks associated with the Network Intrusion Detection system are time-sensitive and are expected to operate continuously, safeguarding the privacy of data. Detecting attacks at the earliest possible stage can significantly enhance server security, and the fog architecture is instrumental in achieving this objective. Employing a locally executed Machine Learning (ML) algorithm plays a crucial role in this process. Our focus is on addressing the intrusion detection challenges prevalent in IoT networks, where IoT devices are susceptible to various network attacks such as flooding attacks, man-in-the-middle attacks, and port scanning. To identify and thwart these attacks, a Network Intrusion Detection System (NIDS) is deployed at fog nodes that are strategically positioned throughout the network. These fog nodes scrutinize the network traffic from all connected devices, comparing it with patterns from previous traffic categorized as an attack [19]. Upon detecting a match, the administrator is promptly notified to initiate further action. The primary objective is to identify and respond to such attacks as soon as they attempt to compromise the system. Fog nodes, which can range from routers and switches to cameras and industrial controllers, may not possess high computational power. Therefore, a lightweight NIDS is imperative for effective intrusion detection.

  - **Face detection:** Numerous cloud-based APIs, such as Google's Cloud Vision and AWS Rekognition, offer people counting or face detection capabilities. Nevertheless, these services necessitate a consistent and dependable internet connection. Facial recognition-enabled security cameras are capable of real-time

identification and tracking of individuals. This technology is employed in various public settings like airports and other areas to bolster security measures and surveil potential risks. People counting or face detection application requires real-time processing [20]. Often, huge volumes of data is transmitted through cameras in a short span of time. In addition, transmitting huge volumes of data to the cloud server for processing is likely to be expensive. Fog computing can help us to address above issues, by analyzing data at the edge of the network, rather than sending it out to remote servers for processing. This paradigm helps not only in reducing costs, but also in improving the responsiveness of the application. Utilizing strategically positioned people counting devices in a retail store allows for the collection of valuable customer data [21]. Interpreting this data provides managers with insights into the store's performance and highlights areas that may require improvement. The retail stores may be benefited in the following ways:

* Optimize staff scheduling: Utilizing a door counter to gauge store traffic enables the store to identify peak hours and days, ensuring adequate staffing to assist customers during those busy times. Conversely, by analyzing foot traffic data, one can pinpoint periods with the lowest in-store visitors, and schedule only the necessary employees during those times.
* Customer behavior: Installing a cost-effective door counter near a store's entrance offers valuable insights into the number of customers entering on specific days and peak times. Examining foot traffic data provides a customer-centric perspective of a business. For example, one might observe consistent store traffic on weekdays with increased activity on weekends, or discover more visitors during midday compared to the afternoon. Empowered by this data, one can enact necessary modifications, such as hiring extra staff, or making adjustments to the store's operating hours.
* Leasing Valuation: A precise people counting solution provides one with the count of individuals entering and exiting malls, highlights popular and well-performing zones, and identifies frequently visited areas by customers. This data becomes instrumental in persuading tenants of the fairness of the rent, substantiating lease valuation claims with concrete numbers. One can leverage this information during lease negotiations and gain insights into how external factors, such as public holidays, impact consumer behavior.
* Plan ahead: A customer counter serves as a crucial instrument for strategic planning in a retail enterprise. By identifying peak hours, days, and even weeks, one can proactively prepare to make those periods as smooth and stress-free as possible for both the store and its customers.

Apart from these, fog computing can be advantageous in various other domains including smart vehicles, Smart Cities, Smart Buildings, Manufacturing, smart farming, Industry,

Ubiquitous Computing, and more.

1.1.4 Challenges in Fog Computing

While fog computing offers numerous benefits, it also presents certain challenges that must be addressed, in order to enhance its feasibility [22]:

- **Reliability and Fault tolerance:** Edge devices are susceptible to malfunctions, sporadic connectivity, and interruptions in the network [23]. Developing resilient fault-tolerant mechanisms to manage device failures, network disruptions, and ensuring the continuous provision of services becomes essential in fog computing environments.
- **Interoperability and Standardization:** The fog computing environment consists of various devices, protocols, and platforms [24]. Ensuring compatibility among distinct vendor-specific solutions and communication protocols, standardizing interfaces, and APIs poses a challenge, impeding the smooth collaboration and integration within the fog ecosystem.
- **Limited resource capability:** The computing and storage capabilities of edge resources are constrained in comparison to conventional data centers [25]. It is essential to utilize these resources efficiently to attain the highest possible overall system utilization.
- **Privacy and Security:** Given the distributed nature of the infrastructure in fog computing, emerging security and privacy concerns pose significant challenges [26]. Ensuring data integrity, safeguarding communication channels, preventing unauthorized access, and addressing privacy issues become crucial tasks in fog computing environments.
- **Resource Management and Orchestration:** Effectively coordinating and overseeing resources and services across diverse fog nodes presents a challenge [27]. Careful attention is required to address dynamic resource provisioning, load balancing, service discovery, and efficient task scheduling to guarantee optimal resource utilization and performance.

1.2 Research Objectives

Our overall objective in this thesis is to explore parallel and distributed Machine Learning model training and inference on edge networks. This objective ties together all the research problems in this thesis. The objectives of this dissertation are as follows -

- **To perform partitioned training and testing of Machine learning models on edge architectures**: In case of online model training and inference in edge networks requires a safe and reliable parallel computing architecture to

achieve improved performance with optimal resource utilization. To address this challenge, we propose an efficient machine learning model partitioning algorithm that considers the safety constraint and requirements of edge networks, and includes the triple-modular redundancy (TMR) technique for trusted computation.

- **To investigate dynamic hierarchical Intrusion Detection task offloading in IoT Edge Networks:** As the web of IoT is growing, more concerns about its security and privacy are becoming prevalent. IoT devices are endangered by various types of attacks, such as port scanning and man-in-the-middle attacks. Monitoring attacks using traditional intrusion detection approaches is computationally intensive, and requires significant storage space. IoT devices, being resource-constrained, may not be able to store data and analyze attacks in real time using these traditional intrusion detection approaches. We have proposed a lightweight intrusion detection system which is applicable for resource-constrained edge devices.

- **To perform data-driven Deep Neural Network task offloading on edge networks:** Deep Neural Networks (DNN) have exhibited good performance in the case of image-based classification, and regression problems. One of the main concerns of training these models well is the usage of a huge amount of data for learning. However, edge devices, being resource constrained by definition, may not be able to handle huge DNN workloads. Hence, the training of the DNN model needs to be done in a distributed manner in the edge network, so that the data is processed in real-time. We propose a framework for the optimal computation offloading of application data-points on various edge devices by proposing a Mixed Integer Programming (MIP) based approach, which minimizes the training time of the given workload, and maximizes the data-point processing ratio

- **To predict outcomes in cardiac surgery using AI models based on non-linear time series data:** The patients prediction tasks need to be performed in real-time, so that the medical staff can take immediate action. But, sending out data to a remote server or Cloud for analysis may introduce unwanted delays. Hence, we have performed the data analysis on an edge device which facilitated in rapid decision-making.

1.3 Contributions

This thesis proposes four fog network based algorithms, one for each above mentioned objectives.

The first work discusses the partitioning of training and testing algorithms for ML models on Edge architecture. The efficiency of the proposed work has been demonstrated in the result section by comparing the proposed PSVM-EA (Partitioning Support Vector Machine on Edge Architectures) framework with a non-partitioned and random partitioned approach on various datasets. In order to enhance the safety and reliability

of the online model training process, the Triple Modular Redundancy (TMR) technique has been incorporated for trusted computation.

In the second work, a fog based lightweight intrusion detection system FC-IDS (Fog Cluster-based Intrusion Detection System) is discussed. The experiments demonstrate that the proposed FC-IDS framework has recorded a very low response time and cost of deployment, in comparison to the remote Cloud server. The FC-IDS framework is meant for binary classification (i.e. attack or normal). Next, we have discussed the FCAFE-BNET framework which can perform multi-class classification. The detailed performance analysis of FCAFE-BNET has been done with respect to various state-of-the-art techniques.

The third work introduces the $D^2 - TONE$ algorithm which considers the network conditions and computational capacity of edge devices before offloading the tasks on the edge network. The computational and transmission times have been estimated using ML models based on certain features, which significantly improves $D^2 - TONE$ algorithm performance in comparison to other approaches.

The last work discusses the methodology of predicting outcomes in cardiac surgery based on non-linear time series data is discussed. These tasks need to be performed in real-time, so that the medical staff can take immediate action. But, sending out data to a remote server or Cloud for analysis may introduce unwanted delays, hence the data analysis has been performed on an edge device.

1.4 Organization of the Thesis

The thesis comprise of seven chapters. The chapter 2 reviews the literature on partitioned training and testing of Machine learning models on edge architectures, intrusion detection systems on fog architecture, various task offloading techniques on edge networks, and real-time analysis of patient health during cardiac surgery.

Chapters 3, 4, and 5 are dedicated to addressing partitioned training and testing of Machine learning models, intrusion detection systems, and various task offloading techniques on the proposed Fog computing frameworks respectively. The structure of these chapters follows a consistent pattern. Initially, each chapter provides a formal description of the problem statement, accompanied by an explanation of the motivation behind the proposed framework. The contributions of the framework are then outlined in the context of the challenges it addresses. Following this, a comprehensive methodology is presented. Each chapter also includes details about the experimental setup, datasets, and baseline algorithms used for performance comparison. Finally, the outcomes of the experiments are presented and analyzed across different settings and parameters.

Chapter 6 discusses, the real-time analysis of patient health during cardiac surgery. The structure of the chapter is described as follows: firstly, the chapter provides the motivation behind the proposed framework. Followed by a detailed methodology comprising of data collection, pre-processing, feature engineering, and feature importance. Lastly, the results section discusses the performance metrics, sociodemographic and Clinical Determinants, and performance evaluation of the proposed framework with other ML models.

The last chapter highlights the conclusion that we draw from our work. It discusses the different areas in Fog networks where scheduling plays an important role in maximizing system utilization and reducing the cost, as compared to using traditional remote data centers. It also emphasizes on the possible future extensions of this work.

Chapter 2

Related Work

The main goal of this thesis is to explore parallel and distributed Machine Learning model training and inference on edge networks. The idea is to leverage the edge for executing these ML tasks in a parallel and distributed manner, resulting in quicker training and inference. Specifically, security and healthcare have been selected as the application domains where our proposed algorithms have been tested. With this as the overall goal, this chapter provides a concise overview of cutting-edge advancements in the domain of fog computing, specifically focusing on Partitioning ML models, lightweight Intrusion Detection System, Data-Driven DNN Task Offloading, and real-time edge based analysis of patient health during cardiac surgery. It outlines the fundamental principles, constraints, and enhancements associated with these pivotal contributions in the field.

Dividing machine learning (ML) models for training on edge nodes is a method that includes spreading the training tasks among various devices or nodes, typically positioned at the network's periphery. This proves advantageous in situations where the data is scattered or when the model exceeds the capacity of a single device. In the past, researchers have tried to partition the ML model using various techniques, such as: data partitioning, model partitioning, and federated learning. When segmenting models for edge training, it is crucial to strike a balance between communication overhead, computational efficiency, and the overall performance of the training process. Experimentation and a thorough analysis of the particular use case are vital for determining the most efficient partitioning strategy. Below are some primary benefits of partitioning ML models on edge nodes for training and inference(testing):

- Real-Time Adaptation: Edge nodes facilitate the immediate adjustment of models in response to alterations in data distribution. This can proves especially advantageous in dynamic environments where the attributes of the data can change over time.

- Scalability: Edge nodes allow for decentralized training, even when a constant network connection is unavailable. Nodes have the capability to function autonomously and coordinate updates once connectivity is reestablished, enhancing the resilience of the training process.

- Resilience to Network Outages: Edge nodes support scalable training by distributing the workload among numerous devices. This promotes the effective utilization of resources and facilitates the incorporation of extra-edge devices, as required.

- Low-Latency Inference: As models are trained at the edge, they can be locally deployed for inference, resulting in predictions with minimal latency. This is essential for applications that demand real-time or near-real-time responses.

2.1 Partitioning ML models on edge architectures

A number of researchers [28, 29, 30] have proposed partitioning Deep Neural Networks (DNN) for speeding up ML model training. Guanghui Zhu et al. [31] propose Forest-Layer, which is a scalable and efficient partitioning mechanism for deep forests. The Ray platform was used to implement this distributed task-parallel system. The authors introduced a few optimization techniques at the system level in order to improve parallelization. Li Zhou et al. [32] proposed an algorithm which finds the optimal partitions of the network model for execution on the edge devices. The system recalculates the points of optimal partitions at certain intervals. The selection of the recalculation interval is crucial, as it can degrade the system's performance by increasing the rescheduling overhead. In addition, Md Maruf et al. propose a machine learning-based prediction for task offloading to minimize the task overload and meet the application requirements [33],[34]. The convolution split algorithm proposed by Shengyu Fan et al. [30] takes into account the size of the kernel, and then expands its feature map accordingly. Moreover, the use of sparse matrix-vector (SpMV) multiplication has improved the performance by increasing the speed and decreasing the memory consumption, while calculating the convolution layer.

Recent studies [35, 36, 37] explore a diverse array of techniques employed in optimizing machine learning models for edge networks. Table 2.1 shows the comparison of various methods, ranging from federated learning to multi-agent systems, focusing on computational speed-up, energy efficiency, and data privacy. Despite these advancements, there is less discussion on model safety and reliability during execution in edge networks, which we strive to address in this work. For example, Tan *et al.*[38] utilized a GPU-based parallel implementation of an SVM model, resulting in significant speed-ups in training times. However, the necessity for GPUs may not be realistic for edge networks, given their resource constraints and cost implications.

Likewise, Li *et al.* [39] implemented a federated learning approach that emphasizes privacy preservation by training a global model with local data. This method investigates collaborative data sharing in vehicular edge networks (VENs) with AI-empowered mobile/multi-access edge computing (MEC) servers. Furthermore, Zhou *et al.*[40] proposed FedACA, an adaptive, communication-efficient learning algorithm to reduce communication overhead in federated learning on edge devices. These methods, though remarkable, do not explicitly address safety and reliability during model partitioning.

Another study is the work of Zhao *et al.* [41], which designed a collaborative mobile

edge computing system involving multiple unmanned aerial vehicles (UAVs) and edge clouds (ECs). Despite its impressive results in minimizing execution delays and energy consumption, it largely overlooks the safety and reliability concerns inherent to model partitioning and integration.
Regarding safe execution, Gu *et al.*[42] present a 'Safe Fail' technique for machine learning models in cyber-physical systems, emphasizing safer decisions based on out-of-distribution (OOD) instance detection. In parallel, Hilbrich[43] proposes a 'correctness by construction' approach that safely utilizes task parallelism in multi-core embedded systems.

The majority of research in the literature concentrates on partitioning Deep Neural Networks (DNN), with limited emphasis on partitioning other fundamental machine learning algorithms. It is important to highlight that deep learning models have demonstrated superior effectiveness in numerous scenarios, particularly those involving image, audio, and text data. Nevertheless, traditional machine learning approaches remain relevant and may be preferred under specific applications like: Fraud Detection, Recommender Systems, Text Classification, etc. In contrast to these existing works, our proposed approach seeks to integrate the benefits of computational efficiency and safe execution by employing optimal ML Model Partitioning with TMR on edge systems. Triple Modular Redundancy (TMR) is a robust method employed in safety-critical contexts to attain elevated levels of fault tolerance and reliability. Through the replication of crucial elements and the comparison of their outputs, it guarantees consistent functionality even when faults or failures occur. This redundancy strategy is indispensable in sectors where the repercussions of system breakdowns can be dire, potentially safeguarding lives and averting catastrophic incidents.

2.2 Intrusion Detection System on Fog Architectures

Security concerns in edge networks are noteworthy because of the decentralized and distributed nature of edge computing. It is crucial to establish secure communication between edge devices and cloud services, encompassing safeguarding data during transit, and ensuring the integrity of communication channels. Edge networks may face vulnerability to Distributed Denial of Service (DDoS) attacks, where a substantial volume of traffic can inundate the network, leading to disruptions. Employing strategies to mitigate DDoS attacks is imperative to sustain service availability. Security measures need to be integrated into edge cloud environments to guard against data breaches, unauthorized access, and various other cyber threats. In the ongoing evolution of edge computing, maintaining a vigilant and proactive approach to addressing security issues is essential for preserving the overall integrity of edge networks.
Edge networks encompass communication among devices and potentially with central servers or the cloud. An IDS actively monitors network traffic, discerns suspicious patterns, and aids in averting unauthorized access, thereby bolstering overall network

Table 2.1: Comparison of our approach with different techniques in the literature

Paper	**Parallel Computing Technique**	**Devices**	**Safety/ Reliability**	**ML Models**	**Performance Evaluation**
Tan *et al.* [38]	CUDA and OpenMP	GPUs	No	SVM	Speedup of 18.5× in training, 81.9× in testing
Li *et al.* [39]	Federated Learning	MECSs	Implicit	DQN	Fast convergence, optimal data sharing, privacy protection
Zhou *et al.* [40]	FedACA	CPUs	Implicit	CNN, ResNet-18	Outperformed FedAsync by 4.20% to 8.04%
Zhao *et al.*. [41]	MATD3	UAVs	No	MATD3 with two hidden layers	Efficient task splitting and offloading, faster convergence
Hilbrich *et al.*. [43]	Multi-function integration	No	Resource validation	N/A	Improved resource assignment, task scheduling, system reliability
Thaha *et al.* [37]	DNN Partitioning and Offloading	Fog Nodes	No	DNNs (e.g., AlexNet, VGG)	Latency decrease 40%-60%, acceleration of 2.6 to 4.2 times
Proposed approach	Optimal ML Model Partitioning	CPUs	TMR (safety & Reliability)	SVM and RF	Speedup of 56.3%, Accuracies: 85%-90% (SVM), 82%-93% (RF)

security. In the event of a security breach, an IDS furnishes crucial information for incident response, aiding in the identification of the intrusion's source and nature. This enables organizations to implement suitable measures to mitigate the impact. Given the real-time processing demands of edge computing, an IDS at the edge is adept at swiftly detecting and mitigating security threats, averting potential disruptions and ensuring the uninterrupted functioning of critical applications. Unlike traditional IDS systems, which may strain resources and induce performance issues on resource-limited edge devices, tailor-made IDS solutions for edge computing can be devised to operate efficiently within the constraints of these devices. Recognizing the dynamic nature of edge environments, where devices may operate in diverse conditions, an IDS designed for edge computing can exhibit adaptability, accommodating the unique characteristics of edge devices and the ever-changing landscape of edge networks.

There are two primary types of Intrusion Detection Systems (IDS): Network-based Intrusion Detection Systems (NIDS), and Host-based Intrusion Detection Systems (HIDS). Each type has a specific role in monitoring and detecting potential security threats. Network-based Intrusion Detection Systems (NIDS) are created to observe and scrutinize network traffic to identify suspicious activities or patterns indicative of a security threat. Operating in a passive mode, NIDS analyze network packets, searching for anomalies or recognized attack signatures. They are commonly positioned at strategic locations within a network, such as network gateways or subnets. This placement allows NIDS to provide a consolidated perspective on network activity, enhancing their effectiveness in detecting attacks spanning multiple hosts or devices.

Host-based Intrusion Detection Systems (HIDS) concentrate on observing actions on individual hosts or devices, searching for indications of unauthorized access or irregular behavior. HIDS are directly installed on individual computers or servers, observing user activities, system logs, and file integrity. They can identify uncommon patterns or deviations from typical behavior. This affords an intricate perspective on activities on particular hosts, proving efficacious in identifying localized attacks and insider threats.

Several intrusion detection systems (IDS) have been proposed in recent years for monitoring attacks in the IoT network. Various ML techniques have been employed in order to improve IDS performance. Often, these systems experience poor performance because of a variety of reasons, which we discuss later in this section. Li et al. in [44] proposed an IDS using a cluster of Neural Networks (NN). The IDS uses Anomaly Behaviour Analysis (ABA-IDS) for ensuring secured fog node availability in the IoT network. The adaptive scheme has a high detection rate for various anomalies, like system glitches, cyber-attacks, and misuses with low overheads. In [45], authors proposed an Enhanced Hybrid IDS (EHIDS) to find the optimal set of weights and biases of Artificial NN (ANN) using a genetic algorithm. After obtaining the trained ANN, the model is deployed to the fog network for classifying attacks. The framework is verified using UNSW-NB15 and ToN_IoT datasets.

In [46], the authors proposed a multi-attack classification model ICNN-FCID for fog

networks by integrating Long-Short Term Memory (LSTM) with Convolutional NN (CNN). The ICNN-FCID approach has been verified using the benchmark NSL-KDD dataset. The IDS framework proposed in [47] has been developed by hybridizing various Machine Learning (ML) algorithms like KNN, Random Forest, Decision tree, and XGBoost. The proposed technique uses user behavior patterns for securing smart homes. The authors used NSL-KDD and CSE-CICIDS 2018 datasets for experimentation. In [48], the authors proposed an attack detection system named CPS-NIDS for a Cyber-Physical Network. The Principal Component Analysis has been used in order to select features. The authors used several ML models like SVM, Random forest, and Logistic regression on extracted features for detecting attacks. The proposed framework has been evaluated on various NIDS-based datasets, like WSN-DS, KDD-Cup-1999, CICIDS 2017, SDN-IoT, and UNSW-NB15. In [49], the authors have proposed a framework using the Persistent regularization algorithm. The Cholesky Factorization is applied using Online Sequential Extreme Learning Machines (CF-OSELM). The proposed approach has been utilized to detect IoT-based attacks in fog devices, sending the attack report to the centralized cloud for detailed analysis.

An ensemble based IDS has been proposed in [50] using Naive Bayes, Logistic Regression, and Decision trees. The CICIDS 2017 dataset has been used for evaluating the proposed technique for binary and multi-class classification. In [51], the authors employed Explainable AI techniques, such as Shapley Additive exPlanations (SHAP), RuleFit, and LIME in order to explain the classifier's decision. The classifier used for IDS was a Deep Learning NN. They have analyzed the proposed framework using the following datasets: UNSW-NB15 and NSL-KDD. In order to detect R2L (Remote to Local) and U2R (User to Root) attacks, the authors have used Bi-directional LSTM (Bi-DLSTM) [52]. The benchmark NSL-KDD dataset has been used for validating the proposed model. In [53], the authors have used the Gorilla Troops optimizer (GTO) method along with Bird Swarm algorithm (BSA) for feature selection. The proposed GTO-BSA has been used for finding the optimal solution of features. The authors have compared the performance of GTO-BSA with other meta-heuristic algorithms along with the GTO algorithm [54]. The study shows that the GTO-BSA approach outperforms all of the other meta-heuristic algorithms.

In [55], the authors have proposed a host-based intrusion detection system using Multi-Layer Perceptron (MLP). The feature space has been reduced using n-gram transformation based on vector space representation. The proposed model has been tested on ADFA-LD and ADFA-WD. The experimentation has been carried out using raspberry pi as a fog device. The power consumption of fog devices has been estimated using voltage and current demand. In [56], the authors have used multiple fog nodes in a local area network for detecting attacks using ML algorithms. The ML tasks are offloaded to the fog cluster for faster inference. The experiments were conducted using various ML algorithms, and the best results were obtained using the XGBoost model. The experiments show that the latency and cost of deployment of the raspberry pi cluster are much less than that of the cloud server. The authors have validated the framework performance

Table 2.2: Comparison of various methodologies with the Proposed Technique.

Methodology	Edge-cloud collaboration	Network conditions considered	Applicable for HIDS/NIDS	Applicable for real-time applications	Binary/ Multi-class classification
FCAFE-BNET (Proposed)	Yes	Yes	Both HIDS & NIDS	Yes	Multi-class
EHIDS [45]	No	No	NIDS	Yes	Binary
ABA-IDS[44]	No	No	HIDS	No	Binay
CF-OSLEM [49]	No	No	HIDS	No	Binary
ICNN-FCID [46]	No	No	NIDS	No	Multi-class
CPS-NIDS [48]	No	No	NIDS	No	Binary
Ensemble IDS [50]	No	No	NIDS	No	Multi-class
BiDLSTM-IDS [52]	No	No	NIDS	No	Multi-class
Explainable-IDS [51]	No	No	NIDS	No	Binary
LW-MLP [55]	No	No	HIDS	Yes	Binary
FC-XGB [56]	No	No	HIDS	Yes	Binary

on the ADFA-LD dataset. In [57], the authors combined various ML classifiers such as Random forest, KNN, and Decision trees in order to build an IDS. The anomaly detection has been done on fog devices, whereas the attack classification has been carried out on the cloud server. The authors used KDDTest-21 and KDDTest+ for analyzing the proposed approach. In [58], the authors used the Random forest model for classifying attacks in a network. The features were analyzed and selected manually, based on the characteristics of the attacks. The performance of the classifier was validated using NSL-KDD and KDD-Cup99. In [59], the authors proposed a real-time IDS, using auto-encoder and isolation forest. The Auto-IF technique has been tested on fog devices using NSL-KDD for binary-class classification settings.
In Table 2.2, we compare our proposed approach (FCAFE-BNET) with some of the recent state-of-the-art techniques. Despite the fact that various ANN based IDS approaches have been proposed in the past few years, the above discussed methodologies have the following shortcomings that need to be addressed:

- The feature selection approach adopted by these methodologies, such as the wrapper method, and the filter method are quite outdated and fail to capture various sensitive features. Due to this, the classifier gives a poor performance on multi-class classification.
- The previous Intrusion detection work are either applicable to NIDS or HIDS.
- The methodologies proposed previously do not consider network conditions and network congestion. Also, the recent methodologies use either a fog device or the cloud for identifying attacks.

2.3 Data-Driven DNN Task Offloading on Edge Networks

Task offloading is one of the most crucial decisions in an edge network. Several offloading approaches have been proposed in the past, which can be broadly classified into two categories: (a) Mathematical optimization, and (b) Artificial intelligence (AI) based algorithms. The mathematical optimization approach can be carried out in the following ways: i) Mixed integer programming (MIP), ii) Game theory, and iii) Heuristics. A comparison of various offloading approaches is given in Table 2.3. The MIP approach helps in optimizing multi-objective functions having different offloading constraints like energy consumption, communication delay, and latency, based on the underlying motivation for research [60], [61]. The MIP approach helps in finding the optimal offloading solution satisfying given constraints. In [62], the author's objective is to minimize the energy consumption of mobile devices with latency constraints in a multi-user system. The MIP objective in [63] includes network delay and processing time for IoT-based mission-critical applications. Similarly, in [64], the authors schedule real-time vehicular tasks based on deadlines on appropriate processors with the objective of minimizing communication delay. In [65], the authors offloaded real-time tasks based on security

and deadline constraints. In [66], the authors proposed dynamic user allocation in stochastic edge networks using Lyapunov optimization algorithm. However, the authors did not consider various communication/transmission aspects (like packet loss, jitter), which are very crucial in real-world scenarios. The MIP approaches assist in providing optimal or near-optimal solutions for offloading tasks. However, all the above approaches fail to consider dynamic network conditions while making offloading decisions. As the above-discussed approaches use static estimating techniques for modeling the inputs in the Mixed integer programming approach, inaccurate estimations might degrade the offloading decisions over time. In [67], [68] and [69], the authors employed game theory for optimizing the revenue for edge/cloud providers, maximizing efficiency of resource allocation, and maximizing spectrum efficiency respectively. However, the game theory approach is incapable of handling dynamic network conditions. Also, when the number of users increases significantly, then the game theory approach increases in complexity. Various heuristic-based approaches for task offloading have been proposed in recent years. The heuristic offloading approach proposed in [70] uses the transmission channel properties and energy consumption models of transmission and computation to find the offloading scheme. The authors consider energy and time constraints for solving the computational offloading problem. In [71], the authors have used an offloading algorithm that tries to adapt the dynamic behavior of the edge network by taking into account the residual energy of mobile devices present in the network. However, the proposed approach fails to minimize the offloading cost. In [72], authors used the Markov random field approach for balancing the workload and lowering energy consumption in edge networks. However, the proposed approach fails to accurately estimate user density, which can be resolved by integrating ML techniques. Also, the framework suffers from scalability issues. In [73], authors equally distribute the data-points to all the devices present in the network, for performing distributed Stochastic Gradient Descent (SGD) in a synchronous manner. In this approach, the author trains the ML model in a distributed manner using Synchronous SGD, without considering the heterogeneity of the edge devices. The authors in [74] claim to provide an effective workload balancing solution for an IoT network with homogeneous servers using the balls and bins theory. Though the cost of offloading is low, the approach does not address the heterogeneity of mobile devices. In addition, the binary decision of offloading is too simplistic to capture the complexities of the edge network. Though the overhead of handling user requests in the case of the heuristic approach is negligible, the performance exhibited by various heuristic algorithms varies drastically in dynamic edge environments. Therefore, this approach needs to be investigated carefully for finding optimal offloading solutions for the edge paradigm.

The mathematical task offloading approaches discussed so far may fail to handle dynamic network situations while allocating tasks to edge resources. Specifically, these mathematical solutions may be incapable of capturing varying conditions in the end-to-end network model. Therefore, many ML-based approaches like linear regression and logistic regression have been proposed to make offloading decisions by using historical data to learn

Table 2.3: Comparison of various offloading approaches.

	MIP	ML	Heuristics	Game theory	Proposed (MIP+ML)
Low complexity	✓		✓		✓
Optimality	✓			✓	✓
Real-Time Decision	✓	✓	✓		✓
Capture Dynamic Network Conditions		✓			✓
Long-term solution		✓			✓

useful behaviors and patterns of the dynamic network [75], [76]. Resource monitoring tools have been used for collecting huge amounts of data in cloud and edge environments. In [77], the authors have used support vector machines for efficiently utilizing energy in a cloud environment. The Deep learning approach has been used in [78] in order to minimize the offloading time and the computation overhead in a given network. In [79], the authors employ the K-Nearest Neighbour scheme for reducing latency and energy consumption in the cloud. These ML approaches help to provide offloading solutions in real-time, but may fail to provide an optimal solution in multi-edge networks.

In order to holistically adapt dynamic network situations and provide long-term optimal/near-optimal offloading solutions in real-time, we employ ML algorithms for enhancing the mathematical task offloading optimization solution with various constraints in a multi-edge environment. However, the MIP-driven approach may be expensive for large-scale scenarios [80]. In order to address this shortcoming, we have used a branch-and-bound based solution, due to which the feasible offloading solution is generated in real-time.

2.4 Non-linear time-series based AI model to predict outcomes in cardiac surgery

In cardiac surgery, outcome prediction tools can be beneficial in ensuring continuity of care and planning resource allocation. As medical information systems and artificial intelligence technologies have advanced, ML algorithms have become increasingly valuable for individualised medicine [81]. If the outcome could be predicted accurately, clinicians could offer more effective treatment strategies to patients following cardiac surgery.

Cardiovascular surgery is considered a challenging operation to perform, as it adversely affects circulation and physiology [82]. In cardiac surgery, there has been an increasing interest in risk prediction models for clinical use. European decision-making guidelines cite several risk stratification methods, although these scores cannot replace clinical judgment and multidisciplinary discussions. The original EuroSCORE, EuroSCORE II, and STS scores are the most widely used scores for predicting mortality after cardiac surgery. However, some studies have shown that these scores have limitations in some surgeries or patient groups [83].

Rather that executing these prediction tasks on third party cloud service providers, the tasks for predicting patient outcomes can be performed on private edge devices, leading to quicker prediction times. Doing so would also address the privacy issues involved with a third party public cloud provider.

Many studies have examined mid-term or long-term mortality after cardiac surgery. Wu et al. [84] developed a risk score that predicted mortality following isolated CABG surgery with a C-statistic ranging from 0.768 to 0.783 for mortality at 1, 3, 5, and 7 years of follow-up. The application of ML approaches has been increasing due to the need for more precise prediction models. A recent meta-analysis of 15 studies indicated that ML models provide better discrimination when compared with conventional LR models when predicting operative mortality after cardiac surgery [85].

Models with ML show potential for capturing non-linear relationships and interactions among features without the need to specify all interactions manually, as with LR. Furthermore, ML algorithms are more efficient than traditional statistical methods because they do not rely on assumptions about data distribution and can perform more complex calculations. ML-based clinical models predicting short-term mortality in cardiac surgery have demonstrated AUC values between 0.74 and 0.79 [83]. In cardiac surgical operations, Zhou et al. [86] and Ong et al. [87] discovered that RF models predict short-term mortality better than other models. Furthermore, multiple investigations found that the XGBoost technique outperformed other ML algorithms in predicting surgical or in-hospital mortality [83]. S.Angraal et al. [88] predicted the mortality and hospitalisation in heart failure by using various ML models. The best AUC (0.72) is achieved in their work by using the RF model. Some of the features used by the model for predicting mortality over 3-years are blood urea nitrogen (BUN) level and body mass index. The RF model achieved a recall value of 0.70 for mortality. In [89], Koponen et al. have used various statistical analysis techniques, such as t-test and z-test, for comparing patient characteristics and clinical characteristics of outcome groups to assess mortalities. The proposed statistical approach achieves an AUC value of 0.70 for mortality up to 1-year. Ruan et al. [90] have proposed a general-purpose representation approach using RNN based denoising autoencoder (RNN-DAE) to summarise electronic health records. By using the RNN-DAE method, the proposed approach achieves an AUC (0.78), accuracy (0.77) and an F1 score (0.44).

All the work discussed for mortality prediction suffers from the following limitations.

Firstly, the work present in the literature uses either statistical techniques or classical non-linear ML models like SVM, RF, DT, and XGB. However, researchers have not examined linear ML models (probabilistic models). Secondly, the discussed approaches have used patient characteristics or clinical characteristics, i.e. static data, to classify the patient's mortality. Lastly, the discussed approaches have used either oversampling or undersampling techniques for handling imbalance. In our proposed approach, we have examined Linear ML models (probabilistic algorithms), such as GNB, BNB, LDA, and LR, and classical non-linear models. It was found that probabilistic algorithms offer better performance than non-linear models. Further, we used time-series data instead of just clinical test reports (static data) for mortality prediction. Also, we have examined the performance of classifiers with combined oversampling (SMOTE) and undersampling (Near-Miss) techniques. The proposed approach has shown significant improvement in the performance of the classifier. The improvement was because probabilistic algorithms like GNB and LR are critical to handling uncertainties caused due to insufficient data. Thus, these algorithms have a high potential to perform well in an imbalanced classification problem.

Chapter 3

A Framework for Partitioning ML Models on Edge Architectures

3.1 Introduction

Current IoT applications generate huge volumes of complex data that requires agile analysis in order to obtain deep insights, often by applying Machine Learning (ML) techniques. A support vector machines (SVM) is one such ML technique that has been used in object detection, image classification, text categorization, and Pattern Recognition. However, training even a simple SVM model on big data takes a significant amount of computational time. Due to this, the model is unable to react and adapt in real-time. There is an urgent need to speedup the training process. Since organizations typically use the cloud for this data processing, accelerating the training process has the advantage of bringing down costs. In this work, we propose a model partitioning approach that partitions the tasks of Stochastic Gradient Descent based Support Vector Machines (SGD-SVM) on various edge devices for concurrent computation, thus reducing the training time significantly. The proposed partitioning mechanism not only brings down the training time, but also maintains the approximate accuracy over the centralized cloud approach. With a goal of developing a smart objection detection system, we conduct experiments to evaluate the performance of the proposed method using SGD-SVM on an edge based architecture. The results illustrate that the proposed approach significantly reduces the training time by 47%, while decreasing the accuracy by 2%, and offering an optimal number of partitions.

With the increasing demand for edge computing in cyber-physical system (CPS) applications, ensuring the safety and reliability of machine learning models running on edge devices during online model training and inference is essential. Although data and model parallelism offer significant advantages for large machine learning model training, adopting parallel computing architecture in edge networks is challenging. It introduces safety concerns while splitting and integrating machine learning models over different computing nodes, which can pose risks to the integrity and reliability of the system. Therefore, online model training and inference in edge networks requires a safe parallel computing architecture to achieve improved performance with optimal resource utilization. To address this challenge, we propose an efficient machine learning model partitioning algorithm that considers the safety constraint and requirements of edge networks,

and includes the triple-modular redundancy (TMR) technique for trusted computation. Compared to the non-partitioning approach, our proposed approach achieves a significant speedup of approximately 56.3% in net training time, making it more efficient and suitable for real-time applications in edge networks.

Recent advancements in edge networks have led to a paradigm shift in data processing across various CPS applications, such as robotics, health monitoring, and autonomous driving systems [91]. The significant increase in machine learning (ML) use, known for its real-time decision-making and cost reduction attributes, has established an essential role for edge networks in facilitating ML algorithms, such as Support Vector Machines (SVM), Random Forests (RF), and Deep Neural Networks (DNNs) for continuous adaptation in CPS applications [92]. Simultaneously, the complex task of online model training presents substantial challenges, including model parallelization/workload partitioning, safety concerns from splitting ML models across edge devices, and legacy single-core processor designs [93]. These challenges complicate the acceleration of ML model performance, and potentially risk system integrity and reliability, thus emphasizing the need for safe parallel computing architectures for efficient model partitioning and integration across edge devices.

Recent studies have explored various techniques to optimize edge-based machine learning model training for CPS applications [35]. While these approaches offer promising solutions, they primarily focus on data and model parallelism, without adequately addressing the safety and reliability concerns associated with ML model partitioning. Furthermore, in the works of Wen Sun *et al.*, [94], Guangxu *et al.* [95], and Sina *et al.* [96], the primary emphasis is on reducing training time and offloading training; however, these solutions have less discussion in the context of safety and reliability of the online model training process. Additionally, these approaches do not comprehensively address the issues of ML model distribution across edge devices, leading to potential resource inefficiency and system vulnerabilities. In this context, an approach that establishes optimal model splits, ensures safety and reliability, and aligns with safety standards like IEC 61508, ISO 26262, and UL 4600 is essential.

This paper proposes a framework for efficiently partitioning machine learning model splits for online training on edge networks, considering their safety constraints and requirements. Our framework aims to minimize training time and communication latency, ensuring that ML models are reliably trained and updated on edge devices without compromising safety, performance, or resource utilization. The framework includes an algorithm that intelligently partitions ML models into smaller sub-models that can be safely executed across multiple edge devices, leveraging their parallel computing capabilities.

Further, to enhance the safety and reliability of the online model training process, our approach incorporates the TMR [97] (Triple Mode Redundancy) technique for trusted computation. TMR is an established Single Event Upset (SEU) technique that replicates the processing of each sub-model across three separate devices, allowing for error detection and correction in the event of any discrepancies between their outputs. By employing

TMR, the proposed algorithm ensures that the system maintains its integrity and reliability, even in the presence of potential faults, hardware compromises, or other safety issues. The framework dynamically manages resources and minimizes training time and latency to optimize performance, while ensuring safety and reliability in edge networks. We evaluate the effectiveness of our proposed approach through extensive experimentation and analysis, demonstrating significant improvements in system performance, safety, and reliability for online model training and inference in edge networks. We conduct a case study on SVM and RF multi-class classifiers by splitting the models into multiprocessor edge devices. The experimental results demonstrate a significant reduction in training time and increased system throughput, without compromising accuracy. The results highlight the potential of combining model partitioning and TMR to address the challenges associated with safe online ML model training on edge networks, paving the way for further exploration and development of safe and reliable edge computing solutions in the evolving landscape of cyber-physical systems.

3.2 Problem Statement

In the context of online model training in edge networks, our goal is to optimize the partitioning of the machine learning model splits across edge devices, balancing the trade-offs between training time, communication latency, and TMR time.

3.2.1 Optimization Problem

The optimization problem can be written as:

$$\min_{s_k \in S, \forall k} \quad \sum_{m_i \in s_k} t_{m_i} + \beta \sum_{m_i \in s_k} l_{m_i} + \gamma \sum_{k=1}^{3} T_{TMR_{k,m_i}} \tag{3.1}$$

The equation 3.1 has three components. The first term in the optimization problem represents the total training time for all splits assigned to edge devices in each partitioning strategy. The second term denotes the total communication latency for all splits assigned to edge devices in every partition, capturing the costs associated with transmitting data between devices. The third term corresponds to the total time spent on TMR, which ensures the system's reliability by incorporating redundancy into the distributed ML model. We introduce two weighting factors, β and γ, to balance the trade-offs between these components:

- β: This weighting factor balances the importance of communication latency in the optimization problem. A higher value of β emphasizes minimizing latency, whereas a lower value focuses more on minimizing training time.
- γ: This weighting factor balances the importance of TMR time in the optimization problem. A higher value of γ emphasizes minimizing TMR time, whereas a lower value focuses on balancing training time and communication latency.

3.2.2 Constraints

- **Model Partition Constraint:** This constraint ensures that the ML model is properly partitioned across the edge devices.

$$\bigcup_{i=1}^{m} s_{k,i} = M; \quad \forall k = 1, \ldots, |S| \tag{CT1}$$

Constraint (CT1) ensures that the union of all model splits across all partitions and edge devices covers the entire set of model splits, which is equal to the full model M.

$$S = \{s_1, s_2, \ldots, s_n\} : s_k \in S, k = 1, \ldots, n \tag{CT2}$$

Here, s_k represents the k^{th} partition assigned to different edge devices. The set S contains all possible assignments of partitions to edge devices. For our example with 6 model splits (m_1, m_2, m_3, m_4, m_5,m_6) and 3 edge devices, the set S could contain the following possible partitions:

$$\begin{aligned} S = \{&(\{m_{1,1}, m_{2,1}\}, \{m_{3,2}, m_{4,2}\}, \{m_{5,3}, m_{6,3}\}), \\ &(\{m_{1,1}, m_{2,1}, m_{3,1}\}, \{m_{4,2}\}, \{m_{5,3}, m_{6,3}\}), \\ &\vdots, \\ &(\{m_{1,1}, m_{2,1}\}, \{m_{3,2}, m_{4,2}, m_{5,2}\}, \{m_{6,3}\})\} \end{aligned}$$

So, tuple $s_k = (s_{k,1}, s_{k,2}, ..., s_{k,e})$, where e is the number of edge devices, and $s_{k,i}$ is the set of model splits assigned to the i^{th} edge device in the k^{th} partition. The optimization problem aims to find the best assignment of partitions ($s_k \in S$) to minimize the total training time and communication latency, while considering TMR.

- **Processing Capability Constraints:**

These constraints ensure that the requirements of assigned model splits (processing, memory, and bandwidth) do not exceed each edge device's capability.

$$\sum_{m \in s_{k,i}} w_m \leq p_i; \quad \forall k = 1, \ldots, |S|; \quad i \in n \tag{CT3}$$

Constraint (CT3) ensures that the processing requirements (w_m) of assigned model splits do not exceed a device's processing power (p_i).

$$\sum_{m \in s_{k,i}} R_m \leq R_{M_i}; \quad \forall k = 1, \ldots, |S|; \quad i \in n \tag{CT4}$$

$$\sum_{m \in s_{k,i}} b_m \leq b_{B_i}; \quad \forall k = 1, \ldots, |S|; \quad i \in n \tag{CT5}$$

Constraint (CT4) guarantees that memory requirements (R_m) of assigned model splits do not exceed a device's memory capacity (R_{M_i}). Similarly, Constraint (CT5) makes sure that the bandwidth requirements (b_m) for transmitting assigned model splits do not exceed a device's bandwidth capacity (b_{B_i}).

- **TMR and Safety Constraint:**

$$\sum_{k=1}^{e} x_{k,j,m_i} = 1; \quad j = 1, \ldots, 3 \tag{CT6}$$

In this constraint, x_{k,j,m_i} represents the binary decision variable for the m_i-th model split assigned to the k^{th} edge device in the j^{th} TMR instance. The constraint ensures that exactly one edge device is assigned to each TMR instance for the m_i-th model split, which is important for ensuring the reliability of the TMR configuration and preventing errors or failures, contributing to the system's overall safety by minimizing the likelihood of incorrect or damaging outputs.

$$\sum_{k=1}^{e} x_{k,j,m_i} T_{TMR_{k,m_i}} \leq T_{fail,m_i} \tag{CT7}$$

The constraint CT7 ensures that the total time for training the redundant models and performing TMR in each TMR instance for the m_i-th model split does not exceed the specified failure threshold T_{fail,m_i}, determined by system designers or domain experts.

3.3 Contributions

The contributions of this chapter can be summarised as follows:

- The proposed framework partitions the weight update operation on multiple edge nodes, in order to train the SGD-SVM model in a parallel and distributed manner. This significantly reduces the training time, without significantly affecting the accuracy.

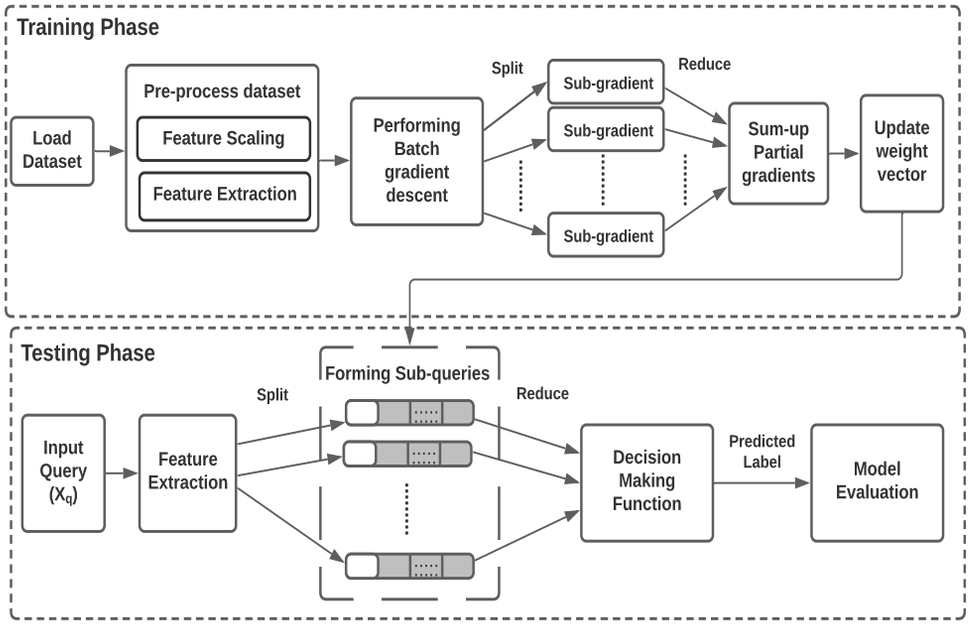


Figure 3.1: Flowchart of Proposed framework.

- The testing of the proposed approach is done in a parallel manner by partitioning the vector multiplication of all the features on the edge architecture.
- The proposed model partitioning approach has been analyzed on various data-sets in order to determine the extent to which various characteristics of the dataset affect the training accuracy, test accuracy, and run-time.
- A machine learning model partitioning algorithm that determines the optimal number of model splits while considering the safety constraints and requirements of edge networks, ensuring efficient and secure parallel execution of ML models across multiple edge devices.
- The integration of the TMR technique to enhance the safety and reliability of the online model training process by incorporating only trusted computation into the model during the execution of partitioned sub-models on edge devices.

3.4 Part A - Partitioning SVM Models on Edge Architectures

3.4.1 System Model

Consider a dataset $D = \{x_i, y_i\}_{i=1}^{N}$, where $x_i \in R^d$ is an input data instance, and $y_i \in \{1, \ldots, C\}$ is a class label for C classes. The SGD-SVM model denoted by M is trained on input data D. This thesis aims to design a framework for partitioning the SGD-SVM model on an edge architecture for parallelization and faster computation. Therefore, this

Table 3.1: Notations

π	Hyperplane / decision boundary
R	Number of edge devices
w_i	Weight of i^{th} feature
M	Machine learning model
α	Learning rate
λ	Regularization constant
ξ	Cost variable
E_i	Number of epochs
D	Dataset
x_i	Input data
y_i	Class label for i^{th} instance
T_o	Class label for test dataset
D_o	class label for train dataset
w_{edge_R}	Weight updation at R^{th} edge device
D_{edge_R}	Sub-query result at R^{th} edge device

thesis problem statement is to partition the training and testing of the model *M*, on *'R'* edge nodes, for which the training time is minimized and the model accuracy is maximized. Notations used throughout this work are defined in Table 3.1

The dataset is first pre-processed by scaling and extracting various features. Then, this processed data is split into training and testing datasets. As shown in Figure 3.1, the model is partitioned and then trained and tested on various edge devices. The partitioning of weight updation tasks is carried out to balance the load among all the edge nodes. This reduces the training and testing time significantly. The partitioned training and testing of SGD-SVM model is discussed in section 3.4.2.1 .

3.4.2 Proposed Approach

This section discusses in detail the partitioned training and testing of the proposed SGD-SVM model on edge architectures.

3.4.2.1 Partitioned Training Algorithm

The objective of training the SVM model by partitioning it on various edge devices with $'n'$ features is to distinctly classify all the data points present in the dataset using an n-D hyperplane. The hyperplane is basically a decision boundary which separates the data into two classes. The number of features used in the model defines the dimensionality of the hyperplane. There might exist many such hyperplanes which separate the data into two classes. However, the motivation is to find that optimal hyperplane π which minimizes the following loss function:

$$L(w,b) = \min_{w^*,b^*} \sum_{\mathrm{i}=1}^{\mathrm{n}} \max(0, 1 - \mathrm{y_i}(\mathrm{w^T}.\mathrm{x_i} + \mathrm{b})) + \lambda. \parallel w \parallel^2$$
$$\text{subject to} \quad \xi_i \geq 0. \tag{3.2}$$

Here, λ is the regularization constant & $\xi_i = max(0, 1 - y_i(w^T.x_i + b))$. Basically, λ is a hyper-parameter. A very low value of λ leads to over-fitting of the trained model. Next, we take the partial derivatives of the loss function with respect to the weights, as shown by the following equation:

$$\frac{\partial L}{\partial w_R} = \frac{\partial}{\partial w_R}\lambda. \parallel w \parallel^2 + \frac{\partial}{\partial w_R}(1 - y_i(w^T x_i + b)) \tag{3.3}$$

Algorithm 1: Partitioning Training Algorithm for SGD-SVM Model

Input : Dataset D, Train_out D_o, Number_of_epochs N, No_Of_EdgeNodes R
Output: Machine_Learning_Model M
$Model$, M = Parameter lists $W_i, i \in 0, 1, ..., n$

Initialize $W_i = 0$, $\alpha = 0.001$, $\lambda = 2 * 1/epochs$

Procedure `Splitter(`*D, M*`)`

- **while** *epochs* $\lesssim N$ **do**
  - $y_i = \sum_{i=0}^{p} W_i * x_i$
  - $mul = y_i * D_{oi}$
  - j=0
  - **for** *each data point j in Dataset D* **do**
    - **if** *mul* $\geq$ *1* **then**
      - cost$\leftarrow$ 0
      - $w_{edge\_1} = \sum_{i=1}^{n/R} w_i - \alpha * (\lambda * w_i)$
      - $w_{edge\_2} = \sum_{i=n/R+1}^{2n/R} w_i - \alpha * (\lambda * w_i)$
      - $\vdots$
      - $w_{edge\_R} = \sum_{i=(R-1)n/R}^{Rn/R} w_i - \alpha * (\lambda * w_i)$
    - **else**
      - $cost \leftarrow 1 - mul$
      - $f_{ij} = x_{ij}.D_{oj}$
      - $w_{edge\_1} = \sum_{i=1}^{n/R} w_i + \alpha.(f_{ij} - \lambda.w_i)$
      - $w_{edge\_2} = \sum_{i=n/R+1}^{2n/R} w_i + \alpha.(f_{ij} - \lambda.w_i)$
      - $\vdots$
      - $w_{edge\_R} = \sum_{i=(R-1)n/R}^{Rn/R} w_i + \alpha.(f_{ij} - \lambda.w_i)$
    - **end**
    - $j = j + 1$
  - **end**
  - $epoch = epoch + 1$
- **end**

The partial derivative of the regularization part is represented by the following equation:

$$\frac{\partial}{\partial w_R}\lambda. \parallel w \parallel^2 = 2.\lambda.w_R \tag{3.4}$$

The partial derivative of the penalty part is represented by the following equation:

$$\frac{\partial}{\partial w_R}(1 - y_i(w^T x_i + b)) = \begin{cases} 0, & \text{if } y_i(w^T.x_i + b) \geq 1 \\ -y_i.x_{iR}, & \text{else} \end{cases} \tag{3.5}$$

Algorithm 2: Partitioning Testing Algorithm for SGD-SVM Model

Input : Test Dataset D_{test}, Test_out T_o Machine Learning Model M, No_Of_EdgeNodes R
Output: Accuracy L
$Model$, M = Parameter lists W_i , $i \in 0, 1, ..., n$
class=[]
Procedure `DecisionFunc(`*D, M*`)`

```
for X_q ∈ D_test do
    D_edge_1 = Σ_{i=1}^{n/R} w_i * x_qi
    D_edge_2 = Σ_{i=n/R+1}^{2n/R} w_i * x_qi
            ⋮
    D_edge_R = Σ_{i=(R−1)n/R}^{Rn/R} w_i * x_qi
    D_Reduce = D_edge_1 + D_edge_2 + ... + D_edge_R
    if D_Reduce > 1 then
        class.append(1)
    else
        class.append(-1)
    end
    if T_o == class[i] then
        po = po + 1
    else
        ne = ne + 1
    end
end
acc = po / (po + ne) * 100
return
```

In case the model predicts the class of the query point correctly, we need to update the weights with the regularization gradient (equation 3.4) only. The following equation shows the weight updation for correctly classified points:

$$w = w - \alpha.(2\lambda w) \tag{3.6}$$

In case the model predicts the class of the query point incorrectly, we need to update the weights with the regularization gradient (equation 3.4) and the loss gradient (equation 3.5). The following equation shows the weight updation for misclassified points:

$$w = w - \alpha.(2\lambda w - y_i.x_i) \tag{3.7}$$

The weight updation process is carried out on various available edge nodes (as shown in Algorithm 1). For a given dataset, this process of updating weights is carried out in a parallel and distributed manner for several epochs until we get an optimized SVM model.

Table 3.2: Details of different datasets

Dataset Name	Instances	Features	Format	No. of classess
Iris [98]	154	4	Text	3
Traffic signs [99]	39,209	3x32x32	Images	43
CIFAR-10 [100]	60,000	3x32x32	Images	10
Fruits [101]	10,901	3x110x110	Images	6

The next section discusses about deploying the above trained model on different edge nodes for testing the trained SGD-SVM model.

3.4.2.2 Partitioned Testing Algorithm

After training using the proposed approach, the SVM model is used to predict future query data '$X'_q \in D_{test}$. The model's testing is done in a parallel and distributed manner. Vector multiplication is done among the query data and respective weights of the feature on 'R' edge devices (shown in Algorithm 2). These sub results on each edge node are represented as D_{edge_1}, D_{edge_2},.....,D_{edge_R}. The sub results reduce to 'D'_{Reduce}, after summation. If this value is greater than 1, then the data point is classified as a positive (+1) class. Otherwise, if the reduced value 'D'_{Reduce} is less than 1, then the data point is classified as a negative (-1) class. In order to calculate the accuracy of the model while testing, the following formula is used:

$$acc = \frac{po}{po + ne} * 100 \tag{3.8}$$

Here, po depicts the correctly classified data points in the test data, and ne depicts the misclassified data points in the test data.

3.4.3 Experimental Results & Analysis

3.4.3.1 Implementation setup

All the experiments were conducted on five raspberry pi 4GB devices acting as edge nodes. Each raspberry pi is equipped with 4 GB RAM, quad core Cortex-A72 (ARM v8) 64-bit SoC along with a 1.5GHz processor frequency. The proposed method was implemented using Python language (version 3), and the MXNET frameworK [102] has been used for processing the ML tasks in a parallel and distributed manner. The built-in socket class of python has been used for networking. Pandas is used for analyzing, cleaning, exploring, and manipulating data. The Sklearn library is used for machine learning and statistical modeling including classification, regression, and dimensionality reduction. MatPlotLib is used for creating high-quality visualizations and graphs. Whereas, Numpy is used to perform a wide variety of mathematical operations on arrays.

3.4.3.2 Dataset

Experiments have been performed using four datasets listed in Table 3.2. All the datasets vary in terms of dimensionality, size, number of classes, and similarity between the present classes. These datasets are as follows: Iris dataset [98], Traffic signs dataset [99], CIFAR-10 dataset [100], and Fruits dataset [101]. We now describe each dataset briefly.

- Iris Dataset: This dataset consists of 150 instances. The number of attributes for each instance are 4 - sepal length, petal length, sepal width, petal width. There are 3 classes, and each class has 50 instances. The task is to classify the plant based on its 4 attributes.

- Traffic Dataset : The classes in this dataset are unevenly distributed. Some classes have 250 instances, while other classes have 2500 instances. Due to this, the dataset is highly imbalanced. Each instance is a 32 x 32 tiny colour image.

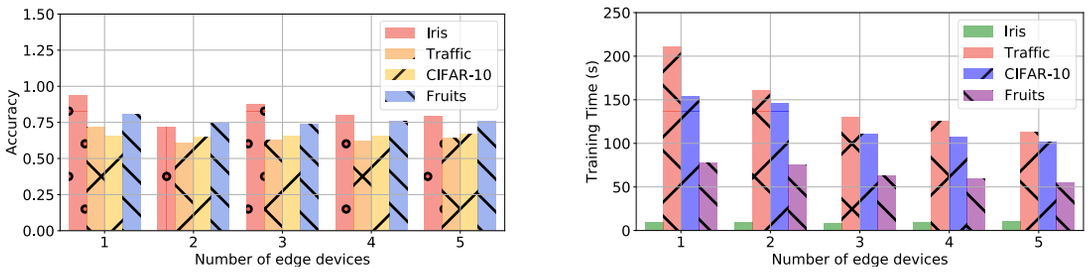
- CIFAR-10 Dataset: This dataset has 10 classes, such as - birds, cats, ships, horses, trucks etc. Therefore, the dataset consists of mutually exclusive, and very distinct classes.

- Fruits Dataset: This dataset consists of 110 x 110 colour images of fruits. The task is to classify whether the fruit present in the image is fresh or rotten. This dataset comprises of 6 classes.

3.4.4 Results & Discussion

3.4.4.1 Effect of Number of edge devices on accuracy & training time

This experiment shows how well the proposed approach performs against different sizes of datasets. The proposed method's behaviour for the SVM model is investigated using the four datasets that are specified in Table 3.2. This experiment runs the SVM model after partitioning it into different edge nodes, starting from one to five, for ten epochs. Figure 3.2 demonstrates the corresponding results for accuracy and training time for partitioning the model incorporating the proposed approach. In Figure 3.2 (a), it is observed that the model training accuracy slightly varies for the different number of partitions, for all the datasets. But, if we look into the corresponding dataset training times in Figure 3.2 (b), we can see as the number of partitions (Edge nodes) increases, the training time decreases significantly. Although the training time does not change much for small-size datasets, the large-size datasets may benefit using the proposed approach. Hence, the experimental results depict the efficacy of the model partitioning approach in terms of reducing the training time, and maintaining a decent accuracy simultaneously.



(a) (b)

Figure 3.2: (a) Accuracy comparison for different datasets, (b) Training time comparison for different datasets

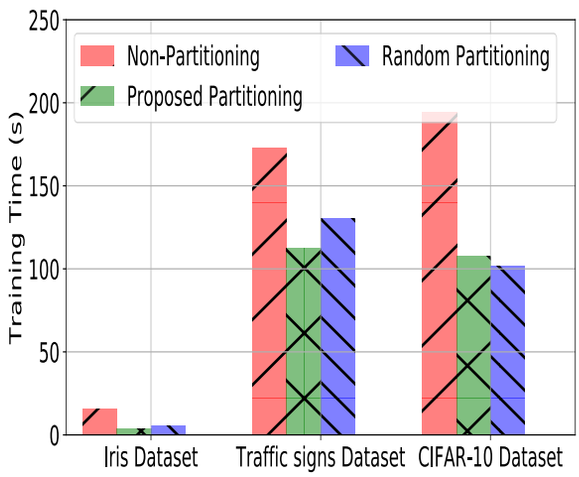
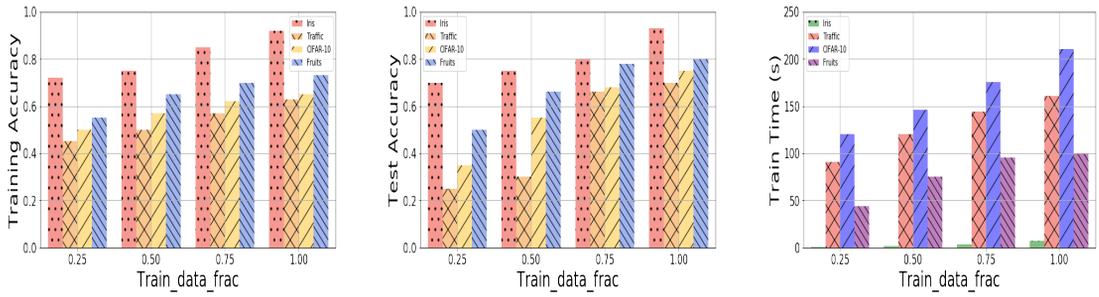


Figure 3.3: Comparison of model partitioning approaches (non-partitioning, proposed partitioning and random partitioning) using SVM



(a) Train Accuracy vs fraction of training data (b) Test Accuracy vs fraction of training data (c) Training time vs fraction of training data

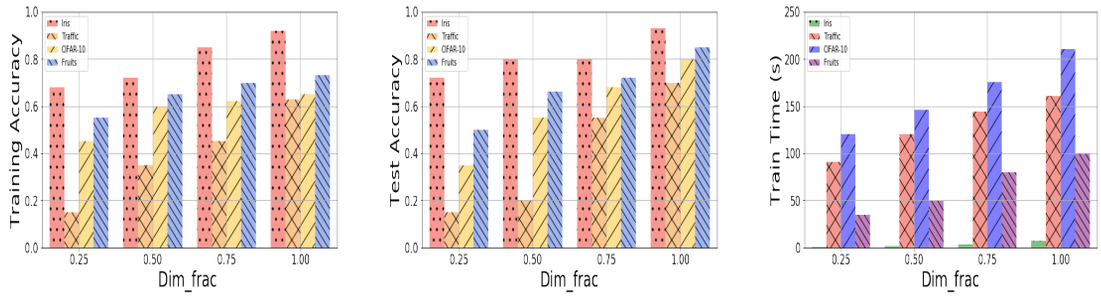
Figure 3.4: Effect of varying number of data points on different datasets

3.4.4.2 Comparison of the proposed partitioning approach with non-partitioning and random partitioning approaches

This experiment shows how the SVM model's partitioning affects the model training time over the Iris dataset, the traffic signs dataset, and the CIFAR-10 dataset. The proposed approach uses a fixed number of edge devices arrived at after theoretical analysis. The partitioned units of the model are distributed to these five available edge devices that can compute in parallel. Moreover, we compare the results of the proposed model with non-partitioning and random partitioning approaches. The non-partitioning approach is the regular model that runs on a single edge device without a model partition, where the random partitioning approach partitions the model randomly between 1 to 5 edge devices. Figure 3.3 shows the comparison of training time for different approaches. The number of epochs is set to 300 for the Iris dataset, 100 for the Traffic and CIFAR-10 datasets. The number of partitions in the proposed approach is estimated prioritizing the accuracy over training time. In the case of the non-partitioning approach, the training time for the SVM model is higher than any other approach for achieving maximum accuracy. However, the proposed partitioning approach offers lower training times for Iris and Traffic signs datasets, over random and non-partitioning approaches. The proposed approach considers the number of partitions for all datasets as 5. On the other hand, the random partitioning approach considers the number of partitions for Iris, Traffic, and CIFAR-10 datasets as 4, 3, and 5, respectively. The required training times using the proposed approach are 3.8000s, 112.4300s, and 107.3243s for the above datasets. Although the random partitioning approach for the CIFAR-10 dataset offers a shorter training time than the proposed approach, a high accuracy is achieved in the proposed approach. This shows the necessity of finding the optimal number of partitions, which we plan to explore in our future work.

3.4.4.3 Effect of varying number of data-points on Accuracy & training time

In this experiment, the performance of proposed partitioning algorithm has been measured when the number of data points are reduced to 75%, 50% & 25% respectively (as shown in figure 3.4). When the training examples are reduced to 75% (i.e. Train_data_frac=0.75),



(a) Training Accuracy vs fraction of features (b) Test Accuracy vs fraction of features (c) Effect of varying number of features on different datasets

Figure 3.5: Effect of varying number of features on different datasets

then the test and train accuracy of all the datasets are not affected much. But, the training time is reduced for all datasets due to less number of computations. When the data points are reduced to 50%, then the performance of the proposed approach was observed to be good for the Iris and Fruits datasets, because they have less classes and therefore, a large number of data points of all the classes are available for model learning. Iris performed well in all cases because it has linearly separable data with 4 dimensions. The traffic dataset shows the worst performance when the training dataset is reduced beyond 75%, because it has a large number of classes (i.e. 43). So, the training examples available for each class in the training data are less. The poor quality of images and imbalance of data points for certain classes have contributed to the poor performance of the traffic dataset. As the data points are reduced, the training time experiences a reduction as well. The best results for all the datasets were recorded for Train_data_frac=0.75, because it records almost equivalent training accuracy with good generalization performance (test accuracy), and reduced training time, as compared to the case when all the data points in the datasets were considered.

3.4.4.4 Effect of varying number of features on Accuracy & training time

In this section, the performance of the proposed approach was recorded when the number of features were reduced to 75%, 50% and 25%. The fruits dataset performed well, even when the features were reduced to 50% (as shown in figure 3.5). This is due to high dimensionality and good resolution of the images (i.e. good image quality). The CIFAR-10 and traffic datasets offered a sub-par performance when 50% of the features were used. This is because both datasets have 32x32 tiny colour images of poor resolution. Even though CIFAR-10 and traffic datasets have the same resolution of images, CIFAR-10 performed well in comparison because all the classes belong to different domains. CIFAR-10 contains images of dogs, birds, trucks, ships, horses etc. Therefore, features needed to represent each class are very different. For instance, if there are images of trucks and dogs in the dataset, it will be easy to correctly classify them with less number of features. In the Traffic dataset, in order to identify traffic signs, all features are important because 10 km speed limit sign is very similar to 70 km speed limit sign.

Therefore, missing few features could lead to misclassification.
In conclusion, when the ML tasks (i.e. weights updation operation) are performed in parallel and distributed manner then the training time is reduced significantly without much affecting the accuracy. While in case of datasets having less number of classes, the data points used for training the SVM model can be reduced significantly without much affecting the training/test accuracy. Due to this the training time of the model reduces significantly. On the other hand, when the dataset comprise of diverse classes, then the dimensionality of the dataset (i.e. number of features) can be reduced significantly without affecting the accuracy performance. The dimensionality reduction results in reduced number of computations and hence the time taken for training the model is reduced.

3.5 Part B - Towards Safe Online ML Model Training and Inference on Edge Networks

3.5.1 System Model & Assumptions

We consider an autonomous vehicle system application requiring real-time decision-making in tasks such as object detection, which is facilitated by online ML model training. The system comprises edge devices, including onboard computers and individual processors within multiprocessor embedded systems, alongside a centralized server responsible for model training and management. Edge devices communicate with each other and the server via wired and wireless networks. The server can also link with external units like Roadside Units (RSUs) to distribute model training tasks efficiently. By employing edge devices, tasks are performed closer to the data source, reducing latency and boosting system performance [103]. Figure 3.6 illustrates this architecture. In this setup, an initial pre-trained model, based on a representative dataset, is updated incrementally as new data arrives. Symbols used throughout this work are defined in Table 3.3.

3.5.1.1 Assumptions

- The system generates real-time data from various sensors like cameras, lidars, radars, and GPS.
- It uses supervised machine learning models, SVM and RF, for decision-making.
- Edge networks, ranging from single to multiprocessor systems, have each processor acting as an edge device, sharing data, instructions, and synchronization signals.
- The server manages model training tasks, workload distribution among edge devices, and result aggregation.
- Safety and Reliability: Safety encompasses the prevention of undesirable consequences resulting from vulnerabilities or incorrect predictions, whereas

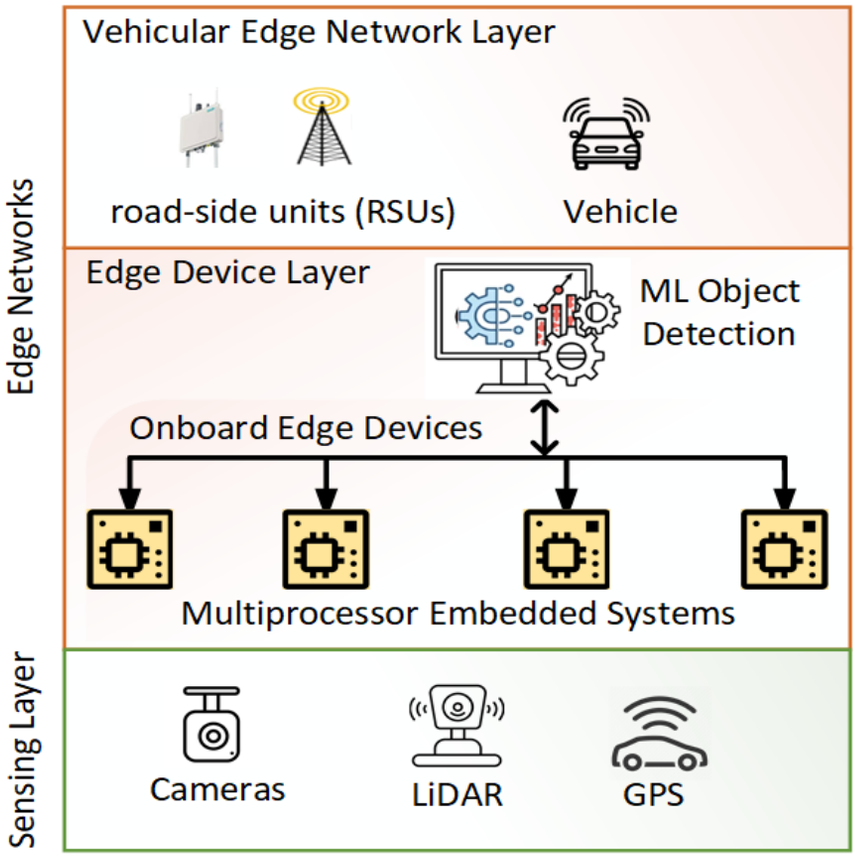


Figure 3.6: Machine learning-enabled edge networks

reliability pertains to maintaining consistent performance and accurate ML models during continuous updates.

3.5.2 Proposed Approach & Methodology

The proposed framework presents a parallel computing architecture that splits an ML model into different processing units on edge networks, ensuring safe execution. Figure 3.7 shows the detailed workflow of the proposed framework, where the optimal split decision is taken by minimizing the net training time. The split decision module implements a partitioning algorithm that determines how the model should be split and where the partitioned models should be run, accelerating the application performance. We discuss the various steps of our proposed approach as follows:

3.5.2.1 Model Partitioning

Given edge devices' computational constraints, we propose Algorithm 3 to find the optimal partitioning s^*, and mapping of model splits to edge devices. This algorithm accommodates the computational needs of models like SVM and RF, that have been used in this study.

Table 3.3: Symbols and their descriptions

Symbol	Description
s_k	k^{th} partition strategy of ML model (refers to a tuple of model splits)
S	Set of all possible partitions of the machine learning model.
m_i	The i^{th} split in the machine learning model.
t_{m_i}	Training time for split m_i.
l_{m_i}	Communication latency for split m_i.
$T_{TMR_{k,m_i}}$	Training time required for TMR for the m_i-th split of k^{th} partition.
$s_{k,i}$	Set of model splits assigned to i^{th} edge device in the k^{th} partition.
s^*	Optimal partition
e	Number of edge devices
d_i	Edge devices i
p_i	Processing capacity of edge devices d_i
m	Submodel which refers to $s_{k,i}$
ϵ	Convergence threshold for the objective function improvement
T_{fail,m_i}	Worst case or failure time for the i^{th} model split

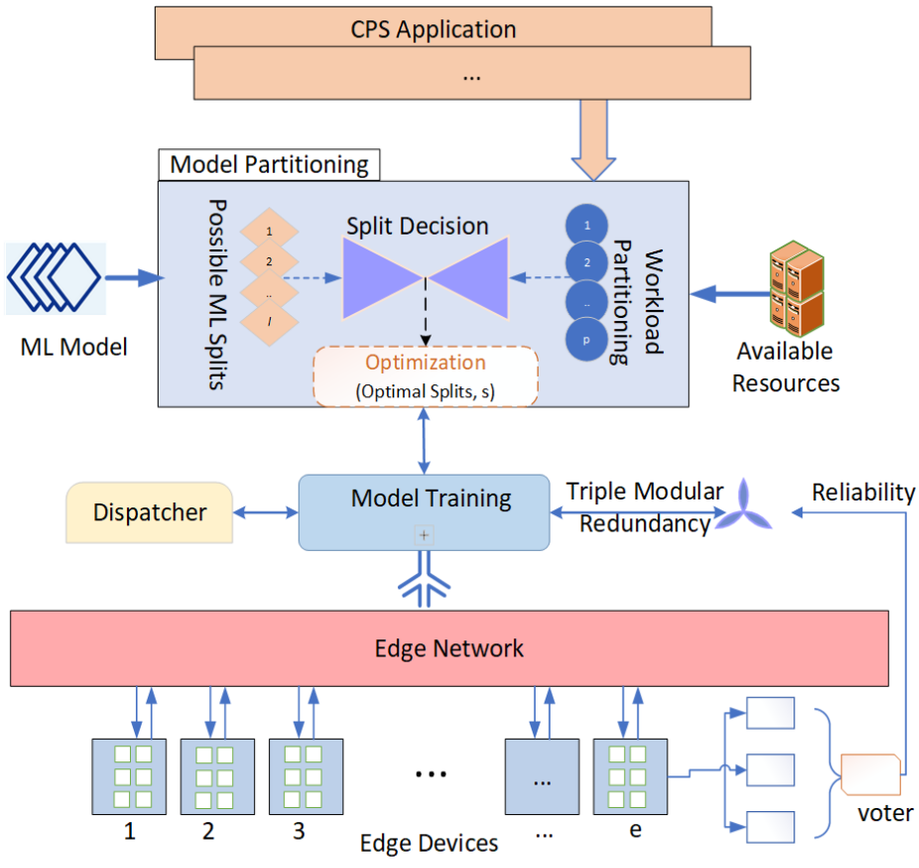


Figure 3.7: Proposed framework for ML model Partitioning

Algorithm 3: Optimal Model Partitioning and Mapping for Edge Networks

Input: ML model M, edge devices processing capability p_i for device d_i, $\forall i \in n$, β, γ, communication latency l_{m_i}, training data

Output: Optimal partitioning s^* and mapping of model splits to edge devices

`/* Initialize search space randomly */`

$S \leftarrow s_1, s_2, \ldots, s_k$

while $\Delta f(s_k) \geq \epsilon$ **do**

  `/* Evaluate the obj function for each` s_k `*/`

  for *each partitioning* $s_k \in S$ **do**

    $f(s_k) \leftarrow$ evaluateObjectiveFunction(s_k) based on Equation (3.1)

  end

  `/* Update the search space */`

  $S \leftarrow S \pm \Delta s_k$ to improve $f(s_k)$

  Sort edge devices: sort(P, p_i)

  `/* Assign model splits to edge devices */`

  for *each submodel* m **do**

    $p_{\min} \leftarrow \arg\min_{p_i \in P} p_i$ s.t. (CT3)

    `/* Update the available capacity */`

    $p_{\min} \leftarrow p_{\min} - w_m$

  end

  Calculate l_{m_i} for each model split m_i

  Sort model splits based on l_{m_i}: sort(M, l_{m_i})

  `/* Schedule` $m_i \in m$ `on edge & update time */`

  for *each model split* m_i **do**

    d_j: $d_j \leftarrow d_j \cup m_i$; s.t. to (CT6) and (CT7)

    $t_{m_i} \leftarrow$ calculateTrainingTime(m_i, d_j)

    $TMR_{m_i} \leftarrow$ calculateTMRTime(m_i)

  end

  Calculate $\Delta f(s_k) = f(s_k^{\text{old}})$ - $f(s_k^{\text{new}})$

end

`/* Select the best partitioning */`

Select $s^* = \arg\min_{s_k \in S} f(s_k)$

Train split m_i on edge device with training data

For each model split m_i, perform TMR: $y_{m_i} \leftarrow$ majority($y_{m_i}^1, y_{m_i}^2, y_{m_i}^3$)

return s^*, mapping of model splits to edge devices

3.5.2.2 Algorithm Overview

The proposed algorithm iteratively explores the solution space S with a set of random model partitionings $s_1, s_2, \ldots, s_k$. It evaluates each partitioning based on an objective function, and updates the search space to improve the function value. The search process continues until the change in the objective function $\Delta f(s_k)$ is below a threshold ϵ. For each partitioning $s_k \in S$, the algorithm evaluates the objective function $f(s_k)$ using Equation (3.1), and updates the search space accordingly.

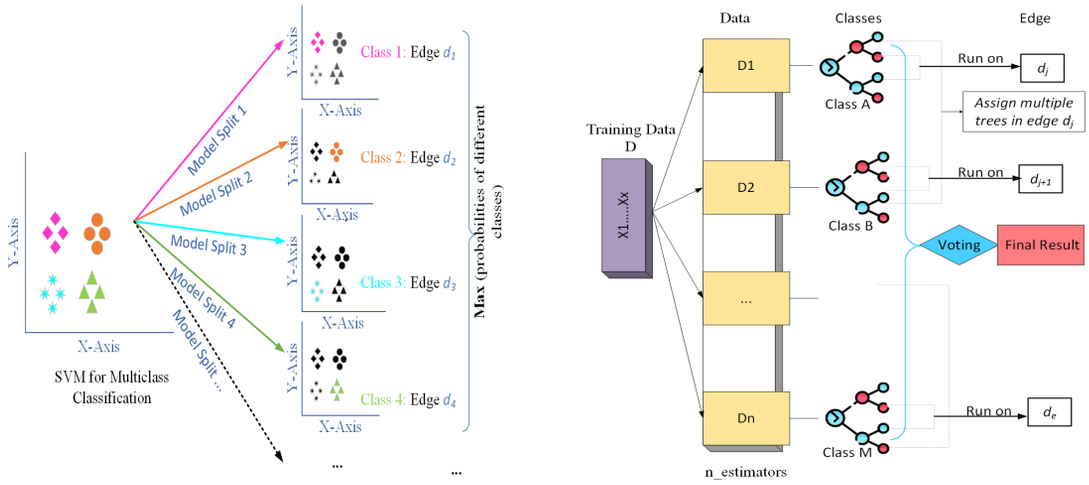
Edge Device Mapping: Post identifying the optimal partitioning, the algorithm maps each model split to an edge device, based on its processing power. It sorts edge devices by processing capacity ($\text{sort}(P, p_i)$), and assigns each sub-model m to the edge device with the least available capacity ($p_{\min}$), in line with Constraint (CT3), updating the device's available capacity.

Scheduling and TMR Constraints: The algorithm schedules model splits for training on edge devices, considering communication latency l_{m_i} and TMR constraints. Model splits, ordered by their communication latency, are scheduled on assigned edge devices (d_j), optimizing training time, while adhering to TMR constraints. Training and TMR times for each model split m_i on edge device d_j are computed and updated.

Training and TMR Integration: After selecting the optimal partitioning s^*, the framework conducts online training for each model split m_i on assigned edge devices, enabling real-time adaptation. Concurrently, it implements TMR for each m_i by selecting the output with at least two matching instances. The framework then returns the optimal partitioning s^* and model-to-device mapping. This optimal partition minimizes training time, while ensuring worst-case execution or failure threshold T_{fail} is not exceeded, and the processor utilization remains within acceptable limits. This problem is solved using mixed integer linear programming (MILP), allowing continuous model improvement in a dynamic context.

Time Complexity Analysis of Algorithm 3: The time complexity of Algorithm 3 is driven by its key operations. The initialization of the search space S takes constant time, $O(1)$. The main loop, iterates over the total number of possible partitions, n_S, until the objective function change ($\Delta f(s_k)$) is below a threshold ϵ, has time complexity $O(E \cdot n_S)$ for the evaluation of the objective function and $O(U \cdot n_S)$ for updating the search space, where E is the time taken by objective function evaluation, and U represents the time consumed in updating the search space. Sorting and assignment of edge devices and model splits result in complexities of $O(e \log e)$ and $O(n_m \cdot e)$ respectively, where e is the number of edge devices, and n_m is the number of model splits. Finally, the calculation of latency and scheduling of model splits add complexities $O(n_m \log n_m)$ and $O(n_m)$, respectively. The overall time complexity can be approximated as $O(E \cdot n_S + U \cdot n_S + e \log e + n_m \cdot e + n_m \log n_m + n_m)$. This explains the algorithm's scalability and efficiency with larger datasets and complex partitioning scenarios.

SVM on Edge Devices: SVMs can be effectively deployed on edge devices by partitioning the training dataset and training multiple binary classifiers in parallel. For



(a) SVM model partition for parallel computing (b) RF model partition for parallel computing

Figure 3.8: (a) SVM model partition for parallel computing, (b) RF model partition for parallel computing

multi-class classification, one-vs-one or one-vs-rest approaches can be adopted, each binary classifier distinguishing between class pairs or one class against the rest, respectively. The partitioned classifiers can be assigned to different edge devices. After individual classifier training, outputs are combined using majority voting or other ensemble techniques to determine the final class label. This parallel structure, illustrated in Figure 3.8a, allows for scalable, efficient SVM deployment on edge devices.

RF on Edge Devices: Training RF models on edge devices involves dividing decision trees and allocating them to different devices d_j. This method reduces the overall training time by leveraging the combined processing power of the edge devices. After training, outputs from individual trees are combined for the final prediction, as shown in Figure 3.8b.

3.5.2.3 Dispatching Partitioned Models

This step assigns partitioned models to edge devices, maintaining execution order via associated threads. We use a global queue to join all processes and return the trained models to the master edge device. The master device combines all models to predict the training input. During partitioned model training, the master device ensures the correct integration of all processes.

3.5.2.4 Safe Integration using TMR

Safe execution is essential when partitioning models across edge devices. To counter issues like resource unavailability, aging, hardware compromise, data corruption, and side-channel attacks that may cause edge devices to produce incorrect results, we propose the integration of the TMR technique. This proven method enhances system reliability in edge network machine learning training. TMR, implemented in parallel on three devices, uses a majority voting system to eliminate single failure points. To ensure trusted computation, we calculate the training module's reliability. The reliability for training

a partitioned model m_i at time t can be calculated using Equation 3.9, where R is the reliability of correct execution.

$$\begin{aligned} R(t) &= R^3(m_i,t) + 3(1 - R(m_i,t)) * R^2(m_i,t) \\ R(t) &= 3R^2(m_i,t) - 2R^3(m_i,t) \end{aligned} \tag{3.9}$$

In the above Equation 3.9, $R^3(m_i,t)$ represents the probability that all three edge devices produce the correct output, and $3(1 - R(m_i,t)) * R^2(m_i,t)$ is the probability that two out of the three edge devices produce the correct output, while one fails. The final equation simplifies this computation.

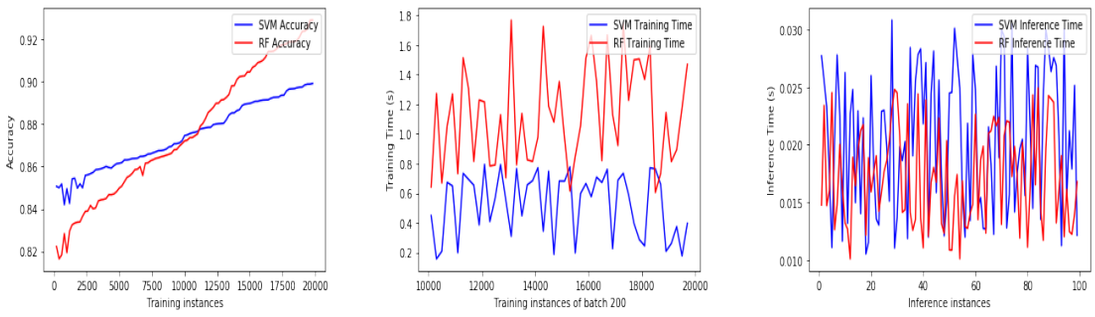
3.5.3 Experimental Results & Analysis

To evaluate the effectiveness of our proposed framework, we conducted different experiments on parallel computing architecture, for multi-class classification problems using SVM and RF. We investigate the performance of the model partitioning algorithm for minimizing the net training time.

Dataset: The dataset employed in this study, titled "Traffic, Driving Style, and Road Surface Condition" [104], was sourced from Kaggle and initially used by Ruta *et al.* [105] to develop machine learning models for Internet of Things (IoT) applications. The data, comprising 24,957 data points, were collected from two vehicles, a Peugeot 207 1.4 HDi, and an Opel Corsa 1.3 HDi, using an OBD device paired with a smartphone. The dataset encompasses 14 features such as "altitude change", "engine load", "speed variance", "fuel consumption", etc. A comprehensive feature list and descriptions are available at [105]. The dataset is categorized into three sub-problems: road surface conditions, road traffic conditions, and driving style.

Environment: The experiments were conducted on multiprocessor systems, representative of advanced edge networks, with a shared network, and a Linux 5.1.0-52-generic (x86_64) kernel integrated into the Ubuntu 20.04 LTS operating system. The multiprocessor systems consist of four Xeon(R) CPU E5-2623 v4 @ 2.60GHz processors, each endowed with eight cores, 4 GB of RAM, a 256 KiB L1 cache, and a 2 MiB L2 cache. This configuration provides a total of 16 GB RAM across the system, accommodating the SVM and RF models employed in our study. Additionally, the shared network offers sufficient bandwidth to meet the data flow requirements of our setup.

We opted for multiprocessor systems as an edge network to emulate modern edge devices' multicore structure. This allows us to explore parallel processing, resource allocation, and crucial inter-processor communication. This choice makes our study reflective of current edge capabilities, and ensures relevance for real-world edge computing scenarios.



(a) (b) (c)

Figure 3.9: (a)Training accuracy for online model training with input size 200 (b)Training time for different training instances (c) Inference time for different training instances

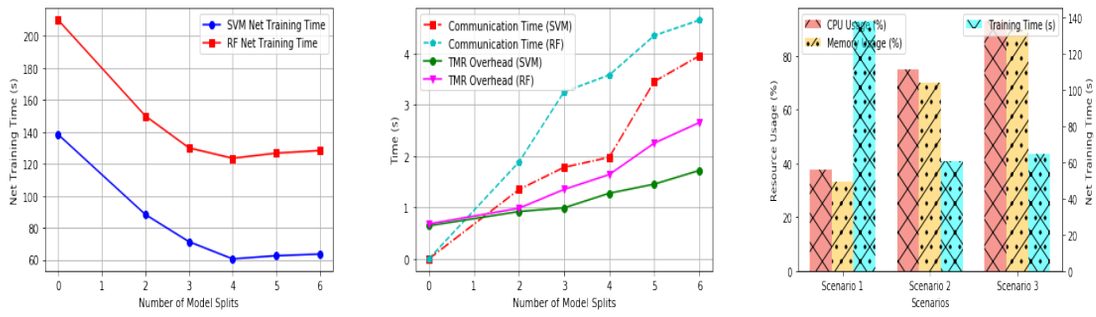
3.5.3.1 Analysis of ML Model Parallel Computing on Edge

In this experiment, we used a pre-trained model as the basis for online model training. This initial model was trained on a dataset of 10,000 samples, which captured the general characteristics of the problem. For the online training phase, we utilized a batch size of 200 for partial model training, applying a learning rate of 0.001. The Algorithm 3 determined the optimal model splits by considering the communication latency and processing capability of each processor in the multiprocessor system. This approach enabled the efficient distribution of the model into four partitions across two edge devices or processors. For the training and testing datasets, the input feature shapes were: (19965, 14) and (4992, 14). The online model training process incrementally updated the model using the remaining 9,965 samples beyond the initial 10,000 samples, allowing the model to adapt to new data patterns, over time.

3.5.3.2 Accuracy

The SVM and RF algorithms were evaluated based on their training accuracy. From Figure 3.9a, it can be observed that the SVM model has a stable performance over the RF model. The SVM accuracy ranges from approximately 85% to 90%, while the RF accuracy ranges from around 82% to 93%. The performance improvement in the SVM model can be attributed to its ability to find an optimal hyperplane that maximizes the margin between different classes, making it well-suited for high-dimensional datasets, like the one used in this experiment.

The learning rate (0.001) in this experiment impacts model training by modulating weight updates. Lower learning rates promote thorough solution space exploration, potentially increasing model accuracy, despite slower convergence. Conversely, higher rates may lead to faster convergence, risking model accuracy due to the potential overshooting of the optimal solution. Our chosen rate of 0.001 seems to strike a balance between fast convergence and accuracy.



(a) (b) (c)

Figure 3.10: (a)Training time for different number of model splits (b)Communication time and TMR overhead for different numbers of splits(c) CPU and memory usage (%) for SVM model training

3.5.3.3 Training and Inference Time

We analyzed the online model training time for both SVM and RF algorithms, which were used to process a batch size of 200 for the remaining 9,965 training samples on top of the pre-trained model. The training time for SVM and RF exhibited different trends, as shown in Figure 3.9b. For the SVM, the training process utilizes the One vs. All (OVA) approach for multi-class classification. This enables the training of multiple binary classifiers in parallel, which results in reduced training time. On the other hand, the RF algorithm constructs an ensemble of 100 random decision trees, which can be computed in parallel by distributing the trees evenly among the available processors. The optimization algorithm finds the optimal distribution of the trees in a *for loop* parallelization, minimizing the overall training time.

Both classifiers were implemented using the *sklearn* library with default hyper-parameters, ensuring consistent configurations across the models. During the training process, the model split was executed on a separate processor, and the inter-processor communication time was recorded to assess the impact of parallel computation. We observe that the SVM generally takes less time in training than the RF. This difference can be attributed to the efficiency of the OVA approach in SVM training, which allows for faster parallelization and computation compared to the RF's decision tree construction and aggregation process.

Furthermore, we examined the inference times for 100 sample test inputs using the trained SVM and RF models. The inference times for both SVM and RF models are illustrated in Figure 3.9c. It is apparent that both SVM and RF algorithms typically exhibited lower inference times. However, the RF model tended to have faster inference times compared to SVM. The maximum inference time for SVM was approximately 30.5 ms, whereas RF had an inference time of about 24.9 ms. In real-world CPS applications, inference time is crucial in determining the system's responsiveness, particularly when real-time decision-making is essential.

3.5.3.4 Comparing Net Training Time and Communication Latency

3.5.3.4.1 Net Training Time

In this experiment, we analyzed the net training time for different numbers of model splits, using both SVM and RF. We considered five different scenarios: no split (0), 2 splits, 3 splits, 4 splits, 5 splits, and 6 splits. For the no-split case (model split zero), the model is trained as a single unit, without any partitioning. From split 2 onwards, the model is divided into the respective number of splits.
As illustrated in Figure 3.10a, training time generally decreased with increasing splits, with exceptions noted for SVM, at 5 and 6 splits. This indicates that the workload distribution across sub-models reduced training time siginficantly, up to 4 splits. Beyond this, no significant improvements were observed, while additional communication latency was incurred. The optimal split number for both SVM and RF was found to be 4, resulting in net training times of 60.48 and 123.5 seconds, respectively.

3.5.3.4.2 Communication Latency and TMR Overhead

As shown in Figure 3.10b, communication latency and TMR overhead were analyzed for various model splits. Communication latency tended to increase with the number of splits. This increase in communication latency can negatively affect the optimal number of model partitioning splits, as it can offset the benefits of reduced training time achieved through parallelization.
For the optimal number of splits (in this case, 4), SVM and RF models exhibited total latencies of 1.9784 and 3.58 seconds, respectively. Corresponding TMR overheads were 1.2785 and 1.6424 seconds, respectively. By integrating TMR into model training and operating devices in parallel, we ensured reliability, without significant delays in model training.

3.5.3.5 Resource Utilization Comparison:

In this experiment, we compared resource utilization across three SVM model training scenarios. Each considers different splits, and we measure CPU usage, memory usage, and training time with online training data (see Figure 3.10c).

- **Scenario 1:** Single-edge device training (no split).
- **Scenario 2:** Training with two edge devices (4 splits).
- **Scenario 3:** Training with three edge devices (6 splits).

3.5.3.5.1 Scenario 1: Single-edge training (no split)

Training was conducted on a single-edge device, utilizing one core. The remaining cores handled TMR configurations. CPU usage was observed to be 37.5% (3 cores out of 8), and memory usage was 33%, for 4GB. As only one core was active, training took longer

(138.35 seconds), which may not always be efficient for real-time applications. This delay could impede real-time applications, highlighting the need for more efficient multi-core processing solutions.

3.5.3.5.2 Scenario 2: Training with two edge devices (4 splits)

In this scenario, the model was divided into four splits, and trained on two devices. Each edge device used two of its cores to run two model splits in parallel, and allocated the other cores for TMR.
As the number of utilized cores increased, the CPU usage rose, from 37.5% to 75%, and memory usage rose to 70.2%,for 8GB. However, due to parallel processing, training time reduced to 60.48 seconds, showing a 56.3% improvement over Scenario 1. This approach allows for more efficient and suitable training for real-time applications.

3.5.3.5.3 Scenario 3: Training with three edge devices (6 splits)

With six model parts assigned across three devices, more cores were used, leading to 93.2% CPU usage, and 90% memory usage. Training time reduced further to 64.48 seconds, this was observed to be slightly longer than Scenario 2. This indicates that while more edge devices and model splits can increase resource usage, the improvement in training time may not necessarily scale proportionally.
In summary, optimal balance in device count, model splits, and resource utilization is key for efficient training in real-time applications. Scenario 2 appears to be the most efficient, but the optimal configuration will depend on specific application needs, and edge device resources.

3.5.4 Threats to Validity

Suitability of edge networks for real-time applications: While training machine learning models on edge networks offers distinct advantages such as: privacy preservation, reduced overhead, and quicker decision-making, it may not be universally suitable for all real-time applications. The practicality and efficacy of our approach can vary depending on the application's unique requirements and constraints. However, edge-based training can provide substantial benefits for applications requiring low-latency responses, such as autonomous vehicles, drones, and industrial IoT systems.
Impact of learning rate on online machine learning model training: The learning rate directly affects resource efficiency, adaptability, convergence speed, and overall model performance in edge networks. Balancing the learning rate is key for resource utilization and convergence. Future work will focus on learning rate optimization for edge-based training.
Generalizability of the proposed approach: We chose SVM and RF models for our initial investigation, due to their distinctive learning algorithms and wide usage in various applications. Although these models often exhibit a manageable computational footprint

for lower-resource devices, their complexity and demands can vary substantially depending on the problem and dataset specifics. This research showcases our approach's potential with these models, and future studies will extend it to more complex machine learning models, such as DNNs, to test its broad applicability and performance.

3.6 Conclusion

With the growing demand for data processing, ML models have become more complex, and may exceed the computational power of individual machines. As a result, training large ML models necessitates optimal model partitioning to enable parallel computing architectures in edge networks. Our proposed framework tackles this by distributing partitioned models across these networks. To ensure safety, our architecture uses the TMR technique for trusted computation, enhancing system reliability.

Chapter 4

An Intrusion Detection System on Fog Architecture

4.1 Introduction

Due to advancements in technology, electrical appliances are now increasingly inter-connected. The goal of Internet -of-things (IoT) is to access every appliance or device through the Internet. This is done in order to operate these gadgets from remote locations. The goal is to improve our day-to-day life. However, this technology raises serious privacy and security issues. As IoT devices are resource-constrained, it is impractical to secure them using traditional approaches. Hence, a light-weight Intrusion detection system (IDS) is required. In this work, we implement a machine learning based Network Intrusion Detection (NID) system in a multi-node fog environment using a Raspberry Pi cluster on a local area network. The proposed Pi-IDS system has been evaluated on ADFA-LD datasets. These datasets comprise of new generation system calls for various attacks on different applications. The proposed fog architecture offers significant advantages in terms of latency, energy consumption and cost over traditional cloud or dedicated personal computer systems. The experiments show that we are able to achieve a Recall of 89%in ADFA-LD with the XGBoost model. The proposed system was able to predict intrusions with an inference time 130 ms, in comparison to Cloud-based inference time of 735 ms, with an estimated running cost of 201 INR/month, in comparison to the Cloud cost of 2051 INR/month.

Further, to enhance the performance in the case of multi-class intrusion attacks, we implemented a lightweight distributed IDS framework, called FCAFE-BNET (**F**og based **C**ontext **A**ware **F**eature **E**xtraction using **B**ranchy**NET**). The proposed FCAFE-BNET approach considers versatile network conditions, such as varying bandwidth and data load before allocating inference tasks on Cloud/Edge resources. Early exit DNN has been used to obtain faster inference generation at the edge. As in many cases, the weights that the model learns in the initial layer may be qualified enough to perform the required task, such as classification. Instead of increasing the computational complexity by using subsequent layers of Deep Neural Network (DNN) for generating the inference, we have used the early-exit mechanism in DNN. The mechanism of DNN with early-exit layers helps to predict a wide range of testing samples through these early-exit branches when they cross the threshold, which maintains the confidence values corresponding to the

inference. By employing this approach, we achieve a faster inference with significantly high accuracy. The proposed FCAFE-BNET framework works for both Network-based and Host-based IDS: NIDS and HIDS. Our experiments demonstrate that, in comparison to recent approaches, the proposed FCAFE-BNET approach has achieved a 39.12% - 50.23% reduction in total inference time on benchmark real-world datasets: NSL-KDD, UNSW-NB 15, ToN_IoT, ADFA_LD.

Internet-of-Things (IoT) interconnect our day-to-day devices through many communication channels [106]. These channels can be in numerous forms, such as the Internet, GPS or Bluetooth. The main goal is to gather information from multiple sources, and use it smartly for various purposes. As the web of IoT is growing, more concerns about its security and privacy are becoming prevalent [107]. IoT devices are endangered by various types of attacks, such as port scanning and man-in-the-middle attacks. Monitoring attacks using traditional intrusion detection approaches is computationally intensive, and requires significant storage space. IoT devices, being resource-constrained, may not be able to store data and analyze attacks in real time. Therefore, these computationally intensive intrusion detection tasks are sent to the cloud for performing inference. The number of IoT applications is increasing rapidly, and sending huge amounts of data for attack monitoring may be overwhelming for present network capabilities. So, in order to reduce the network latency and monitor the attacks in real time, the edge computing paradigm may be employed.

Edge computing may be used to monitor attacks at the circumference of the network, within the proximity of the source generating the data. By placing the computing device closer to the point of data generation, the attacks in the network may be monitored in real-time, which in turn protects the network from security threats. Edge devices are crucial in scenarios where real-time tasks demand adequate computing and storage capabilities, as they help reduce latency by fast processing of data within the network. Hence it can be inferred that Edge computing is well suited for IoT security [108]. It is highly reliable and solves the high network latency issue that the cloud lacks.

Network Intrusion Detection Systems (NIDS) are installed at numerous nodes within an IoT network periphery to detect these attacks [109]. These nodes are chosen to cover the entire network. NIDS installed at various points keep observing the network traffic passing through. The administrator is notified if the observed traffic matches any previous malicious traffic. These nodes can be in various forms, such as routers, cameras, or other electronic controllers. For instance, in order to monitor and track the flow of the network from different sources, the IDS deployment can be done on the gateways, so that other connected clients & hosts can be monitored, regardless of their firmware and operating system. These nodes are less robust computationally, so we need a lightweight NIDS.

In order to reduce the computation, we have used an early-exit mechanism in DNN, to obtain faster inferences. The early-exit mechanism accelerates the inferences by generating output at the initial layers of the DNN, thus reducing the computational complexity of the mission-critical tasks. The early-exit mechanism in DNN leads to an accuracy-latency

trade-off i.e. the overall inference latency is reduced using the early-exit approach, but it might adversely affect the inference accuracy. In the proposed work, we have tried to balance out latency and accuracy, while executing tasks on edge-cloud synergy. The performance of the proposed framework has been evaluated on benchmark datasets of NIDS, like NSL-KDD [110], and CICIDS2017 [111]. The proposed framework is evaluated on new-generation IoT and Industrial IoT based NIDS datasets, such as UNSW-NB15 [112], and ToN_IoT [113]. These datasets have gathered various cyber-attacks and normal events from telemetry data of Industrial IoT and IoT sensors. The data collection is done on a large-scale real-life IoT network with three layers: Cloud, Edge/Fog, and IoT devices. The proposed technique aims to extend globally to recent and older NIDS datasets and other IDS, such as Host Intrusion Detection Systems (HIDS) [114]. To achieve this objective, we have converted the given datasets to 2-Dimensional (2D) vectors (i.e. 2D images). The pre-processing method for the above task (i.e. image conversion) is different for each dataset, in order to obtain the best results. The results are validated on a benchmark dataset of HIDS, such as ADFA-LD [115]. Most of the researchers in the past have worked on binary classification (i.e. attack or not attack), without considering the nature of the attack. A few researchers proposed methods for multi-class classification, but the performance of these frameworks was not up to the mark. We observe that decoding the type of attack, along with the presence of the attack can help to deal with the attack. In this work, we have proposed a noble feature extraction technique, which performs well in the case of multi-class classification.

4.2 Problem Statement

Assuming that our application needs to meet its deadlines, but can bear moderate accuracy losses, we need to balance out the accuracy and the latency. So, in our work, we have optimized the DNN right-sizing such that the inference accuracy is maximized within the defined latency requirement. In upcoming sections, we discuss in detail how we optimise the above latencies in order to make our system fast as well as robust under various dynamic environments.

4.3 Contributions

The major contributions of our proposed work in this chapter can be summarized as under:

- The training and testing of ML models has been done in a distributed manner, using a Raspberry pi cluster as a fog environment.
- In order to study the performance of the proposed technique on cloud and fog infrastructure, The trained model has been deployed on the cloud, as well as on the pi-cluster. The experiments show that the pi-cluster (fog) took less time for inference, as compared to the cloud.

- In order to deal with class imbalance, the SMOTE technique is applied, which has improved the performance of the proposed approach significantly. Feature reduction is performed using PCA. This reduces the computations significantly, and thus reduces the training & testing time of the model, without affecting the accuracy much.
- The proposed FCAFE-BNET framework optimizes DNN right sizing (i.e. early-exit mechanism) for maximizing the accuracy within the given latency requirement through edge-cloud collaboration. The efficacy of the proposed approach is demonstrated on the test-bed using network traces of real-world datasets.
- The proposed FCAFE-BNET framework considers network versatile environments (i.e. varying bandwidth, data load) for better performance in real-world scenarios. For performing fast inferences in case of low bandwidth, the inference task is performed on fog-cluster/fog-device with reduced computational complexity, instead of offloading it to the Cloud.
- The proposed data transformation and feature extraction method distinguishes the various attack patterns effectively and significantly improves the multi-class IDS performance.

4.4 Part A - An Intrusion Detection System on Fog Architecture

4.4.1 Proposed Framework

As shown in Figure 4.1, the proposed framework comprises of four phases: feature extraction, feature selection, selecting the machine learning model, deploying and evaluating the trained model on a Raspberry Pi Cluster and the Cloud.

4.4.1.1 Feature Extraction

The considered dataset consists of system call traces. There are different files which consist of a number of system call traces. Each file is labelled according to the class it belong to, such as : Add-User, Hydra-FTP, Normal etc. We use the modified vector space representation technique on system call traces for generating our dataset, upon which we apply various Machine Learning algorithms for predicting potential attacks. The **n-gram** frequency technique is used for feature extraction. N-gram is a sequence of 'n' consecutive system calls, each system call being one word. We generate all possible n-grams, then we calculate the frequency of each unique n-gram. Next, we select the top-m occurring n-grams as our **features** for the dataset. The values of 'n' and 'm' has been finalized through experimentation. Each system call trace present in the data serves as single instance in the generated dataset. Once the features have been extracted from all the available traces (both "normal" and "attack"), for each trace (file) available in

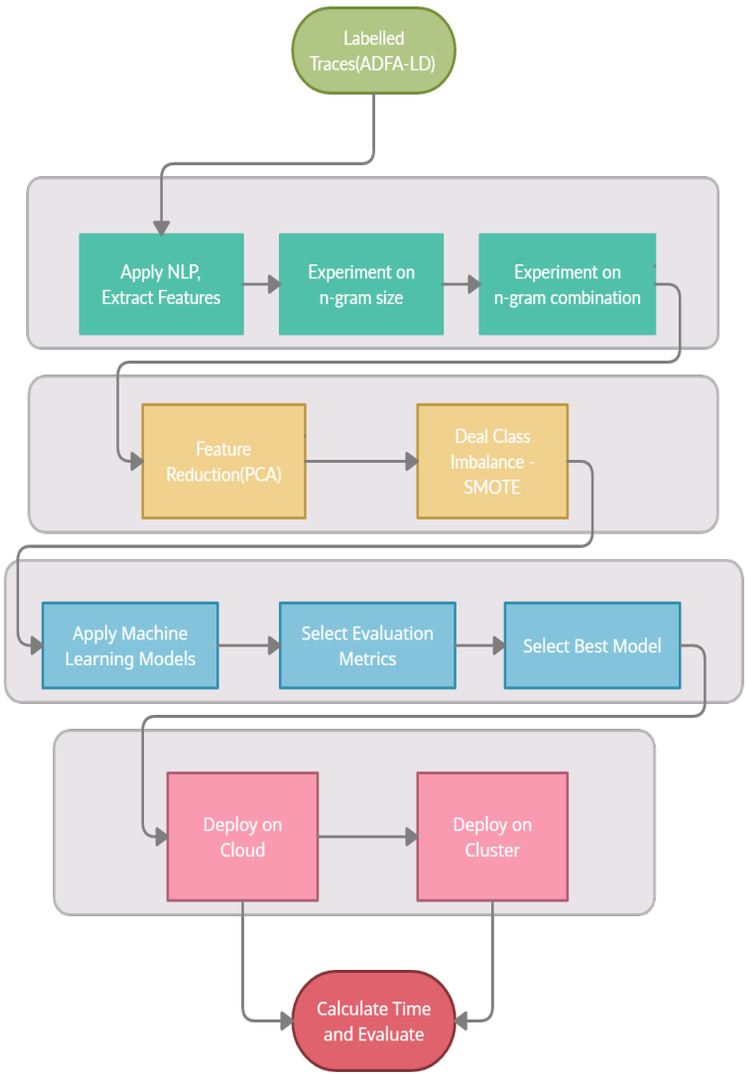


Figure 4.1: Graphical representation of our work in steps

the dataset, the selected features are searched. The frequency of occurrence of selected features in the given trace serves as an instance in the dataset. The label is the same as the trace name (filename). The data is converted from system call sequence to a tabular format, where each row depicts the particular system call trace and each column depicts the selected n-gram feature or label.

Algorithm 4: Fog-cluster Inference Algorithm

Input: *'M' Trained model , 'n' system call traces, 'k' worker fog nodes*
Output: *Inference time 'T', labels*
At Master Fog Node:

- Deploy trained model *'M'*
- *'n'* System call instances

for each worker fog node ∈ k:

- *Master node sends 'n/k' instances & Model 'M'*
- *Inference results from all 'k' fog worker nodes is sent back to master node*

Return: *Inference Time & Predicted Labels*

4.4.1.2 Feature Selection

Now that the dataset is built in *csv* format, we apply standard data cleaning and pre-processing techniques. In the data cleaning phase, we use an anomaly detection technique called the Interquartile Range Method[116] to remove extreme outliers, which might affect the accuracy of our model. We also convert the dataset to be of binary classes, i.e we classify all attacks under category **attack** and normal instances as category **normal**. The distribution of the dataset is still biased, with 90 percent of the instances belonging to the normal category. The distribution of the training data is shown in Figure 4.2(a).

To deal with data imbalance, we applied **SMOTE**(Synthetic Minority Oversampling Technique)[117]. This is an oversampling technique which increases the number of instances of the minority class. SMOTE works with the concept of drawing a line between close minority class data points, and generating points on the line. We apply this technique only on our train data, to make our model more sensitive to the 'attack' class of data. The class distribution of train data after SMOTE is present in Figure 4.2(b). The Principal Component Analysis (PCA) technique has been used for dimensionality reduction. This projects the data into axes of maximum variance, thus helping in reducing the number of selected features from our data, while holding maximum information possible.

4.4.1.3 Selecting Machine Learning Models

On the processed dataset, we have trained various Machine Learning models, namely: Decision Tree, Random Forest, KNN, SVM, & XG Boost classifier. In order to find the

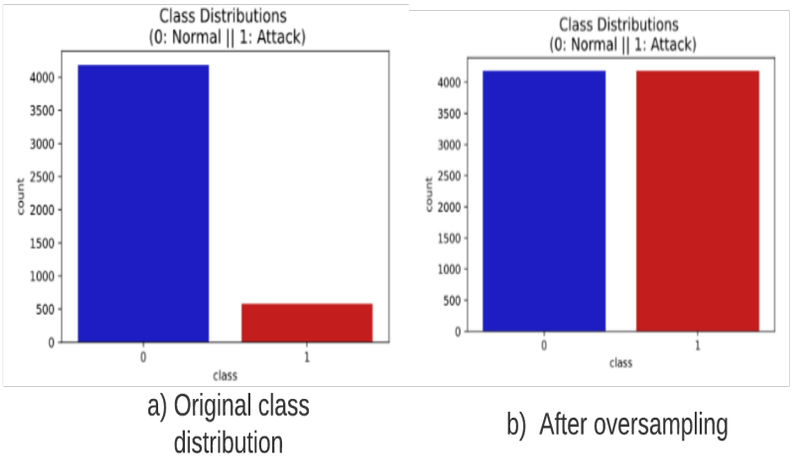


Figure 4.2: Original Class Distribution

best parameters to maximize the performance, GridCV search was used. After comparing the results obtained by GridCV search, the best model has been selected for deployment on both cloud and pi cluster for further experimentation.

4.4.1.4 Deploying Model

We used Microsoft Azure cloud services for deploying our model. Specifically, we created a web application instance on the cloud, and deployed our model in the back-end of the web application. The web page is accessible by the user, and the user can upload the data file to get the classification, and would also be notified of the time taken by the model to do the classification. The total time can be calculated using following equation:

$$Inference_t = Upload_t + Run_t + Download_t + 2 * latency \tag{4.1}$$

The trained model has been deployed on the master fog node along with the processed data. The job of the master fog node is to distribute the process (trained model) and data to all the worker fog nodes (as shown in Algorithm 4). The total run-time is calculated by summing the time taken to send the data along with the trained model, running the classification algorithm , and receiving the processed data (results).

4.4.2 Experimental Setup

We conducted our experiments under two environments: fog & cloud. This section discusses the setup for both the environments, metrics used to evaluate the performance, and the dataset used for experimentation.

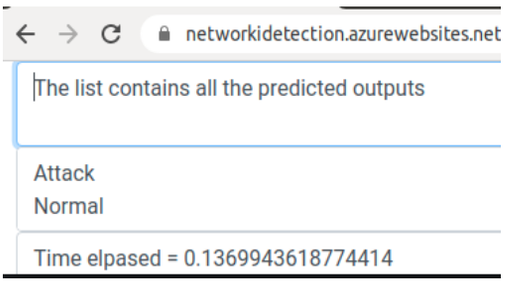


Figure 4.3: Deployed Web Application

4.4.2.1 Fog Setup

The experiments were conducted on a cluster formed by two Raspberry Pi 4 Model B, and one Raspberry Pi 3 Model B. Each raspberry pi 4 has Cortex-A72 (ARM v8) 64-bit SoC, along with 4 GB RAM and quad core with 1.5GHz processor frequency. The raspberry Pi 3 is equipped with 1GB of RAM. The networking was done using a network switch with 100 Mbps bandwidth, and three Ethernet cables. Three 32 GB Class 10 Micro SD Cards have been used as internal storage for raspberry pi OS and various files. The Distributed Machine Learning Cluster was implemented on each device using Apache Spark and Hadoop. All the three Raspberry Pis acted as edge devices. Among the three, one Raspberry Pi 3 was considered to be the master node, and two raspberry pi 4s acted as worker nodes. The two worker nodes did all the training and classification, and sent the results back to the master node.

4.4.2.2 Cloud Setup

In order to deploy our model on the cloud, we selected Microsoft Azure Cloud service. We created a web application on Azure in order to create an interactive Web Page to send test data and receive classifications. The configuration of our Cloud machine was: B2 Category Machine 200 with total ACU and 3.5 GB RAM. The server selected for cloud machine was in Central India (Pune). The estimated cost for this cloud machine was 2051 INR/Month. The software requirement of our Cloud was: Flask framework to create the Web Application, Python 3.8 for back-end machine learning, and HTML and JavaScript for front-end web pages. This creates a website where the user can select and upload data, and get classification predictions directly on the web page. The user also gets information about the time it took for the cloud machine to run the classification task (as shown in Figure 4.3).

4.4.3 Evaluation Metrics & Dataset

We have evaluated the inference time for both fog cluster and Cloud. Also, we have compared the cost of deployment for both fog cluster and cloud. The impact of varying n-components of PCA and oversampling on performance parameters (i.e. precision, recall and accuracy) is discussed. The performance of various ML algorithms in terms of precision, recall and f1-score is discussed in detail.

Table 4.1: Types of attacks

Class	Attack Type	Number of Instances
Adduser	Adds new sudo user	91
Meterpreter	Access of interactive shell to attacker	75
Java_Meterpreter	Java based interactive shell	124
Webshell	Web Server based attack	118
Hydra_FTP	Brute Force on FTP server	162
Hydra_SSH	Brute Force on SSH server	176

4.4.3.1 Evaluation Metrics

The metrics used to evaluate the performance of the framework were: $Precision = \frac{TP}{TN+FP}$, $Recall = \frac{TP}{TP+FN}$, and $Accuracy = \frac{TP+TN}{TN+TP+FN+FP}$. Recall tells us how many attacks we are able to detect out of the total number of attacks. An attack which goes unnoticed can do much more harm than some normal instances classified as attacks, which makes recall an important metric here. At the same time, we cannot ignore other metrics completely, so we also keep a tab on other metrics, while giving more importance to Recall.

4.4.3.2 Dataset

The ADFA-LD (Australian National Defence Force Linux Dataset)[118] was created by the Australian National Defence Force Academy. It is a host-level intrusion detection dataset currently used widely for testing of intrusion detection products. The dataset comprises of system-call traces generated on Ubuntu Linux 11.04 version with certain categories of attacks carried out on the system, which are labelled accordingly. The data consists of hundreds of text files with continuous system call traces, labelled through its name as its class. As shown in Table 4.1, the dataset comprises of six different types of network attacks performed on Linux systems.

4.4.4 Results & Discussion

4.4.4.1 Finding the best values of n-grams & top 'm' features

We conducted experiments to study how the selection of top 'm' n-gram features affects the training accuracy of the proposed model. As shown in Figure 4.4, the maximum training accuracy is reported at m = 150, for 3-gram features. In case of 2-gram features, the maximum training accuracy is reported at m = 60. The best training accuracy for 1-gram features is reported at m = 45. The 1-gram and 2-gram features offer poor performance for m less than 15, while 3-gram has poor performance for m ranging from 10 to 75. In order to further investigate the test performance of the selected features, we have trained the proposed model with 1-gram, 2-gram, 3-gram features separately, as well as in combinations (1+2-grams and 1+2+3-Grams) with 'm' corresponding to maximum value

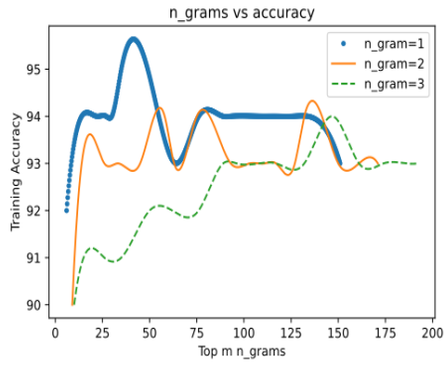


Figure 4.4: Varying size of top selected features for different n-grams

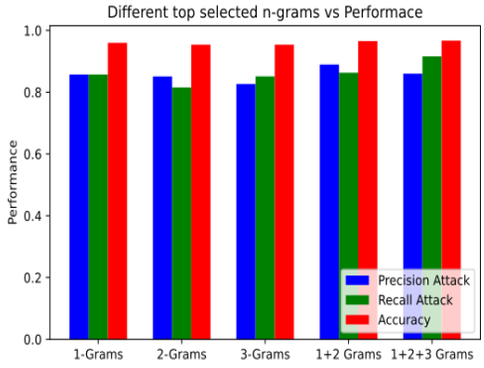


Figure 4.5: N-gram combinations with their Performance

of training accuracy reported (as shown in Figure 4.5). The test data comprises of 90% normal data, and 10% attack data.

As shown in Figure 4.5, all the n-gram combinations reported very high accuracy. But, as the data is highly imbalanced, accuracy may not be a good measure, as it gives biased results due to the class with more instances. So, in order to deal with class imbalance while testing, Recall may be a better measure. Figure 4.5 depicts that the best recall for attacks corresponds to 1+2+3-Grams. So, we have used 1-gram, 2-gram and 3-gram features collectively with m = 45, 60 and 150 respectively. Therefore, the total number of features extracted is 255 (i.e. m = 255).

4.4.4.2 Effect of Data Processing on performance

As shown in Figure 4.6, the maximum Recall, Precision and Accuracy is reported when the 83 Principle components (i.e. n-components = 83) are selected. The initial dimensionality of the dataset is 255, as shown in the previous section. Now, after applying PCA, it has reduced to 83. The inference time of the model before PCA is 443 *ms*, while after applying PCA, it has reduced to 130 *ms*. Due to a reduction in dimensionality, the computational and communication time has reduced significantly. Before applying SMOTE on the XGBoost model, we obtained a Recall of 79 percent, and a Precision of 92 percent (as shown in Figure 4.7). However, after applying SMOTE, the performance (recall) of the model improved significantly. This occurred because the model was trained

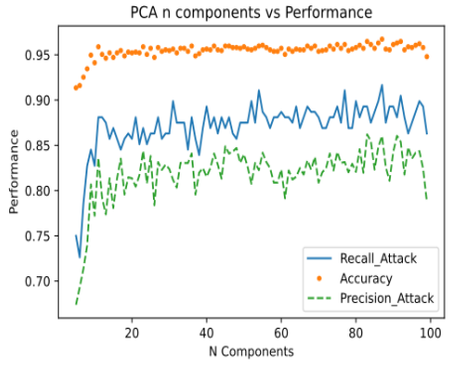


Figure 4.6: Varying n-components of PCA

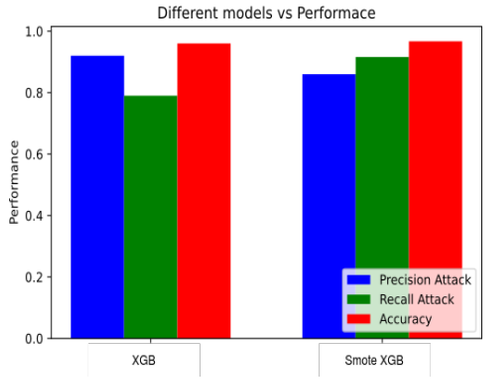


Figure 4.7: Before and after SMOTE(Oversampling)

better on the minority class, and was able to better classify it, thus increasing the Recall.

4.4.4.3 Evaluating performance on different ML Models

For evaluating the performance based on the evaluation metrics discussed, we trained all the discussed models. The best hyper-parameter values were selected for training all the ML models based on GridCV search. We plotted the performance of all the models on the finalized dataset for attacks, see Figure 4.8 for the results. Table 4.2 depicts the performance for both "attack" and "normal" cases. Here, we see that Support Vector Classification offered best Recall, but the Precision was quite low. The Random Forest model offered a good balance between precision and recall with 96 percent accuracy. The best recall was seen in XGBoost model, with good precision and accuracy. So, we selected the Gradient Boosting technique (XGBoost) model to be deployed on both the cluster and the cloud.

4.4.4.4 Performance Evaluation on the Cloud

After finalizing the Machine Learning model, we used the setup Web Application on the Microsoft Azure Cloud to calculate the time required for classification. The inference time was calculated using Equation (4.1). The calculation of upload time and download time were done with the help of the upload speed and download speed of the available

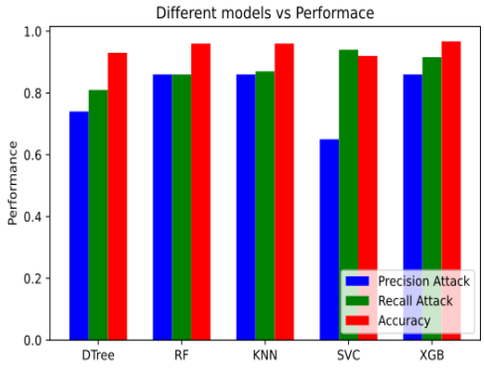


Figure 4.8: Evaluating different ML models

Table 4.2: Evaluation Metrics for all Classes

	Attack			**Normal**		
Models	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
DTree	74	81	77	97	95	96
RF	86	86	86	98	98	98
KNN	86	87	88	98	98	98
SVC	65	94	77	99	92	95
XGB	83	89	86	98	97	98

cloud machine (as shown in Figure 4.9). We calculated the speeds and latency of the cloud using the web-application provided by Microsoft. We calculated the latency using Microsoft Azure website, as shown in Figure 4.10. Here, we ping the server selected location (Pune, India), and return the latency. We took the average of the latency, which came out to be **169 ms**. The upload time was calculated using the size of our file, and the upload speed to the Central India server (Figure 4.9), which came out to be 2.57 MB/s.

This factor may vary from place to place, and places with bad/unstable internet connections might suffer from much higher inference time, because of this factor. The upload time calculated using this speed came out to be **260 ms**, for 684 KB of uploaded data. The download time of the data (classifications of the instances) was calculated using the download speed, which was 5 MB/s. The download time came out to be **1.4 ms**, for 7.5 KB of downloaded data. The run time was calculated using a python script, which ran along the code and displayed the time in the output of the web app, see Figure 4.3 for the result. This came out to be **136 ms**. The total inference time was **735.6 ms**. The experimentation has been done using a batch size of 1000 instances: the smaller is the batch size, the more is the advantage offered by the fog infrastructure, because of reduced latency. The cost of the cloud as estimated by Azure was 2051 INR/Month.

Geography	Region	Physical Location	Upload Progress	Upload Speed
Asia Pacific	Central India	Pune	100%	2.57 MB/s

Figure 4.9: Upload Speed

Region	Average Latency (ms)
Central India (Pune)	169 ms

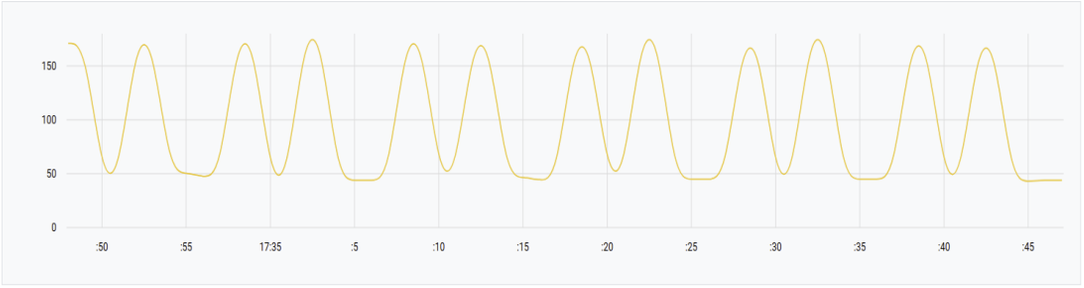


Figure 4.10: Latency from Cloud

4.4.4.5 Performance Evaluation on the fog Cluster

When we ran the ML job on the cluster, the inference time came out to be **130 ms**. The inference time was averaged out of one thousand runs, and was run on the same data as was run on the cloud. When we ran inference for only a single run, the inference time was close to **300-450 ms**, but under a continuous load, the connections and data pipelines were already built, which brought down the average run time. The cost calculations of the cluster were done considering the maximum load on all the Raspberry Pis. We calculated the power consumption as follows:

$$Power = Voltage\_usage * Current\_usage \tag{4.2}$$

The standard Voltage consumed by the Raspberry pi 4 model B is 5V. The current required by the device varies with the peripherals attached to it. We connected a Keyboard and an HDMI cable, which required approximately 400 mA of current. The Energy consumed by the fog device was calculated using:

$$Energy\_consumption = Power * CPU\_time \tag{4.3}$$

Using the above equations, the maximum total power consumption of our system came out to be 40 Watts. Considering the rate of electricity in the place of experimentation, the total cost for one month of running the system came out to be INR 201.6. This is the maximum possible cost of the entire cluster. This is far less than the estimated cloud cost, even if we extend our fog architecture by a few nodes or change in electricity cost rate. The fog & cloud comparison is shown in Table 4.3.

Table 4.3: Time and Cost Comparison

Infrastructure	**Inference Time**	**Cost**
Fog Cluster	*130 ms*	*201 INR/Month*
Cloud	*735 ms*	*2051 INR/Month*

4.5 Part B - Dynamic Hierarchical Intrusion Detection task offloading in IoT Edge Networks

4.5.1 Motivation

Data present in the network is often critical, and its privacy must be protected. IoT server security can be increased multi-fold if we can identify the attacks timely, as the delay in attack detection may lead to serious repercussions. Delays can be exploited by malicious attackers to achieve unauthorized entry into the sensitive information transmitted through IoT devices. Furthermore, postponed identification can provide attackers with the opportunity to intensify their assaults, extending their influence to different segments of the network, compromising additional devices, and inflicting extended harm. Figure 4.11 depicts the graph between latency (i.e. transmission delay and processing delay) for various approaches: device-only, edge-only, and cloud-only, for varying network bandwidths. In this experiment, a raspberry pi, a laptop, and a Microsoft Azure virtual machine have been employed to emulate IoT device, edge device, and the cloud respectively. As shown in the figure, the end-to-end latency (inference time) varies significantly for the different approaches with varying network bandwidths. Therefore, network bandwidth plays a significant role in determining the end-to-end latency. Instead of solely executing on the cloud/edge/IoT device, we need to define a framework that can interplay between the three layers (i.e. cloud, edge, and IoT devices), harnessing their individual merits based on the dynamic network conditions, so that the attacks can be detected in real-time.
Moreover, as the dataset contains numerical/categorical values with vast ranges, identifying patterns may be quite difficult. Wide variations in numerical data can magnify the influence of noise present in the dataset. This noise has the potential to conceal authentic patterns, ultimately resulting in the emergence of false findings or misinterpretations. The sensitivity of different ML algorithms can be affected by the distribution and scale of the data. Extensive ranges may necessitate extra pre-processing or adjustments to enhance the efficacy of these techniques. A more robust approach needs to be formulated and developed. This will help in multi-class IDS performance improvement, while maintaining the context.

4.5.2 System Model

In this section, we discuss our system model and various assumptions. We have considered an IoT device based network. Our model considers a region having numerous resource constrained devices that are linked with each other on the network. The network maintains

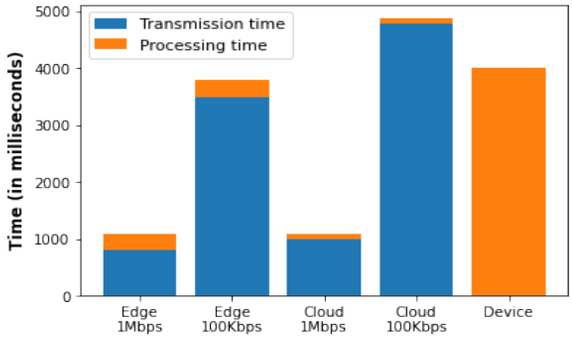


Figure 4.11: Inference time of different approaches on AlexNet under varying bandwidths.

the flow of data as well as other sequential information required for data communication. Below, we provide a formal description of the necessary parameters with respect to our considered model.

We need to minimize the inference time I_T in order to reduce the attack detection latency so as to make it real-time. I_T depends on data processing delay and data transmission delay, which make up the overall latency of the system.

Data processing delay(P_T) depends on multiple factors. In simple terms, the processing is related to the computation power of the node. It may not be feasible to have access to high Computational nodes at all time, due to, for example, economics. Our approach tends to complete the provided task (i.e. inference instances) using Cloud/edge resources, offering satisfactory performance.

The data processing delay is defined using the following equation:

$$P_T = P_{RT} + M_T \tag{4.4}$$

Here, P_{RT} is the data instance pre-processing time, and M_T is the model prediction time. In order to reduce the processing delay when the bandwidth is low and the number of data points for inference are high, we process the data points on the local cluster (discussed in Algorithm 5). In the case of multiple processing nodes, the concept of parallelism helps in improving performance, while trading-off some extra resources. However, in case of low bandwidth and less number of data points, processing happens on a single edge device.

We have further tried to reduce the above processing latencies by performing "dynamic right-sizing" of the deep neural network (i.e. Early-exit mechanism). Specifically, the exit point for the neural network is decided based on the entropy calculation, which is computed using following formula:

$$Entropy = y * log(y) \tag{4.5}$$

In above equation, y is the output vector. High entropy signifies no confidence (i.e. low accuracy), and low entropy signifies high confidence (i.e. high accuracy). So, the threshold value of entropy will define the exit-point of DNN. If the threshold value is set as high, then the DNN model may exit at early layers with low latency, with sub-optimal accuracy.

If the threshold value is set as low, then the DNN model will exit at later layers with high confidence output, but high latency.
Data transmission delay(D_T) depends on network traffic and the undergoing network bandwidth. D_T is computed using following equation:

$$D_T = Instance\_size/Bandwidth \tag{4.6}$$

In order to minimize the transmission delay, we are making use of a local cluster in case of high traffic with low bandwidth. The contribution of D_T to the detection latency becomes less significant in the case of high bandwidth. Considering the above delays we can write the final inference time as below:

$$I_T = D_T + P_T \tag{4.7}$$

Here, I_T gives the overall inference latency.
Problem Definition : Assuming that our application needs to meet its deadlines, but can bear moderate accuracy losses, we need to balance out accuracy and latency. So, in our work, we have optimized the DNN right-sizing such that the inference accuracy is maximized within the defined latency requirement. In upcoming sections, we discuss in detail how we optimise the above latencies in order to make our system fast as well as robust under various dynamic environments.

4.5.3 Proposed Work

In the proposed work, we consider two types of IDS: NIDS and HIDS. The datasets used in experimentation under the NIDS domain are: NSL-KDD [110], CICIDS 2017 [111], ToN_IoT [113], and UNSW-NB15 [112]. In the case of HIDS, we have used the benchmark ADFA-LD dataset [115]. The proposed work comprises of two phases: Data pre-processing phase and inference task allocation phase. The data pre-processing is discussed in this section, while the inference task allocation algorithms are discussed in Section 4.5.8.
The data pre-processing is carried out in four steps: feature extraction, selecting features, data normalization, and feature transformation. Even though we followed a consistent procedure for all IDS datasets, the specific instructions within each step depend on the type of IDS i.e. NIDS or HIDS. The graphical representation of this procedure can be found in Figures 4.12 & 4.13, illustrating the sequence of steps involved. After applying the above pre-processing steps, we obtain an RGB image which is given as input to BranchyNet. RGB images were created due to the following reasons:

- As the dataset contains numerical/categorical values with vast ranges, identifying patterns manually using n-grams may be quite difficult, and may miss some useful patterns. This may ultimately result in the emergence of false findings or misinterpretations, which may lead to the poor performance of the models in terms of multi-class classification.

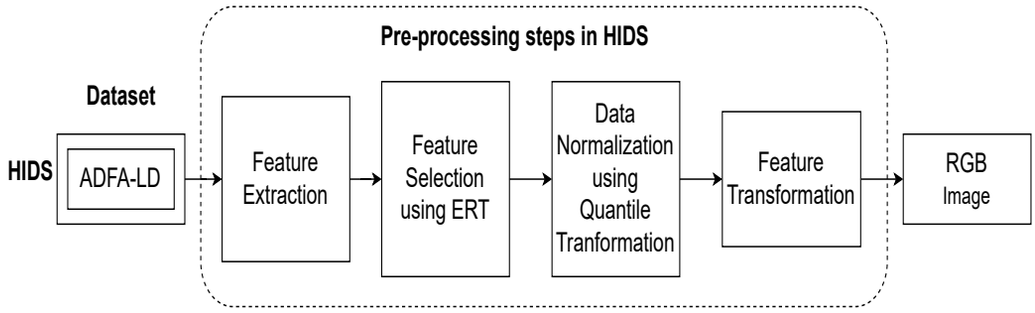


Figure 4.12: Pre-processing steps for HIDS dataset

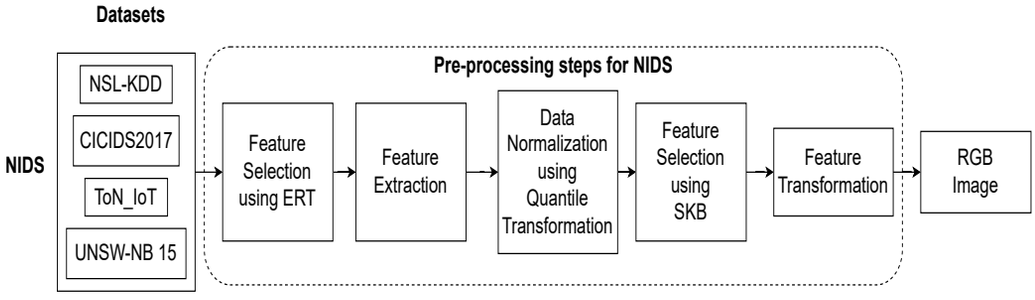


Figure 4.13: Pre-processing steps for NIDS datasets

- Utilizing multiple convolutional and pooling layers allows DNNs to efficiently capture the spatial correlations and patterns in the images.
- Through numerous iterations, DNNs can autonomously learn object features, removing the need for manual feature-engineering tasks.

The upcoming sections will provide an explanation of the pre-processing steps employed.

4.5.4 Feature Extraction

The procedure of feature extraction varies for each IDS (i.e. HIDS and NIDS) and is described individually. This distinction is necessary due to the unique context found in the collected data of each IDS.

4.5.4.1 Feature Extraction of NIDS

As stated in Table 4.4, all NIDS datasets used in our work i.e. NSL-KDD, CICIDS2017, UNSW-NB15, and ToN_IoT comprise of numerical & categorical features. Considering this, the feature is depicted as single or multiple pixels. In general, the features have either non-negative or nominal numerical values. Given the heterogeneous nature of the features, it is more advantageous to employ a feature extraction approach to unify each input feature. Categorical features are treated as a single pixel and computed using equation (4.8).

$$f_c = \frac{a_c * 255}{c_m} \tag{4.8}$$

Here, a_c represents the actual feature value, while c_m signifies the category's maximum

value. Categorical features present in the datasets can take either discrete or symbolic values. If the original dataset includes symbolic features, they are converted into discrete values.

The numerical values may exhibit continuous characteristics without specific limits. Due to the substantial variation in maximum values across various features, we opted to distribute these values across multiple pixels instead of consolidating them into single pixel. Our initial step involved examining the maximum values of each feature independently within the training dataset. In total, two distinct procedures are employed. The first procedure involves calculating the pixel count and their corresponding values for numerical features, using equations (4.9) & (4.10).

$$NoP = \begin{cases} \frac{a_c}{255}, & a_c \leq 25,500 \\ \frac{a_c}{255^2}, & a_c > 25,500. \end{cases} \tag{4.9}$$

$$V(NoP) = (NoP - \lfloor NoP \rfloor) * 255 \tag{4.10}$$

In this scenario, NoP represents the total count of pixels allocated for the numeric values. NoP is further broken down into the integer part, denoted as $\lfloor NoP \rfloor$, representing the count of pixels set to 255. Additionally, $V(NoP)$ denotes the value of the $(\lfloor NoP \rfloor + 1)^{th}$ pixel. Any remaining pixels, if applicable, are assigned a value of 0. The overall pixel count for each feature is determined by rounding up NoP to the nearest integer value, which corresponds to the maximum value of the feature. To prevent computational slowdown caused by features with excessively large values, we opted to divide features that have maximum values exceeding 25,500 by 255^2. This precautionary measure is particularly crucial in the case of the CICIDS 2017 dataset, which contains larger values.
The second approach for handling numerical values follows the same principle as described in equations (4.9) & (4.10). However, it is specifically implemented for features that represent data size, such as destination bytes and source bytes. Given that the range of values for these features is considerably greater than that of others, relying solely on the previous method would be inefficient. As a result, we devised a different approach for these features. We assigned a total of 16 pixels to represent this feature, which were divided into four quadruple sections. The initial four pixels depict the samples with data sizes below 1 Kilobyte, the next four pixels depict the samples having data size ranging from 1 Kilobyte to 1 Megabyte, the next four pixels represent samples ranging from 1 Megabyte to 1 Gigabyte, and the final four pixels represent samples with data sizes exceeding 1 Gigabyte. The calculation of pixel values still follows equations (4.9) & (4.10), but in this case, the data is converted into the appropriate data size measurement prior to computing the pixel values. The advantage of distributing the feature value across multiple pixels, as opposed to a single pixel, is that it helps prevent the normalization step from diminishing the influence of other features with lower values. This is important because such features might have a more significant contribution to the classification of a particular intrusion.

Table 4.4: Datasets with Data Type and sample data.

DATASET	**TYPE OF DATA**	**SAMPLE DATA**
NSL-KDD	NUMERICAL/ CATEGORICAL	1, udp, http_data, SF, 443, 0, 0, 1, 0, 0, 001, 1, 0, 0, ...
CICIDS2017	NUMERICAL/ CATEGORICAL	3445, 111,745,690, 34, 16, 6445, 1152, 405, 0, 201.5, 204.7242045, 72...
ADFA_LD	CATEGORICAL	6,5,43,6,45,123,6,195,120,6,6,134,145,1,1,251,243, 245,1,0,0,1...
UNSW-NB15	NUMERICAL/ CATEGORICAL	0.000011 udp – INT 2 0 496 0 90909.093750 254 . . . 1 2 0 0 0 1 2 0 normal 0
ToN_IOT	NUMERICAL/ CATEGORICAL	1554198358 3.122.49.24 1883 192.168.1.152 529 tcp - 80549.53 1762852 41933215...

4.5.4.2 Feature Extraction of HIDS

In the case of HIDS, we have used the ADFA-LD dataset, which consists of system call traces of varying lengths, that cannot be directly utilized. The dataset is created by capturing the system calls in a Ubuntu Linux 11.04 operating system that has been fully patched [119]. This particular Linux version is built on kernel v2.6 which comes with 326 system calls by default. However, for our specific requirements, we opted to employ a feature vector with a size of 350 to accommodate potential custom system calls. Each element within this feature vector corresponds to a unique system call, and its value represents the ratio of that system call's occurrence to the overall size of the trace, as defined in equation (4.11).

$$CR_j = \frac{|C_j|}{|S_T|} \; \forall j, 1 \leq j \leq 350 \tag{4.11}$$

In this context, CR_j denotes the ratio of the j^{th} system call within the final feature vector. $|C_j|$ represents the total count of occurrences of the j^{th} system call in the given trace, while $|S_T|$ represents the size of the trace. The subsequent step of our pre-processing procedure involves removing redundant or less significant features from the samples, as discussed in the next subsection.

4.5.5 Feature Selection

By determining the feature set prior to inputting it into BranchyNet, we save additional processing time that would otherwise be spent on the DNN learning features that may have less relevance compared to other important features for the ultimate classification. Consequently, feature selection helps reduce the feature space, leading to improved performance and classification time [120]. The feature selection method employed remained the same for all datasets, with the only variation being the order in which it was performed. For NIDS datasets, feature selection was conducted before feature extraction to achieve optimal results. However, for the HIDS datasets, feature selection

was carried out after the feature extraction step. In our study, we employed two feature selection methods, namely Extremely Randomized Trees (ERT) [121] and Select K-Best (SKB) along with Chi-squared score function, utilizing the classification tools provided by scikit-learn [122]. A score is assigned to each feature using these feature selection methods, indicating its importance relative to other features. Notably, the ERT classifier demonstrated superior performance and was primarily utilized in this research. SKB, on the other hand, was exclusively applied to the NIDS datasets following feature extraction. Its purpose was to eliminate unessential pixels (features) that may have a lesser impact on classification, thereby improving performance, while simultaneously reducing the input size. ERT, or Extremely Randomized Trees, constructs an ensemble model that builds upon the conventional top-down decision tree approach. However, it incorporates two key distinctions. Firstly, the splitting of nodes within ERT is carried out in a completely random manner, differing from other decision-tree based classifiers. Secondly, ERT employs the entire training data during the node splitting process, as opposed to using a bootstrap replica. The number of features to be eliminated is determined based on the feature importance score, as discussed earlier, while also taking into account the desired shape of the image. The optimal size of the image is determined through an iterative process of trial and error while training the network. To illustrate, in the case of the ADFA-LD dataset, which initially consists of 350 features and a final image size of 10 by 10 pixels, we would need to eliminate the 250 features with the lowest ranking. In the case of NIDS datasets, we have subsequently decreased the resolution of the resulting samples by applying the SKB feature selection process to all pixels. This particular step proved beneficial in reducing the pixel count by 30-50%, resulting in improved classification time and enhanced accuracy of the IDS.

4.5.6 Data Normalization

To ensure consistency with the pixel value range of images, which typically spans from 0 to 255, it is necessary to normalize the network data within the same scale. Two commonly utilized normalization techniques are: min-max normalization and quantile normalization, both of which aim to convert data values to a standardized range. However, due to the limitations of min-max normalization in handling outliers, the proposed framework [123] adopts quantile normalization instead. This method transforms the feature distribution into a normal distribution and recalculates the values of all features based on this distribution. Consequently, the majority of variable values tend to cluster around the median values, thereby providing effective handling of outliers.

4.5.7 Feature Transformation

To enhance the data representation, we expanded the existing dataset by introducing two additional layers, effectively transforming a single sample into an RGB image. The effectiveness of RGB encoding, as opposed to grayscale encoding, has been explored in a separate study [124]. The findings revealed a significant improvement in the accuracy

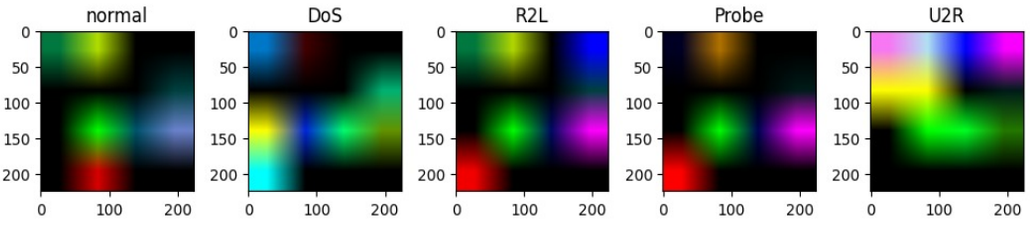


Figure 4.14: RGB images obtained after pre-processing steps for NSL-KDD dataset

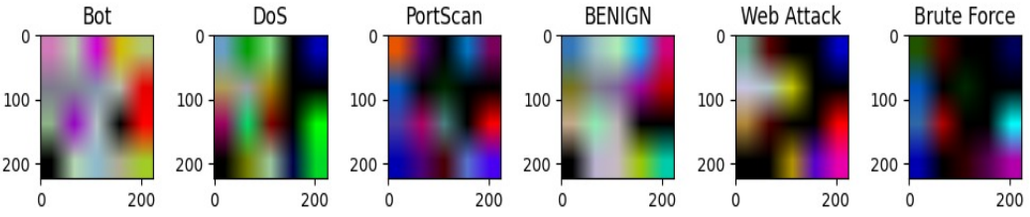


Figure 4.15: RGB images obtained after pre-processing steps for CICIDS2017 dataset

of the trained model, while utilizing RGB encoding. In our study, the RGB channels generation is done in 4 steps. The initial step involves reshaping the original data into a 2D vector. Subsequently, for the second channel, we horizontally mirror the original 2D data. The third channel is created simply by subtracting the value of each pixel from the maximum value of the current sample. Once all channels are generated, we append the standard deviation of each column and row to the corresponding column and row. After the generation of all three channels, these channels are combined and reshaped into an N x M x 3 vector suitable for DNN input. The RGB images obtained after pre-processing steps for NSL-KDD, CICIDS2017, UNSW-NB15, ToN_IoT, and ADFA_LD datasets are shown in Figure 4.14, Figure 4.15, Figure 4.16, Figure 4.17, and Figure 4.18 respectively.

4.5.8 Proposed algorithms

The previous section focused on the feature preprocessing and transformation for obtaining the RGB image corresponding to each data instance. We will now demonstrate the necessary models that will carry out the inferences, along with their location, as per the dynamic network conditions. Specifically, we propose a dynamic algorithm called $FCAFE-BNET()$, that offers the location (Cloud/edge-cluster/edge-device) of inference in such a manner that will lead to the least trade-off between inference latency and

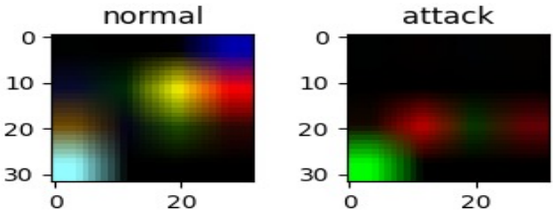


Figure 4.16: Final RGB images obtained for UNSW-NB15 dataset

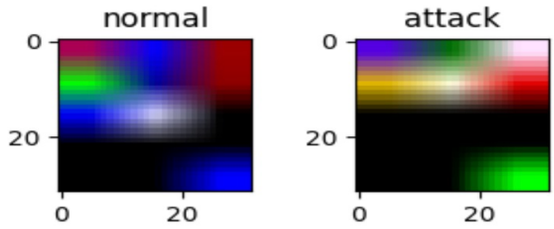


Figure 4.17: Final RGB images obtained for ToN_IoT dataset

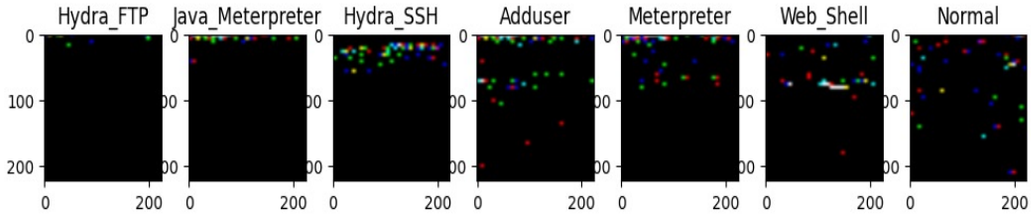


Figure 4.18: RGB images obtained after pre-processing steps for ADFA_LD dataset

corresponding accuracy. The pseudo-code for $FCAFE - BNET()$ is given in Algorithm 5. As discussed in our system model, we consider a network of IoT devices consisting of a continuous flow of data containing system calls and other networking parameters necessary for flow control and data transmission. The proposed algorithm monitors the network traffic in order to avoid any malicious attacks in real-time. For a given interval, we collect the data instances from the network traffic, along with the bandwidth details. Next, we need to generate inferences within a given latency constraint. To do that, we check whether the value of I_T i.e. total inference time (discussed in section 4, system model) is within the latency requirement represented as L_R in Algorithm 5. In Algorithm 5, we defined two thresholds: T_{BA} and T_{NI}, which are the threshold of bandwidth and the number of data instances, respectively. If the recorded bandwidth (B_A) exceeds the defined threshold T_{BA}, it implies that we can execute that instance on the cloud without violating the latency requirements. Upon failing the above condition, we proceed by the density of data instances. When the density of data instances is high, the number of current instances (N_I) exceeds T_{NI}, so we send the instances collected to the master node of the cluster executor, discussed in Algorithm 6 (i.e. $Local - Cluster - Executor()$). This ensures that inferencing of a large number of instances is performed within given latency constraints, when the recorded bandwidth is not sufficient for execution on the cloud executor. When the number of instances does not violate the imposed threshold, we prefer the local executor for the generation of inference, as discussed in Algorithm 7 (i.e. $BNETalgorithm$). Next, we discuss the functioning of modules $BNET()$ and $Local - Cluster - Executor()$, which are responsible for local and cluster execution of data instances, respectively.

We have defined about the need for $Local - Cluster - Executor()$ earlier in Section 4.5.8. Now, we will demonstrate how it handles the instances while generating the inferences. There is a master node whose job is to carry out the instance partitioning among the worker

nodes. Upon receiving the instance set, the master nodes distributes it equally over the cluster, in order to obtain fast inferencing. The data instances I_C are first pre-processed and transformed into an RGB image, using the steps stated in section V. This is done on each cluster node. All the nodes contain the model that carries out the prediction, after obtaining the RGB images. Upon inference generation, the results are added into I_R, which is maintained by the master node. The master node then sends the inference report back to the network administrator, who analyzes the results and takes necessary action such as blocking the suspicious nodes.

Algorithm 5: FCAFE-BNET(B_A, I_C)

Require: Bandwidth and Network Traffic
B_A: Bandwidth recorded after t time interval
I_C: Instance-Collector()
N_I: Number of instances(I_C)
I_T: Total inference time
L_R: Latency Requirement
Ensure: Location of inference generation

```
while I_T ≤ L_R do
    if B_A > T_BA then
        Execute on cloud
    else if B_A <= T_BA AND N_I > T_NI then
        Run on Local-Cluster-Executor(I_C)
    else
        /* Run on local Edge device using Algorithm 7 */
        BNET(I_C)
    end if
end while
```

Algorithm 6: Local-Cluster-Executor (I_C)

Require: Data Instances, I_C
N: Sizeof(I_C)
K: Number of worker nodes in the local cluster
Ensure: Inference Result, I_R

```
Send data instances to the master node
for each worker nodes : do
    PP_I: Preprocess(I_C) using steps discussed in Section V
    T_I: Transform(PP_I) into RGB image using steps discussed in Section V-D
    Process N/K data instances using trained model
    Keep on appending the results into I_R
end for
Return I_R
```

The pseudo-code and workflow of the proposed $BNET$ algorithm is represented in Algorithm 7 and Figure 4.19. In the case of $BNET$, the model is deployed on the local fog device. So, it needs to perform fast inferencing, as it only contains a single computation node. Firstly, all the data instances are pre-processed and transformed into RGB images on the fog node, using the steps stated in Section V. We have used the

Algorithm 7: BNET (I_C)

Require: Data Instances
  N: Number of exit points
Ensure: Inference Result
  for i *in* I_C **do**
    PP_I: Preprocess(I_C) using steps discussed in *Section V*
    T_{PP}: Transform(PP_I) into RGB image using steps discussed in *Section V-D*
    for $j \leftarrow 1$ to N **do**
      $X \leftarrow FExit_j(T_{PP})$
      $Y \leftarrow Softmax - Activation(X)$
      $E \leftarrow Entropy(Y)$
      if $E < T$ **then**
        add label(argmax(Y)) to I_R
        Break
      end if
    end for
  end for
  Return I_R

Branchynet framework of deep neural networks in the $BNET$ algorithm. Unlike usual deep neural network models, branchynet consists of multiple exit points. These multiple exit points help in balancing the latency-accuracy requirements. Here, we iterate over each data instance and calculate the value of entropy corresponding to each instance in the output vector. If the value of entropy is below the defined threshold (T), then we output the data instance from that exit point itself. Otherwise, we proceed with further exit points. Similar to $Local - Cluster - Executor()$, here also we keep on adding the inferences to the output vector I_R. The threshold T imposed on the exit points has been defined experimentally, and it varies with the nature and complexity of the input dataset.

Table 4.5: Dataset description.

Datasets	Test Data Points		Train Data Points	
	Attack	Non-Attack	Attack	Non-Attack
ToN_IoT	74512	73022	210311	103198
UNSW NB15	45332	37000	119341	56000
CICIDS2017	33626	136219	78217	318087
NSL-KDD	12833	9711	58630	67343
ADFA-LD	104	3398	422	563

Table 4.6: Total inference time comparison for various techniques on NIDS based datasets.

Dataset	FCAFE-BNET (Proposed)	FC-XGB	CAFE-CNN	GTO_BSA	GTO
NSL-KDD	**11.41**	145.50	47.71	10205.83	9719.66
CICIDS-2017	**45.31**	170.34	87.68	2270.91	6988.46

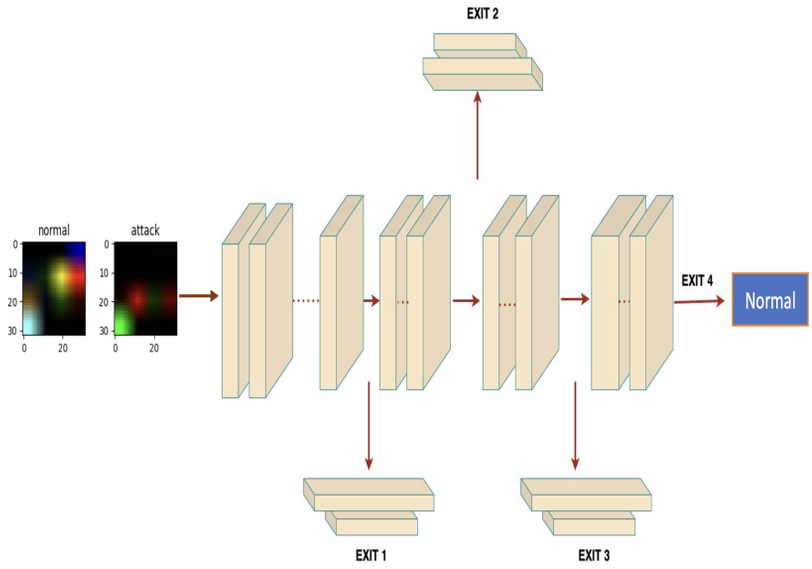


Figure 4.19: Workflow of the proposed BNET algorithm

Table 4.7: Total inference time comparison for various techniques on HIDS dataset.

Dataset	FCAFE-BNET (Proposed)	LW-MLP	FC-XGB	CAFE-CNN
ADFA-LD	**15.36**	817	650	49.72

4.5.9 Results

In order to evaluate the performance of our proposed $FCAFE - BNET$ algorithm, we have deployed a prototype using Azure cloud and Raspberry Pis. The cloud setup has been done using Microsoft Azure cloud service, with 16GB RAM. The local cluster has been deployed using five Raspberry Pi 4s Model B, having 4GB RAM and Cortex-A72 ARMv8 processor having quad core with processing frequency of 1.5GHz. Each Raspberry Pi is installed with 32 GB MicroSD cards fro internal storage. In the cluster comprising of five Raspberry Pis, one Raspberry Pi acts as a master node, and rest of the four Raspberry Pis act as worker nodes. The networking of the cluster has been done using ethernet, and a network switch with 100 Mbps bandwidth. Apache Hadoop and Spark [125] frameworks have been employed in order to harness the distributed ML cluster. A single Raspberry Pi 4 has been used as the local fog device. In order to incorporate Branchy DNN for multi branch DNN training, we have used Chainer [126] and BranchyNet [127] frameworks on local fog devices. The AlexNet [128] model with three exit points has used for training over various IDS datasets. The AlexNet comprises of eight layers, out of which the first five layers are convolutional layers and last three layers are fully connected layers. We have selected the odd convolutional layers as the three exit points. The description of training set and test set of all the datasets used in the study is given in Table 4.5. The ADFA-LD, UNSW-NB 15, and ToN_IoT datasets have been evaluated for binary class classification. The evaluation metrics used are: Accuracy, Precision, Recall, F1-score [129], and total inference time (I_T, described in section 4.5.2). The NSL-KDD and CICIDS 2017 datasets have been evaluated for multi-class classification, so we have used the weighted average of

the precision, recall, F1-score [129], and Accuracy along with total inference time.

Table 4.8: Total inference time comparison for various techniques on IoT based NIDS datasets.

Dataset	FCAFE-BNET (Proposed)	EHIDS	CF-OSELM	ABA-IDS	ICNN-FCID
ToN_IoT	**9.76**	16.68	19.94	20	22.42
UNSW-NB 15	**17.9**	29.89	35.73	35.85	40.19

4.5.9.1 Multi-class classification performance:

The performance of FCAFE-BNET has been compared with various other approaches discussed in Chapter 2, Section 2.2, for the multi-class classification problem on NSL-KDD, CICIDS 2017, and ADFA_LD datasets. As shown in figure 4.20, in case of the NSL-KDD dataset, FCAFE-BNET performs well in comparison to the latest proposed approaches. The performance boost is due to it's unique pre-processing steps that convert the data into RGB images, which helps the DNN in extracting various patterns useful to detect different classes of attacks. Also, the performance of FCAFE-BNET is better than CAFE-CNN due to fine-tunning the pre-trained model (i.e. AlexNet in our case) on given NSL-KDD dataset, instead of using untrained CNN in the case of CAFE-CNN and starting from scratch. Meta-heuristic algorithms, such as GTO-BSA and GTO have performed well because of finding the optimal feature set using optimization algorithms. But, as shown in Table 4.6, the total inference time is quite high, for both GTO-BSA and GTO. This is because both of the optimization algorithms take a long time to figure out the best solution. Hence, these approaches may not be suitable for real-time applications, like intrusion detection in IoT networks. Comparatively, our proposed approach takes significantly less amount of time to generate inferences.

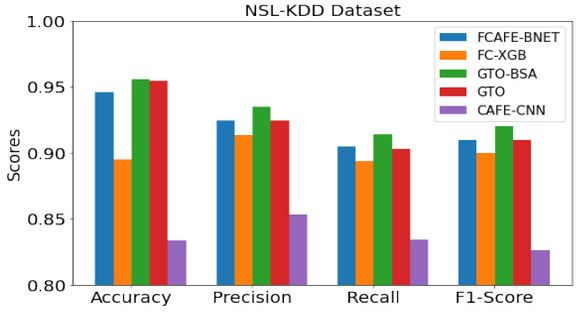


Figure 4.20: Performance comparison on NSL-KDD dataset.

As shown in Table 4.6, the inference generation time is the least in the case of FCAFE-BNET, because of two reasons. Firstly, FCAFE-BNET helps in locating the appropriate inference generation location, depending upon the dynamic network conditions. In addition, the low inference time is due to the use of the early-exit mechanism in the local fog device. This reduces the computation by exiting the output from early layers with a certain confidence interval, instead of traversing the complete DNN for the output. The poor performance of FC-XGB in terms of inference time is observed, because

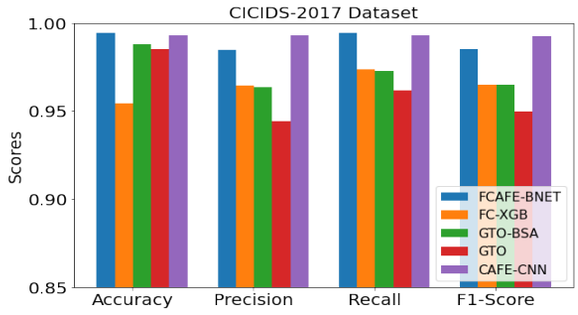


Figure 4.21: Performance comparison on CICIDS 2017 dataset.

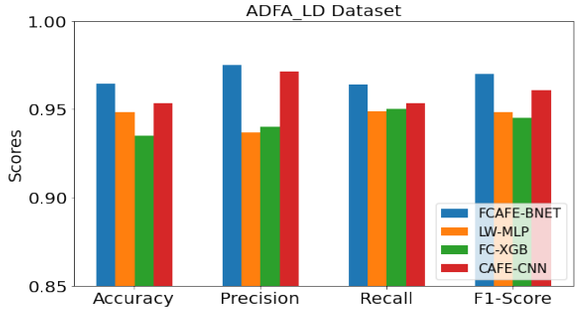


Figure 4.22: Performance comparison on ADFA_LD dataset.

it always offloads the inference task on the local cluster, without considering the dynamic network condition.

As shown in Figure 4.21, in case of the CICIDS 2017 dataset, FCAFE-BNET approach has outperformed other recent approaches in terms of the weighted average of accuracy, precision, recall, and F1-score. This is because the RGB images obtained for different attacks after performing pre-processing steps (discussed in Section 4.5.3) are very different from each other, as shown in Figure 4.15. Due to this, the DNN correctly classifies different attacks. The CAFE-CNN has performed equally well in terms of all the scores, due to the use of CNN. But, as shown in Table 4.6, the inference time of an instance using CAFE-CNN is quite high as compared to FCAFE-BNET, due to the latter's early-exit mechanism. The number of computations is reduced significantly due to early-exit mechanism. Hence, the resource constraint fog device can quickly perform inference tasks without compromising the accuracy score.

As shown in Figure 4.22, in the case of ADFA_LD, FCAFE-BNET has performed the best in terms of all scores, because of adopting a good feature extraction and transformation method, discussed in section 4.5.3. Also, using a pre-trained DNN model i.e. AlexNet has significantly enhanced the performance. The performance of LW-MLP and FC-XGB is observed to be the worst, because of selecting the features manually by using the n-gram technique. Also, in case of the LW-MLP, the authors have used a shallow MLP model for inference, which resulted in poor performance of the LW-MLP approach. The XGboost model performed well in the case of tabular data, but as the ADFA_LD dataset comprises of system call files, due to which the FC-XGB approach did not offer optimal results. As shown in Table 4.7, the proposed FCAFE-BNET offers the best results in the case of total

inference time as well. This is because in the case of LW-MLP, the experiments were carried out on a single local fog device. Whereas, in the case of FC-XGB, the inference task is always offloaded on the local cluster. However, both of the approaches did not take into consideration the network conditions and task load for inference generation task allocation. CAFE-CNN offers better performance due to the use of GPU resourced local device for experimentation, but the authors have not considered a fog based environment in their work. The proposed FCAFE-BNET offers the least total inference time, due to locating the right inference generation location using Algorithm 5.

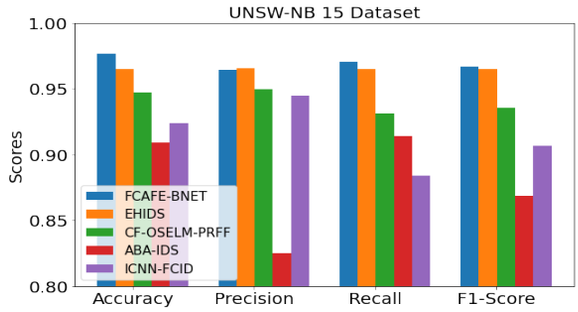


Figure 4.23: Performance comparison on UNSW-NB15 dataset.

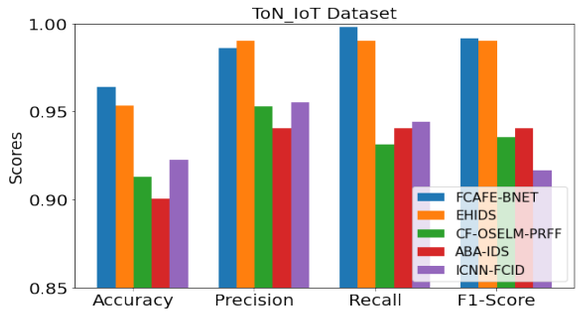


Figure 4.24: Performance comparison on ToN_IoT dataset.

4.5.9.2 Binary-class classification performance:

In this section, the proposed FCAFE-BNET approach has been compared for binary class classification problem with EHIDS [45], ABA-IDS [44], ICNN-FCID [46], and CF_OSLEM [49] approaches, using benchmark UNSW-NB 15 and ToN_IoT datasets. As shown in Figure 4.23 & 4.24, the ABA-IDS offers the worst performance in the case of all the scores, for both datasets. In ABA-IDS, node profiling was done before passing it to ANN. However, these collected node attributes do not help much in correctly classifying the attacks, thus showing the worst performance in comparison to other recent approaches. Both CF-OSELM and ICNN-FCID approaches have shown moderate performance in terms of all the scores for both datasets, shown in Figure 4.23 & 4.24. In both approaches, the authors did not adopt any feature selection method, which resulted in poor performances of CF-OSLEM and ICNN-FCID. In comparison to the recent approaches, the proposed FCAFE-BNET has performed well in terms of all the scores for both datasets. This is

because of refining the data by pre-processing, and transforming it before passing it to the DNN. The EHIDS has performed equally well because of using genetic algorithms for optimizing ANN weights and biases. However, as shown in Table 4.8, the EHIDS approach has taken a longer time to generate inference than the proposed FCAFE-BNET. This is because of FCAFE-BNET's policy of allocating the inference task to the right location i.e. fog device/ local cluster/ Cloud, instead of always allocating to the fog device. The ABA-IDS approach took a long time to generate the inference, because node profiling is a time consuming process. Finally, ICNN-FCID records the highest inference time for both UNSW-NB 15 and ToN_IoT. This is because the approach integrates two computational intensive models i.e. CNN and LSTM, for generating inference.

Table 4.9: Performance of FCAFE-BNET with 3 exit points on NSL-KDD dataset with varying Entropy threshold

Entropy Threshold (T)	Accuracy (%)	Time (s)	Exit (%)		
			Exit pt 1	Exit pt 2	Exit pt 3
0.00001	95.02	4.454	0.0	0.0	100.0
0.0001	94.78	4.445	0.0	7.0	93.0
0.001	94.54	4.180	0.0	17.55	82.45
0.05	94.08	3.743	4.37	30.18	65.45
0.01	93.95	3.134	10.34	32.32	57.34
0.75	93.83	3.234	23.65	17.66	58.69
1.5	**92.75**	**2.945**	**48.45**	**6.21**	**45.34**
2.5	84.64	2.457	57.34	2.43	40.23
5.0	69.45	2.005	68.14	1.52	30.34
10.0	54.43	1.045	99.5	0.5	0

4.5.9.3 Effect of Entropy threshold on FCAFE-BNET performance:

In this section, we discuss the effect of the entropy threshold on the performance of the proposed FCAFE-BNET, in terms of accuracy and inference time. The experiments were conducted on the benchmark NSL-KDD dataset by varying the entropy threshold value, represented as $'T'$ in Algorithm 7. The Branchy AlexNet comprises of three exit points represented as: Exit pt. 1, Exit pt. 2, and Exit pt. 3, respectively. Here, Exit pt. 1 depicts early exiting from the first convolutional layer, whereas Exit pt. 2 depicts early exiting from the third convolutional layer, and Exit pt. 3 depicts exiting from the last convolutional layer (i.e. complete AlexNet network). As shown in Table 4.9, when the value of $'T'$ is very low, a large number of data points exit from Exit pt. 3 (i.e. exit from the last layer). Correspondingly, the accuracy recorded by the inference model is the maximum. However, note that the inference time corresponding to this case is high as well, because of exiting from the last layer. In case of setting a very high entropy threshold, a large number of data points exit from Exit pt. 1. Here, the DNN model exits the outputs at an early layer, with comparatively lower confidence. Therefore, the inference time recorded by the inference model is extremely low, with low accuracy. On the other hand, when the value of $'T'$ is set to 1.5, then both the performance metrics i.e.

accuracy and inference time are balanced. This is because approximately 55% of the data points exit from early layers, due to which the inference time recorded is low. Also, the accuracy recorded is high, because the DNN model exits the outputs from both early and last layers, based on the entropy threshold.

4.6 Conclusion

IoT applications are vulnerable to various attacks, due to which many researchers have proposed different Intrusion Detection Systems (IDS) that can secure the IoT network. Often, these approaches fail to detect various classes of attack efficiently. The poor performance is the result of adopting outdated feature selection, extraction, and transforming methods. In addition, these approaches do not take into account versatile network conditions. The proposed FCAFE-BNET approach improves the multi-class IDS performance by exploiting various pre-processing steps that help in identifying various attack patterns correctly. The proposed FCAFE-BNET algorithm takes into account dynamic network conditions before allocating the tasks to different fog layers i.e. Cloud/cluster/fog device. Moreover, the use of the early-exit mechanism in the local fog device speeds up the inference by reducing the number of computations, without adversely affecting the performance. The proposed technique is not only applicable to NIDS and HIDS datasets, but can be extended to other forms of IDS as well, by making minor changes in the pre-processing step. The fog-edge architecture is a promising way to deal with various attacks. Through experimentation, we observe that both the cost and latency (inference time) of the fog (Pi Cluster) is less than a similar powered Cloud web application. In future work, we plan to consider other critical evaluation metrics for IoT systems, such as storage efficiency and energy consumption.

Chapter 5

Data Driven DNN Task Offloading on Edge Networks

5.1 Introduction

Edge computing aims to reduce bandwidth bottleneck and latency by performing computation close to the end-users, making it a viable option for application offloading. Due to its limited computational capacity, the edge may need to be considered with the cloud for offloading. Computation offloading of tasks is challenging, as it depends on: the availability of limited resources, dynamically changing network conditions, and concurrent user access. Mathematical task offloading approaches may be incapable of capturing dynamic network situations in large end-to-end network models. We propose D^2-TONE (Data-driven Deep Neural Network Task Offloading on the Network Edge), an approach that employs Machine Learning algorithms for accurately estimating offloading delays, such as computational and transmission delays. D^2-TONE holistically adapts to dynamic network situations, and provides optimal/near-optimal offloading solutions in real time. In addition, the proposed algorithm employs distributed execution of DNN tasks on edge devices/cloud data centers. Experiments reveal that D^2-TONE reduces the training time by 1.55 to 2.77 times, compared to baseline approaches. In addition, the edge-based D^2-TONE offers an improvement of 55-76% in the data processing ratio, in comparison to other offloading approaches.

The emergence of Internet-of-things (IoT) has led to the generation of huge volumes of data. There are several real-time IoT-based applications, like smart cities [130], self-driven cars [131], and industrial monitoring [132], for which the generated data needs to be processed within a given deadline [133]. The data generated through these IoT applications is usually processed using a Cloud data center (CDC). Due to the rapid increase in IoT applications, it is estimated that the volume of data generated by such applications will exceed today's network bandwidth capacity [134]. In addition, the computation capacity of modern gadgets has increased significantly in recent years. These advancements in user device hardware have led to the emergence of the "Edge Computing" paradigm.

Edge computing [135] helps in processing data near its point of generation, rather than sending the complete data to the remote cloud for processing. As the data gets processed in the proximity of users, the response time is reduced, leading to an improved user experience, enabling different real-time applications, such as vehicle OTA. Generally,

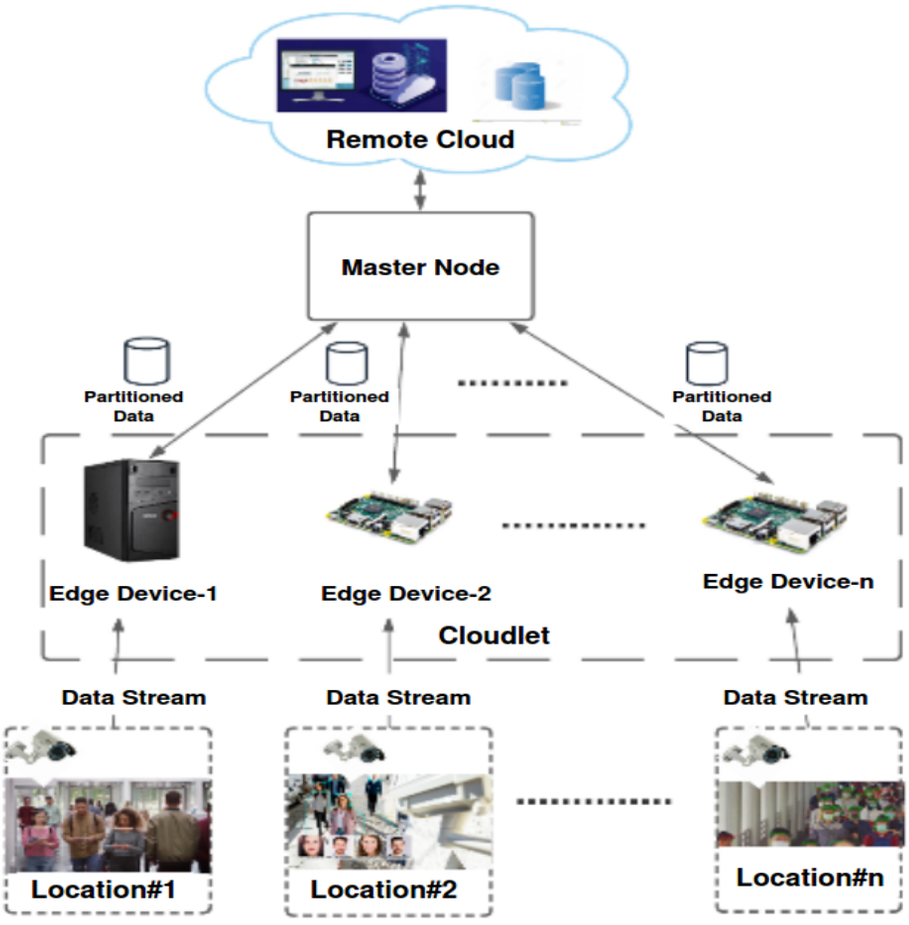


Figure 5.1: System architecture for crowd counting application.

user devices have limited resources with respect to task load [136]. Hence, computation offloading is carried out in order to balance the load among edge devices. The main goal of offloading is to distribute the task partitions among the nearby edge devices, so that the computation time and energy consumption of the system may reduce [137]. The major challenge exists in utilizing these finite heterogeneous user devices or edge resources efficiently. Mathematically optimized task offloading approaches need to appropriately model the inputs, such as: available resources, transmission delay, computational delay, task distribution size, and end users, according to certain objectives, such as minimizing the delays or energy consumption of mobile devices. In order to do this, an accurate estimation of these inputs is required. Misleading estimations may degrade the task offloading performance over time. The estimation of the computational time itself is a non-trivial task, as it requires application profiling software, which may be incompatible with the heterogeneous resources in the edge network. Moreover, edge networks are highly prone to disruptions and noise. Estimating transmission time is a tedious task, as the standard network measurement techniques fail to capture the complexities in the network. In order to address the above shortcomings, we use data-driven estimation approaches such as machine learning algorithms. This approach provides more accurate estimates, and hence, helps in efficient offloading of tasks in a multi-edge environment.

Recently, there has been an increase in the usage of surveillance cameras in various

public places for safety and security. This has brought the problem of crowd analysis to the fore. The crowd counting problem aims at detecting faces in order to estimate the number of people present in a given location. Crowd analysis has applications in several domains: monitoring safety and security, early detection of overcrowding, appropriate crowd management, and movement in public spaces. ML techniques have been used in order to extract useful information from huge volumes of data generated through IoT devices. Among various ML algorithms, Deep Neural Networks (DNN) have exhibited good performance in the case of image-based classification, and regression problems [138]. One of the main concerns of training these models well is the usage of a huge amount of data for learning [139]. However, edge devices, being resource constrained by definition, may not be able to handle huge DNN workloads. Hence, the training of the DNN model needs to be done in a distributed manner in the edge network, so that the data is processed in real-time [140].

In this work, we consider the people counting (face detection) application for shopping malls and retail stores [141]. Several advantages exist in installing traffic counter devices in malls and stores: optimizing sales, improving store operations, evaluating new ad campaign effectiveness, and visitor analytics [142]. All of the above factors have a direct impact on the profitability of the retail store [143]. There are several cloud-based APIs that are available for face detection, like Google's Cloud Vision and AWS Rekognition. However, these services require reliable and stable internet connectivity. The edge computing use case for crowd counting is shown in Figure 5.1. The data in the form of video streams/images is transmitted from cameras to nearby edge devices for processing and storage. If the local edge device is incapable of processing the received data in a given deadline, then the data is sent to the master node/ server for processing. The Master node decides if the data needs to be processed on a remote cloud, or on the Cloudlet (Edge network), based on the tasks deadlines. If the deadline constraint is hard (i.e., real-time tasks), then the Master node sends the partitioned data to edge devices present in the Cloudlet. Otherwise, the data may be sent to the remote cloud for storage and processing. This use case faces various challenges: computing resource heterogeneity, device-to-server ratio, congested, and noisy networks. These challenges make it difficult to accurately estimate the offloading cost. Moreover, the complexities of such networks can not be effectively handled by static network profiling approaches, as these approaches can degrade the offloading performance over time with inefficient task scheduling.

5.2 Contributions

The major contributions of the proposed work are:

- We propose data-driven machine learning techniques for estimating the computation and transmission offloading costs for edge networks. These techniques are based on various dataset parameters, ML parameters, and hardware/network parameters for dynamic workloads. This leads to more accurate predictions of offloading costs as

Table 5.1: Estimated computation offloading cost (C)

ED \ J	j_1	j_2	...	j_l
1	$C(1, j_1)$	$C(1, j_2)$	...	$C(1, j_l)$
2	$C(2, j_1)$	$C(2, j_2)$	...	$C(2, j_l)$
⋮	⋮	⋮	⋮	⋮
n	$C(n, j_1)$	$C(n, j_2)$	...	$C(n, j_l)$

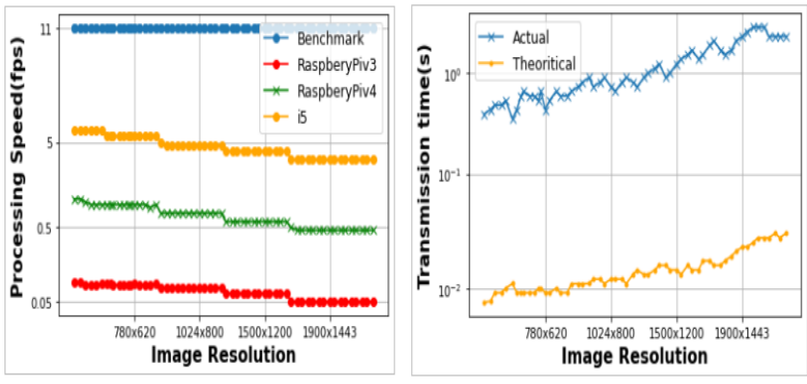
compared to static estimation methods.

- We propose a framework for the optimal computation offloading of application data-points on various edge devices by proposing a Mixed Integer Programming (MIP) based approach, which minimizes the training time of the given workload, and maximizes the data-point processing ratio.

- The proposed framework significantly reduces the training time of DNN model by updating model parameters in a parallel and distributed manner on various edge devices, without sacrificing the performance of the trained model.

5.3 Motivation

We conducted an experiment to measure the processing speed and transmission time for varying network conditions and hardware types, the results of which are shown in Figure 5.2. The computation time depends on the processing speed of the hardware (i.e. $computation\ time = \frac{Number\_of\_frames}{Processing\_speed(infps)}$). However, accurately estimating the processing speed is not a trivial task in a large heterogeneous network. As shown in Figure 5.2a, the processing speed varies tremendously with different hardware types. The benchmark used in our study is a face recognition application [144], where the processing speed for all CPUs is considered to be static (i.e., 11 frames per second). Similarly, in Figure 5.2b, the theoretical ($transmission\_time = \frac{Data\_size}{Bandwidth}$) transmission time estimates are much lower than the actual (measured) transmission time. The inaccuracy is due to merely relying on the link bit rate (Mbps) for estimating the transmission time. In the real world, this depends on dynamic network conditions. Therefore, if these static computation and transmission time estimating techniques are used to model the inputs of mathematical optimization task offloading approaches (like MIP), then the inaccurate estimations might degrade the offloading decisions over time. Hence, for providing a more accurate estimation of computation and transmission times, we propose data-driven estimation approaches (machine learning algorithms), which help in optimizing task offloading in a multi-edge-cloud environment.



a) Processing speed Vs Image resolution b) Transmission time Vs Image resolution

Figure 5.2: Experiment analysis on a) Processing speed for varying hardware types and b) Transmission time for varying network conditions.

5.4 System Model

We now discuss our proposed system model. The key notations are depicted in Table 5.2. We consider $'n'$ heterogeneous edge devices, such that $ED = \{1, 2, ..., n\}$, where each edge device $e \in ED$ may have diverse computational capacities. The data point batch j_x comprises of images/frames that need to be processed. The set of all data point batches is represented by: $J = j_1, j_2, ..., j_l$, such that the data size of $j_1 < j_2 < ... < j_l$. The computation offloading cost of $'j'_q$ data batch, where $j_q \in J$ on edge device $'e'$, $e \in ED$ is represented by $C(e, j_q)$. The computation offloading cost $C(e, j_q)$ defines the time taken to process data batch j_q on edge device e. Table 5.1 shows the estimated computation offloading cost $C(e, j_q)$, where $e \in ED$ & $j_q \in J$. The data-points batch j_q, $j_q \in J$ selected for processing on edge device $'e'$, $e \in ED$ is termed as local dataset $'D'_e$ for device e. Similarly, the local dataset for all the $'n'$ edge devices $e \in ED$ are represented as $D_1, D_2, ..., D_e, ..., D_n$. The training time taken to finish the given task $'D'_e$ on edge device $e \in ED$ is computed using the following equation:

$$U(e, D_e) = C(e, D_e) + tt(D_e, e_m, e), \forall e, e_m \in ED \tag{5.1}$$

Here, $C(e, D_e)$ is the estimated computation offloading cost of edge device $'e'$ on its local dataset D_e. Next, $tt(D_e)$ represents the estimated transmission time of dataset D_e from master edge device e_m to e. The transmission time $tt(D_e, e_m, e)$ comprises of data transfer time and propagation delay. The training time taken to finish the given task is:

$$U(e, D_e) = C(e, D_e) + \frac{s(D_e)}{bw(e_m, e)} + pd(e_m, e), \forall e, e_m \in ED \tag{5.2}$$

Here, $\frac{s(D_e)}{bw(e_m,e)}$ denotes the data-transfer time, where $s(D_e)$ is the data size of D_e, and $bw(e_m, e)$ is the bandwidth of network connectivity between the edge devices e_m and e. Next, $pd(e_m, e)$ represents the propagation delay. As the edge nodes have limited processing capability in comparison to cloud CD, therefore, queuing of tasks (data-points) may occur when large data-points batches (tasks) are offloaded on edge devices. Hence, we

Table 5.2: Symbols and Notations

Sets:	
J	Set of data point batches
ED	Set of edge devices
Variables:	
β	Continuous positive variable that depicts the deadline.
$\xi(e, j_c)$	Binary variable assigned 1 if data point batch $j_c \in J$ is allocated to device $e \in ED$, and 0 otherwise.
$U(e, D_e)$	Continuous positive variable that denotes the time taken to train the DNN model with the assigned data-points batch D_e at edge device $e \in ED$.
$U(ED, D)$	Continuous positive variable that denotes the total time taken to train the DNN model with all data-points in the dataset (D) assigned to edge devices ED.
Parameters:	
$C(e, j_c)$	Estimated computation offloading cost of batch j_c at device e $\forall j_c \in J, e \in ED$.
D	Number of data-points in Dataset (Dataset size).
n	Number of available edge devices.
$tt(j_c)$	Transmission time of batch j_c.
$pd(j_c)$	Propagation delay of batch j_c.
$l_k(w)$	loss function defined on the weight vector w and sample data k.
$L_e(w)$	Local loss function defined on the weight vector w at node $'e'$.
$L(w)$	Global loss function defined on the weight vector. w
$W_e^l(T)$	Local parameter for each node e with iteration index $T = 0, 1, 2, ..,$ and so on.
$W^g(T)$	Global parameter at master node with iteration index $T = 0, 1, 2, ..,$ and so on.

consider that all edge devices maintain an individual data-points queue to buffer the DNN tasks. The cloud CD is computationally rich, so it has no queuing delays, as tasks are scheduled immediately. The queuing delay of the offloaded task depends on the computing speed of the edge device and the current state of the assigned queue. So, if the queue buffer is congested with a large number of DNN tasks (data-points), then it might lead to a long queuing delay. However, another factor that highly governs the queuing delay is the computation capacity of the device. For example, if the computing capacity of the edge is high, then more data-points will be processed in less amount of time, which results in shorter queuing delays and a high service rate. The queue backlog of edge device e at $t + 1^{th}$ time instance, is defined as follows:

$$b(e, t + 1) = |b(e, t) - \Theta(e, t) + a(e, t)| \tag{5.3}$$

Here, $b(e, t)$ is the queue length/backlog of e at t^{th} instance. $\Theta(e, t)$ denotes the DNN tasks that left the queue of e at t^{th} instance, after being processed by e. The number of DNN tasks that arrived at t^{th} instance on edge device e is represented by $a(e, t)$. The estimated queuing delay (time taken to process the queue backlog) is added to the training time of the DNN model. The queuing delay is represented by $qd(e)$. Hence, the total training time taken to finish the given DNN task after including all delays, is defined as follows:

$$U(e, D_e) = C(e, D_e) + \frac{s(D_e)}{bw(e_m, e)} + pd(e_m, e) + qd(e) \tag{5.4}$$

The real-time task $'D'_e$ assigned to edge device $'e'$, where $e \in ED$, must complete its computation within the given deadline β.

$$U(e, D_e) \leq \beta, \forall e \in ED \tag{5.5}$$

Therefore, if all the edge devices $e \in ED$ complete the assigned task $'D'_e$ in the given deadline $'\beta'$, the total training time $'U(e, D_e)'$ to process all the tasks / data-points in the complete dataset $'D'$ such that $\sum_{e=1}^{n} D_e = D$ is :

$$U(ED, D) \leq \beta \tag{5.6}$$

5.5 Problem Formulation

The crowd counting use case involves offloading of facial recognition tasks from end devices like cameras to nearby cloudlets/servers. The inherent challenges in this multi-edge scheduling are as follows: (a) accurate offloading cost estimation in varying network conditions, (b) limited storage and computational capabilities of end devices, (c) poor offloading decisions may introduce network congestion, and (d) availability of limited offloading resources for computation and processing. We now discuss the problem formulation of our proposed scheme that attempts to mitigate the above concerns.

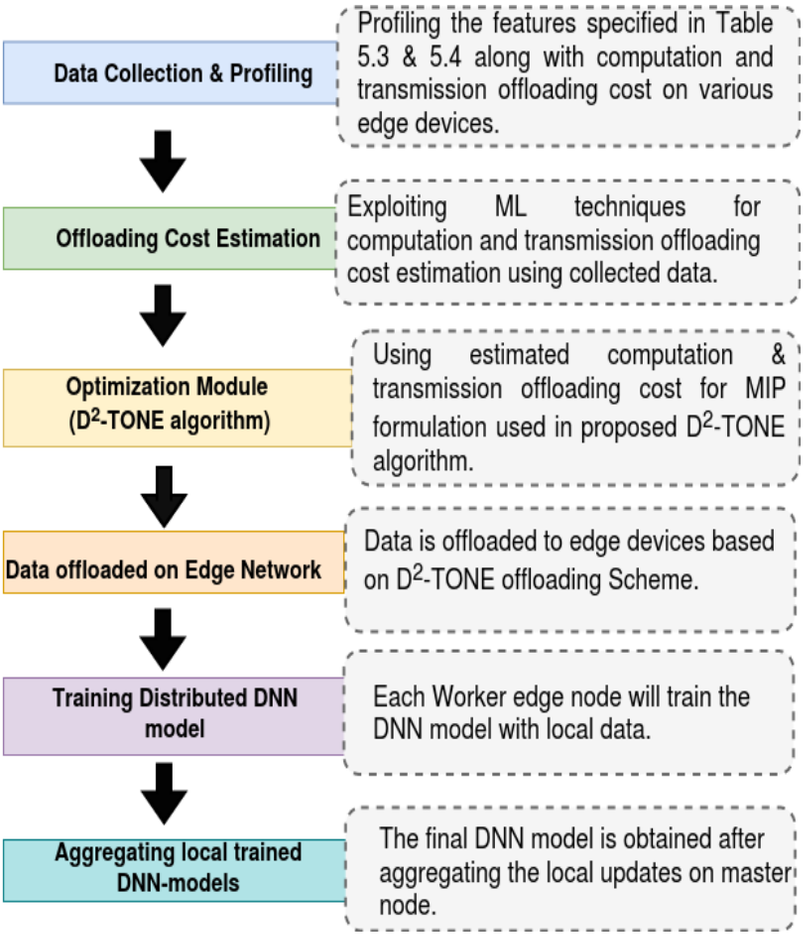


Figure 5.3: Workflow of Proposed framework.

5.5.1 Proposed DNN Task Offloading Framework

The proposed $D^2 - TONE$ (Data Driven DNN Task Offloading on Network Edge) framework can be used for scheduling applications such as face detection on multi-edge architectures. In order to deal with inaccurate estimations made by traditional computational and transmission offloading cost measuring techniques, we employ data-driven ML models trained on historical device data for computation and transmission offloading cost prediction. The trained ML model helps in predicting device performance more accurately, which in turn helps in optimizing the task offloading schedules on edge networks. An overview of proposed framework is shown in Figure 5.3. First, in the data collection phase, the profiling of the device and network parameters is carried out. Next, these parameters are used by the ML models as input features for predicting computation and transmission offloading costs. Next, the offloading cost predictions are used by the optimization module for deriving the task offloading schemes. On the basis of the task offloading schedule given by the optimization module, the tasks are dispatched to the edge devices present in the network. After the data is offloaded to the edge devices, the training of the DNN model is initiated in a distributed manner, with local datasets. Finally, the aggregation of local parameter updates is carried out on the master edge device, and the final trained DNN model is obtained.

5.5.2 Predicting offloading cost

In order to determine/predict the computation and transmission offloading cost of edge devices with varying computation capacity (i.e., hardware configuration), network conditions and dataset characteristics, we have exploited various ML models such as: Decision tree, Linear regression, Support vector machines, Random forest, Multi-layer perceptron, and K-Nearest Neighbour. These models help us to determine the approximate computation and transmission time for each edge device in the face of varying network conditions. This allows the varying task load/data size to get processed on different edge devices, according to their computational capacity. The features that have been considered for training ML models, for predicting computation offloading cost $C(e, j_c)$, and transmission offloading cost $tt(j_c, e_m, e)$, where $e, e_m \in ED$ & $j_c \in J$, are listed in Table 5.3 and Table 5.4, respectively. Further, the performance of various ML models used for estimating computation & transmission offloading costs is discussed in section 5.8.1

5.5.3 MIP scheduling problem

The optimization problem is formulated using $'n'$ heterogeneous edge devices, where each edge device $e \in ED$ has a different computational capacity. The dataset comprises of $'d'$ data-points. The training algorithm needs to run for $'v'$ number of epochs. The total number of data-points that need to be processed is $D = v.d$. Table 5.1 shows the computation offloading cost $C(e, j_c)$, $\forall e \in ED, j_c \in J$. This has been computed using a data-driven approach, as discussed in Section 5.5.2. We begin our scheduling problem

Table 5.3: Features used for estimating computation (Processing speed) offloading cost

data Attributes:	
• Image Height (Pixels) • Data Size (bytes) • No. of data-points	• Image Width (Pixels) • No. of features
ML Attributes:	
• Number of Epochs • Number of neurons per layer • Number of layers	• Number of model parameters • Learning rate
H/W Features::	
• Processor Speed (Mhz) • Storage (KB) • LAN operating freq. (Mhz)	• RAM (KB) • Switch Bandwidth (Mbps) • Ethernet cable (feets)

Table 5.4: Features used for estimating Transmission time offloading cost

data Attributes:	
• Image Height (Pixels) • Data Size (bytes) • No. Of data-points	• Image Width (Pixels) • No. of features
H/W Features:	
• Bitrate (Mb/sec) • Packet Loss (%) • LAN operating freq. (Mhz)	• Ethernet cable (feets) • Mean RTT (sec) • Jitter (sec)

formulation by defining various variables, constraints, and parameters. The first binary variable is defined as follows:

$$\xi(e, j_c) = \begin{cases} 1, & \text{if batch } j_c \text{ is assigned at device } e. \\ 0, & \text{otherwise.} \end{cases} \tag{5.7}$$

Here, $j_c \in J$, $c = (1, 2, .., l)$, $e \in ED$, and J is the set of all data-points batches.

5.5.3.1 Optimization problem

The optimization problem that needs to be solved is represented as:

$$Minimize \quad U(e, D), \quad \forall e \in ED \tag{5.8a}$$

$$\text{subject to} \sum_{c=1}^{l} \xi(e, j_c) \leq 1, \quad \forall e \in ED \tag{C1}$$

$$\sum_{e=1}^{n} \sum_{c=1}^{l} \xi(e, j_c).j_c = D, \quad \forall j_c \in J, \forall e \in ED \tag{C2}$$

$$U(e, D) \leq \beta, \quad \forall e \in ED \tag{C3}$$

The constraint C1 ensures that at most one batch of data-points $j_c \in J$ is allocated to each edge device. The C2 constraint ensures that the summation of data-points allocated to all edge devices is equal to the total number of data-points present in the dataset. D denotes the number of data-points present in the dataset and $n = |ED|$ (number of edge devices), where ED is the set

of all edge devices. The continuous positive variable $U(e, j_c)$ depicts the time taken to finish the assigned batches at edge device $'e'$.

$$U(e, j_c) = (C(e, j_c) + tt(j_c, e_m, e) + qd(j_c)).\xi(e, j_c), \quad \forall e, e_m \in ED, j_c \in J \tag{5.9}$$

Here, $C(e, j_c)$ is the total computation offloading cost of batch $'j_c'$ at edge device $'e'$, the transmission time of data batch j_c from master edge device e_m to e is represented by $tt(j_c, e_m, e)$. The queuing delay of batch j_c is represented by $qd(j_c)$. The continuous variable $U(e, D)$ in constraint C3 depicts the total time taken to process data-points $'D'$ at various edge devices $e \in ED$. Finally, we introduce the continuous variable $'\beta'$ that depicts the maximum training time among all the edge devices. The following constraint holds:

$$U(ED, D) \leq \beta \tag{5.10}$$

We have adopted the Mixed Integer Programming (MIP) method to solve equation (5.8a), which is a well known NP-hard problem. We have used the IBM ILOG CPLEX v12.10.0 optimizer to find optimal offloading solutions by solving equation (5.8a). Since the problem is NP-hard, it takes a long time to generate the offloading solution. In order to reduce the offloading solution generation time, we have reduced the complexity of algorithm using a branch-and-bound method (i.e. optimality gap), discussed in section 5.7.3.

5.6 Proposed Algorithms

5.6.1 Proposed $D^2 - TONE$ algorithm

We now discuss the steps of the proposed $D^2 - TONE$ algorithm (Algorithm 8). First, the training time on the local edge device (U_{local}) is estimated using ML models. If the training time of assigned data-points in dataset D on the local device is within the assigned deadline, then all the data-points/tasks are scheduled on the local edge device. Otherwise, the training time for each edge device in the network is estimated using ML models. If the total training of all edge devices present in the network having an equal number of data-points (represented by U_{EN}) is less than the specified deadline, the data-points are scheduled on all such edge devices. If the local edge device and edge devices with equal data-points do not meet the deadline condition, then the total training time of the cloud (U_{cloud}) is checked to see if it is within the deadline. If the above conditions do not satisfy, then the optimal offloading solution is provided by solving the *optimization equation* discussed in section 5.5.3.1. In order to reduce the offloading solution generation time, we have used the optimization equation in the worst-case scenario. By introducing this heuristic, the complexity of the algorithm reduces significantly, without violating deadline constraints. Additionally, in order to deal with the complexity of optimization problem, we opted for a branch and bound solution of the optimization problem with a 1-2% optimality gap. The results in Section 5.8.7 show that the offloading solution generation time reduces significantly after introducing the branch and bound solution, without significantly affecting the performance.

Algorithm 8: $D^2 - TONE$ Algorithm

Require: $D :=$ Data points in the dataset.
$\beta :=$ Deadline.
$U_{local} :=$ Total training time to process data-points in D on the local edge device.
$U_{EN} :=$ Total training time on edge network with each edge device having an equal number of data-points.
$U_{Cloud} :=$ Total training time to process data-points in D on remote Cloud server.
Ensure: $S_o :=$ Offloading Schedule.
if $U_{local} \leq \beta$ **then**
Schedule dataset 'D' on *local* edge device.
else if $U_{EN} \leq \beta$ **then**
Schedule data-points $'D'$ on *edge network* having local dataset $D_1 = D_2 = ... = D_n$ & $\sum_{e=1}^{n} D_e = D$
else if $U_{Cloud} \leq \beta$ **then**
Schedule on cloud architecture
else
Schedule data-points $'D'$ on *edge network* based on offloading solution provided by *optimization equation* (5.8a)
end if
return S_o

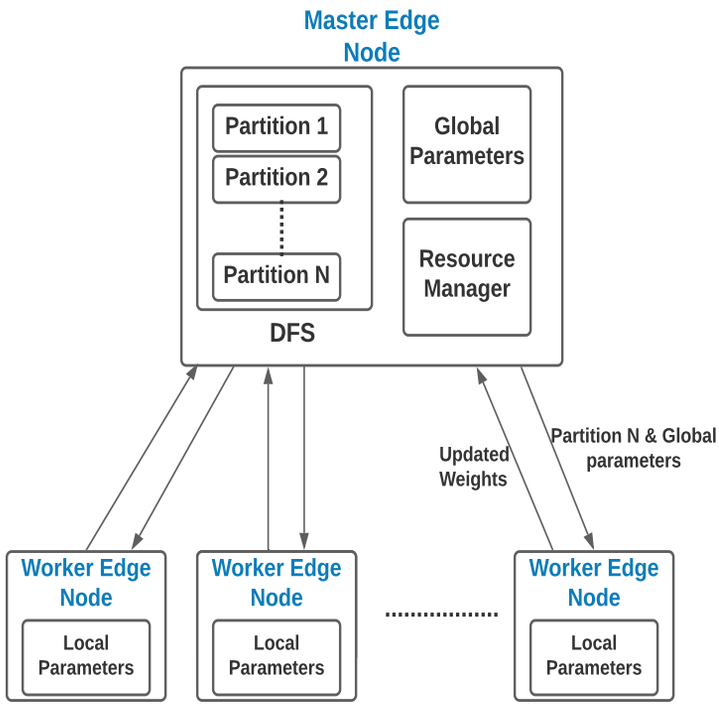


Figure 5.4: Distributed Deep Neural Network Learning framework.

5.6.2 Distributed DNN learning algorithm

The $D^2 - TONE$ algorithm decides the local dataset for each edge device, based upon which the distributed DNN learning takes place. The Distributed Deep neural network learning framework is shown in Figure 5.4. The machine learning model is learned using training data. The sample data $'k'$ used for training comprises of two parts. The first part is the features/Input vector $'x'_k$ of the ML model (i.e., image pixels). The second part is the label/Output scalar $'y'_k$ (i.e., value in case of regression problem). The learning of the ML model is facilitated using a loss function, which is defined on the weight vector $'w'$ corresponding to its sample data $'k'$. The model training process aims to minimize this prediction error (i.e., minimizing the loss function) on a given set of sample training data $'k'$. The loss function is denoted by $'l(w, x_k, y_k)'$. In short, we represent $l(w, x_k, y_k)$ as $l_k(w)$. The loss function which we consider for the regression problem is the Mean Square Error (MSE). So, when the $l_k(w)$ is the MSE, then it is represented as:

$$l_k(w) = \frac{1}{2}||y_k - W^T x_k||^2 \tag{5.11}$$

Let the number of edge nodes selected by $D^2 - TONE$ algorithm be $'M'$. Their local datasets are given as: $D_1, D_2, ..., D_e, ..., D_M$. The local dataset for each edge device e is based upon the $D^2 - TONE$ algorithm. The loss function at node $'e'$ with sub-dataset $'D'_e$ is represented as follows:

$$L_e(w) = \frac{1}{|D_e|} \sum_{k \in D_e} l_k(w) \tag{5.12}$$

Here, $|D_e|$ represents the size of the dataset at node $'e'$. The global local function is computed using all the distributed datasets such that $D_e \cap D_{e'} = \emptyset$ for $e \neq e'$ and $D = \sum_{i=1}^{M} D_e$. The global loss function is defined as:

$$L(w) = \frac{\sum_{k \in \cup_e D_e} f_k(w)}{|\bigcup_e D_e|} = \frac{\sum_{e=1}^{M} D_e L_e(w)}{D} \tag{5.13}$$

The value of $L(w)$ (i.e., the global loss function) is computed by distributing the complete information (dataset) among $'M'$ nodes. The main goal of learning the DNN model is to minimize $L(W)$ (i.e., global loss function), which can be represented as:

$$w^* = argmin L(w) \tag{5.14}$$

The solution to Equation (5.14) is computed using the Gradient-descent method for a distributed environment, as computing close-form solution for Equation (5.14) is inherently complex in distributed settings. So, we use the distributed DNN learning approach that exploits the gradient learning technique in order to solve Equation(5.14) in a distributed manner.

The local parameter for each node e is depicted by $W_e^l(T)$, where $T = 0, 1, 2, ...$ represents the iteration index. When $T = 0$, the same local parameter values are assigned to all the nodes. But when $T > 0$, the local loss function ($L_e(w)$) is used to update the local parameters ($W_e^l(T)$) of edge node $'e'$, depending upon the parameter values at $T - 1$ on each node $'e'$. This updation of local parameters using $L_e(w)$ and local data-points/dataset D_e is termed as *local update*. The master node will perform global aggregation after several local updates. In order to synchronize the local parameter of each node e at time interval $'T'$, the global parameter at time $'T'$ is assigned to the local parameter of each worker node. Usually, the local parameter at each worker node changes after a global update/aggregation (i.e. $W_e^l(T) \leftarrow W^g(T)$), which takes place after time interval

Algorithm 9: Master_Node_Update(k, S_ϕ, β)

Require: $k :=$ Training Data points.
$\beta :=$ Maximum training time.
$J :=$ Data points batches from x_k & y_k.
$a :=$ Time after which global parameters sync.
$S_\phi :=$ The dictionary of selected edge devices $e \in ED$ with their corresponding data points batches $D_e = j_c, \forall\ j_c \in J$ using $D^2 - TONE$ Scheduling algorithm.
Ensure: $W^g :=$ Trained global parameters of the DNN model. {Initialize global parameters}

```
w(t) ←Initialization of weights
W_epoch ← w(t)
S_φ ← ⟨e, j⟩∀ed ∈ ED, j ∈ J
Master node sends the data-points batches/partitions 'D'_ed to edge nodes e_i ∈ S_φ according to D^2 − TONE scheduling algorithm.
while (Train_time ≤ β) do
    if T is not multiple of a then
        {Local parameter updates}
        for all Edge devices e ∈ S_φ in parallel do
            W_e^l(T) ← Worker_Node_Update(e, W_e^l(T), D_e)
            T ← T + 1
        end for
    else
        {Global parameter update}
        W_e^g ← Σ_{∀e∈S_φ} (1/S_φ). W^e_{epoch+1}
        T ← T + 1
    end if
end while
W^g ← W_e^g
return W^g
```

$'a'$ (as shown in Algorithm 9). The Local updation (Algorithm 10) of parameters at all the edge nodes e is done using the following rule:

$$W_e^l(T) = W_e^l(T-1) - \alpha.\Delta L_e(W(T-1)) \tag{5.15}$$

Here $\alpha > 0$, depicts the step size. The global update is performed after every $'a'$ time interval, using the following equation:

$$W^g(T) = \frac{\sum_{e=1}^{M} D_e W_e(T)}{D} \tag{5.16}$$

The global parameter $W^g(T)$ is computed by taking the weighted average of local parameters of all the nodes. These local and global updates take place until the time limit $'\beta'$ (deadline) is reached.

5.7 Experimental setup

5.7.1 Data Collection & Profiling

In order to determine the computation $C(e, D_e)$ and transmission $tt(D_e)$ offloading cost for the crowd count application [145], the sub-sampling of images with varying image resolution

Algorithm 10: Worker_Node_Update(e, W, D_e)

Require: e := Worker edge device in $D^2 - TONE$ Schedule.
D_e := The data points batch associated with given worker edge device e.
W := Updated global weights.
α := Learning rate.
Ensure: W^l := Trained local parameters of the DNN model.
{Initialize local parameters}
1: $W_l \leftarrow W$
2: B := split D_e into sub-batches
3: **for all** Edge devices $e \in S_\phi$ in parallel **do**
4: **for all** batch $b \in B$ in parallel **do**
5: $W_e^l(T) \leftarrow W_e^l(T-1) - \alpha.\Delta L(W_e^l(T-1))$
6: **end for**
7: **end for**
8: **return** $W_e^l \ \forall e \in ED$ to Master edge node.

(480x640, 1024x684), with 200 images per resolution group has been done. For capturing the ground-truth computation and transmission cost of the face detection application, a varying number of data-points (data sizes) were executed on edge devices. For dynamically estimating computation and transmission costs, we have used the Scikit-learn machine learning library in Python for different batches having varying numbers of data-points. The computation and transmission offloading cost was measured in seconds. The profiling of RAM, processor freq, storage, switch bandwidth, & operating frequency values has been done while measuring the computation offloading cost. We employed three types of machines with heterogeneous computational characteristics: Raspberry Pi v4 (3 nodes), Raspberry Pi v3 (2 nodes) & Intel-core i5 (2 nodes). The total number of samples recorded for computation offloading cost estimation was 1000. The features used for predicting $C(e, D_e)$ are listed in Table 5.3. The features used for predicting transmission time are listed in Table 5.4. The mean Round trip time (RTT) and Bit rate are important in determining the speed and capacity of the connection. On the other hand, jitter and packet loss determine the connection interruptions. The LAN operating frequency and cable length determine the strength of the link/connection. The total number of samples recorded for transmission cost estimation was 1000.

5.7.2 Baseline approaches & Dataset used

We compare our proposed approach with the following baseline approaches:

- Static Estimate with MIP (SE-MIP): In [64], the static computation and transmission techniques have been used to model the inputs in MIP formulation using equation (5.8a). In order to statically estimate the computation cost $C(e, D)$, the following formula has been used:

$$C(e, D) = \frac{D}{P_s} \tag{5.17}$$

Here, D is the number of frames (data-points), and P_s is the average speed of processing/computing data-points. The processing/computation speed (P_s) is measured in data-points per second. The value of (P_s) is taken to be 11 frames per second, which is attained from benchmark results [144]. In order to statically estimate transmission time the

following formula has been used:

$$tt(j_c) = \frac{Data\_size(j_c)}{Bandwidth} \tag{5.18}$$

- Homogeneous Edge Only (HEO): In [73], all the devices present in the edge network receive an equal amount of data-points (DNN tasks). The heterogeneity factor is not considered while offloading DNN tasks, i.e., all the devices are considered to have the same processing speed.
- Centralized: In [146], the data-points (DNN workload) are processed only on the local edge device.
- Cloud_only: in this approach, we have sent the entire DNN workload to the cloud for processing.
- Random: in this approach, the data-points batches are distributed uniformly either to the set of edge devices, or the cloud, at random.

In order to evaluate the performance of our proposed $D^2 - TONE$ algorithm on the regression problem, we have used the crowd counting dataset [145], which includes 2000 RGB images - frames in a given video. The images with a resolution of 480x680 resolution were collected after placing a webcam in a mall. The number of persons (objects) varies in every image/frame. We need to determine the number of persons in each frame.

5.7.3 Evaluation Metrics

In order to evaluate the performance of the proposed approach, the following parameters have been used in our experiments:

- **Actual_Train_Time_Variation (ATTV):** The solution to $D^2 - TONE$ is compared when the actual (ground-truth) values of $C(e, D)$ and $tt(D_e)$ are known with the solution to $D^2 - TONE$ when $C(e, D)$ and $tt(D_e)$ are either predicted using ML models or statically estimated with $Static\_Est.$ (SE-MIP). The actual (ground-truth) value of computation and transmission offloading costs are obtained by actually processing the data point batches on edge devices. When the actual value of computation and transmission offloading costs are used in the $D^2 - TONE$ algorithm, we refer to U as U_{Actual}. When computation and transmission offloading costs are statically estimated in $D^2 - TONE$, we refer to U as U_{SE-MIP} - the statically estimated solution. Lastly, in case of data-driven approach, we have used two best pairs of ML models (from Table 5.5) for estimating computation and transmission offloading costs, which are represented as RFR+MLP and DT+SVM. In case of RFR+MLP, Random Forest Regressor and Multi-layer perceptron models are used for estimating $C(e, D)$ and $tt(D_e)$ respectively. In case of DT+SVM, Decision tree and Support vector machine models are used for estimating $C(e, D)$ and $tt(D_e)$ respectively. When computation and transmission offloading costs are predicted using RFR+MLP & DT+SVM approaches in $D^2 - TONE$ algorithm, we refer to U as: $U_{RFR+MLP}$ & U_{DT+SVM} solutions respectively.
  The Actual Train Time Variation (ATTV) is calculated by measuring the variation percentage for $U_{RFR+MLP}$, U_{DT+SVM}, and U_{SE-MIP} solutions $w.r.t$ the actual solution (U_{Actual}). Hence, $ATTV$ is defined as:

$$ATTV = \frac{|U_{Actual} - U_{Approach}|}{U_{Actual}} * 100 \tag{5.19}$$

Here, *Approach* ∈{RFR+MLP, DT+SVM, SE-MIP.}

- **Data-points Processing Ratio (DPR):** The input, i.e., the DNN tasks consist of a set of data point batches. The DPR is defined as the percentage of data-points that finish processing before their deadline (denoted by D') divided by the total number of data-points scheduled on the edge/cloud (denoted by D), so as to train the DNN model in a distributed manner. DPR is calculated using the following equation:

$$DPR = \frac{D^{'}}{D} * 100 \tag{5.20}$$

- **Mean Square Error (MSE):** We use mean square error (MSE) as the loss function for the regression problem, where the prediction is a scalar value. This function helps us to estimate the performance of the trained DNN model using various approaches. The MSE is calculated by taking the mean of the squared difference between actual and predicted values:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^{'})^2 \tag{5.21}$$

Here, n is the number of data-points, y_i is the actual value (i.e., the actual number of persons in the frame), and $y_i^{'}$ is the predicted/estimated value (i.e., predicted number of persons in the frame using given approach). The model/approach with the least value of MSE is considered to be the best.

- **Queue Backlog:** This is the average number of DNN tasks (data-points) that are queued per edge device during the given simulation period. More is the queue backlog; more is the queuing delay.

- **Deadline Factor:** This is used to measure the real-time performance of the proposed algorithm. A smaller deadline indicates tight deadline (real-time tasks), whereas a large deadline value indicates a looser deadline (batch processing) [147],[65]. The average real-time value of face recognition varies from 400-800ms in [148], [149]. The resulting value for $'d'$ (deadline factor) is taken as 400ms. In our experiments, we measure the performance of various approaches when the deadline factor 'd' increases and decreases by 25% and 50% (represented as 1.25d, 1.50d, 0.75d, and 0.50d respectively).

- **Number of Parameter Updates:** In the Gradient descent method, several parameter update steps are performed on various edge devices, using equation (5.15). These parameter updates are performed in order to satisfy equation (5.14). In general, a larger number of parameter updates performed on data-points results in fine-tuning of the model on the given dataset, and hence a good performance of the trained model.

- **Time to Generate Offloading Solution (TGOS):** This is the time required to obtain the offloading solution using the $D^2 - TONE$ algorithm.

- **Optimality Gap:** The optimality gap helps in reducing TGOS. We have used the IBM CPLEX optimizer in order to find optimal offloading solutions by solving the equation (5.8a). The CPLEX solver sometimes obtains good integer solutions very quickly, but keeps on examining several additional solutions in order to demonstrate that the provided solution is

Table 5.5: Computation offloading cost prediction results using various ML models

Approach	**Computation time (RMSE)**	**Transmission time (RMSE)**
Decision Tree (DT)	0.243 ± 0.851	0.543 ± 0.461
Linear Regression (LR)	0.794 ± 0.134	0.944 ± 0.851
Support vector machine (SVM)	0.494 ± 0.243	0.394 ± 0.451
Multi-layer Perceptron (MLP)	0.347 ± 0.034	**0.343 ± 0.340**
Random forest (RFR)	**0.234± 0.044**	0.422 ± 0.144
K-Nearest Neighbour (KNN)	0.444 ± 0.094	0.843 ± 0.781
Static_Estimate(SE)	10.243 ± 3.851	14.134 ± 0.649

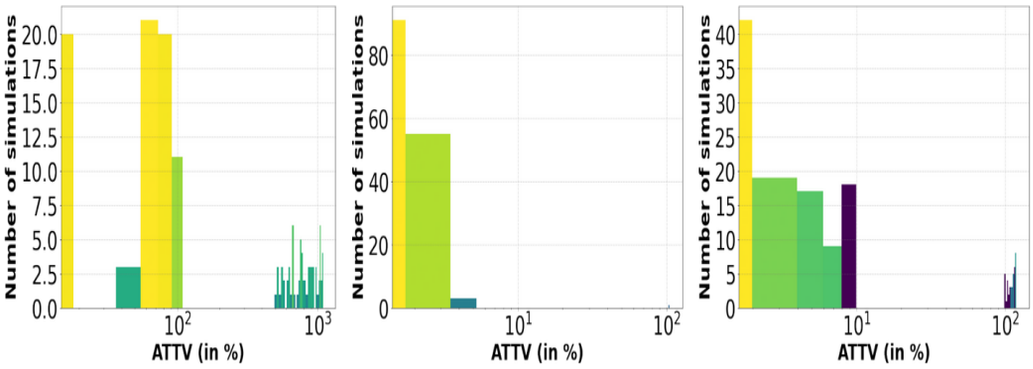


Figure 5.5: Number of simulations corresponding to Actual train time variation ($ATTV$) for SE-MIP, RFR+MLP and DT+SVM solutions.

optimal. So, in order to speed up the TGOS, we have introduced "optimality tolerance". An optimality gap of X% means that the CPLEX solver will stop immediately on obtaining a feasible integer solution that is confirmed to be within X% of the optimal solution.

5.8 Results & Discussion

5.8.1 ML models for offloading cost prediction

In this experiment, we evaluated the performance of the static estimation approach with other ML algorithms for predicting the computation $C(e, D_e)$ and transmission $tt(D_e)$ offloading costs. The aim of the experiment is to select the best approach that can predict the computation and transmission costs with high accuracy i.e. low Root Mean Square Error (RMSE). As shown in Table 5.5, the RMSE value is very high for the static estimation technique (Static_Estimate) in comparison to other ML based algorithms. Thus, the data-driven approach (ML algorithms) for predicting the computation and transmission offloading cost is more beneficial than statically estimating the computation and transmission offloading cost (Static_Est). The ML models evaluated were: Decision tree, Linear regression, Support vector machines, Random forest, Multi-layer perceptron, and K-Nearest Neighbours. The evaluation results of all ML models with RMSE values are shown in Table 5.5. The ML model with the least value of RMSE would be preferred.

The Random Forest Regressor (RFR) approach offers the best results for computation offloading

cost $C(e, D_e)$ with low root mean square error(RMSE) values and low variance. The low variance is due to the voting of several decision trees, which are trained on varying sub-sets of data present in the given dataset. Due to this bagging (voting) technique, the variance and bias of the trained model are reduced significantly. The next best performance for computation offloading cost $C(e, D_e)$ is offered by the Decision tree, as it is capable of capturing complex relationships of variables/features using fine-grained decision boundaries/branches. However, its variance is quite high compared to that of Random forest. A single instance of the decision tree has been considered, whereas, in the case of RFR, several decision trees take decisions, which reduces the variance significantly. The ML model that performed the best in the case of estimating transmission offloading cost was Multi-Layer Perceptron (MLP), followed by Support vector machines (SVM). The MLP model has performed well due to its capability to detect complex patterns and trends from complicated data. SVM has performed well due to its good generalization ability, i.e., the ability to perform well in the case of unseen data.

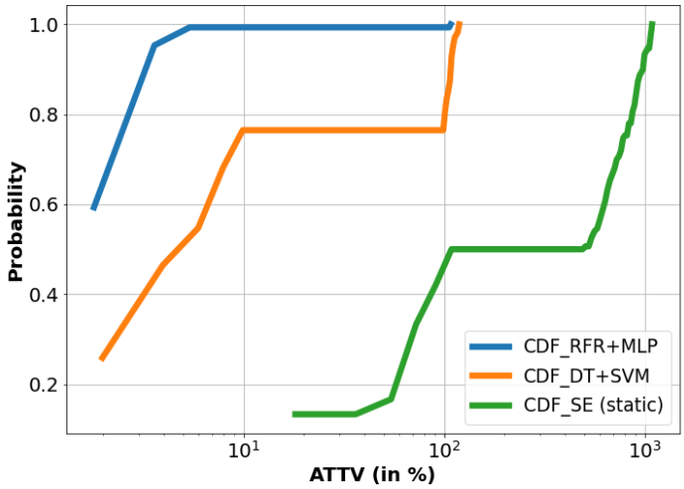


Figure 5.6: Cumulative Distribution Function (CDF) vs $ATTV$.

5.8.2 Effect of offloading costs estimation approach on ATTV

In order to study the effect of offloading cost estimation approach (Static_Estimate or ML based) on $D^2 - TONE$, we ran 150 simulations of task offloading using $D^2 - TONE$ with SE-MIP (i.e., $Static\_Estimate$), RFR+MLP, and DT+SVM models. Figure 5.5 shows the histograms for these models/solutions. The histogram depicts the number of simulations (counts) corresponding to different Actual train time variations (ATTV), which has been defined earlier in section 5.7.3. As shown in Figure 5.5(b), for RFR+MLP, there are 99 simulations (i.e. approximately 66 % of the simulations) out of 150 in which $U_{RFR+MLP}$ is the same as Actual train time U_{Actual} i.e. ATTV= 0%. However, for DT+SVM, there are 52 simulations (i.e. approximately 34 % of the simulations) out of 150 in which U_{DT+SVM} is the same as Actual train time U_{Actual} i.e. ATTV= 0%. This indicates that the selection of an appropriate ML model for predicting $C(e, D)$ and $tt(D)$ is of prime importance. The worst performance is shown by SE-MIP (shown in Figure 5.5(a)), as a very small number of simulations were completed within AATV= 10 %. This is because SE-MIP considers the average processing speed for varying hardware types to be static, i.e., 11 frames per second, and the processing speed is highly correlated with the hardware configuration and image resolutions. In addition, SE-MIP fails to capture dynamic network conditions. This set of experiments shows that the proposed data-driven approach ($D^2 - TONE$) with RFR+MLP model

offers a better performance.
Figure 5.6 shows the cumulative distribution function for ATTV. From the figure, we observe that about 99% of the simulations finished within 0-7% of actual train time variation (ATTV), when the RFR+MLP approach was used for predicting $C(e, D)$ and $tt(D)$ in $D^2 - TONE$. However, when the DT+SVM approach was employed, only 78% of the simulations completed within AATV= 10 %. Interestingly, the remaining 22% took 3-4 times (approx. ATTV = 100%) training time w.r.t actual training time U_{Actual}. Finally, the SE-MIP approach recorded the worst performance, as approximately 55% of the simulations took ten times more training time w.r.t actual training time (i.e., ATTV= 800-1000%).

5.8.3 Comparative analysis of DNN task processing time.

In this experiment, we have examined the DNN task processing time, which consists of computation time, transmission time (data transfer time + network latency), and queuing delay, for different approaches. As shown in Figure 5.7, the lowest computation time is recorded in the case of the Cloud_only approach. This is due to the high computation power of the device deployed in the cloud data center. The worst computation time is recorded in the case of the Centralized approach, in which all the data-points which need to be processed are queued on the local edge device. Due to the limited computation capacity of the edge device, the time taken to execute the given data-points batch is quite high. The proposed $D^2 - TONE$ algorithm offers reduced computation time in comparison to the SE-MIP, HEO, and centralized approach. This is because the data-points are offloaded to the edge/cloud based on their computational capacity, which results in a good performance for $D^2 - TONE$.

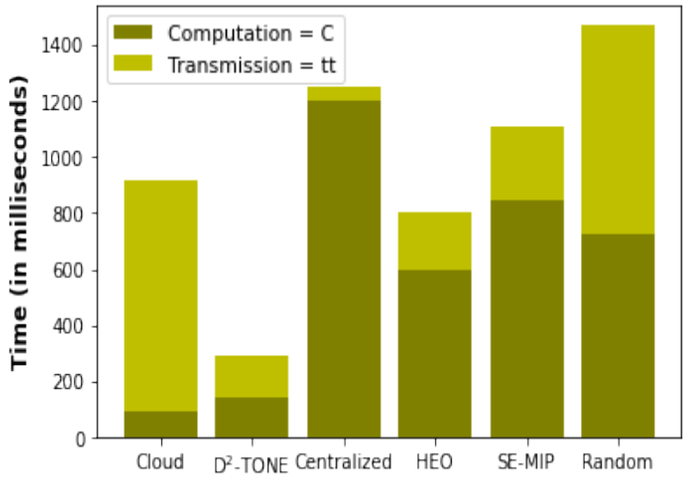


Figure 5.7: DNN task process time for various approaches.

The transmission delay comprises of the data transfer time and the network delay (propagation delay). The worst performance in terms of transmission delay is observed in the case of the Cloud_Only approach. This is due to the high latency caused because of network traffic in the wide area network. Also, when more devices are connected through the internet, it results in bandwidth reduction. All these reasons contribute to the poor performance of the Cloud_only approach. In comparison, negligible transmission delay is experienced in the case of the Centralized approach. Here, the data is not transmitted to the network and is processed on the local edge device. However, due to a high computation time, the overall performance of the centralized approach is not acceptable for a real-time scenario. The proposed $D^2 - TONE$ approach offers

significantly reduced transmission times along with computational times, as compared to SE-MIP, HEO, centralized and Cloud_only approaches. The reason for D^2-TONE's performance in terms of transmission delay is due to precisely estimating the network delays based on the dynamic network conditions and offloading the DNN tasks accordingly. Due to this, the assigned DNN tasks get processed in less amount of time, which results in faster training of the given DNN model. In the case of SE-MIP, the transmission delays of DNN tasks are estimated using static techniques, which do not capture the dynamic condition in the network. This results in poor offloading decisions. Similarly, in the case of HEO, an equal amount of DNN tasks are offloaded to all the devices available in the network, which results in network congestion. Hence, training time increases significantly due to poor offloading decisions. The best overall performance is offered by D^2-TONE because it intelligently offloads the DNN tasks in the network.

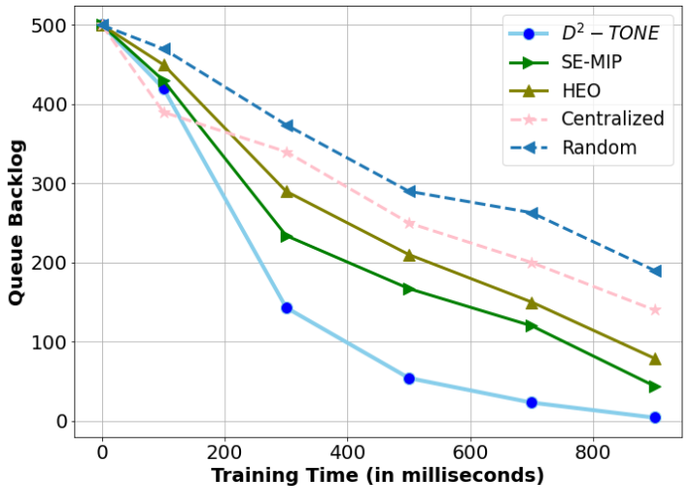


Figure 5.8: Queuing Backlog vs. Training time.

Figure 5.8 depicts the device queue backlog for the compared approached. Initially, the queuing backlog is observed to be less in the case of the Centralized approach. Unlike other approaches, no transmission delays are involved in the Centralized approach, so processing of the data-points takes place fairly quickly. However, after a certain time duration, the performance of the Centralized and Random approaches in terms of queue backlog deteriorates. In the case of the Centralized approach, this is due to an increase in the unprocessed data-points in the edge device queue, which is in turn due to the limited computing capacity of a single edge device. In the case of the Random approach, the high queue backlog is due to uniformly offloading data-points to randomly selected edge devices without considering their computation capacity. While employing D^2-TONE, the number of unprocessed data-points in the edge device queue is quite less in comparison to other approaches. This is because D^2-TONE offloads the data-points according to the computing capacity of the devices in the network.

5.8.4 Effect of data size on DPR

In this experiment, we observe the effect of data size on the data-points processing ratio (DPR) that has been defined earlier in section 5.7.3. The results are shown in Figure 5.9. As shown in the Figure 5.9, we increased the data-point load on edge devices by varying the data size to be processed on the edge from 100KB to 500KB, and observed its corresponding effects on DPR. As the data size increases, more data-points are left unprocessed, which results in low DPR values. This trend is applicable for all approaches: SE-MIP, HEO, Cloud, D^2-TONE & Centralized.

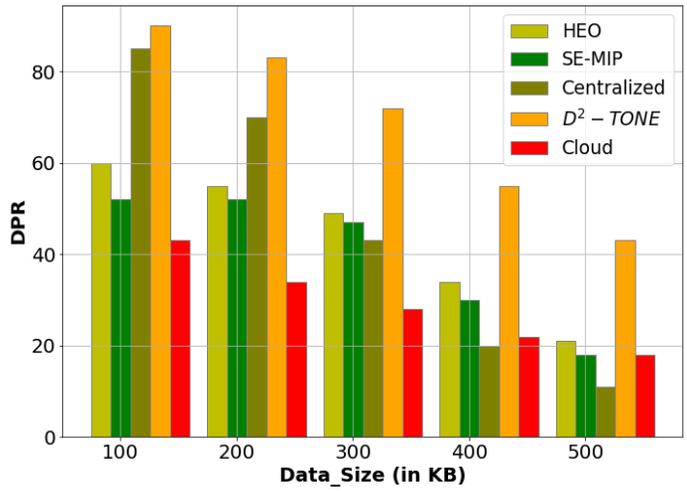


Figure 5.9: Effect of Data size on DPR.

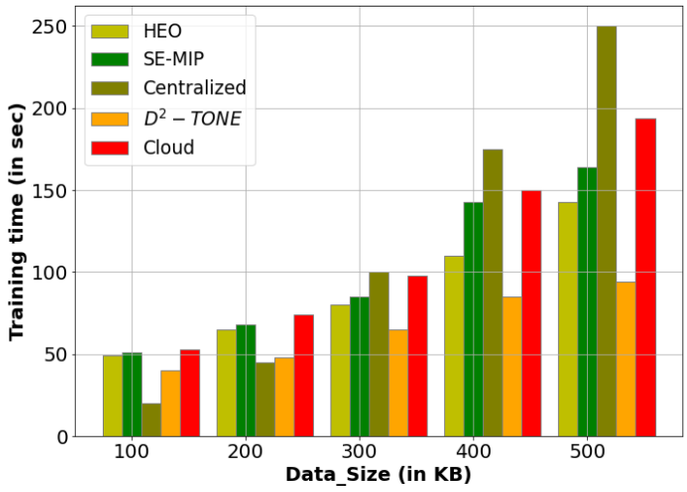


Figure 5.10: Effect of data size on train time.

However, with the proposed data-driven $D^2 - TONE$ approach, more data-points are processed (for gradients updates) within a given deadline in comparison to other approaches. This is due to assigning the tasks according to device processing speeds and dynamic network conditions. On the other hand, in the HEO approach, tasks/data-points are distributed equally in the edge network without consideration of hardware configuration (processing speeds). Therefore, the number of gradient updates (tasks) performed within a given deadline on the edge network is less, which results in low DPR. When the data size is 100KB (few data-points), then the DPR is observed to be high for the Centralized approach as compared to the HEO approach. This is because a small number of data-points can easily be processed on a single-edge device. However, when the data size increases in the Centralized approach, the number of data-points processed on a single-edge device becomes low due to limited processing capacity. This results in a low DPR value for this approach.

5.8.5 Effect of data size on training time of DNN model

In this experiment, we examined the effect of data size on the training time of DNN model using various approaches. As shown in Figure 5.10, we have increased the DNN task (i.e.,

data-points) load on the network by varying the data size from 100KB to 500KB and observed its corresponding effects on training time. When the data size is 100KB and 200KB, then the best performance is offered by the Centralized approach. This is because the computational capacity of the local edge device is sufficient to handle such small data sizes. However, when the data size increases significantly, then the centralized approach shows the worst performance due to its limited computational capacity. All the distributed training approaches (i.e., SE-MIP, HEO, and $D^2 - TONE$) perform well in comparison to the centralized approach when the DNN task load is high. This is due to reduced queuing delay and distributed execution of DNN tasks. $D^2 - TONE$ performs particularly well on both small and large data sizes. However, a noticeable performance difference can be observed when we considered data sizes greater than 300KB. The reason for a reduced training time in the case of $D^2 - TONE$ is due to carrying out the training process in a distributed manner and utilizing the limited edge resources to their full capacity, by accurately estimating computational and transmission delays using the best ML models (RFR+MLP). In SE-MIP and HEO, the training is performed in a distributed manner, but the training time is quite high in both scenarios. The reason for high training time in the case of SE-MIP is due to inaccurate estimations of various delays made using static techniques. On the other hand, HEO shows poor performance because, in this approach, heterogeneity factor of edge resources are not considered, due to which queuing backlogs increases on slower devices and faster devices remain idle after processing the assigned DNN tasks. Therefore, the available resources are not fully utilized in the case of SE-MIP and HEO. Finally, the Cloud_Only approach took significant time to train the DNN model using the same amount of data-points. This is because of the huge transmission delays introduced by the network traffic.

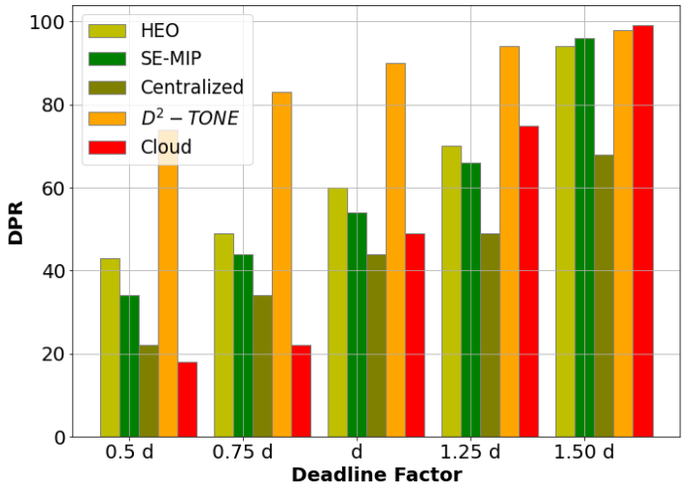


Figure 5.11: Effect of deadline factor on DPR.

5.8.6 Effect of deadline value on DPR

In this experiment, we varied (increased/decreased) the Deadline Factor (DF) and observed its corresponding effect on the Data-points Processing Ratio (DPR). The DF value of the assigned data-points was increased/decreased by 25% & 50%. As shown in Figure 5.11, as the DF increases, more data-points are processed. Therefore, the DPR value corresponding to DF='1.25d' & DF='1.50d' (i.e. when deadline factor is increased 25% & 50% respectively) increases significantly. When DF='1.50d' (DF is increased by 50%), then the DPR values for all the approaches are observed to quite high, except for the *centralized* approach. This leads to the processing of almost

all the data-points (i.e., DPR=100%). The poor performance of the *Centralized* approach is due to the limited computing capacity of the local edge device. However, when DF decreases by 25% & 50%, the proposed $D^2 - TONE$ offers a higher DPR as compared to all other approaches. The superior performance of $D^2 - TONE$ is due to the incorporation of dynamic network conditions and computational capacities of the edge devices present in the edge network before assigning data-points. Thus, the task scheduling using $D^2 - TONE$ helps in handling interactive tasks within the specified deadlines, hence, improving the DPR. When DF decreases by 25% & 50%, the worst DPR value is recorded when all the DNN tasks (data-points) are sent to the cloud for processing. This is due to the high network latency , which results in a low DPR value.

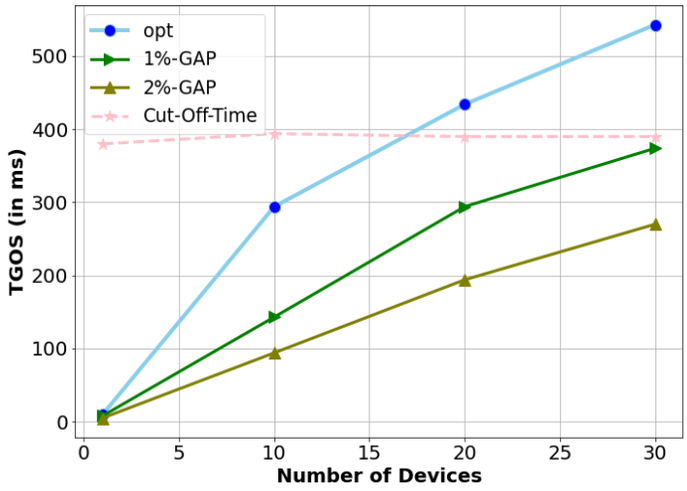


Figure 5.12: Effect of the number of devices on TGOS.

5.8.7 Effect of number of devices on TGOS

In this experiment, we explore the effect of an increase in the number of edge devices on time to generate offloading solutions (TGOS), defined earlier in section 5.7.3. The results are shown in Figure 5.12. Solving the *optimization problem* using equation (5.8*a*) for actual training time (U_{actual}) is NP-Hard and complex, even for a moderately sized edge network of 25-30 nodes with heterogeneous computing capacities. In order to deal with this complexity, we opted for a branch-and-bound solution of the *optimization problem* with a 1-2 % optimality gap. Here, we use the IBM Cplex optimizer. As shown in Figure 5.12, due to this branch and bound approach, the *optimization problem* solution scales well. The 1-2% optimality gap solution to the *optimization problem* has been generated earlier than the cut-off time. The optimal / near-optimal solution is provided in a lesser amount of time because the CPLEX solver will stop immediately on obtaining a feasible integer solution that is within 1-2% of optimal. Since solving the optimization problem takes a significantly lesser amount of time, more time will be available for training the model, before the deadline expires. Therefore, the number of parameter updates will increase significantly with an increased training time, which results in superior learning in a distributed environment.

Figure 5.13 depicts the increase in parameter updates when the 1-2 % optimality gap is introduced. As shown in Figure 5.13, the parameter updates increased significantly when the 1-2% optimality gap has been introduced in the $D^2 - TONE$ algorithm in comparison to when the *optimization problem* using equation (5.8*a*) is solved for 0% optimality gap.

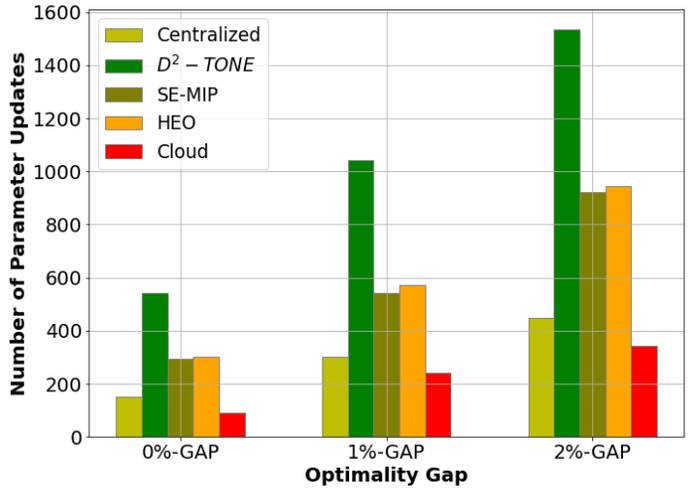


Figure 5.13: Number of parameter updates vs. Optimality gap.

5.8.8 Comparison of baseline approaches for crowd counting application

In this experiment, we used the crowd counting dataset [145] to compare the performance of our proposed $D^2 - TONE$ approach with the SE-MIP and HEO (equal data-points distribution in the network) approaches. The results are shown in Figure 5.15. As shown in Figure 5.15(a), the predicted person count varies significantly with the actual person count in the case of the SE-MIP approach. This happens because very few parameter updates took place on the edge network, due to the use of static estimation techniques for computation and transmission time estimation, resulting in poor model training. The HEO approach with an equal distribution of data-points on various edge devices performed better than the SE-MIP approach. Here, more parameter updates took place versus the SE-MIP approach. However, the HEO approach could not perform well when the person count exceeded 20. As shown in Figure 5.15(c), the proposed $D^2 - TONE$ approach records the best performance among all the approaches, due to a large number of parameter updates taking place, resulting in superior model learning. This increase in the parameter updates occurs due to offloading the tasks (data-points) according to the hardware configuration of the edge devices present in the heterogeneous edge network.

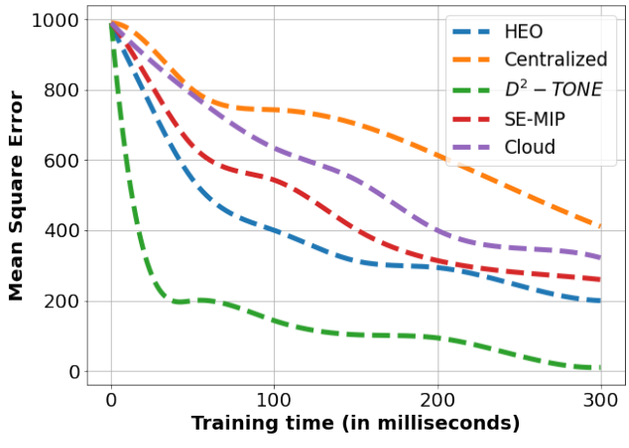
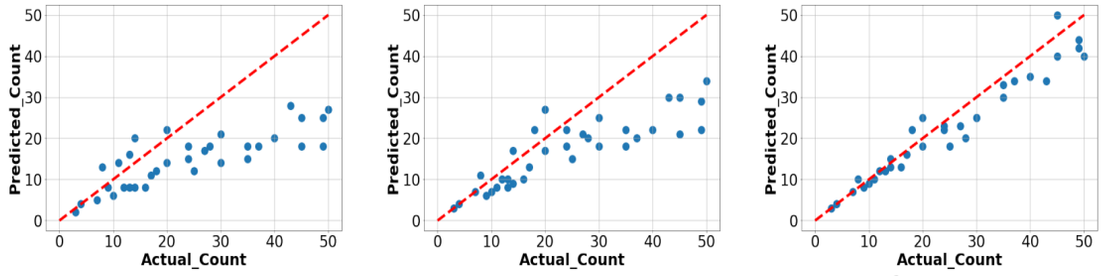


Figure 5.14: Plot of MSE versus training time.

However, the proposed framework $D^2 - TONE$ does not perform well when the person count



a) Plot for $SE-MIP$ approach. b) Plot for HEO approach. c) Plot for D^2-TONE approach.

Figure 5.15: Performance of $SE-MIP$, HEO and D^2-TONE approaches in crowd counting application.

exceeds 35 per frame. This is due to partly occluded objects. As shown in Figure 5.14, The Mean Square Error (MSE) of the *Centralized* approach doesn't decrease much with training time, as very limited parameter updates take place on an edge device. The poor performance (i.e. high Mean Square Error) of cloud is because of high transmission delay, due to which very less time is available for training the DNN model. The performance of the HEO approach is better than the SE-MIP approach, but the best performance is offered by the D^2-TONE approach. This is because D^2-TONE takes into consideration the number of parameters, layers, epochs, and processing capability of the hardware for estimating the offloading cost and schedules the data-points accordingly on the network.

5.9 Conclusion

Accurate offloading cost estimation in the face of varying network conditions is a challenging problem. The proposed D^2-TONE approach utilizes data-driven task offloading in order to predict the offloading cost on heterogeneous multi-edge networks. We investigated several classical ML models for predicting offloading costs and discussed the importance of model selection on D^2-TONE's performance. Extensive experiments revealed that the proposed D^2-TONE approach provides near-optimal offloading solutions. Moreover, the proposed solution is scalable for moderately sized networks, with a 1-2% optimality gap. D^2-TONE has been observed to perform well in terms of maximizing DPR and minimizing MSE in comparison to other approaches. This is due to a larger number of parameter updates taking place while training the DNN model in a distributed manner, within a given deadline. We also studied the effect of various data sizes on D^2-TONE's performance. The experiments revealed that the proposed D^2-TONE approach offers significantly smaller training times in comparison to other approaches. In the future, we would like to extend our work to a wireless network setting.

Chapter 6

A non-linear time-series based AI model to predict outcomes in cardiac surgery

6.1 Introduction

Adverse lifestyles have led to increased cardiac complications, further accelerating the burden of cardiac surgeries in tertiary care hospitals. For optimum management of cardiac surgical patients in the hospital, it is essential to have an accurate idea regarding the patients' expected ICU stay and hospital stay. Additionally, the forecasting of the survival outcome of patients is also essential for ICU management. This study aims to develop artificial intelligence models based on non-linear time-series data of blood pressure and heart rate to predict the ICU stay, hospital stay, and survival outcome of cardiac surgical patients. The intraoperative heart rate and blood pressure data of 1077 patients undergoing cardiac surgeries at a single tertiary care hospital were recorded every minute. The raw data was processed to remove artifacts. Next, feature engineering and oversampling were performed. Then, various classification and regression models were trained and tested. The prediction results were evaluated on the following performance metrics: area under the curve (AUC), accuracy, F1-score, RMSE, and R2-score. The Gaussian Naive Bayes + Logistic Regression (GNB+LR) model is the best model for survival analysis, having the highest AUC of 0.72, Accuracy of 83%, and an F1-score of 0.86. The Gradient boosting (GB) model is the best model for the analysis of hospital stay, offering the highest R2-score (0.023). The XGBoost regressor is the best model for ICU stay analysis, offering the highest R2-score (0.125). Artificial intelligence models based upon the intraoperative time series data were developed to analyze outcomes in cardiac surgery with high accuracy. These models can be used in cardiac surgeries to predict the ICU stay, hospital stay, and overall survival of the patients for better ICU management at the hospital.

Cardiovascular disease has become the prominent cause of morbidity and mortality in India during the past decades. Genetic factors and acquired modern lifestyle risk factors seem to be the primary cause of high incidence. Cardiovascular disease is managed by contemporary methods, like percutaneous coronary revascularization and surgical methods. Coronary artery bypass graft surgery (CABG) was first performed in India in 1975, about 13 years after its advent in 1962. In the mid-1990s, some 10,000 CABG surgeries were performed annually in India. At present, the annual number is about 60,000 [150]. The usual challenges faced by clinicians are predicting the duration of hospital stay or ICU stay and the overall survival outcomes post-surgery. The ability to obtain accurate predictions of survival outcomes can improve the efficacy of healthcare institutions in allocating, coordinating, and expending limited healthcare resources for treating new patients[151]. Since every patient has a different clinical history, demographic profiles, predisposing risk factors,

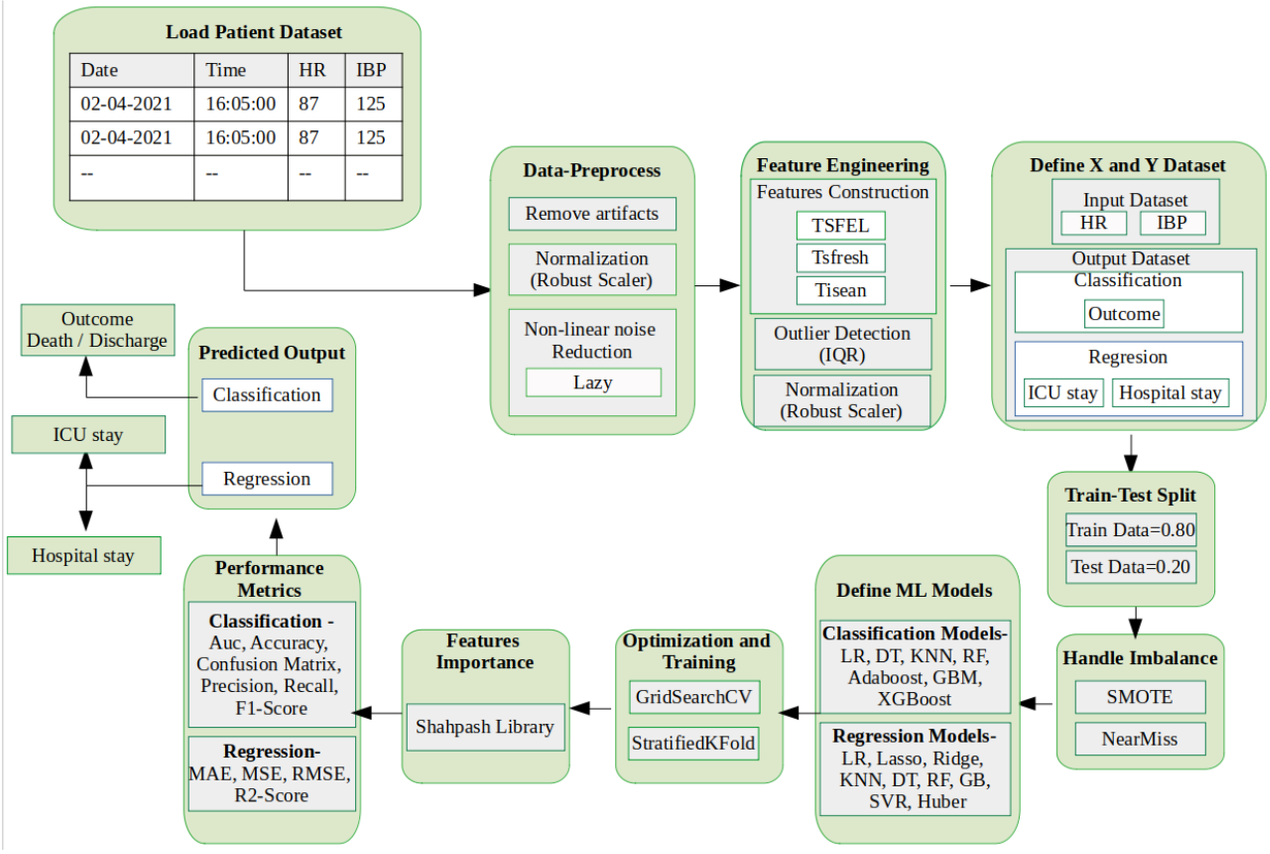


Figure 6.1: Methodology of the proposed framework

and traditional methods cannot offer accurate and reliable predictions for survival outcomes. [152].

Artificial intelligence (AI) and machine learning (ML) are evolving techniques in healthcare. Several models are available that take the static parameters of cardiac patients as input and, using inbuilt algorithms, give an accurate idea about the survival outcomes. Among all these models, the most accepted are EuroSCORE II and STS Score. These models predict the survival outcomes based on static parameters like age, sex, ventricle dysfunction, creatinine clearance, pulmonary hypertension and surgical intervention [153]. Since existing models predict the survival outcomes based upon static parameters at the preoperative stage, the accuracy of such models is challenging in the case where complications arise during intraoperative or postoperative stages.

Apart from static parameters, a considerable amount of physiological time series data is generated in cardiac surgeries, which could be used to predict the survival outcomes. [154]. Since invasive blood pressure (IBP) and heart rate (HR) data are available in the intraoperative stage, this time series data could be used to predict the survival outcomes. The present study aims to develop a model which uses intraoperative non-linear time-series data of IBP and HR to predict the ICU stay, hospital stay, and survival outcome in cardiac surgeries. Specifically, we aim to construct an end-to-end data analysis pipeline which incorporates artefacts removal, non-linear noise reduction and feature engineering. We also adopt the synthetic minority oversampling technique (SMOTE) and NearMiss technique to alleviate the inadequate classification generated by imbalanced data. Last but not least, we aim to evaluate the performance of different ML models and ensemble models by optimizing their hyperparameters in the prediction of survival outcome, ICU stay and hospital stay.

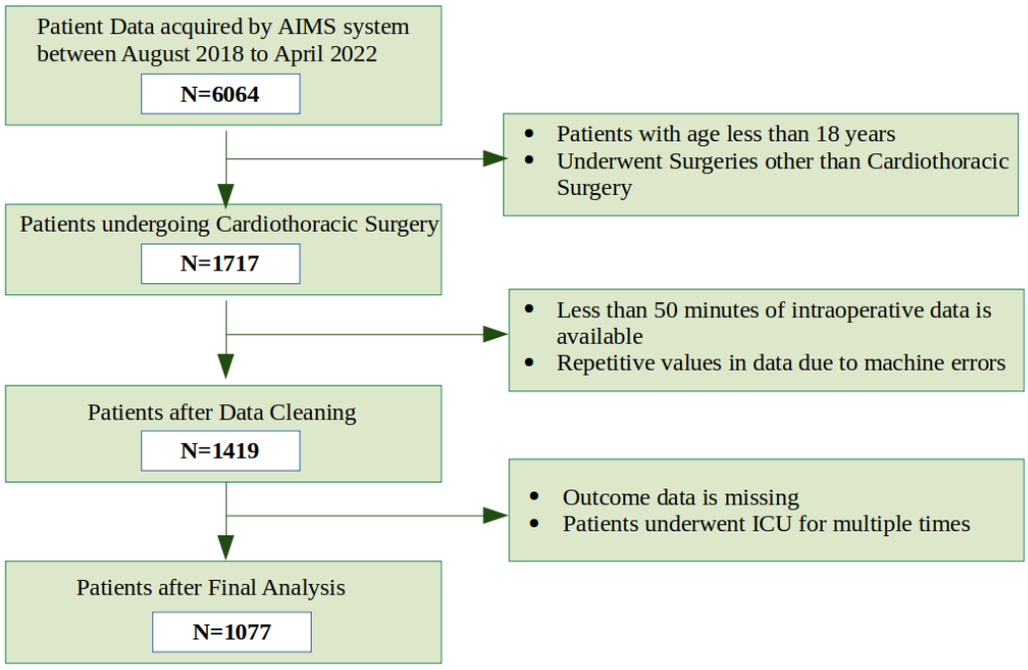


Figure 6.2: Flow chart depicting that inoperative data of total 6064 patients was captured by the AIMS system. Data of 4987 patients was excluded from the final analysis, based upon inclusion/exclusion criteria. Data from 1077 patients was used for the final analysis.

6.2 Methodology

The overall methodology followed in developing the model is shown in Figure 6.1. First, a data preprocessing mechanism that incorporates artefacts removal, normalization, and noise filtering was implemented for HR and IBP non-linear time series. Afterwards, a feature engineering procedure that includes features construction from various in-built libraries, anomaly detection, and normalization was performed for classification and regression analysis of the time series. Hereafter, SMOTE and NearMiss techniques were applied to alleviate the poor classification produced by the imbalanced data. Finally, for classification analysis, the balanced data samples were combined with hyperparameter optimization in classification ML models to generate the final survival outcome of Death and Discharge. However, for regression analysis, samples with Discharge as a survival outcome were combined with hyperparameter optimization in regression ML models to predict the duration of hospital stay and ICU stay.

6.2.1 Data Collection

This was an observational study collecting data from a single tertiary care hospital over three years, from April 2019 to March 2022. Anaesthesia Information management system (AIMS) installed at the cardiac surgical operation theatre in the host institute captured the time series data from patients intraoperatively. The patient-specific parameters utilized for the study include HR and IBP, captured at time intervals of 1 minute. The data was stored centrally in the servers.

6.2.2 Data Pre-Processing

The pre-processing of data consists of the following steps: removal of artefacts, data normalization, and non-linear noise reduction (see Figure 6.2). The data was cleaned based on the IBP feature.

All rows of data consisting of IBP values lower than 20 were removed. Further, context-based labelling was performed to label the data that was not missing at random correctly. Data missing before attaching monitors to the patient was labelled as "-9999", data missing between arterial cross-clamping and declamping was labelled as "-8888", and all other data missing during cardiopulmonary bypass was labelled as "-7777". If a row had both HR and IBP missing outside the context, i.e. data was missing at random, the entire row was removed. For normalization of data, RobustScaler [155] was used. RobustScaler removes the median and scales the data according to the quartile range. The nonlinear time series might suffer from noise caused by the device or the environment where the measurements are performed. Noise affects the interpretability and should be removed and not be considered for further data analysis. Thus, we have used a function called "lazy" from the Tisean package, for nonlinear noise reduction [156]. It performs simple nonlinear noise reduction by replacing the middle coordinate of each embedding vector with the local average of this coordinate.

6.2.3 Feature Engineering

We now discuss the various libraries that have been used to construct features from the time series data. The first library is TSFEL. This python library depends on Numpy and SciPy, which provide efficient numerical functions for multivariate time series data. The TSFEL features can be grouped into three different domains, which are: temporal, statistical, and spectral [157]. Some of the features considered in the TSFEL library are "unique", which returns the percentage of unique values in the time series data, "Wentropy(S)", which computes the Shannon entropy of the time series data using wavelets, and "Negative Turning", which returns the number of negative turning points of the time series data. The second library is tsfresh. Tsfresh is a python package that automatically computes hundreds of related time series features, and studies the properties of the series by incorporating 72 time-series feature functions for each fixed time window [158]. Series data, "longest strike above mean", which returns the length of the longest consecutive sub-sequence in the time series that is bigger than the mean of time series data, and "count below mean", which returns the number of values in time series that are lower than the mean. The third library is Tisean. Tisean incorporates several functions for non-linear time series analysis. Moreover, this package includes an algorithm for stationarity/non-stationarity testing, non-linear noise reduction, and non-linear time series prediction. Some of the features investigated in the Tisean package are: the Lyapunov exponent, which computes the largest Lyapunov exponent (LLE) of time series data to indicate the chaotic nature of time series, fsle (finite-size Lyapunov exponents), which measures the divergence of nearby trajectories to resolve predictability, and order entropy, which computes the unpredictability of fluctuations in a time series [156].

6.2.4 Defining Datasets

The input datasets include the HR and IBP time-series parameters. The survival outcome has been considered an output data for classification analysis. The outcomes were labelled either as discharge or death. For regression analysis, ICU stay and hospital stay have been considered as output for the patient datasets with discharge as the survival outcome. In addition, the patients whose ICU and Hospital stay duration was detected as an outlier were removed from the regression analysis using the interquartile (IQR) method. The IQR [159] method is the difference between the third quartile (Q3) and first quartile (Q1) of the time series data. The upper limit has been evaluated as Q3 + 1.5 x IQR, and the lower limit has been evaluated as Q1 - 1.5 x IQR. A patient

is considered an outlier if their data is beyond these limits.

6.2.5 Train-Test Split for Regression Model

Data leakage during model evaluation can create bias; therefore, we split the data into the train and test data sets. Out of 1077 cases, 781 were used to train the model, and the remaining 296 cases were used to test the model. There were 57 death cases and 724 discharge cases in the train data. There were 29 death cases and 267 discharge cases in the test data.

6.2.6 Handling Imbalance for Classification Model

Data imbalance was observed in our data due to the high number of discharge cases and a low number of death cases. To address this data imbalance, we used the synthetic minority oversampling technique (SMOTE), which creates synthetic data samples associated with the minority death class along with the line segments which join the nearest neighbours of the minority class. This technique minimizes the classifier over-fitting issue by enlarging the minority class decision region [160]. We also employed the Near-Miss under-sampling technique, which increases the spacing between the two classes by removing the instances of the majority class with the smallest distance from the minority class. For the final model testing, we used SMOTE and Near-Miss techniques. A death discharge ratio of 1:3 was obtained by oversampling the death cases by 20%, and undersampling the discharge cases by 30%. Finally, we obtained 144 death cases and 434 discharge cases for classification model testing.

6.2.7 Selecting ML Model

The extracted features of HR and IBP non-linear time series have been used to predict the survival outcome (Death & Discharge), Hospital stay, and ICU stay. We have explored various Linear and Non-linear ML models for predicting survival outcomes. The Non-Linear models explored are as follows: Decision Tree (DT) Classifier [161], K Nearest Neighbor (KNN) Classifier [162]. Moreover, ensemble learning-based models such as Random forest (RF) Classifier [163], AdaBoost Classifier [164], Gradient Boosting Machine (GBM) Classifier [165], and XGBoost (XGB) Classifier [166] have been used. In addition, linear ML models such as Logistic Regression (LR)[167], Linear Discriminant Analysis (LDA), Gaussian Naive Bayes (GNB) and Bernoulli Naive Bayes (BNB) were also considered for predicting survival outcomes.
The models used for predicting ICU stay and hospital stay of the patients are linear Regression (LR) [168], Lasso Regression [169], Ridge Regression [170], K Neighbors Regressor [171], DT Regressor [172], RF Regressor [173], GB Regressor [174], XGB Regressor [175], Support Vector Regressor (SVR)[176], and Huber Regressor [177].

6.2.8 Optimization and Training

Grid search (GS) is the most commonly used technique for optimizing the hyperparameters of ML models. It adjusts the parameters based on the step size within the specified range and then evaluates the model's performance for each combination of hyperparameters [178]. Moreover, the k-fold cross-validation (CV) algorithm has been adopted to enhance the robustness and avoid overfitting. In this model, a 5-fold GridsearchCV has been adopted, in which the entire dataset is split into five groups. The evaluation scores are accumulated for each group and then summarized

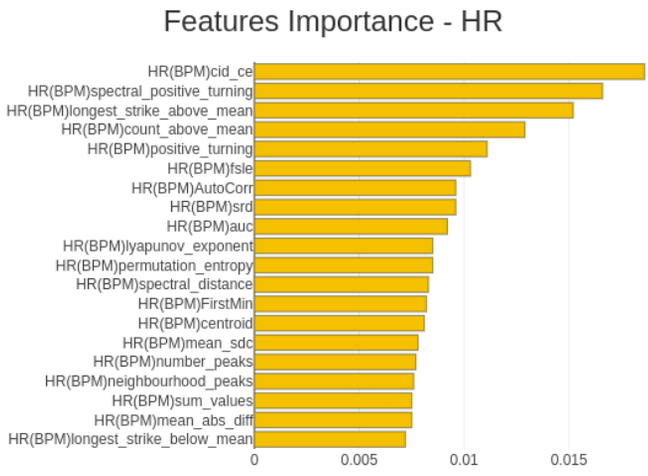


Figure 6.3: Feature Importance - HR

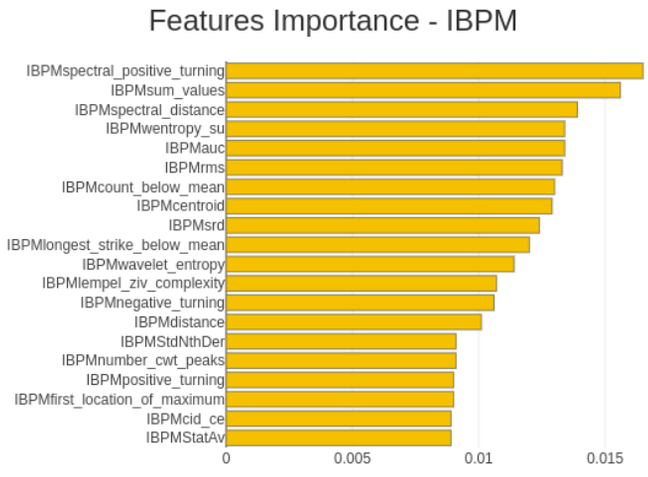


Figure 6.4: Feature Importance - IBPM

at the end to evaluate the model performance. Thereby, a subset of hyperparameters with the best accuracy and performance are selected for model training of the present dataset.

6.2.9 Feature Importance

The ML frameworks offer minimal insights on the influence of various features employed for prediction due to the black-box nature of their operations. Thus, for the correct interpretation of the ML model for prediction, the Shapash library has been adopted in our proposed model to analyze the feature importance [179]. Figures 6.3 and 6.4 depict the respective contribution of the top 20 features in predicting HR and IBP. It can be observed that positive spectral turning is the topmost significant feature for both HR and IBP.

6.2.10 Performance metrics

This section discusses the performance metrics for evaluating both classification and regression ML models. The area under the curve (AUC) is the most intuitive metric that has been utilised to evaluate the classification models. It is a measure of the ability of a classifier to differentiate among classes and is used as a summary of the receiver operating characteristic (ROC) curve.

The higher the AUC, the better the model's performance differentiating between the death and discharge classes. In addition, accuracy, confusion matrix, recall, precision, and F1-score have also been used to estimate the overall model performance [180].

The accuracy metric outlines the performance of a classification model as the number of accurate predictions divided by the entire set of predictions. The accuracy is expressed by equation (6.1) below.

$$Accuracy = \frac{T_P + T_N}{T_P + F_P + T_N + F_N} \tag{6.1}$$

Here T_P is the number of True Positives, F_P is the number of False Positives, T_N is the number of True Negatives, and F_N is the number of False Negatives.

The precision or specificity evaluation metric depicts the percentage of patients associated correctly with the discharge class, as predicted by the models. In contrast, recall or sensitivity shows the percentage of patients associated correctly with death class, as predicted by the models. Both the precision and recall are expressed in equations (6.2) and (6.3), respectively.

$$Precision = \frac{T_P}{T_P + F_P} \tag{6.2}$$

$$Recall = \frac{T_P}{T_P + F_N} \tag{6.3}$$

Furthermore, the confusion Matrix is an N x N matrix utilised to estimate the interpretation of a classification model, where N is the number of target classes. It analyses the actual values with those predicted by the ML models. The F1-Score is the harmonic mean of precision and recall, as given in equation (6.4). It considers both F_P and F_N, and provides equal weightage to precision and recall. Thereby, it executes well on an imbalanced dataset.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{6.4}$$

Next, mean absolute error (MAE), mean square error (MSE), root mean square error (RMSE), and coefficient of determination (R^2 score or R-squared) metrics [181] were used to estimate the forecasting errors and analyse the performance of regression ML models. These metrics are expressed in terms of the following equations:

$$MAE = \frac{1}{m}\sum_{i=1}^{m} \|X_i - Y_i\| \tag{6.5}$$

$$MSE = \frac{1}{m}\sum_{i=1}^{m} (X_i - Y_i)^2 \tag{6.6}$$

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m} (X_i - Y_i)^2} \tag{6.7}$$

$$R^2 score = 1 - \frac{\sum_{i=1}^{m} (X_i - Y_i)^2}{\sum_{i=1}^{m} (\overline{Y} - Y_i)^2} \tag{6.8}$$

$$\overline{Y} = \frac{1}{m}\sum_{i=1}^{m} Y_i \tag{6.9}$$

Here, X_i is the predicted i^{th} value, Y_i is the actual i^{th} value, and m is the total number of test samples.

Normalised values of MAE, MSE, and RMSE nearer to zero, along with an R^2 score nearer to

Table 6.1: Sociodemographic and clinical determinants of patients

Variable	**Sub domain**	**Mean ± SD, or N (%)**
Age		46.47 ± 15.58 years
Sex	Male	644 (59.80%)
	Female	432 (40.11%)
	Sex Unspecified	1 (0.09%)
Previous Cardiac Surgery		6 (0.55%)
Serum Creatinine		0.90 ± 0.62 mg/dL
Unstable Angina		40 (3.71%)
LV Ejection	<30%	42 (3.89%)
	30-50%	304 (28.26%)
	>50%	190 (17.64%)
Pulmonary Hypertension	Mild	31 (2.87%)
	Moderate	47 (4.36%)
	Severe	46 (4.27%)
Urgency	Elective	730 (67.78%)
	Emergency	347 (32.22%)
Surgical Procedures	CABG	248 (23.03%)
	CABG+AVR	22 (2.04%)
	CABG+DVR	6 (0.56%)
	CABG+MVR	12 (1.11%)
	ASD	52 (4.83%)
	AVR	112 (10.40%)
	DVR	81 (7.52%)
	MVR	184 (17.08%)
	TVR	3 (0.28%)
	Surgery on thoracic aorta	67 (6.22%)
	Other Cardiac Surgeries	160 (14.86%)
	Non cardiac thoracic surgeries	130 (12.07%)
Post infract septal rupture		13 (1.20%)
Outcome	Death	86 (7.99%)
	Discharge	991 (92.01%)
ICU Stay		5.48 ± 6.52 days
Hospital Stay		11.67 ± 8.36 days

unity, are the primary measures to select the best model with the lowest prediction error for ICU stay and hospital stay.

6.3 Results

6.3.1 Sociodemographic and Clinical Determinants

The mean age of the patients in the present study was 46.47 ± 15.58 years. The sex ratio was skewed towards males, with male to female sex ratio of 1.49. There were 6 (0.55%) patients that revealed a history of previous cardiac surgery. The mean creatinine level was 0.90 ± 0.62 mg/dL. Unstable angina was documented in 40 (3.71%) patients. LV ejection was < 30% in 42 (3.89%) patients, 30-50% in 304 (28.26%) patients, and > 50% in 190 (17.64%) patients. There were 46 (4.27%) patients that exhibited severe pulmonary hypertension, whereas moderate

Table 6.2: Prediction of Survival Outcome

Pipeline	Models	AUC	F1-Score	Acc	Precision		Recall	
					PC	NC	PC	NC
Remove artifacts ↓ Scaling (Robustscaler) ↓ Features Extraction ↓ Scaling (Robustscaler) ↓ Handle Imbalance (SMOTE + Near-Miss) ↓ Hyper-parameter Optimization ↓ Prediction	DT	0.46	0.73	0.78	0.93	0.08	0.67	0.35
	KNN	0.58	0.83	0.78	0.94	0.13	0.82	0.35
	RF	0.50	0.82	0.81	0.94	0.13	0.85	0.29
	ADA	0.61	0.80	0.73	0.95	0.13	0.76	0.37
	GBM	0.62	0.81	0.76	0.95	0.15	0.79	0.32
	XGB	0.62	0.84	0.80	0.94	0.15	0.84	0.35
	GNB	0.67	0.84	0.83	0.94	0.14	0.89	0.30
	BNB	0.58	0.76	0.69	0.94	0.11	0.70	0.41
	LDA	0.56	0.80	0.79	0.93	0.09	0.86	0.18
	LR	0.64	0.78	0.71	0.95	0.11	0.73	0.44
	GNB + LR	0.72	0.86	0.83	0.94	0.16	0.88	0.43

DT=Decision Tree, KNN= K Nearest Neighbors, RF=Random Forest, ADA= Adaboost, GBM= Gradient Boosting Machine,XGB= XGBoost, GNB= Gaussian Naive Bayes,BNB= Bernoulli Naive Bayes, LDA= Linear Discriminant Analysis, LR= Logistic Regression, AUC= Area under the curve,Acc= Accuracy, PC= Discharge Class, NC= Death Class

and mild pulmonary hypertension was there in 47 (4.36%) and 35 (2.87%) patients, respectively. Among total surgeries, 730 (67.78%) were elective surgeries, whereas 347 (32.22%) were emergency surgeries. Surgical procedure adopted include CABG in 248 (23.03%) patients, CABG+aortic valve replacement(AVR) in 22 (2.04%) patients, CABG+double valve replacement(DVR) in 6 (0.56%) patients, CABG+ mitral valve replacement(MVR) in 12 (1.11%) patients, Surgery on thoracic aorta in 67 (6.22%) patients, atrial septal defect closure(ASD) in 52 (4.83%) patients, AVR in 112 (10.40%) patients, DVR in 81 (7.52%) patients, MVR in 184 (17.08%) patients, and tricuspid valve replacement in 3 (0.28%) patients. Next, 160 (14.86%) were other cardiac surgeries, 130 (12.07%) were non-cardiac thoracic surgeries, and 13 (1.20%) patients had post infract septal rupture. A total of 991 (92.01%) patients were discharged successfully, whereas mortality occurred in 86 (7.99%) patients. The mean ICU stay was 5.48 ± 6.52 days, and the mean hospital stay was 11.67 ± 8.36 days, as shown in Table 6.1.

6.3.2 Performance evaluation of Survival outcomes

For the classification models to predict survival outcomes, the maximum AUC value was recorded for GNB+LR (0.72), followed by GNB (0.67), LR (0.64), XGB (0.62), GBM (62), BNB (0.59), and RF (0.50). As shown in Figure 6.5, GNB+LR has shown a very balanced sensitivity and

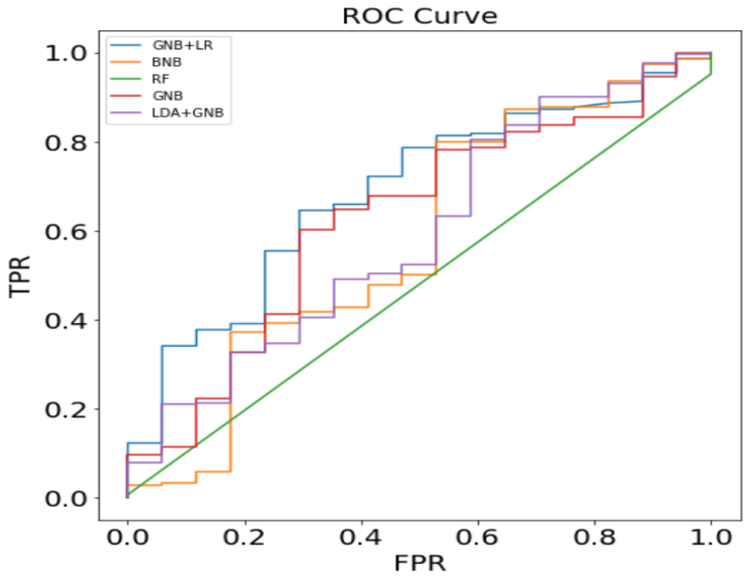


Figure 6.5: AUC Comparison

specificity for different values of FPR. Therefore, the performance of GNB+LR is more robust compared to any other model. As shown in Table 6.2, the highest F1-score has been recorded by GNB+LR (0.86), followed by GNB (0.84) and XGB (0.84). The highest accuracy was achieved by both GNB+LR (0.83) and GNB (0.83). All the models offered good precision values for the Discharge class. However, all the models recorded low precision values for the Death class, which resulted in false warnings. The best precision value for Death was recorded for GNB+LR (0.16), whereas the worst precision value for Death was recorded for DT (0.08) and LDA (0.09). The best test recall value for the Discharge class was recorded for GNB (0.89), followed by GNB+LR (0.88), and LDA(0.86). The best test recall value for the Death class was recorded for LR (0.44), followed by GNB+LR (0.43) and BNB (0.41). GNB+LR offers the best overall performance in terms of AUC, F1-score, accuracy, recall, and precision.

6.3.3 Performance evaluation of Hospital Stay

Table 6.3 depicts the experimental results for the regression models to predict the hospital stay, where the performance of these models has been evaluated using various performance evaluation metrics such as MAE, MSE, RMSE, and R2_score. For predicting hospital stay, the lowest RMSE was achieved using GB regressor (5.238), followed by XGB regressor (5.354), RF regressor (5.504), Ridge regression (5.568), K neighbours regressor (5.576), DT regressor (5.678), Lasso regression (5.847), SVR regressor (5.915), Linear regression (11.197), and Huber regressor (12.765). The highest performance for hospital stay analysis was registered using the metrics MSE and RMSE, obtained by the GB regressor.

When evaluated using both MAE and R2-score, the GB regressor was the best model for the analysis of hospital stay, with values of 3.548 and 0.023, respectively. The least performing algorithms, in this case, were linear regression and Huber regressor, with R2-score values of -3.461 and -4.799, respectively.

Table 6.3: Prediction of Hospital Stay

Pipeline	ML Models	MAE	MSE	RMSE	R2 Score
Remove artifacts ↓ Scaling (Robustscaler) ↓ Features Extraction ↓ Outlier Detection (IQR) ↓ Scaling (Robustscaler) ↓ Hyper-parameter Optimization ↓ Prediction	Linear Regression	4.764	125.379	11.197	-3.461
	Lasso Regression	3.856	34.194	5.847	-0.216
	Ridge Regression	3.651	31.005	5.568	-0.103
	K Neighbors Regressor	3.717	31.102	5.576	-0.106
	DT Regressor	3.841	32.241	5.678	-0.147
	RF Regressor	3.820	30.302	5.504	-0.078
	GB Regressor	3.548	27.446	5.238	0.023
	XGB Regressor	3.615	28.671	5.354	-0.020
	SVR(kernel =rbf)	3.856	34.994	5.915	-0.245
	Huber Regressor	11.610	162.96	12.765	-4.799

DT=Decision Tree, RF=Random Forest, GB= Gradient Boosting,XGB= XGBoost, MAE=Mean Absolute Error,MSE=Mean Square Error, RMSE= Root MSE

Table 6.4: Prediction of ICU Stay

Pipeline	ML Models	MAE	MSE	RMSE	R2 Score
Remove artifacts ↓ Scaling (Robustscaler) ↓ Features Extraction ↓ Outlier Detection (IQR) ↓ Scaling (Robustscaler) ↓ Hyper-parameter Optimization ↓ Prediction	Linear Regression	12.360	21710	147.34	-4149.1
	Lasso Regression	3.247	545.87	23.363	-103.34
	Ridge Regression	2.927	374.75	19.358	-70.635
	K Neighbors Regressor	1.659	5.340	2.310	-0.020
	DT Regressor	1.690	5.536	2.352	-0.058
	RF Regressor	1.634	4.994	2.234	0.045
	GB Regressor	1.582	4.551	2.133	0.129
	XGB Regressor	1.536	4.577	2.139	0.125
	SVR(kernel =rbf)	1.876	7.0	2.645	-0.338
	Huber Regressor	5.432	271.06	16.463	-50.814

DT=Decision Tree, RF=Random Forest, GB= Gradient Boosting, XGB = XGBoost, MAE=Mean Absolute Error,MSE=Mean Square Error, RMSE= Root MSE

6.3.4 Performance evaluation of ICU Stay

Table 6.4 presents the experimental results for the regression models to predict the ICU stay, where the performance of these models has been evaluated using various performance evaluation metrics such as MAE, MSE, RMSE, and R2_score. For predicting ICU stay, the lowest RMSE was achieved using KN regressor (3.28) followed by SVR rbf (5.29), RF regressor (5.60) followed by GB regressor (5.62), DT regressor (5.92), Ridge regressor (6.66), SVR Linear (6.84), Lasso regressor (6.95) and LR (7.57). The highest performance for ICU stay analysis was registered using the metrics MSE, and RMSE, obtained by the GB regressor.

When evaluated using both MAE and R2-score, the XGB regressor was the best model for the analysis of ICU stay, with values of 1.536 and 0.125, respectively. The models with the least performance were Lasso regression and Linear regression, with R2-score values of -103.34 and -4149.1, respectively.

6.4 Discussion

Several earlier studies had predicted the length of hospital stay and ICU stay post cardiac surgery. Tsai et al. [182] developed an artificial neural network (ANN) based model to predict the length

of stay in the pre-admission stage for inpatients diagnosed with heart failure. Using ANN and linear regression, the study predicted hospital stay correctly with MAE values of 3.83 and 3.76, respectively. However, no other performance criteria were mentioned. Another study proposed by Triana et al. [183] developed a model to predict the post-surgery length of stay for patients who undergo coronary artery bypass grafting (CABG). ANN was the best performing model for predicting the length of stay, achieving an RMSE of 3.342, an MAE of 1.853, and an R2-score of 0.212. Furthermore, Fang et al. [184] used a Bayesian Neural Network (BNN) model to predict the length of stay on the eICU-CRD (collaborative research database). With prior knowledge of the weights of NN, BNN achieves an MAE of 1.955044 and an R2-score of 0.097909 for ICU stay prediction. Moreover, another study conducted by Kadri et al. [185] introduced a novel approach based on a deep learning-driven generative adversarial network (GAN) model for predicting the patient length of stay in the emergency department (ED). The GAN model was the best performing model among all the deep learning models, achieving an RMSE of 100.309 and an MAE of 61.722. The developed model in the present study can effectively predict the length of hospital stay, and ICU stay after cardiac surgery. By referring to Table 6.3, we observe that the highest performance for hospital stay analysis was registered using the metrics MAE (3.548), RMSE (5.238), and R2-score (0.023), obtained by the IQR+GB regressor model. GB randomly samples the train datasets to obtain sample subsets of datasets and then trains the learner to reduce the residuals created by the previous learner. As a result, GB forces the prediction value close to the actual value, which improves the regression performance for the final integration. When comparing the performance of all regression models for ICU stay analysis, as displayed in Table 6.4, it was observed that the highest performance was registered using the metrics MSE (4.551) and RMSE (2.133), obtained by the IQR+GB regressor model. However, the model developed with IQR and XGB happened to be the best model in terms of MAE (1.536) and R2-score (0.125) for ICU stay analysis. XGB includes significant adaptation techniques such as shrinkage and instances subsampling which alleviates the over-fitting problem. A positive R2-score in both hospital stay and ICU stay analysis indicates that IQR is an effective method to detect anomalous observations in non-linear time series data. Thus, adopting the IQR outlier detection method in the present study enhances the regression performance of both hospital stay and ICU stay. Hence, the present study achieved a high prediction accuracy for both hospital stay and ICU stay analysis, compared to the models adopted in previous studies.

The performance of various Linear and Non-linear ML models has been evaluated using metrics like accuracy, precision, recall, F1-score, and AUC. However, as the data in our case is highly imbalanced, accuracy alone may not be a reliable metric for evaluating the performance of various trained ML classifiers/models. For example, the ML model has been tested on a dataset having 12% of "Death" cases. So, the classifier/model achieves an accuracy of 0.88, even if the model consistently predicted the "Discharge" outcome. Therefore, balancing the data and selecting the right metrics for evaluating the ML models/classifiers is essential.

We used the performance metrics for selecting the optimal model/classifier: Recall (Sensitivity) and AUC. It is essential to identify "Death" cases (i.e., high-risk patients). Therefore, we focus more on having a high recall of the "Death" class and an acceptable recall of the "Discharge" class. However, in real-world warning systems, it is desirable to have a high value of specificity (Precision) to avoid false warnings. We note that the count of positive samples ("Death" class) is much less than the count of negative samples ("Discharge" class). Due to this, even a small value of FPR brings about a large number of false warnings (FP), which results in a low precision value. Therefore, in the case of imbalanced datasets, it is hard to obtain an acceptable precision (specificity) value and high recall (sensitivity). Hence, it is essential to balance the specificity

and sensitivity criteria. The ROC curve (AUC) helps graphically visualise the trade-off between specificity and sensitivity for a given classifier. Also, the AUC measure is not dependent on data imbalance. Hence, it provides an unbiased evaluation parameter for the model/classifier's performance.

A strength of the present study is that it implemented a non-linear time series based artificial intelligence machine learning model for predicting the outcome of cardiac surgeries. To the best of our knowledge, no study has used the intraoperative blood pressure and heart rate data as an input to an AI model to predict patient outcomes. Recently, Fernandes et al, used intraoperative hypotension, vasopressor-inotropes, and cardiopulmonary bypass to predict mortality post cardiac surgery. The XGB model was found to predict mortality better with area under the receiver operating characteristic curve, 95% confidence interval (CI): 0.88(0.83-0.94); positive predictive value, 0.10(0.06-0.15); specificity 0.85 (0.83-0.87), and sensitivity 0.75 (0.57-0.90) [186].

There were a few limitations in our research, which we now discuss. To begin with, we used data from a single tertiary care institution. A multicenter registry and prospective investigations may be required to corroborate these findings. Second, our findings are on adult cardiac surgical patients; they cannot be applied to other groups, such as children or non-cardiac surgical patients. Finally, we did not incorporate other static parameters which could improve model performance. Future work could be based on the addition of static parameters and a distributed framework for data engineering, model training, and validation. The distributed framework will provide the advantage of scaling the solution for big data analysis and real-time response. Furthermore, RNNs cannot handle the long-term dependencies due to vanishing/exploding gradient problem. LSTM is a good option for such sequences that have long term dependencies, and is powerful when the data contains time series.

6.5 Conclusion

In this study, several ML models were trained on the specific non-linear time series dataset with HR and IBP parameters to predict patients' survival outcome, ICU stay, and hospital stay, determining risks following cardiac surgery. By applying the SMOTE and Near-Miss techniques, hyper-parameter optimisation algorithms, and 10-fold cross-validation, the performance of hybrid classifiers for survival outcome identification was systematically investigated. From the experiments, the GNB+LR ensemble model was the best model for survival analysis, offering the highest AUC of 0.72, an accuracy of 0.83, and an F1 score of 0.86. The GB regressor was found to be the best model for the analysis of hospital stay, having the lowest RMSE (5.238). The XGB regressor was the best model for ICU stay analysis offering the highest R2-score (0.125). The results demonstrated that a combination of ensemble ML models and refined feature engineering could accurately predict patient mortality.

Chapter 7

Conclusion and Future Work

The thesis explores parallel and distributed AI/ML on the edge. Specifically, security and healthcare have been picked as the use cases. The first work explores ML model partitioning on the edge, the second one predicts intrusion detection on edge. The third work performs DNN task offloading on edge, and the fourth works predicts outcomes of cardiac surgery patients on the edge. We hold the view that fog computing offers advantages to applications necessitating swift response times and can concurrently reduce the volume of traffic directed towards cloud data centers. This thesis addresses four distinct research challenges, namely: Partitioning Machine Learning Models on Edge Architectures, Intrusion Detection System on Fog Architecture, Data-Driven Deep Neural Network Task Offloading on Edge Networks, and non-linear time-series based AI model to predict outcomes in cardiac surgery.

In the Chapter 3, we proposed a PSVM-EA (Partitioning Support Vector Machine on Edge Architectures) framework that partitions the weight update operation on multiple edge nodes, in order to train the SGD-SVM (Stochastic Gradient Descent based Support Vector Machines) model in a parallel and distributed manner. This significantly reduces the training time, without majorly affecting the accuracy. The testing of the proposed approach was done in a parallel manner by partitioning the vector multiplication of all the features on the edge architecture. Further, to enhance the safety and reliability of the online model training process, we incorporated the Triple Modular Redundancy (TMR) technique for trusted computation. TMR is an established Single Event Upset (SEU) technique that replicates the processing of each sub-model across three separate devices, allowing for error detection and correction in the event of any discrepancies between their outputs. By employing TMR, our proposed algorithm ensures that the system maintains its integrity and reliability, even in the presence of potential faults, hardware compromises, or other safety issues. The Optimal ML Model Partitioning framework dynamically manages resources and minimizes training time and latency to optimize performance, while ensuring safety and reliability in edge networks. We conducted a case study on SVM and RF multi-class classifiers, by splitting the models into multiprocessor edge devices.The experimental results demonstrate a significant reduction in training time, and increased system throughput, without compromising accuracy. Our proposed approach achieves a significant speedup of approximately 56.3% in net training time compared to the non-partitioning approach, making it more efficient and suitable for real-time applications in edge networks.

An Intrusion Detection System on Fog Architecture is presented in Chapter 4. The FC-IDS (Fog Cluster-based Intrusion Detection System) framework comprises four phases: feature extraction, feature selection, selecting the machine learning model, deploying and evaluating the trained model on a Raspberry Pi Cluster and the Cloud. The training and testing of ML models have been done in a distributed manner, using a Raspberry pi cluster as a fog environment. The experiments show that the pi-cluster (fog) took less time for inference as compared to the cloud. In order to deal with class imbalance, the Synthetic Minority Oversampling Technique (SMOTE) was applied, which improved the performance of the proposed approach significantly. Feature reduction was

performed using Principal Component Analysis (PCA). This reduces the computation significantly, in turn, reducing the training & testing time of the model, without affecting the accuracy by much. The proposed FC-IDS system has been evaluated on the Australian Defence Force Academy Linux Datasets (ADFA-LD). These datasets comprise new generation system calls for various attacks on multiple applications. The proposed fog architecture offers significant advantages in terms of latency, energy consumption, and cost over traditional cloud or dedicated personal computer systems. Further, to enhance the performance in the case of multi-class intrusion attacks, we used a context-aware feature extraction approach. The proposed feature extraction approach is applicable to both HIDS and NIDS. In addition, the FCAFE-BNET algorithm exploits the early exit mechanism, due to which exiting the DNN from intermediate layers is possible after the desired result is obtained, instead of passing through the entire model, in order to reduce the processing time. The proposed algorithm has been examined using various IDS datasets, like NSL-KDD, ADFA_LD, UNSW-NB, and ToN-IoT. Our proposed technique has recorded a significant reduction in the total inference time, as compared to other state-of-the-art techniques.
In Chapter 5, we propose a data-driven task offloading algorithm that combines Mixed Integer Programming (MIP) and Machine Learning (ML) approaches to find optimal/near-optimal offloading solutions, which helps in migrating the computation to available edge devices in the network, based on network conditions and computational capacities. The proposed $D^2 - TONE$ algorithm offers superior performance due to its adaptation to the dynamic network conditions, while offloading the tasks. The experiments show that the training time is reduced significantly by using the proposed $D^2 - TONE$ approach, in comparison to other state-of-the-art techniques.
In chapter 6, our work aims to develop a model that uses intraoperative non-linear time-series data of IBP and HR to predict the ICU stay, hospital stay, and survival outcome in cardiac surgeries. Specifically, we aim to construct an end-to-end data analysis pipeline that incorporates artifact removal, non-linear noise reduction, and features engineering. We also adopt the synthetic minority oversampling (SMOTE), and NearMiss techniques to alleviate the inadequate classification generated by imbalanced data. Finally, we aim to evaluate the performance of different ML models and ensemble models by optimizing their hyperparameters in the prediction of survival outcome, ICU stay, and hospital stay.

7.1 Future Work

The potential future directions of the work done in this thesis are as follows:

- In the Chapter 3, we have proposed approaches that partitions the SVM and RF models on edge architectures. The future work could be to extend this work for various other ML algorithms. Also, the experiments were performed on CPUs in the present work, which could be extended further to GPUs.
- One interesting future work of Chapter 4 could be considering other critical evaluation metrics for IoT systems, such as storage efficiency and energy consumption. In the present work, the proposed framework has been evaluated based on accuracy, recall, F1-score, and inference time.
- In Chapter 5, the distributed DNN model learning is carried out on edge devices, which were connected using switch and ethernet cables. But, in order to make our research more realistic, we would like to extend our work to a wireless network setting.
- Future work in case of Chapter 6 could be incorporating other static parameters, which

could improve model performance. Another exciting work could be introducing a distributed framework for data engineering, model training, and validation. The distributed framework would provide the advantage of scaling the solution for big data analysis, and real-time response.

7.2 Implications of This Thesis

The rationale behind this thesis and the issues it tackles are highly pertinent within the realm of Fog/Edge Computing, which is gaining traction notably due to the proliferation of Internet of Things (IoT) devices within networks. The substantial influx of data stemming from these IoT devices poses a significant challenge in data management. Moreover, leveraging the computational and storage capabilities inherent in these IoT devices offers potential advantages, particularly in facilitating rapid response times for users by processing data at the network's edge. This thesis puts forth various algorithms aimed at: Partitioning Machine Learning Models on Edge Architectures, Intrusion Detection System on Fog Architecture, Data-Driven Deep Neural Network Task Offloading on Edge Networks, and non-linear time-series based AI model to predict outcomes in cardiac surgery.

One significant outcome of this thesis is to lay the foundations for developing a general framework for Intrusion Detection System on fog architecture. The proposed framework exploits context-aware feature extraction approach which helps in detecting various types of intrusions. The adaptation to the dynamic network conditions while offloading the tasks is also addressed in the proposed framework. Another key objective of this thesis is to optimize the utilization of fog resources based on the latency demands of the application. Applications with lower latency requirements will be assigned resources closer to the network, and so on.

This thesis offers a comprehensive depiction of the fog computing paradigm, encompassing a synopsis of the cutting-edge contributions of different DNN task offloading on edge networks, ML task partitioning on Edge Architectures, Intrusion Detection System on Fog Architecture, and real-time outcomes prediction in cardiac surgery. The foundational principles and constraints, as well as the enhancements made to these foundational contributions, have been underscored. Additionally, this thesis introduces potential avenues for future research that are worth exploring.

References

[1] He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE network*, 32(1):96–101, 2018.

[2] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[3] U Cisco. White paper: Cisco annual internet report (20182023). 2020.

[4] Mohammed Laroui, Boubakr Nour, Hassine Moungla, Moussa A Cherif, Hossam Afifi, and Mohsen Guizani. Edge and fog computing for iot: A survey on current research activities & future directions. *Computer Communications*, 180:210–231, 2021.

[5] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98: 289–330, 2019.

[6] Koustabh Dolui and Soumya Kanti Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[7] PJ Escamilla-Ambrosio, A Rodríguez-Mota, E Aguirre-Anaya, R Acosta-Bermejo, and M Salinas-Rosales. Distributing computing in the internet of things: cloud, fog and edge computing overview. In *NEO 2016: Results of the Numerical and Evolutionary Optimization Workshop NEO 2016 and the NEO Cities 2016 Workshop held on September 20-24, 2016 in Tlalnepantla, Mexico*, pages 87–115. Springer, 2018.

[8] Khaja Mannanuddin, Srinivas Aluvala, Y Sneha, E Kumaraswamy, E Sudarshan, and K Mahender. Confluence of machine learning with edge computing for iot accession. In *IOP Conference Series: Materials Science and Engineering*, volume 981, page 042003. IOP Publishing, 2020.

[9] Babatunji Omoniwa, Riaz Hussain, Muhammad Awais Javed, Safdar Hussain Bouk, and Shahzad A Malik. Fog/edge computing-based iot (feciot): Architecture, applications, and research issues. *IEEE Internet of Things Journal*, 6(3):4118–4149, 2018.

[10] Xiang Sun and Nirwan Ansari. Edgeiot: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, 54(12):22–29, 2016.

[11] Michele De Donno, Koen Tange, and Nicola Dragoni. Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog. *Ieee Access*, 7:150936–150948, 2019.

[12] Ola Salman, Imad Elhajj, Ayman Kayssi, and Ali Chehab. Edge computing enabling the internet of things. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 603–608. IEEE, 2015.

[13] Simar Preet Singh, Anand Nayyar, Rajesh Kumar, and Anju Sharma. Fog computing: from architecture to edge computing and big data processing. *The Journal of Supercomputing*, 75: 2070–2105, 2019.

[14] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–41, 2019.

[15] Mithun Mukherjee, Lei Shu, and Di Wang. Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Communications Surveys & Tutorials*, 20(3): 1826–1857, 2018.

[16] Gohar Rahman and Chuah Chai Wen. Fog computing, applications, security and challenges, review. *International Journal of Engineering & Technology*, 7(3):1615–1621, 2018.

[17] S Mohan Kumar and Darpan Majumder. Healthcare solution based on machine learning applications in iot and edge computing. *International Journal of Pure and Applied Mathematics*, 119(16):1473–1484, 2018.

[18] Pietro Spadaccino and Francesca Cuomo. Intrusion detection systems for iot: opportunities and challenges offered by edge computing and machine learning. *arXiv preprint arXiv:2012.01174*, 2020.

[19] Prabhat Kumar, Govind P Gupta, and Rakesh Tripathi. A distributed ensemble design based intrusion detection system using fog computing to protect the internet of things networks. *Journal of ambient intelligence and humanized Computing*, 12:9555–9572, 2021.

[20] Sung In Cho and Suk-Ju Kang. Real-time people counting system for customer movement analysis. *IEEE Access*, 6:55264–55272, 2018.

[21] Valério Nogueira, Hugo Oliveira, José Augusto Silva, Thales Vieira, and Krerley Oliveira. Retailnet: A deep learning approach for people counting and hot spots detection in retail stores. In *2019 32nd SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*, pages 155–162. IEEE, 2019.

[22] Haochen Hua, Yutong Li, Tonghe Wang, Nanqing Dong, Wei Li, and Junwei Cao. Edge computing with artificial intelligence: A machine learning perspective. *ACM Computing Surveys*, 55(9):1–35, 2023.

[23] Ali T Atieh. The next generation cloud technologies: a review on distributed cloud, fog and edge computing and their opportunities and challenges. *ResearchBerg Review of Science and Technology*, 1(1):1–15, 2021.

[24] H Sabireen and VJIE Neelanarayanan. A review on fog computing: Architecture, fog with iot, algorithms and research challenges. *Ict Express*, 7(2):162–176, 2021.

[25] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data*, pages 37–42, 2015.

[26] Ahmed M Alwakeel. An overview of fog computing and edge computing security and privacy issues. *Sensors*, 21(24):8226, 2021.

[27] Cosmin Avasalcai, Ilir Murturi, and Schahram Dustdar. Edge and fog: A survey, use cases, and future challenges. *Fog Computing: Theory and Practice*, pages 43–65, 2020.

[28] Juyong Kim, Yookoon Park, Gunhee Kim, and Sung Ju Hwang. Splitnet: Learning to semantically split deep networks for parameter reduction and model parallelization. In *International Conference on Machine Learning*, pages 1866–1874, 2017.

[29] Emad MalekHosseini, Mohsen Hajabdollahi, Nader Karimi, Shadrokh Samavi, and Shahram Shirani. Splitting convolutional neural network structures for efficient inference, 2020.

[30] Shengyu Fan, Hui Yu, Dianjie Lu, Shuai Jiao, Weizhi Xu, Fangai Liu, and Zhiyong Liu. Cscc: Convolution split compression calculation algorithm for deep neural network. *IEEE Access*, 7:71607–71615, 2019.

[31] Guanghui Zhu, Qiu Hu, Rong Gu, Chunfeng Yuan, and Yihua Huang. Forestlayer: Efficient training of deep forests on distributed task-parallel platforms. *Journal of Parallel and Distributed Computing*, 132:113–126, 2019.

[32] Li Zhou, Mohammad Hossein Samavatian, Anys Bacha, Saikat Majumdar, and Radu Teodorescu. Adaptive parallel execution of deep neural networks on heterogeneous edge devices. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pages 195–208, 2019.

[33] Md Al Maruf and Akramul Azim. Extending resources for avoiding overloads of mixed-criticality tasks in cyber-physical systems. *IET Cyber-Physical Systems: Theory & Applications*, 5(1):60–70, 2019.

[34] Md Al Maruf and Akramul Azim. Requirements-preserving design automation for multiprocessor embedded system applications. *Journal of Ambient Intelligence and Humanized Computing*, 12:821–833, 2021.

[35] Md Maruf and Akrumal Azim. Optimizing DNNs Model Partitioning for Enhanced Performance on Edge Devices. *Proceedings of the Canadian Conference on Artificial Intelligence*, jun 5 2023. https://caiac.pubpub.org/pub/ly32gqd5.

[36] M. Sahi, M. A. Maruf, A. Azim, and N. Auluck. A framework for partitioning support vector machine models on edge architectures. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 293–298, 2021.

[37] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco. Distributed inference acceleration with adaptive dnn partitioning and offloading. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 854–863, 2020.

[38] K. Tan, J. Zhang, Q. Du, and X. Wang. Gpu parallel implementation of support vector machines for hyperspectral image classification. *In IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(10):4647–4656, 2015.

[39] X. Li, L. Cheng, C. Sun, K. Lam, X. Wang, and F. Li. Federated-learning-empowered collaborative data sharing for vehicular edge networks. *IEEE Network*, 35(3):116–124, 2021.

[40] S. Zhou, Y. Huo, S. Bao, B. Landman, and A. Gokhale. Fedaca: An adaptive communication-efficient asynchronous framework for federated learning. In *IEEE*

International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), pages 71–80, 2022.

[41] N. Zhao, Z. Ye, Y. Pei, Y. Liang, and D. Niyato. Multi-agent deep reinforcement learning for task offloading in uav-assisted mobile edge computing. *IEEE Transactions on Wireless Communications*, 21(9):6949–6960, 2022.

[42] X. Gu and A. Easwaran. Towards safe machine learning for cps: infer uncertainty from training data. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 249–258, 2019.

[43] R. Hilbrich. How to safely integrate multiple applications on embedded many-core systems by applying the "correctness by construction" principle. *Advances in Software Engineering*, 2012.

[44] Jesus Pacheco, Victor H Benitez, Luis C Felix-Herran, and Pratik Satam. Artificial neural networks-based intrusion detection system for internet of things fog nodes. *IEEE Access*, 8: 73907–73918, 2020.

[45] Doaa Mohamed and Osama Ismael. Enhancement of an iot hybrid intrusion detection system based on fog-to-cloud computing. *Journal of Cloud Computing*, 12(1):1–13, 2023.

[46] K Kalaivani and M Chinnadurai. A hybrid deep learning intrusion detection model for fog computing environment. *Intelligent Automation & Soft Computing*, 30(1), 2021.

[47] Faisal Alghayadh and Debatosh Debnath. A hybrid intrusion detection system for smart home security based on machine learning and user behavior. *Advances in Internet of Things*, 11(1):10–25, 2021.

[48] Vinayakumar Ravi, Rajasekhar Chaganti, and Mamoun Alazab. Recurrent deep learning-based feature fusion ensemble meta-classifier approach for intelligent network intrusion detection system. *Computers and Electrical Engineering*, 102:108156, 2022.

[49] Narendra Mohan et al. A novel intrusion detection technique based on fog computing using cholesky factorization based online sequential extreme learning machines with persistent regularization. *International Journal of Control and Automation*, 12(6):117 – 126, Dec. 2019. URL http://sersc.org/journals/index.php/IJCA/article/view/2026.

[50] Adeel Abbas, Muazzam A Khan, Shahid Latif, Maria Ajaz, Awais Aziz Shah, and Jawad Ahmad. A new ensemble-based intrusion detection system for internet of things. *Arabian Journal for Science and Engineering*, pages 1–15, 2021.

[51] Zakaria Abou El Houda, Bouziane Brik, and Lyes Khoukhi. "why should i trust your ids?": An explainable deep learning framework for intrusion detection systems in internet of things networks. *IEEE Open Journal of the Communications Society*, 3:1164–1176, 2022.

[52] Yakubu Imrana, Yanping Xiang, Liaqat Ali, and Zaharawu Abdul-Rauf. A bidirectional lstm deep learning approach for intrusion detection. *Expert Systems with Applications*, 185: 115524, 2021.

[53] Saif S Kareem, Reham R Mostafa, Fatma A Hashim, and Hazem M El-Bakry. An effective feature selection model using hybrid metaheuristic algorithms for iot intrusion detection. *Sensors*, 22(4):1396, 2022.

[54] Benyamin Abdollahzadeh, Farhad Soleimanian Gharehchopogh, and Seyedali Mirjalili. Artificial gorilla troops optimizer: a new nature-inspired metaheuristic algorithm for global optimization problems. *International Journal of Intelligent Systems*, 36(10):5887–5958, 2021.

[55] Belal Sudqi Khater, Ainuddin Wahid Bin Abdul Wahab, Mohd Yamani Idna Bin Idris, Mohammed Abdulla Hussain, and Ashraf Ahmed Ibrahim. A lightweight perceptron-based intrusion detection system for fog computing. *applied sciences*, 9(1):178, 2019.

[56] Mansi Sahi, Mahip Soni, and Nitin Auluck. An intrusion detection system on fog architecture. In *2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, pages 591–596. IEEE, 2021.

[57] Poulmanogo Illy, Georges Kaddoum, Christian Miranda Moreira, Kuljeet Kaur, and Sahil Garg. Securing fog-to-things environment using intrusion detection system based on ensemble learning. In *2019 IEEE wireless communications and networking conference (WCNC)*, pages 1–7. IEEE, 2019.

[58] Deepa Rani and Narottam Chand Kaushal. Supervised machine learning based network intrusion detection system for internet of things. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE, 2020.

[59] Kishwar Sadaf and Jabeen Sultana. Intrusion detection based on autoencoder and isolation forest in fog computing. *IEEE Access*, 8:167059–167068, 2020.

[60] Chenmeng Wang, Chengchao Liang, F Richard Yu, Qianbin Chen, and Lun Tang. Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. *IEEE Transactions on Wireless Communications*, 16(8):4924–4938, 2017.

[61] Yiming Liu, F Richard Yu, Xi Li, Hong Ji, and Victor CM Leung. Distributed resource allocation and computation offloading in fog and cloud networks with non-orthogonal multiple access. *IEEE Transactions on Vehicular Technology*, 67(12):12137–12151, 2018.

[62] Changsheng You, Kaibin Huang, Hyukjin Chae, and Byoung-Hoon Kim. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 16(3):1397–1411, 2016.

[63] Zhaolong Ning, Peiran Dong, Xiangjie Kong, and Feng Xia. A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things. *IEEE Internet of Things Journal*, 6(3):4804–4814, 2018.

[64] Nitin Auluck, Akramul Azim, and Kaneez Fizza. Improving the schedulability of real-time tasks using fog computing. *IEEE Transactions on Services Computing*, 2019.

[65] Nitin Auluck, Omer Rana, Surya Nepal, Andrew Jones, and Anil Singh. Scheduling real time security aware tasks in fog networks. *IEEE Transactions on Services Computing*, 2019.

[66] Phu Lai, Qiang He, Xiaoyu Xia, Feifei Chen, Mohamed Abdelrazek, John Grundy, John G Hosking, and Yun Yang. Dynamic user allocation in stochastic mobile edge computing systems. *IEEE Transactions on Services Computing*, 2021.

[67] Yang Liu, Changqiao Xu, Yufeng Zhan, Zhixin Liu, Jianfeng Guan, and Hongke Zhang. Incentive mechanism for computation offloading using edge computing: A stackelberg game approach. *Computer Networks*, 129:399–409, 2017.

[68] Quoc-Viet Pham, Tuan Leanh, Nguyen H Tran, Bang Ju Park, and Choong Seon Hong. Decentralized computation offloading and resource allocation for mobile-edge computing: A matching game approach. *IEEE Access*, 6:75868–75885, 2018.

[69] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24 (5):2795–2808, 2015.

[70] Keqin Li. Heuristic computation offloading algorithms for mobile users in fog computing. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(2):1–28, 2021.

[71] Min Guo, Xing Huang, Wei Wang, Bing Liang, Yanbing Yang, Lei Zhang, and Liangyin Chen. Hagp: A heuristic algorithm based on greedy policy for task offloading with reliability of mds in mec of the industrial internet. *Sensors*, 21(10):3513, 2021.

[72] Marios Avgeris, Dimitrios Spatharakis, Dimitrios Dechouniotis, Aris Leivadeas, Vasileios Karyotis, and Symeon Papavassiliou. Enerdge: Distributed energy-aware resource allocation at the edge. *Sensors*, 22(2):660, 2022.

[73] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[74] Shuang Lai, Xiaochen Fan, Qianwen Ye, Zhiyuan Tan, Yuanfang Zhang, Xiangjian He, and Priyadarsi Nanda. Fairedge: A fairness-oriented task offloading scheme for iot applications in mobile cloudlet networks. *IEEE Access*, 8:13516–13526, 2020.

[75] Yaser Jararweh, Manar Bani Issa, Mustafa Daraghmeh, Mahmoud Al-Ayyoub, and Mohammad A Alsmirat. Energy efficient dynamic resource management in cloud computing based on logistic regression model and median absolute deviation. *Sustainable Computing: Informatics and Systems*, 19:262–274, 2018.

[76] Fahimeh Farahnakian, Pasi Liljeberg, and Juha Plosila. Lircup: Linear regression based cpu usage prediction algorithm for live migration of virtual machines in data centers. In *2013 39th Euromicro conference on software engineering and advanced applications*, pages 357–364. IEEE, 2013.

[77] Rongdong Hu, Jingfei Jiang, Guangming Liu, and Lixin Wang. Cpu load prediction using support vector regression and kalman smoother for cloud. In *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, pages 88–92. IEEE, 2013.

[78] Shuai Yu, Xin Wang, and Rami Langar. Computation offloading for mobile edge computing: A deep learning approach. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6. IEEE, 2017.

[79] Fahimeh Farahnakian, Tapio Pahikkala, Pasi Liljeberg, and Juha Plosila. Energy aware consolidation algorithm based on k-nearest neighbor regression for cloud data centers. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 256–259. IEEE, 2013.

[80] Firdose Saeik, Marios Avgeris, Dimitrios Spatharakis, Nina Santi, Dimitrios Dechouniotis, John Violos, Aris Leivadeas, Nikolaos Athanasopoulos, Nathalie Mitton, and Symeon Papavassiliou. Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks*, 195:108177, 2021.

[81] Holger Fröhlich, Rudi Balling, Niko Beerenwinkel, Oliver Kohlbacher, Santosh Kumar, Thomas Lengauer, Marloes H Maathuis, Yves Moreau, Susan A Murphy, Teresa M Przytycka, et al. From hype to reality: data science enabling personalized medicine. *BMC medicine*, 16(1):1–15, 2018.

[82] Atila Kara, Sakir Akin, and Can Ince. Current opinion the response of the microcirculation to cardiac surgery. *Curr Opin Anesthesiol*, 28, 2015.

[83] Yue Yu, Chi Peng, Zhiyuan Zhang, Kejia Shen, Yufeng Zhang, Jian Xiao, Wang Xi, Pei Wang, Zhichao Jin, and Zhinong Wang. Machine learning methods for predicting long-term mortality in patients after cardiac surgery. *Front Cardiovasc Med*, 2021.

[84] Chuntao Wu, Fabian T Camacho, Andrew S Wechsler, Stephen Lahey, Alfred T Culliford, Desmond Jordan, Jeffrey P Gold, Robert SD Higgins, Craig R Smith, and Edward L Hannan. Risk score for predicting long-term mortality after coronary artery bypass graft surgery. *Circulation*, 125(20):2423–2430, 2012.

[85] Umberto Benedetto, Arnaldo Dimagli, Shubhra Sinha, Lucia Cocomello, Ben Gibbison, Massimo Caputo, Tom Gaunt, Matt Lyon, Chris Holmes, and Gianni D Angelini. Machine learning improves mortality risk prediction after cardiac surgery: systematic review and meta-analysis. *The Journal of Thoracic and Cardiovascular Surgery*, 2020.

[86] Ying Zhou, Si Chen, Zhenqi Rao, Dong Yang, Xiang Liu, Nianguo Dong, and Fei Li. Prediction of 1-year mortality after heart transplantation using machine learning approaches: A single-center study from china. *International Journal of Cardiology*, 339:21–27, 2021.

[87] Chin Siang Ong, Erik Reinertsen, Haoqi Sun, Philicia Moonsamy, Navyatha Mohan, Masaki Funamoto, Tsuyoshi Kaneko, Prem S Shekar, Stefano Schena, Jennifer S Lawton, et al. Prediction of operative mortality for patients undergoing cardiac surgical procedures without established risk scores. *The Journal of Thoracic and Cardiovascular Surgery*, 2021.

[88] Suveen Angraal, Bobak J Mortazavi, Aakriti Gupta, Rohan Khera, Tariq Ahmad, Nihar R Desai, Daniel L Jacoby, Frederick A Masoudi, John A Spertus, and Harlan M Krumholz. Machine learning prediction of mortality and hospitalization in heart failure with preserved ejection fraction. *JACC: Heart Failure*, 8(1):12–21, 2020.

[89] Timo Koponen, Johanna Karttunen, Tadeusz Musialowicz, Laura Pietiläinen, Ari Uusaro, and Pasi Lahtinen. Vasoactive-inotropic score and the prediction of morbidity and mortality after cardiac surgery. *British journal of anaesthesia*, 122(4):428–436, 2019.

[90] Tong Ruan, Liqi Lei, Yangming Zhou, Jie Zhai, Le Zhang, Ping He, and Ju Gao. Representation learning for clinical time series prediction tasks in electronic health records. *BMC medical informatics and decision making*, 19(8):1–14, 2019.

[91] J. M. G. Sánchez, N. Jörgensen, and M. Törngren. Edge computing for cyber-physical systems. *ACM Trans. Cyber-Phys. Syst.*, 2022.

[92] S. Hamdan, S. Almajali, M. Ayyash, H. B. Salameh, and Y. Jararweh. An intelligent edge-enabled distributed multi-task learning architecture for large-scale iot-based cyber–physical systems. *Simulation Modelling Practice and Theory*, 122:102685, 2023.

[93] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[94] W. Sun, J. Liu, and Y. Yue. Ai-enhanced offloading in edge computing: When machine learning meets industrial iot. *IEEE Network*, 33(5):68–74, 2019.

[95] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang. Toward an intelligent edge: Wireless communication meets machine learning. *IEEE communications magazine*, 58(1):19–25, 2020.

[96] S. Shahhosseini, D. Seo, A. Kanduri, T. Hu, S. Lim, B. Donyanavard, A. M. Rahmani, and N. Dutt. Online learning for orchestration of inference in multi-user end-edge-cloud networks. *ACM Transactions on Embedded Computing Systems*, 21(6):1–25, 2022.

[97] T. Arifeen, A. S. Hassan, and J. Lee. Approximate triple modular redundancy: A survey. *IEEE Access*, 8:139851–139867, 2020.

[98] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[99] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2013.

[100] Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

[101] Fruits dataset kaggle. Sriram Reddy Kalluri. https://www.kaggle.com/sriramr/fruits-fresh-and-rotten-for-classification. [Online; Last accessed 24 Oct 2020].

[102] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems, 2015.

[103] M. A. Maruf, A. Singh, A. Azim, and N. Auluck. Faster fog computing based over-the-air vehicular updates: A transfer learning approach. *IEEE Transactions on Services Computing*, 2021.

[104] Kaggle. Traffic, driving style, and road surface condition, 2023. Available online: https://www.kaggle.com/datasets/gloseto/traffic-driving-style-road-surface-condition (accessed on 20 March 2023).

[105] M. Ruta, F. Scioscia, G. Loseto, A. Pinto, and E. Di Sciascio. Machine learning in the internet of things: A semantic-enhanced approach. *Semantic Web*, 10(1):183–204, 2019.

[106] Eryk Schiller, Andy Aidoo, Jara Fuhrer, Jonathan Stahl, Michael Ziörjen, and Burkhard Stiller. Landscape of iot security. *Computer Science Review*, 44:100467, 2022.

[107] Eric Gyamfi and Anca Jurcut. Intrusion detection in internet of things systems: A review on design approaches leveraging multi-access edge computing, machine learning, and datasets. *Sensors*, 22(10):3744, 2022.

[108] Santosh Kumar Das and Vikash Kumar. Iot security enhancement system: A review based on fusion of edge computing and blockchain. *Constraint Decision-Making Systems in Engineering*, pages 204–218, 2023.

[109] Rami J Alzahrani and Ahmed Alzahrani. A novel multi algorithm approach to identify network anomalies in the iot using fog computing and a model to distinguish between iot and non-iot devices. *Journal of Sensor and Actuator Networks*, 12(2):19, 2023.

[110] NSL-KDDdataset,https://www.unb.ca/cic/datasets/nsl.html.

[111] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[112] Nour Moustafa and Slay Jill. The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. *Information Security Journal: A Global Perspective25*, (1-3):18–31, 2016.

[113] ToN_IoTdatasets(2020),https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-ton-iot-Datasets/.

[114] Victor Chang, Lewis Golightly, Paolo Modesti, Qianwen Ariel Xu, Le Minh Thao Doan, Karl Hall, Sreeja Boddu, and Anna Kobusińska. A survey on intrusion detection systems for fog and cloud computing. *Future Internet*, 14(3):89, 2022.

[115] ADFA-LDdataset,https://github.com/verazuo/a-labelled-version-of-the-ADFA-LD-dataset.

[116] Credit fraud detector. https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets.

[117] Smote for imbalanced classification with python. https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/.

[118] a-labelled-version-of-the-adfa-ld-dataset. https://github.com/verazuo/a-labelled-version-of-the-ADFA-LD-dataset.

[119] Frances Osamor and Briana Wellman. Deep learning-based hybrid model for efficient anomaly detection. *International Journal of Advanced Computer Science and Applications*, 13(4), 2022.

[120] Tansel Dokeroglu, Ayça Deniz, and Hakan Ezgi Kiziloz. A comprehensive survey on recent metaheuristics for feature selection. *Neurocomputing*, 2022.

[121] Amin Aminifar, Matin Shokri, Fazle Rabbi, Violet Ka I Pun, and Yngve Lamo. Extremely randomized trees with privacy preservation for distributed structured health data. *IEEE Access*, 10:6010–6027, 2022.

[122] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.", 2022.

[123] Siti-Farhana Lokman, Abu Talib Othman, Muhamad Husaini Abu Bakar, and Shahrulniza Musa. The impact of different feature scaling methods on intrusion detection for in-vehicle controller area network (can). In *Advances in Cyber Security: First International Conference, ACeS 2019, Penang, Malaysia, July 30–August 1, 2019, Revised Selected Papers 1*, pages 195–205. Springer, 2020.

[124] Taejoon Kim, Sang C Suh, Hyunjoo Kim, Jonghyun Kim, and Jinoh Kim. An encoding technique for cnn-based network anomaly detection. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2960–2965. IEEE, 2018.

[125] Akaash Vishal Hazarika, G Jagadeesh Sai Raghu Ram, and Eeti Jain. Performance comparision of hadoop and spark engine. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pages 671–674. IEEE, 2017.

[126] Seiya Tokui, Ryosuke Okuta, Takuya Akiba, Yusuke Niitani, Toru Ogawa, Shunta Saito, Shuji Suzuki, Kota Uenishi, Brian Vogel, and Hiroyuki Yamazaki Vincent. Chainer: A deep learning framework for accelerating the research cycle. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2002–2011. ACM, 2019.

[127] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[128] Zheng-Wu Yuan and Jun Zhang. Feature extraction and image retrieval based on alexnet. In *Eighth International Conference on Digital Image Processing (ICDIP 2016)*, volume 10033, pages 65–69. SPIE, 2016.

[129] Joseph A. Cottam, Natalie C. Heller, Christopher L. Ebsch, Rahul Deshmukh, Patrick Mackey, and George Chin. Evaluation of alignment: Precision, recall, weighting and limitations. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 2513–2519. IEEE, 2020. doi: 10.1109/BigData50022.2020.9378064.

[130] Johan Barthélemy, Nicolas Verstaevel, Hugh Forehead, and Pascal Perez. Edge-computing video analytics for real-time traffic monitoring in a smart city. *Sensors*, 19(9):2048, 2019.

[131] Md. Al Maruf, Anil Singh, Akramul Azim, and Nitin Auluck. Faster fog computing based over-the-air vehicular updates: A transfer learning approach. *IEEE Transactions on Services Computing*, pages 1–1, 2021. doi: 10.1109/TSC.2021.3099897.

[132] Gustavo Caiza, Morelva Saeteros, William Oñate, and Marcelo V Garcia. Fog computing at industrial level, architecture, latency, energy, and security: A review. *Heliyon*, 6(4):e03706, 2020.

[133] Mainak Adhikari, Mithun Mukherjee, and Satish Narayana Srirama. Dpto: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing. *IEEE Internet of Things Journal*, 7(7):5773–5782, 2019.

[134] Ola Salman, Imad Elhajj, Ali Chehab, and Ayman Kayssi. Iot survey: An sdn and fog computing perspective. *Computer Networks*, 143:221–246, 2018.

[135] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.

[136] Mingjin Gao, Rujing Shen, Jun Li, Shihao Yan, Yonghui Li, Jinglin Shi, Zhu Han, and Li Zhuo. Computation offloading with instantaneous load billing for mobile edge computing. *IEEE Transactions on Services Computing*, 2020.

[137] Jie Zhang, Hongzhi Guo, Jiajia Liu, and Yanning Zhang. Task offloading in vehicular edge computing networks: A load-balancing solution. *IEEE Transactions on Vehicular Technology*, 69(2):2092–2104, 2019.

[138] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, 19 (1):447–457, 2019.

[139] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24(2): 394–407, 2019.

[140] Yakun Huang, Xiuquan Qiao, Pei Ren, Ling Liu, Calton Pu, Schahram Dustdar, and Junliang Chen. A lightweight collaborative deep neural network for the mobile web in edge cloud. *IEEE Transactions on Mobile Computing*, 2020.

[141] Qian Huang, Kane Rodriguez, Nicholas Whetstone, and Steven Habel. Rapid internet of things (iot) prototype for accurate people counting towards energy efficient buildings. *J. Inf. Technol. Constr.*, 24:1–13, 2019.

[142] Sami Abdulla Mohsen Saleh, Shahrel Azmin Suandi, and Haidi Ibrahim. Recent survey on crowd density estimation and counting for visual surveillance. *Engineering Applications of Artificial Intelligence*, 41:103–114, 2015.

[143] Vishwanath A Sindagi and Vishal M Patel. A survey of recent advances in cnn-based single image crowd counting and density estimation. *Pattern Recognition Letters*, 107:3–16, 2018.

[144] Y. an, j. wu, and c. yue, "cnns for face detection and recognition," https://github.com/fusio-wu/cs231a project, 2017.

[145] Crowd counting dataset :. https://www.kaggle.com/fmena14/crowd-counting.

[146] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[147] Mithun Mukherjee, Vikas Kumar, Qi Zhang, Constandinos X Mavromoustakis, and Rakesh Matam. Optimal pricing for offloaded hard-and soft-deadline tasks in edge computing. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[148] Yitu Wang and Takayuki Nakachi. A privacy-preserving learning framework for face recognition in edge and cloud networks. *IEEE Access*, 8:136056–136070, 2020.

[149] Anis Koubaa, Adel Ammar, Anas Kanhouch, and Yasser AlHabashi. Cloud versus edge deployment strategies of real-time face recognition inference. *IEEE Transactions on Network Science and Engineering*, 9(1):143–160, 2021.

[150] Shi-Jer Lou, Ming-Feng Hou, Hong-Tai Chang, Hao-Hsien Lee, Chong-Chi Chiu, Shu-Chuan Jennifer Yeh, and Hon-Yi Shi. Breast cancer surgery 10-year survival prediction by machine learning: A large prospective cohort study. *Biology*, 11(1):47, 2021.

[151] Douglas Kondziolka, Phillip V Parry, L Dade Lunsford, Hideyuki Kano, John C Flickinger, Susan Rakfal, Yoshio Arai, Jay S Loeffler, Stephen Rush, Jonathan PS Knisely, et al. The accuracy of predicting survival in individual patients with cancer. *Journal of neurosurgery*, 120(1):24–30, 2014.

[152] Upendra Kaul and Vineet Bhatia. Perspective on coronary interventions & cardiac surgeries in india. *The Indian journal of medical research*, 132(5):543, 2010.

[153] Niv Ad, Sari D Holmes, Jay Patel, Graciela Pritchard, Deborah J Shuman, and Linda Halpin. Comparison of euroscore ii, original euroscore, and the society of thoracic surgeons risk score in cardiac surgery patients. *The Annals of thoracic surgery*, 102(2):573–579, 2016.

[154] Kristien Van Loon, F Guiza, Geert Meyfroidt, J-M Aerts, Jan Ramon, Hendrik Blockeel, Maurice Bruynooghe, Greta Van den Berghe, and Daniel Berckmans. Prediction of clinical conditions after coronary bypass surgery using dynamic data analysis. *Journal of medical systems*, 34(3):229–239, 2010.

[155] DK Thara, BG PremaSudha, and Fan Xiong. Auto-detection of epileptic seizure events using deep neural network with different feature scaling techniques. *Pattern Recognition Letters*, 128:544–550, 2019.

[156] Rainer Hegger, Holger Kantz, and Thomas Schreiber. Practical implementation of nonlinear time series methods: The tisean package. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 9(2):413–435, 1999.

[157] Marília Barandas, Duarte Folgado, Letícia Fernandes, Sara Santos, Mariana Abreu, Patrícia Bota, Hui Liu, Tanja Schultz, and Hugo Gamboa. Tsfel: Time series feature extraction library. *SoftwareX*, 11:100456, 2020.

[158] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77, 2018.

[159] Jihao You, Edmond Lou, Mohammad Afrouziyeh, Nicole M Zukiwsky, and Martin J Zuidhof. A supervised machine learning method to detect anomalous real-time broiler breeder body weight data recorded by a precision feeding system. *Computers and Electronics in Agriculture*, 185:106171, 2021.

[160] Jiayao Chen, Hongwei Huang, Anthony G Cohn, Dongming Zhang, and Mingliang Zhou. Machine learning-based classification of rock discontinuity trace: Smote oversampling integrated with gbt ensemble learning. *International Journal of Mining Science and Technology*, 2021.

[161] Bahzad Charbuty and Adnan Abdulazeez. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2(01):20–28, 2021.

[162] Lishan Wang. Research and implementation of machine learning classifier based on knn. In *IOP Conference Series: Materials Science and Engineering*, volume 677, page 052038. IOP Publishing, 2019.

[163] Nejdet Dogru and Abdulhamit Subasi. Traffic accident detection using random forest classifier. In *2018 15th learning and technology conference (L&T)*, pages 40–45. IEEE, 2018.

[164] Siji Chen, Bin Shen, Xin Wang, and Sang-Jo Yoo. A strong machine learning classifier and decision stumps based hybrid adaboost classification algorithm for cognitive radios. *Sensors*, 19(23):5077, 2019.

[165] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

[166] Amal Asselman, Mohamed Khaldi, and Souhaib Aammou. Enhancing the prediction of student performance based on the machine learning xgboost algorithm. *Interactive Learning Environments*, pages 1–20, 2021.

[167] Simon Nusinovici, Yih Chung Tham, Marco Yu Chak Yan, Daniel Shu Wei Ting, Jialiang Li, Charumathi Sabanayagam, Tien Yin Wong, and Ching-Yu Cheng. Logistic regression was as good as machine learning for predicting major chronic diseases. *Journal of clinical epidemiology*, 122:56–69, 2020.

[168] Dastan Maulud and Adnan M Abdulazeez. A review on linear regression comprehensive in machine learning. *Journal of Applied Science and Technology Trends*, 1(4):140–147, 2020.

[169] R Muthukrishnan and R Rohini. Lasso: A feature selection technique in predictive modeling for machine learning. In *2016 IEEE international conference on advances in computer applications (ICACA)*, pages 18–20. IEEE, 2016.

[170] Julianna D Ianni, Zhipeng Cao, and William A Grissom. Machine learning rf shimming: Prediction by iteratively projected ridge regression. *Magnetic resonance in medicine*, 80(5): 1871–1881, 2018.

[171] Oliver Kramer. K-nearest neighbors. In *Dimensionality reduction with unsupervised nearest neighbors*, pages 13–23. Springer, 2013.

[172] Shikder Shafiul Bashar, Md Sazal Miah, AHM Zadidul Karim, and Md Abdullah Al Mahmud. Extraction of heart rate from ppg signal: a machine learning approach using decision tree regression algorithm. In *2019 4th International Conference on Electrical Information and Communication Technology (EICT)*, pages 1–6. IEEE, 2019.

[173] Xudong Zhou, Xinkai Zhu, Zhaodi Dong, Wenshan Guo, et al. Estimation of biomass in wheat using random forest regression algorithm and remote sensing data. *The Crop Journal*, 4(3):212–219, 2016.

[174] Upma Singh, Mohammad Rizwan, Muhannad Alaraj, and Ibrahim Alsaidan. A machine learning-based gradient boosting regression approach for wind power production forecasting: A step towards smart grid environments. *Energies*, 14(16):5196, 2021.

[175] Ali Shehadeh, Odey Alshboul, Rabia Emhamed Al Mamlook, and Ola Hamedat. Machine learning models for predicting the residual value of heavy construction equipment: An evaluation of modified decision tree, lightgbm, and xgboost regression. *Automation in Construction*, 129:103827, 2021.

[176] Fan Zhang and Lauren J O'Donnell. Support vector regression. In *Machine Learning*, pages 123–140. Elsevier, 2020.

[177] Yunlong Feng and Qiang Wu. A statistical learning assessment of huber regression. *Journal of Approximation Theory*, 273:105660, 2022.

[178] BH Shekar and Guesh Dagnew. Grid search-based hyperparameter tuning and classification of microarray cancer data. In *2019 second international conference on advanced computational and communication paradigms (ICACCP)*, pages 1–8. IEEE, 2019.

[179] Carlos Matias Scavuzzo, Juan Manuel Scavuzzo, Micaela Natalia Campero, Melaku Anegagrie, Aranzazu Amor Aramendia, Agustín Benito, and Victoria Periago. Feature importance: Opening a soil-transmitted helminth machine learning model via shap. *Infectious Disease Modelling*, 7(1):262–276, 2022.

[180] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. In *Australasian joint conference on artificial intelligence*, pages 1015–1021. Springer, 2006.

[181] Davide Chicco, Matthijs J Warrens, and Giuseppe Jurman. The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation. *PeerJ Computer Science*, 7:e623, 2021.

[182] Pei-Fang Jennifer Tsai, Po-Chia Chen, Yen-You Chen, Hao-Yuan Song, Hsiu-Mei Lin, Fu-Man Lin, and Qiou-Pieng Huang. Length of hospital stay prediction at the admission stage for cardiology patients using artificial neural network. *Journal of healthcare engineering*, 2016, 2016.

[183] Austin J Triana, Rushikesh Vyas, Ashish S Shah, and Vikram Tiwari. Predicting length of stay of coronary artery bypass grafting patients using machine learning. *Journal of Surgical Research*, 264:68–75, 2021.

[184] Jiansheng Fang, Junlin Zhu, and Xiaoqing Zhang. Prediction of length of stay on the intensive care unit based on bayesian neural network. *Journal of Physics: Conference Series*, 1631(1): 012089, 2020.

[185] Farid Kadri, Abdelkader Dairi, Fouzi Harrou, and Ying Sun. Towards accurate prediction of patient length of stay at emergency department: a gan-driven deep learning framework. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–15, 2022.

[186] Marta Priscila Bento Fernandes, Miguel Armengol de la Hoz, Valluvan Rangasamy, and Balachundhar Subramaniam. Machine learning models with preoperative risk factors and intraoperative hypotension parameters predict mortality after cardiac surgery. *Journal of Cardiothoracic and Vascular Anesthesia*, 35(3):857–865, 2021.