# IoT DATA STREAM HANDLING AND ANALYSIS

Thesis Submitted for the award of the Degree of

**DOCTOR OF PHILOSOPHY**

by

**Sanjay Patidar**

(2K18/PHD/CO/015)

Under the Supervision of

**Prof. Rajni Jindal**

Professor

Department of Computer Science and Engineering

Delhi Technological University

&

**Dr. Neetesh Kumar**

Associate Professor

Department of Computer Science and Engineering

Indian Institute of Technology , Roorkee

**Department of Computer Science and Engineering**

**DELHI TECHNOLOGICAL UNIVERSITY**

**(Formerly Delhi College of Engineering)**

**Shahbad Daulatpur , Bawana Road , Delhi-110042, India**

**August 2024**

**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi- 110042, INDIA

# CANDIDATE DECLARATION

I, Sanjay Patidar hereby certify that the work which is being presented in the thesis entitled "**IoT Data Stream Handling and Analysis**" in partial fulfilment of the requirement for the award of the degree of Doctor of Philosophy, submitted in the Department of Computer Science & Engineering, Delhi Technological University is an authentic record of my own work carried out during the period from 2018 to 2024 under the supervision of Prof. Rajni Jindal and Dr. Neetesh Kumar.

The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other Institution.

**Candidate's Signature**

This is to certify that the student has incorporated all the corrections suggested by the examiners in the thesis and the statement made by the candidate is correct to the best of my knowledge.

[illegible]

**Signature of Supervisor** **Signature of Co-Supervisor** **Signature of External Examiner**

**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi- 110042, INDIA

## CERTIFICATE

This is to certify that the work embodied in the thesis entitled "**IoT Data Stream Handling and Analysis**" submitted by **Mr. Sanjay Patidar**, Roll No: 2K18/PHD/CO/15 as a part-time scholar in the Department of Computer Science & Engineering, Delhi Technological University, is an authentic work carried out by his under my guidance. This work is based on the original research and has not been submitted in full or in part for any other diploma or degree of any university to the best of my knowledge and belief.

**Supervisor** | **Joint-Supervisor**

[illegible]

**Prof. Rajni Jindal**
Professor
Department of CSE
Delhi Technological University
Delhi-110042

**Dr. Neetesh Kumar**
Associate Professor
Department of CSE
IIT Roorkee
Haridwar-247667

Date: 20/08/2024

Place: New Delhi

# ACKNOWLEDGEMENT

I owe a debt of my gratitude to my supervisor, Prof. Rajni Jindal, Professor, Department of Computer Science and Engineering, Delhi Technological University, and joint-supervisor, Dr. Neetesh Kumar, Associate Professor, IIT Roorkee, Haridwar, for their valuable guidance, motivation, constant support and encouragement, which has helped me in formulating an urge for this research work. Their exemplary hard work and insightful guidance have always helped me overcome difficulties.

I would like to take this opportunity to thank all the faculty members of the Computer Science & Engineering Department, Delhi Technological University, for their encouragement and support. I am also thankful to all the staff members and research fellows for their untiring support.

I would like to extend deep gratitude to my lovely and pyari dadi Mrs. Kamla Bai W/o Late Pratap Singh Patidar, who blessed me to be a good human being, for always being a pillar of strength and support to manage my personal and professional life. I feel fortunate to have a great personality in my family who has given great moral to me throughout the journey of my life.

Last but not least, without whom I could not imagine being enrolled in a Ph.D., my father and mother, Mr. B.M. Patidar, Mrs. Pushpa for their unwavering encouragement, undeterred faith in me, and being the biggest pillar of strength which supported me till the end. It would not have been possible to complete my research without their love, encouragement, and support. Finally, I would like to express my gratitude to my wife Jyoti Patidar and my daughter Aashvi. Without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

**Mr. Sanjay Patidar**
Roll No: 2K18/PHD/CO/15
Department of Computer Science and Engineering
Delhi Technological University Delhi-110042

# ABSTRACT

Internet of Things (IoT) is an internet connectivity extension to the physical devices on heterogeneous networks to meet application requirements of IoT data on various grounds. IoT devices produce a significant volume of sensitive data, which they transfer to the cloud for processing and decision-making. IoT devices generate enormous amounts of streamed data, necessitating real-time responses. Numerous devices in the IoT enable continuous information generation and communication. When IoT devices generate a large amount of stream data and transmit it through the network, which relies on the internet and low-speed IoT infrastructure, real-time data transmission becomes a challenge. IoT devices face crucial challenges in processing, analyzing, and handling such data, as well as ensuring data security across multiple connected devices. We divided this research work into two subdomains: IoT data stream handling and IoT data stream analysis and we presented their significant contributions to the IoT data stream handling and analysis domain. Out of them, three belong to IoT data stream handling and three to the IoT data stream analysis subdomain.

**First**, we have conducted a systematic literature review of IoT data stream handling and analysis issues to highlight the challenges of the existing work and identify the various application areas as well.

**Second**, we proposed IoT data stream handling models. We developed a secure and energy-efficient hybrid model to address identified challenges and provide an effective solution that uses compression and encryption to improve energy and security efficiency in IoT devices. This hybrid architecture uses ChaCha12-Poly1305 for authenticated encryption with associated data (AEAD) and SZ 2.1 compression. Using the proposed model the device's battery life is also enhanced by 10% while ensuring the security of data. It also reduces both encryption and overall processing time by 95% and 98% respectively for two datasets used in the experiment. Another proposed delta encoding model compresses and encrypts the data stream during IoT network transmission, enabling compression in the form of deltas. We employed a lightweight stream cipher encryption technique to meet security requirements. The proposed delta encoding model is compared with the baseline LDPC encoding on temperature sensor datasets. As an result, the data are compressed up to 37.59%, and the average transmission time and the average power consumption are reduced by 72.57% and 68.86%, respectively.

**Third**, we worked on IoT data stream analysis, which involved analysing and forecasting the data generated by the IoT system. During pandemic time, we proposed a model for prediction of the transmission process of COVID-19 cases that uses the technique of long-short-term memory (LSTM), a deep learning technique suited for real data predictions based on time series stream data. The results shows that predicted model returns efficient results with minimal error for May, 4-15, 2020. This analysis  helps the relevant nation to take decisions in that time. Later, we proposed the grape, apple and potato leaf disease detection network (GAP-LDDN), which uses dual attention techniques to extract features, identify crop diseases, and classify diseases. As a result, model achieves 99.98%,97.88% and 99.88% accuracy for grapes, apple and potato datasets respectively. Then we experimented with machine learning techniques to analyse biofuel-producing plants.

**Lastly**, as an application of our research work, we invented a product as a solution to a problem in the agriculture sector, that was patented and is in the process of being translated into a commercialized product as an IoT and AI-based smart autonomous weed detection and removal system that removes unwanted weed from the crop field as well as collects and processes real-time data to make future predictions that will be used by farmers.

This research successfully provides a more reliable, effective, efficient, optimal, and secure data stream handling and analysis system using Internet of Things technology. The results of the experiments, analysis, and performance evaluation confirm that the provided work creates a practical and dependable IoT environment. Furthermore, the comparison study demonstrates that the proposed method is better than the existing methods that are already in use. Consequently, the results of this research successfully provide an IoT stream data handling and analysis system that is efficient, optimal, and secure by utilizing IoT technology.

# Content

# List of Abbreviations

| | |
|---|---|
| IoT | Internet of Things |
| AAD | Additional Associated Data |
| LDPC | Low Density Parity Check |
| GAP-LDDN | Grapes Apple Potato Leaf Disease Detection Network |
| R-CNN | Regional Convolutional Neural Network |
| M2M | Machine to Machine |
| M2H | Machine to Human |
| H2M | Human to Machine |
| LSTM | Long Short-Term Memory |
| BDA | Big Data Analytics |
| MAC | Message Authentication Code |
| AEAD | Authenticated Encryption with Associated Data |
| AH | Authentication Header |
| ESP | Encapsulating Security Period |
| CoAP | Constrained Application Protocol |
| SDA | Stream Data Analytics |
| OTK | One Time Key for Poly1305 algorithm |
| ABFT | Algorithm Based Fault Tolerance |
| ARX | Add Rotate XOR |
| RQ | Research Question |
| AES | Advanced Encryption Standard |

# List of Figures

# List of Tables

# List of Algorithms

# CHAPTER 1
# INTRODUCTION

This chapter commences by providing an overview of the Internet of Things (IoT) system, emphasizing the associated issues. It then proceeds to introduce the IoT data layers. The chapter also discusses the concept and requirements of the Internet of Things architecture, the motivation behind the research, the problem statement and the contributions made. The final section of the chapter incorporates the thesis structure.

## 1.1 Background

This section provides details of the IoT architecture, explains the associated data characteristics, challenges, limitations and presents the need to integrate the IoT technology with the IoT data processing system.

### 1.1.1 IoT Architectures

The Internet of Things (IoT) is an emerging and widely adopted technology for connecting any physical device to the internet, as well as other devices. It includes several physical objects, such as smart appliances, light bulbs, self-driving cars and fitness devices. IoT devices have embedded sensors, processors and communication hardware to assemble and transfer the data. Generally, a device communicates with other similar devices to act in unison and share information [1–2]. Large sensors and other connected devices in this IoT era generate a vast quantity of data. The future of this IoT era is merging with Industry 5.0, in which device-to-device communication at regular or irregular intervals increases the volume of data. This causes a high level of complexity and requires a high level of optimization in data handling [3]. Industry 5.0, an emerging research domain, generates massive streamed data through trillions of device-to-device communications, leading to a noticeable increase in data volume [4]. Several applications, including transportation, health and industry, require real-time processing of this streamed data, utilizing optimization schemes based on bloom filters to handle the data stream and account for the real-time constraint [5]. This section covers details of the IoT system, identifies the associated challenges, explains the IoT architecture concept and presents the need to integrate the IoT data transmission technology with the IoT system. To build an IoT solution, we must develop a process flow for a specific framework [6]. The IoT architecture generally comprises these four stages, as shown in Figure 1.1.

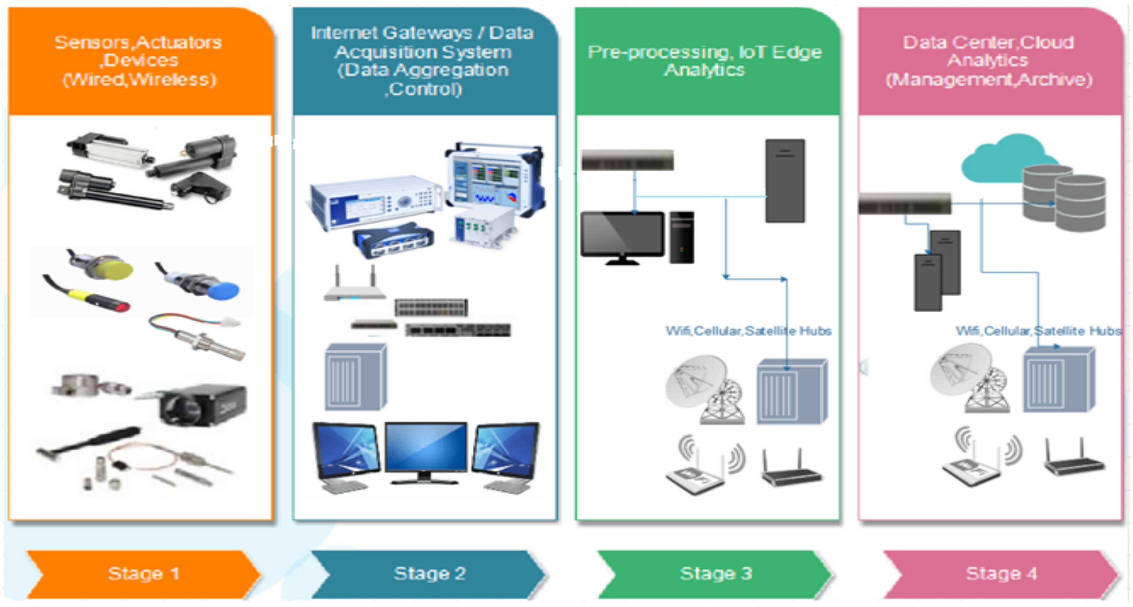

Sensors,Actuators ,Devices (Wired,Wireless)
Internet Gateways / Data Acquisition System (Data Aggregation ,Control)
Pre-processing, IoT Edge Analytics
Data Center,Cloud Analytics (Management,Archive)
Wifi,Cellular,Satellite Hubs
Wifi,Cellular,Satellite Hubs
Stage 1
Stage 2
Stage 3
Stage 4

Figure 1.1: Working Principles of the IoT Layered Architecture

**1. Physical Layer:** It consists of sensors that can gather data. Data is collected from the environment. It identifies the smart objects and collects physical parameters from them [7-8]. An actuator influences environmental change. For example, if there is a temperature change, the sensor will detect it and if the light is low? An actuator will switch on the street lights automatically.

**2. Internet Gateway:** In this layer, data generated by sensors should be pre-processed. The processing stage receives the data in analog form, which requires aggregation and digital conversion of that analog data. In this layer, the internet gateway routes the data over WLANs or networks, enabling real-time data collection and pre-processing. The gateways have some additional functionalities such as data management, data analytics and malware protection.

**3. Data Pre-processing:** In this stage, the data is pre-processed by edge computing or some IT infrastructure. The IT infrastructure performs further analysis of the data in edge computing [8]. The aforementioned two layers are located on the device side, while edge IT processing takes place in other remote or edge offices. Typically, we send IoT data to a server or data center, which consumes significant resources and network bandwidth. Therefore, edge computing systems perform data analytics to reduce IT infrastructure burdens.

**4. Analytics and Management:** At this stage, the IT system manages, stores and analyzes data. The data requires additional processing, but it does not require immediate feedback. Therefore, a cloud-based system or data centre is perfect for this purpose.

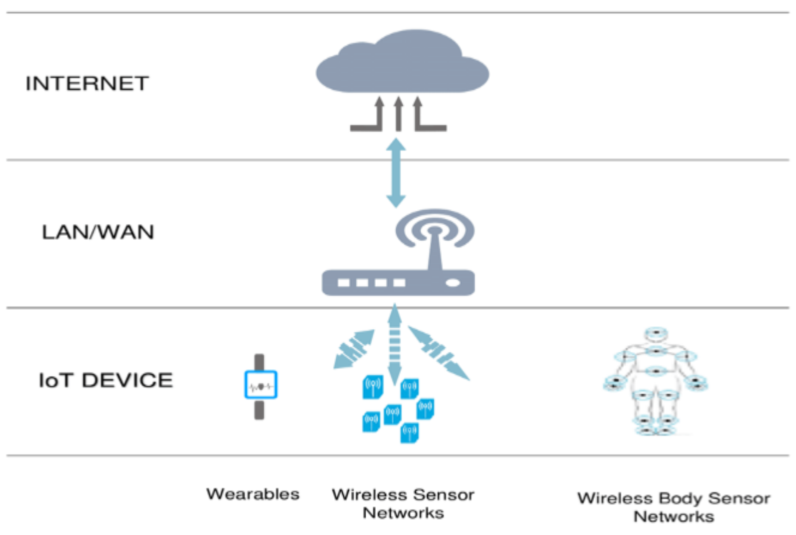


Figure 1.2: Working Principle of the IoT Network Architecture

Furthermore, the essential blocks of the IoT network architecture are IoT devices such as wireless sensor nodes, wearable devices, LAN/WAN connecting devices and the internet. Battery capacity and other computing resource constraints restrict the use of these objects for data collection, transmission and processing. The highest energy consumption in a cloud-based IoT architecture occurs during the data transmission phase [9].

Figure 1.2 shows the architecture of an IoT network, which collects data from smart objects such as sensor devices, wireless sensor networks and some wearable devices. Using LAN and WAN networks, various communication protocols, such as Bluetooth and Wi-Fi, transmit data for data processing, data storage and data filtering. Furthermore, the amount of data transmitted over the internet for storage and processing grows. Therefore, transmitting such a large amount of data while considering real-time application constraints on a network remains a challenging problem [10]. Given these requirements, deploying data compression techniques on IoT devices is a good idea. This work aims to reduce the transmission of bits for data processing, storage and filtering, while maintaining quality of service (QoS) parameters such as transmission delay and energy consumption and meeting the real-time constraint.

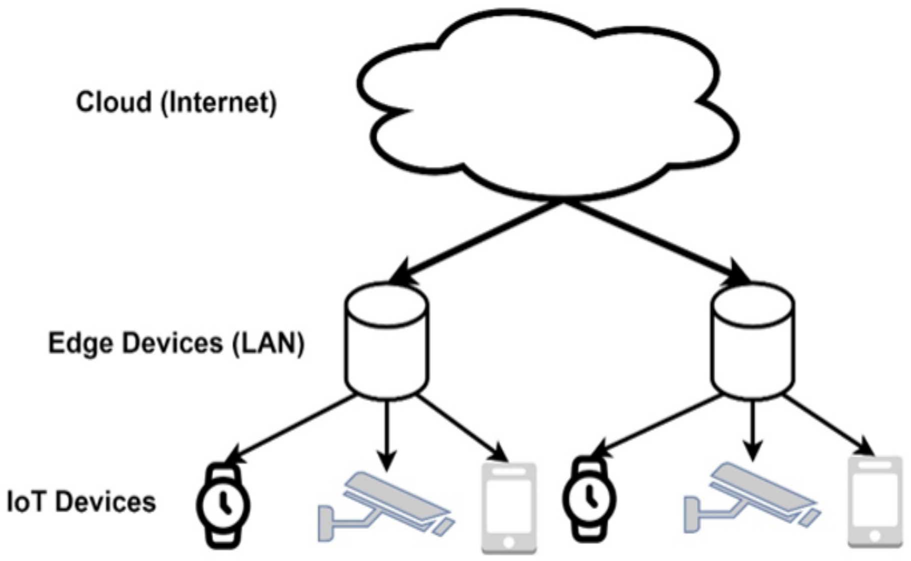


Figure 1.3: Working Principles of the Edge Computing Architecture

The accessibility of cloud computing, which is essential for processing IoT data, has allowed IoT to grow to its current size and complexity on a worldwide scale [11]. However, there are several drawbacks to this approach, including bandwidth issues, network latency in transmission, high energy consumption and security threats. These factors reduce the longevity and reliability of IoT systems. The most significant challenge that IoT devices must address is improving energy efficiency while still maintaining data security [12]. In IoT applications, sensors capture constant real-time streaming of data that must be cloud-based processed. Because of the huge amount of streaming data, transmission is energy-consuming on IoT devices, reducing their efficiency and lifetime. A mobile phone, a local PC, or an IoT gateway are examples of edge devices. By bringing computing capacity closer to IoT devices, the Edge Computing architecture, depicted in Figure 1.3, has emerged as a viable solution for this. We use the data gathered by smart IoT devices like sensors, camera and wearable's. The edge node may then process the data locally before transmitting it to the cloud. The practice of edge computing involves moving cloud services to the network's edge. We can effectively compare it to a decentralized cloud, positioning computing power close to the data source to facilitate local decision-making.

### 1.1.2 IoT Data Characteristics Challenges and Limitations

The massive data generation characteristics of IoT devices have made them be included under the big data category. Investigating the characteristics and features of IoT data is imperative to better understand the requirements for IoT big data analytics (BDA). The general characteristics of IoT data are as follows:

**1. Heterogeneity:** Because IoT systems involve multiple interconnected devices, they generate a vast and diverse amount of data. The data acquisition gadgets in IoT systems collect a variety of IoT data, resulting in data heterogeneity.

**2. Massive Streaming Data:** IoT applications deploy and distribute countless numbers of information-capturing devices. These devices continuously produce streams of information, resulting in a massive amount of unending data.

**3. Noise Data:** The IoT is a network of many interconnected components and devices that involves continuous data interaction and exchange. IoT applications continuously transmit and receive information between the connected devices. Therefore, noise and errors may affect data during such transmission and acquisition processes.

**4. Space and Time Correlation:** Many IoT systems connect sensor components to specific locations, thereby assigning each data item a time-stamp and location.

**Challenges:**

The IoT landscape is encountering unprecedented challenges due to the exponential rise in data generated within Industry 5.0, posing intricate dilemmas in data handling, transmission and analysis. The escalating device-to-device communication within IoT ecosystems amplifies the magnitude of the data influx, straining current data handling methodologies. Heterogeneous networks, characterized by varying speeds and divergent communication protocols, compound the intricacies of managing these diverse data streams. Concomitant factors, such as power consumption constraints and resource limitations within IoT devices, further exacerbate the data handling complexities.

**Limitations of Current Methods:** The existing data transmission methodologies face difficulties when controlled with the dynamic nature of IoT-generated data. The inadequacies inherent in current methods highlight the critical need to optimize data handling strategies, not only to minimize power consumption but also to ensure robust security measures for safeguarding sensitive information traversing these networks.

**Need for Streamlined Solutions:** The absence of streamlined approaches to addressing data reduction, compression and security issues complicates the management of IoT data streams. The dearth of tailored algorithms to optimize time, energy or classification constraints within varying data types, such as text and image datasets, intensifies the complexities faced in data management.

## 1.2 Motivation

Real time data processing over the IoT is a recent practice and they offer handling and analysis for huge amounts of data generated. Despite the numerous benefits of IoT, users prioritize security as the system processes highly confidential data. The integrity of IoT data remains an essential requirement in an IoT data processing system. Moreover, the speed of real-time data transmission in a battery-constrained environment has become a major challenge and issue in existing systems. During real-time data transmission, a significant volume of stream data from IoT devices traverses the network and undergoes processing on low-speed infrastructure. This could potentially impact the entire Internet of Things system. Additionally, we can analyze the vast volume of data generated to forecast future data, thereby resolving systemic issues. Therefore, achieving secure and fast data transmission through data analysis is a significant challenge. This section outlines the following research gaps associated with IoT system.

- The IoT real-time application should be able to communicate with other devices and handle real-time data. It should also deal with human cooperation, such as human-to-machine (H2M), human-to-human (H2H) and machine-to-machine (M2M), in addition to sensing and actuation (M2M).
- Another challenge in the IoT is developing a proper monitoring system to obtain real-time feedback from end users. In the future, it will be necessary to develop an IoT application to monitor various in-process applications, such as surgery and organ replacement monitoring systems, within the health sector. Other areas, such as water security monitoring and smart house creation apps, necessitate proper IoT applications.
- There is also a need to develop applications based on IoT for several agricultural uses, including smart packaging, text message warnings on land defects, intakes and so forth. However, the challenge lies in making these apps user-friendly for farmers.

- The IoT is now in its first stages of development, necessitating the establishment of a standardized communication architecture [13–19]. The design guarantees compatibility across various devices, simplifies the development process and ultimately incorporates standards that facilitate straightforward deployment. Certain reference designs have a role in facilitating IoT communication when it comes to addressing research difficulties in the field of IoT.
- The IoT context encompasses privacy and security standards. The IoT connects nearly trillions of devices, transmitting sensitive data like account details and personal information to ensure safe communication. To avoid these eventualities, security in IoT technologies is critical. As the number of IoT applications rises, privacy becomes the primary challenge, given the trillions of individuals linked to this platform and the vast volumes of data they exchange. We need to develop an IoT system that requires individuals' permission to use their data. Additionally, we can only use this data for specific purposes and within a limited scope.

## 1.3 Problem Statement

The IoT data stream handling and analysis system is receiving a significant amount of attention. The main goal of this thesis is to develop stream data handling and analysis techniques or algorithms that provide a privacy-preserving environment with all security features. The thesis addresses the following issues:

1. **Exploration of IoT Data Stream Techniques:** The first research objective motivates the exploration of various IoT data stream handling techniques to navigate the complexities of managing massive and diverse data streams generated by interconnected IoT devices. Using what we've learned from litrature discussing issues with communicating IoT devices, particularly the details provided in the paper about safe and energy-efficient edge computing, underscores the importance of understanding, scrutinizing and enhancing data handling practices. Unravelling the complexities of these techniques to enhance the efficiency, security, and resilience of IoT data streams serves as the motivation.

2. **Addressing Data Reduction, Compression and Security:** Current research that highlight the flaws in current data transmission methods strongly link to the need to develop algorithms that address data reduction, compression and security issues in IoT data streams. Drawing inspiration from the paper introducing novel approaches to merge compression and encryption

methods, the motivation stems from fortifying data transfer efficiency while ensuring stringent security measures. The significance lies in developing algorithms that mitigate vulnerabilities, reduce transmission times and fortify the integrity and confidentiality of transmitted data.

**3. Optimization for Time, Energy and Classification:** The state of art that were used as inspiration for the design of algorithms that improve time, energy use or classification problems in handling IoT data streams gives the importance of efficient classification and efficient computing. Inspired by this existing state of art, the objective is to propose algorithms that adapt to the dynamic requirements of IoT environments and optimizing performance metrics without compromising accuracy. The motivation here is to enhance resource utilization, reduce energy consumption and improve classification accuracy within text and image datasets, aligning with the evolving demands of IoT ecosystems.

**4. Real-time Data Prediction for Future Behaviour Understanding:** Developing prediction models on real-time IoT data streams to understand future behaviour is motivated by the futuristic insights highlighted in the existing work showcasing predictive analytics. The motivation derives from the necessity to anticipate and comprehend future patterns and trends within the vast IoT generated data. Leveraging insights from these studies, the objective is to harness predictive analytics to enable proactive decision-making, strategic planning and system responsiveness. Understanding real-time data behaviours serves as a cornerstone for adaptive decision-making and fostering resilient IoT environments. These motivations, drawn from the challenges, contributions, and futuristic insights delineated in the referenced studies, underscore the critical necessity and overarching significance of the outlined research objectives. They aim to develop advancements and innovations within the realm of IoT data handling, addressing challenges and capitalizing on opportunities to drive efficiency and resilience in IoT ecosystems.

**5. Anticipating Future Behaviours:** Understanding and predicting future behaviour within real-time data streams represent a significant challenge. These predictions hold paramount importance for proactive decision-making and strategic planning in sectors spanning agriculture, industrial automation and healthcare.

**6. Aligning Objectives with Existing Research:** Despite significant contributions from existing literature in understanding and addressing certain facts of these challenges, gaps and unexplored dimensions persist within IoT data handling techniques, algorithmic designs and predictive modelling. Bridging these gaps forms the impetus for this research, aiming to propose novel solutions, algorithms and predictive models tailored to address the intricate nuances of IoT data management. The convergence of these challenges underscores the critical necessity of novel solutions, algorithms and predictive models tailored to address the intricacies of IoT data handling, reduction, compression, security and prediction, forming the focal points of this research endeavour.

## 1.4 Contribution of the Thesis

This thesis endeavours to make substantial contribution to the evolving landscape of IoT data handling, analysis, security, power consumption and predictive modelling. Anchored in the exploration of diverse research objectives, the contributions of this thesis align with the following pivotal areas:

- Designing algorithms for IoT stream data handling
- Exploration and analysis of IoT data stream techniques
- Innovating predictive modelling for future behavior understanding
- Proposing innovative solutions for IoT data handling and their analysis

The thesis contributions are as follows:

1. The literature review was conducted to explore and highlight the techniques of IoT stream data technology utilized to secure data transmission from IoT devices at the physical layer to cloud storage systems. It also identifies the various application areas for integrating IoT data handling and analysis systems.
2. The proposed improve hybrid SZ2.1 model ensures confidentiality, integrity and power consumption features. It provides confidentiality by utilizing the encryption technique to encode the data. Using the improved SZ2.1 compression technique, the power consumption feature is achieved.
3. The IoT data stream handling using delta encoding is proposed to provide security features with an energy-efficient data transmission system in an IoT environment. The proposed model provides a complete approach to fast and secure data transmission, which also increases the battery power of IoT devices.
4. The proposed work implements the low-density parity check (LDPC) algorithm to ensure confidentiality, integrity and features. It provides confidentiality by utilizing the encryption technique to encode the data.

5. COVID-19 prediction model using a deep learning-based long short-term memory (LSTM) model, which was used during pandemic time for the prediction of future cases and to help save lives.
6. The proposed GAP-LDDN model uses the deep learning based ResNet 101 algorithm to identify and classify plant leaf diseases in real time. It is based on the Internet of Things (IoT) and is used for grape, apple, and potato leaf disease detection.
7. We have implemented the machine algorithms to identify plants that produce biofuel using Mask RCNN.
8. As innovation and application of the proposed work, we invented and own the concept of weed detection and removal system used by farmers to replace existing manual and mechanical system using IoT and AI. The system process real-time data collected by the farmer and helps death detection weed along with the crop in the crop field.

In summary, this thesis addressing and innovating solutions for the multifaceted challenges prevalent in IoT data handling, reduction, compression, security and predictive modelling, thereby contributing to the evolution and advancement of IoT ecosystems.

## 1.5 Thesis Organization

The organization of the thesis is presented in this section which is comprised of seven chapters as listed below:

### Chapter 1: Introduction

This chapter introduces the research work with the problem statement. We also present a brief outline of the thesis in this chapter.

### Chapter 2: Related Work

This chapter presents a research methodology by defining and explaining the research problem in detail. We have also presented a brief description of existing work related to IoT data stream handling and analysis. This chapter also identifies research gaps based on existing study. The following paper has been published from this work:

- Rajni Jindal, Neetesh Kumar and Sanjay Patidar, IoT Stream Data Compression Using LDPC Coding. *Evolution in Computational Intelligence, Frontiers in Intelligent Computing: Theory and Applications (FICTA 2020),* NIT Suratkal, Karnatka, 4-5th January, 2020, Volume 1 , pp. 455-466.

- Sanjay Patidar, Rajni Jindal, Neetesh Kumar, IoT Data Stream Handling, Analysis, Communication and Security Issue: A Systematic Survey, *Wireless Personal Communication,* Springer US, 2024, pp. 1-50. **(SCIE, IF: 1.9)**

## Chapter 3: Preliminaries

This chapter explains essential components of proposed IoT data steam handling and analysis techniques. It covers the technical details related to the proposed IoT data steam handling and analysis structure such as layers, IoT devices, security issues etc. The chapter also describes compression, transmission, prediction, analysis etc. algorithms. The following paper has been published from this work

- Rajni Jindal, Neetesh Kumar and Sanjay Patidar, An Energy Efficient, Secure Model for IoT Streamed Data Handling using Delta Encoding, *International Journal of Communication System,* Wiley, 2022. **(SCIE, IF: 1.82)**
- Sanjay Patidar, Rajni Jindal, Neetesh Kumar, IoT Data Stream Handling, Analysis, Communication and Security Issue: A Systematic Survey, *Wireless Personal Communication,* Springer, 2024, pp 1-50. **(SCIE, IF: 1.9)**

## Chapter 4: IoT Data Stream Handling

This chapter covers a detailed description of the proposed IoT data steam handling model. It presents the complete discussion of data transmission, compression, communication and security features achieved by the proposed model. The following paper has been published from this work:

- Sanjay Patidar, Rajni Jindal and Neetesh Kumar, A Secure and Energy-efficient Edge Computing Improved SZ 2.1 Hybrid Algorithm for Handling IoT Data Stream. *Multimedia Tools Appl*, Springer, 2024. **(SCIE, IF: 3.6)**
- Rajni Jindal, Neetesh Kumar and Sanjay Patidar, An Energy Efficient, Secure Model for IoT Streamed Data Handling using Delta Encoding, *International Journal of Communication System,* Wiley, 2022. **(SCIE, IF: 1.82)**
- Rajni Jindal, Neetesh Kumar and Sanjay Patidar., IoT Stream Data Compression Using LDPC Coding. *In book Evolution in Computational Intelligence, Frontiers in Intelligent Computing: Theory and Applications (FICTA 2020),* NIT Suratkal, Karnatka,4-5th January, 2020, Volume 1, Pp. 455-466.

## Chapter 5: IoT Data Stream Analysis

This chapter presents the IoT data analysis in the proposed technique. It gives the details of formulating an analysis model. The following paper has been published/communicated from this work:

- Sanjay Patidar, Rajni Jindal and Neetesh Kumar, Streamed Covid-19 Data Analysis Using LSTM—A Deep Learning Technique. *Advances in Intelligent Systems and Computing: Soft Computing for Problem Solving (SoCP 2020),* IIT Indore, MP, 18-19th Dec 2020, Vol 1393. Pp. 493-504. **(Awarded Best Paper)**
- Sanjay Patidar, Rajni Jindal and Neetesh Kumar , GAP-LDDN : Leaf Disease Detection Network based on Multitask Learning and Attention Features for IoT Data, *European Journal of Plant Pathology ,* Springer Nature, 2024. **(Communicated)**

## Chapter 6: Applications of Proposed Work

This chapter highlights the innovative application of the proposed work in agriculture domains. The following patent has been published from this work and received a research grant of 50L for commercialise it.

- Sanjay Patidar, Rajni Jindal, Neetesh Kumar, Smart Autonomous Weed Detector and Removal, *Official Journal of the Patent Office, India*, Patent Id : 202011028931, Issue No. 34/2020. **(Patent)**
- Received **Translation Research Project** for development and commercialization of real-time application in product from Technology Innovation Hub foundation, IIT Kharagpur. **(Project)**

## Chapter 7: Conclusion and Future Work

The final chapter presents the conclusion and future scope of the research work. This chapter deliberates the importance of the proposed model for providing data transmission, communication, security, prediction and classification to data generated from IoT systems.

**List of Publications:** This section lists published/accepted/communicated research papers relating to this research work in International/National Journals/Conferences of repute.

**References:** This section is the list of references cited in this research work.

# CHAPTER 2

# LITERATURE REVIEW

This chapter presents a literature review on traditional IoT data stream handling and analysis techniques that integrating handling and analysis of IoT data stream systems. Discuss the review process and their planning. Then conducted the review with the help of formulation of research question ,objective and later explore the research gaps and propose a solution.

## 2.1 Introduction

This subsection represents the literature review to comprehensively study existing and on-going research work. As illustrated in Figure 2.1, the literature review adheres to the outlined framework for the review process. First, the review process identifies the categories for research. The next step involves formulating research questions to review existing techniques. Next, the study presents the category-wise review report, identifying research gaps. Then finally review conclude with formulating the research objective (RO), conclusions and future directions.

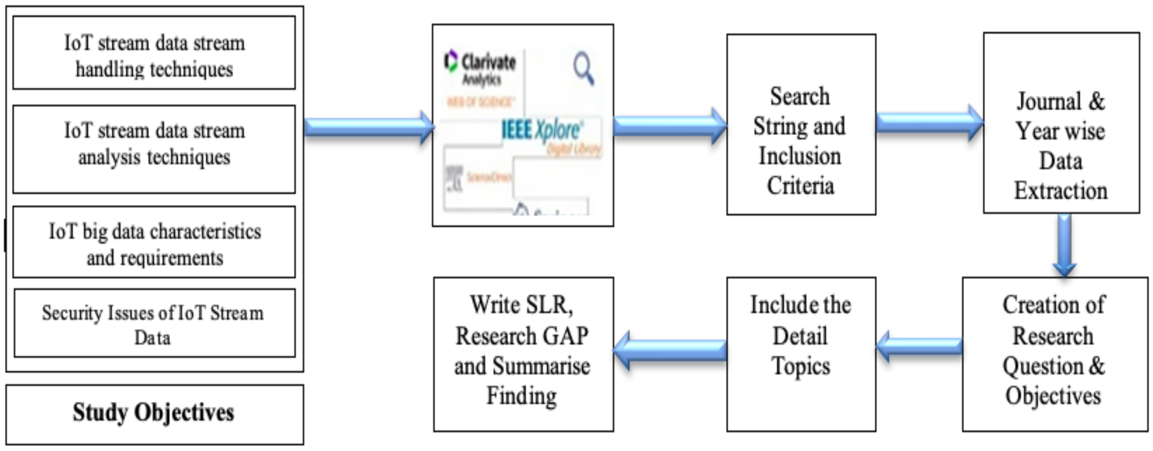

IoT stream data stream handling techniques
IoT stream data stream analysis techniques
IoT big data characteristics and requirements
Security Issues of IoT Stream Data
Study Objectives
Clarivate Analytics
WEB OF SCIENCE
IEEE Xplore Digital Library
ScienceDirect
Search String and Inclusion Criteria
Journal & Year wise Data Extraction
Creation of Research Question & Objectives
Include the Detail Topics
Write SLR, Research GAP and Summarise Finding

Figure 2.1: Framework of the Review Process

## 2.2 Planning the Review

This subsection discuss the strategies used in the literature review. The current study employs the literature review principle methodology to identify and analyze published work related to IoT data stream handling. Thus, we divided the review process into following steps:

- **Study Objectives:** In this sub-section various research objectives are studied and analyzed as per the research questions and their mapping with the above research questions and references of this study, which were discussed in introduction section 1.
- **Selection of Digital Libraries:** The selection of digital libraries is based on how the research studies align with the existing literature. Due to the huge number of research studies available, we used some search criteria to identify the studies, therefore, the search criteria became important. In this study, we have taken the different digital libraries, which are IEEE Xplore, Springer, Elsevier etc.
- **Selection of Search String:** A collection of characters or words that are entered into a search engine in order to pick particular information is referred to as a search string. There is a clear correlation between the information that we give to the search engine of the digital library and the results that we obtain on our search. We must be cautious to choose the keywords in the search string.

  String = (IoT + steam data + handling + analysis + security +communication+ optimization)*(datasets/techniques /evaluation parameters)

  Every digital library listed above have the search string applied, and the publication year given from 2002 to 2024. Therefore, as mentioned in the sections below, the literature was further processed using well defined inclusion-exclusion criteria and quality analysis criteria.
- **Research Inclusion Criteria:** The paper was included in our review if it describes research on IoT stream data handling, analysis, security and optimization under time and classification restrictions. Publications and position papers that omitted experimental results were disqualified. The papers' publication dates, datasets, methods, criteria for assessment and results were taken into consideration when evaluating them.
- **Data Extraction Attributes:** Each research study should develop an objective data extraction method to extract useful information and answer the research questions (RQ). For each research study that complies with the quality analysis requirements, Table 2.1 shows the various attributes for data extraction under systematic literature review (SLR).

Table 2.1: Attributes for Data Extraction

| S. No. | Attribute |
|---|---|
| 1. | Publication Category |
| 2. | Publication Date |
| 3. | Work |
| 4. | Method used |
| 5. | Technique used |
| 6. | Dataset |
| 7. | Results and Performance |

- **Paper's Categories:** Our research revealed that a variety of factors, such as datasets and techniques, can categorize research publications. Based on the publisher and year, we grouped the papers into various groups, such as IoT stream data handling, analysis, and security constraints.

## 2.3 Reporting the Review

In this section, we have conducted a review of IoT stream data handling and analysis into two categories. Then identified the research gaps based on this study.

### 2.3.1 IoT Stream Data Handling

This sub-section presents research work related to IoT data stream handling techniques utilized to improve the IoT system related to proposed model under following sub heading.

#### 2.3.1.1 Secure and Energy-Efficient IoT System

S. Lakshmi Narasimhan et al. [20] proposed the ISABELA algorithm for compression, which uses B-spline interpolation. However, sorting destroys the reallocation of each point, necessitating separate storage. Therefore, this approach produces low compression when dealing with large data sets. Furthermore, this technique depends heavily on the smoothness of the local region data, which is considered a drawback because spikey and abrupt data changes frequently occur in simulated data. This results in low prediction accuracy and degraded compression quality. Z. Chen et al. [21] proposed a compression algorithm called NUMARCK.

It compresses data using vector quantization. It stores two successive iterated sets of data. N. Sasaki et al. [22], just like NUMARCK, also quantize the data distribution. This algorithm effectively overcomes ISABELA's limitations by reducing the dependence on data efficacy. This algorithm is not error-bound, however. Furthermore, the compression ratio achieved is low. P. Lindstrom et al. [23] developed a lossy compressor based on a lifted orthogonal data block transform, offering the option of error-bounded compression. It doesn't rely on the data's smoothness. However, using this method, there is no way to guarantee that the decompressed results will satisfy the error bound. SZ 1.1 handles a significant volume of data generated during the execution of HPC applications. We design the difference between the input data and the decompressed data to stay within the error bound. Experimental demonstrations on the driver stress detection dataset show that an altered version of this technique can achieve a maximum compression ratio of 103. Using physiological signals, Healey et al. [24] suggested a method for measuring stress. We can use these signals to gather continuous driving task performance and provide feedback on the driver's stress. Goldberger et al. [25] explained different components of research resources related to complex physiologic signals. For high-performance computing applications (HPC), Azar et al. [26] suggested a fast error-bounded lossy compression scheme that deals with a wide range of HPC application-generated data. M. Burtscher et al. [27] describe a lossless data compression technique called FPC. Scientists fabricate it to compress data effectively and simultaneously meet the high throughput requirements of scientific computing environments. Although it is a fast-lossless compression technique, it is not capable of achieving the large compression ratios required for IoT applications. P. Lindstrom et al. [28] proposed a data compression technique that, unlike FPC, works well with integer data and variable-precision floating points. When compared to its predecessors, this compression method achieves higher throughput. However, it doesn't attain compression rates appropriate for IoT devices that produce a lot of data. K. L. Tsai et al. [29] proposed the strategy of secure low-power communication. This approach uses incredibly little power and is safe. By using fewer AES encryption cycles, it reduces energy consumption. The authors also recommended key management and the D-Box upgrade procedure. K. L. Tsai et al. proposed the LPADA architecture for the AES cipher [30]. By utilizing the energy-efficient S-Box, power gating strategies and other energy management methods, this architecture aims to reduce the energy consumption of AES. In contrast to traditional AES data encryption, the writers were able to achieve a dynamic power reduction of 62.0%. For IoT systems, S. Roy et al. [31] proposed another encryption technique. It is essentially a lightweight cipher based on cellular automata. While the edge node performs the decryption, the IoT node performs the encryption. It has demonstrated superior performance compared to DES and 3-DES, making it a valuable option for IoT devices with limited resources. Secure and Fast Encryption Routine (SAFER ) proposed by X. Guo et al. [32], is a complexity-reduced Fermat block encryption method. It takes into account the diffusion and confusion theories. The SAFER algorithm forms the confusion layer, while the Fermat number theory transform composes the diffusion layer. Due to the lack of multipliers and the focus on integer operations alone, this method has a low level of complexity. This method is suitable for Internet of Things devices with minimal power. S. Singh et al. [33] proposed a suite of cryptographic algorithms comprising encryption algorithms and hashes to design an efficient system consisting of resource-constrained IoT devices. They also analyzed a variety of lightweight ciphers based on certain characteristics, viz., the number of rounds, size of the key and block. D. A. F. Saraiva et al. [34] talk about how AES, ChaCha20-Poly1305, and a few more algorithms perform in IoT applications. They contrasted the operating times, throughputs and energy requirements of these algorithms. They discovered that the lightweight block ciphers (such as SPECK and LEA) and the ChaCha20-Poly1305 stream cipher are suitable alternatives for devices with limited resources. B. J. Mohd et al. [35] offered a straightforward approach for measuring the effectiveness of light ciphers. We use the model to identify areas where the encryption structure can enhance its effectiveness. We utilize it to forecast the ideal block size and the number of rounds, ensuring high throughput and minimal energy consumption. They also provided an enhanced energy management algorithm that enables a device to handle critical data even when there is a low energy level. A. A. Diro et al. [36] use the Fog computing architecture to ensure distribution and scalability, and offload the security functions from IoT devices to Fog nodes to reduce the burden on resource-constrained IoT devices. They use elgamal-based elliptic curve cryptography for data security, ensuring simplicity and energy efficiency. A. O. Akmandor et al. [37] proposed several architectures for different IoT applications. This order to boost security and energy efficiency while maintaining the devices' intelligence, this article combined decision-making utilizing machine learning and cryptography algorithms on IoT devices. On the IoT device, they applied compressive sensing, compressed signal processing and machine learning inference to achieve high classification accuracy while drastically reducing energy usage. A. Fragkiadakis et al. [38] presented the use of adaptive compressive sensing. The system uses compressive sensing to simultaneously deliver compression and encryption. We choose the compression rate in this study based on the projected data scarcity to ensure the system's effectiveness. Y. Zhang et al. [39] appraised Compressive Sensing's security features. The authors described the ChaCha20-Poly1305 AEAD algorithm as a small and quick implementation for ARM Cortex-M4 processors. J. Qi et al. [40] used a hybrid security and CS-based scheme for ensuring energy efficiency and security in IoT devices. They used compressive sensing for compressing signals, chaotic 8-bit block encryption for privacy, and message authentication codes for authentication. F. de Santis et al. [41] presented a compact and rapid implementation of the ChaCha20-Poly1305 AEAD algorithm for ARM Cortex-M4 processors. The results show that ChaCha20-Poly1305 is a reliable technique to ensure security in IoT devices. To secure data packets in lossy and low-powered networks, S. Luangoudom et al. [42] proposed using an authenticated encryption method based on XSalsa20 and Poly1305.

#### 2.3.1.2 Delta Encoding for IoT Streamed Data Handling

S. Fong et al. [43] proposed a novel and light-weighted feature selection method that uses accelerated particle swarm optimization (APSO). The design specifically targets the mining of stream data on the Y. The authors apply the swarm search approach incrementally and plan to use a new feature selection algorithm to evaluate the results. However, in the case of stationary data, APSO is a time-consuming approach. Thus, computational time efficiency is the main concern in IoT stream data handling. L. Du et al. [44] proposed the convolutional neural network (CNN) hardware accelerator-based streaming architecture. The accelerator performs the energy optimization by compressing irrelevant data movements. The Taiwan semiconductor manufacturing company (TSMC) implemented it, using 65 nm technology and a 5mm core size. This hardware IP has also implemented a tragic sign net and it is verified on the field-programmable gate array (FPGA). The achieved energy efficiency enhances its suitability for integration with IoT devices. However, this work does not account for the crucial role of data transmission over the IoT network in decision-making. H. Jin et al. [45] proposed a pre-scheduling straggler mitigation framework. They developed it to meet the requirements of real-time stream data processing. It optimizes data assignment by leveraging the predictability of iterated batch stream jobs. The authors implemented a lever that contributes to the Apache Spark Streaming extension. The lever introduces a model to evaluate node's capacity. B. Mohd et al. [46] presented a simple model for the performance metrics of lightweight ciphers. We used the model to search for opportunities to improve the performance of the cipher structure. We also used it to predict the optimal block size and the number of rounds required to achieve high throughput and low energy consumption. The work also presented a novel algorithm for energy management that allows a device to process important data with low energy constraints. Y. Zhang et al. [47] reviewed the security aspects of compressing. The authors investigated several compression sensing schemes by studying several performance measurement matrices in various network scenarios, such as IoT, WSNs, WBANs, etc. However, the system experiences a surge in time complexity, necessitating further improvement. T. Lloyd et al. [48] presented a lossless compression technique and proposed a run-length base delta encoding algorithm. For the data transmission mission, this algorithm provides a high-level compression and decompression rate that is suitable for up to 40 GB of real-time data. It also determines the expected throughput of compression and decompression for both the CPU and GPU in the synthetic data set. K. Tsai et al. [49] proposed a secure low-power communication (SeLPC) strategy. This method is safe and requires low power consumption. Furthermore, the authors have suggested the use of key management and the D-Box update procedure. This improves security and lowers power consumption, however, the authors ignored the use of data reduction techniques to reduce the power consumption of the IoT devices. S. Roy et al. [50] proposed an encryption technique for IoT systems known as lightweight cellular automata cipher (LCC) that is based on cellular automata. The IoT node performs the encryption, while the edge node performs the decryption. The results demonstrate enhanced performance compared to DES and 3DES, indicating a potential advantage for IoT devices with limited resources. However, the authors overlooked the potential to enhance IoT network speed and decrease power consumption through encryption.

#### 2.3.1.3 LDPC Coding for IoT Stream Data Handling

D. Puthal et al. [51] proposed a model that uses variable key lengths of 128, 64 and 32 bits, which helps provide real-time security verification to the data stream. Observations suggest that the use of stream ciphers makes sense when dealing with unbounded data streams. So, they use cipher streams that have similar characteristics as stream data. As a result, encryption and compression reduce the transmission time from the IoT device to the collector, improving efficiency. T. Gomes et al. [52] studied the IPsec protocol header compression in 6 Low Power Area Network (6LoWPAN). IPsec ensures the integrity and confidentiality of the transport layer header. The authentication header (AH) and encapsulating security payload (ESP) of IPsec handle data integrity and authentication. The encapsulating security payload employs a cryptographic function to compress the data from 18 to 12 bytes, while the authentication header compresses the data from 24 to 16 bytes. AH saves 8 bytes, ESP with encryption saves 6 bytes and ESP with both authentication and encryption saves 6 bytes. However, ESP faces certain limitations due to the encryption of the upper layer header, which prevents compression.

P. Chang-Seop et al. [53] focus on DTLS (datagram transport layer security) header compression. DTLS is a security protocol that uses the constrained application protocol (CoAP). We further divide DTLS into two layers: the record layer and the handshake layer. We compress the record layer from 104 to 40 bytes and the handshake layer from 96 to 24 bytes. Consequently, the record layer preserves 64 bytes, while the handshake layer saves 72 bytes. L. Xingcheng et al.[54] concentrated on LDPC codes for compression, but they only addressed general data, not specific data and they didn't describe unattributed data in the decompression method. N. Ryo et al. [55] the author proposed a solution to the distributed source encoding and decoding problem. Distributed source encoding, such as sensor networks, is helpful in managing network traffic. The author proposed a block code that can be achieved at any point in the Slepian Wolf region, which is helpful to determine the region's lossless compression coding rate. M. Johnson et al. [56] studied encrypted data compression. Distributed source coding uses key-encrypted data as its source. At the recipient's end, the receiver simultaneously decompresses and decrypts the data using the received key and the encrypted data. For the experiment and simulation, they utilized fictitious data in the form of a binary image measuring 100*100, comprising 706 pixels with a value of 1, which they compressed at a high rate and then decompressed without any loss. A random key stream performs the encryption in this case using a bitwise XOR operation. P. Kishore et al.[57] we proposed resolution progressive compression, a type of lossless compression, to compress encrypted grayscale images more efficiently. When the image's resolution gradually increases, it uses the statistical value of a sampled low-resolution picture. The XOR operation encrypts the image and the key stream consists of bits. The IoT device transmits the data stream as described in Z. Guang et al.[58]. Among stream cipher and block cipher, selecting stream cipher has more advantages. Stream ciphers are known for their fast processing speed and in situations where resources are limited in the IoT stream, their simple structure can significantly improve efficiency. For devices that have limited resources, such as Grain, Trivium, MICKEY and WG-8, apply stream ciphers that are lightweight.

### 2.3.2 IoT Stream Data Analysis

This sub section presents research work related to IoT data stream analysis techniques utilized to improve the IoT system under following sub heading.

#### 2.3.2.1 Analysis and Prediction of Covid-19 Cases during Pandemic

R. C. Das et al.[59] presented a forecast model that considers seven countries, namely the USA, UK, Italy, Spain, France, China and India, for predicting the number of incidences. They have used the Box-Jenkins method for forecasting and the results show that the USA and India have the maximum number of cases. A.S. Fokas et al. [60] presented a numerical algorithm that can calculate the cumulative number of deaths. This is the result of a violation of the social distancing concept. N. Piovella et al. [61] suggested the susceptible exposed infections removed (SEIR) method, which gives a simple way to express the peak and asymptotic values. M. Youssoufa et al. [62] presented an overview of some already proposed methods used in studies, as well as a compendium of available open-source datasets for COVID-19.

#### 2.3.2.2 GAP LDDN for IoT Streamed Data Analysis

The related work is introduced, in this section, in two verticals. The first one is to review of attention-based object detection techniques. Second one provides state of the art overview of multitask learning methods in the similar domain. Over the past few years, a variety of deep learning approaches have been developed and put into use for diagnosing diseases [63–66]. One of the image processing techniques was used by S. Ramesh et al. [67] to reduce the dependency on the farmers to preserve the agricultural production. The authors proposed that paddy leaf diseases be identified and categorized using enhanced DNN and the Jaya algorithm. The RGB photos were converted into HSV images during prepossessing in order to remove the backdrop. Based on the saturation components, binary images were extracted to separate the sick and non-diseased area. Utilizing the Jaya optimization algorithm and an optimized deep neural network, diseases were classified. Transfer learning was used by N. Duong-Trung et al. [68] to categories the rice grain discoloration disease. The discoloration was viewed as posing a concern to the nations that produce rice. To identify strawberry verticillium wilt, X. Nie et al. [69] created a disease detection network based on Faster R-CNN and multi-task learning. According to the symptoms of the detected plant components, the strawberry verticillium wilt detection network (SVWDN) employed channel-attention techniques to extract features from the leaf diseases (i.e., young leaves and petioles). A grape leaf disease detection network (GLDDN) with dual attention methods for feature evaluation, detection and classification was proposed by R. Dwivedi et al. [70] since it recognizes and detects the diseased/infected regions, the testing conducted over a benchmark dataset at the evaluation stage suggests that disease detection networks may be more appropriate than the current approaches.

- **Object Detection Based on Attention Mechanism*:*** Stacking CNN is a technique used in object detection to find sick areas in leaves. In order to solve the picture classification challenges, Y. Sun et al. [71] developed a novel approach employing a genetic algorithm to evolve the topologies and connection weight initialization values of a deep convolutional neural network. F. Wang et al. [72] created the Residual Attention Network (RAN) employing an attention mechanism that can integrate a convolution neural network with cutting-edge feed forward network architecture in an end-to-end training method. The RAN was constructed by stacking attention modules that provide characteristics that are con-scions of attention. The feed-forward and feedback attention processes were combined into a single feed-forward process inside of each attention module using a bottom-up and top-down feed-forward structure. Additionally, Zhang et al. [73], introduced a unique attention-guided network that integrates multi-level contextual information in a progressive manner while being selective in its selection. By leveraging region proposals for detection and classification, R-CNN (Region-based Convolutional Neural Network) [74], significantly improved performance when compared to previous approaches. Later, faster R-CNN was used in conjunction with region proposal network (RPN) to recognize regions and objects, including their classes. A novel ROI-wise Reverse Reweight Network for road marking detection was proposed by X. Zhang et al. [75] and it achieved a 7.24% mAP improvement over the Faster R-CNN baseline. J. Mao et al. [76] combined additional features as a successful way to improve conventional pedestrian identification techniques. An innovative technique for recognizing pedestrians in poor lighting circumstances was put out by Xu et al. [77]. A novel framework for cross-modality learning was used in this work. For the purpose of identifying plant diseases, E. C. Too et al. [78], used four distinct deep CNN architectures: Inception-V4, VGG 16, ResNet and Dense-Net. DenseNet outperformed the competition despite having a high computational cost.
- **Multi-task Learning*:*** In computer vision, multi-task learning is a highly respected approach that is both popular and successful [79-80]. In a CNN image, R. Ranjan et al. [81] concurrently identified the gender, located landmarks, evaluated the position and detected the face. S. Ren et al. [82] used the same feature maps in Faster R-CNN to carry out bounding-box regression and classification simultaneously. A collection of intermediary auxiliary tasks for ultimate depth estimation and scene processing were predicted by Dan et al. [83]. On tomato leaves, H. Durmus et al. [84], employed the AlexNet and SqueezeNet Deep architectures. Additionally, it is stated in their studies that a robotic arm can detect

and remove sick leaves. Existing solutions detect whether there is disease on leaf or not, some authors also try to recognize after detection but suffers from over-fitting which generates sharp decline in their model's performance in real world. This motivates to develop a model that is able to detect diseases then recognize the disease of that particular plant also taken care of over-fitting, once this over-fitting is taken care, the proposed task becomes usable in industry and in real world scenario as well.

### 2.3.2.3 Bio Fuel Plant Analysis and Identification Using Machine Learning

Robert J. Henry proposed that the capability of plants to substitute fossil oil was estimated by studying the range of formation needed, the stretch of land required and the kinds of crops accessible. High yielding crops (50 tonnes/ha) that hold a high transformation efficiency (75%) would need a global area footprint of approximately 100 million ha to substitute current oil usage [85]. J. Wäldchen, et al. proposed that Classification of plants does not remain individually the part concerning botanists plus plant scientists. This is expected instead helpful for huge sections of society as well as the usual citizens. However, the identification of plants with traditional ways is time-taking and baffling for beginners. This generates a great hurdle for beginners engaged in receiving species experience. In current times, computer science study, primarily image processing also exemplary recognition methods, are found inside plant taxonomy which ultimately compensates toward insufficiency in human identification capacities [86]. S. Nie et al. proposed that though, current methods on inshore ship identification rely massively upon hand-made characteristics that require very complex method. Our paper introduces a modern technique to perform the task framed on Mask R-CNN by incorporating Soft-NMS within our skeleton which enhance some robustness [87]. W. Fang, et al. recommended a Mask R-CNN framed on smartphone detection model in specific RBC practice. Analyses show how our model diminishes smartphone scanning rate to one third. Therefore, that machine learning discovery method presents an intelligent alternative to enhance every user's experience within wireless power transmission concerning mobile also IoT gadgets [88]. Z. Emeršič et al. proposed that ear detection is a very important action in the process of ear recognition pipeline since it either makes or breaks the scheme. Though, there exists arguably in the literature about the shortage regarding ear detection approaches feasible. That postures a difficulty toward initiating ear recognition method to more extensive application and employment in commercial systems [89]. N Kumbasar et al. operate for the discovery of hangars within high-resolution airport satellite pictures was conducted employing Mask R-CNN algorithm. Though the discovery of structures inside the satellite pictures have been a general custom, being any of the hangars concealed in various dimensions create a problem concerning the discovery algorithms [90].

### 2.3.3 Identification of Research Gaps

- Until now, researchers have focused on the security and energy efficiency of IoT devices as separate subjects due to their competing ideas [32, 39, 40, 41]. IoT systems are currently using compressive sensing to simultaneously provide security and energy efficiency.
- In cases where the signal is not sparse, the existing approach does not apply. Moreover, the obtained compression ratio is limited and significantly lower than what recent state-of-the-art lossy compression algorithms can achieve.
- Existing work focuses on using a lossy error-bounded compression algorithm to reduce the power consumption of IoT devices, regardless of the characteristics of the input data.
- The literature study reveals that a majority of researchers utilize hardware-based encryption techniques to conserve the power of IoT devices, while others decrease compression by lowering overhead or header compression.
- Some researchers have applied compression techniques to high volumes of raw data. However, the proposed model caters to continuous time-series data in IoT applications, leveraging a low-cost operator to save data compression time and reducing the power consumption of an IoT device by minimizing data transmission time.
- Furthermore, previous research ignored the prior compression percentage, whereas the proposed model allows user preference-based compression for the input data set in advance.
- Many researchers have focused on data compression and network speed efficiency, yet they have overlooked the crucial aspect of power consumption minimization, particularly for battery-powered IoT devices.
- The majority of the researchers have used compression to optimize the stationary data; however, the proposed model deals with both stationary as well as real-time streamed data.
- Most studies focused on IoT data classification and analysis in agriculture problems using machine learning, but there is scope to handle and analyze large real-time IoT data using deep learning technology.

## 2.4 Concluding the Review

After reviewing various literature, it has been observed that there are problem in terms of IoT data transmission, security and power consumption in existing data stream handling system. Hence there is a scope for developing energy efficient and secure model or algorithms for handling huge data generate from IoT devices and transferred to cloud or server to enhance performance, efficiency and reliability.

An essential first step in the review process is defining the research questions. The main objective of this study is to present an overview of recent work on IoT data stream handling and analysis methods. Therefore, six research questions were defined. In this study, we examine the following research question (RQ) to facilitate a comprehensive analysis of IoT streaming data handling and analysis techniques:

RQ1. What is IoT stream data handling and analysis?

RQ2. What are the economic impacts of the increased application of IoT ?

RQ3. How does our everyday lives is affected IoT data?

RQ4. Is storing data over such a computing server model on the Internet through a computing provider a good option in IoT?

RQ5. What are the purpose of IoT data stream handling and analysis?

RQ6. How does IoT data handling algorithms optimize time, energy & classification constraint?

RQ7. What are the various techniques to deal with security issues of IoT data stream handling and analysis ?

This research aims to systematically and rigorously evaluate the literature on IoT stream data handling, analysis and security across various IoT fields within the scope of this study. As a result, we have developed the following research objectives (RO) and mapped them with the above research questions and references for this study.

RO1. To study different IoT data stream handling techniques.

RO2. To design algorithms to deal with IoT stream data reduction, compression and security issue.

RO3. To design algorithms to optimize time, energy or classification constraints in data stream handling on text/image dataset.

RO4. To develop the prediction model on real-time data stream to understand the future behaviour.

Further after the discussion of research study of various researcher and find research gap, prepared research question and research objectives of our research, the contributions of this review are as follows:

- We are examining the characteristics of IoT data and the prerequisites for conducting analytics. The IoT and data remain intricately interconnected. Statistics indicate that by 2030, the number of IoT-connected devices will reach around 30.73 trillion, primarily driven by data.
- Examine different methods of data management and analysis to assess both small and large data sets with diverse data characteristics, with the aim of deriving valuable findings and insights. The IoT refers to a network of interconnected objects, technologies, and people working together to accomplish a shared objective.
- It highlights the unique characteristics of IoT data for analytical purposes. Businesses often communicate these insights as trends, patterns and statistics to strategically use data for more efficient decision-making.
- The study the various challenges that arise in the management of streaming data. System and security architects, as well as data engineers, face several challenges in order to ensure the successful implementation and adoption of data streaming platforms.
- This study explores several concerns that have a negative impact on the platform's usability, operation, maintenance, and security, as well as the IoT data and devices.
- This study examines several methodologies for ensuring safe connectivity in the management and analysis of IoT data streams.
- Methodologies for managing and processing IoT stream data are being explored. The data streams sent to data collectors over a diverse network that includes the internet and low-speed IoT. Streaming data plays a crucial role in IoT applications. However, when a large data stream is transferred, the difference in speed and maximum transfer unit between the IoT and conventional Internet results in additional processing time and resources.

# CHAPTER 3

# PRELIMINARIES

This chapter gives a brief description of the prerequisites for the proposed IoT data stream handling and management architecture. It covers the technical details of the proposed structure for data transmission, such as data speed, compression, encryption, power consumption, security, data analysis and the application of IoT data handling, among others. The chapter also describes the algorithms utilized in the proposed architecture.

## 3.1 IoT Stream Data

Researcher have recommended the deployment of streamed data analysis (SDA) primarily on cloud platforms or high-performance computing systems. The SDA on such platforms is mainly based on incremental processing and data parallelism [91]. Data parallelism refers to the segregation of massive datasets into numerous smaller datasets for simultaneous execution of parallel analytics, while incremental processing refers to the quick processing of a limited set of information in a sequence of tasks. Although incremental processing and data parallelism reduce response time to the SDA architecture, they are not practical for time-critical IoT applications. However, implementing quick analytics on IoT components has additional obstacles, such as a lack of power, storage and computational resources at the data source.

## 3.2 Characteristics of IoT Stream Data

Stream data in the IoT represents inherently continuous, unidirectional and dynamic data flows, typically processed in a single-pass mode. Such a paradigm has favoured it with various common characteristics like timeliness, endlessness, volatility and randomness, as shown in Figure 3.1.

**1. Timeliness:** Ensuring the promptness of information processing requires an infrastructure capable of collecting, processing, transferring and displaying stream data instantaneously. As the value of information may disappear over time, the streaming platform or streaming architecture must execute all the communication and calculations on the fly with the information that has freshly arrived.

**2. Endlessness:** Freshly generated information will continuously append to the information channel as long as the stream-processing mechanism and data sources remain active. Therefore, processing such data (stream data) requires high-level availability to avoid possible data flow interruptions, which may result in the breach of real-time promises.

**3. Randomness:** Randomness is a direct result of the dynamic nature of stream data. There can be heterogeneous, unforeseeable factors that influence the processing sequence. For instance, the information generation procedure may introduce randomness due to the independent deployment of information sources in distinct conditions, making it impossible to guarantee the order of information arrival across distinct streams.

**4. Volatility:** The majority of the stream data will be deserted once such data finishes traversing through the employed stream processing system, thereby making the existence of information quite volatile.

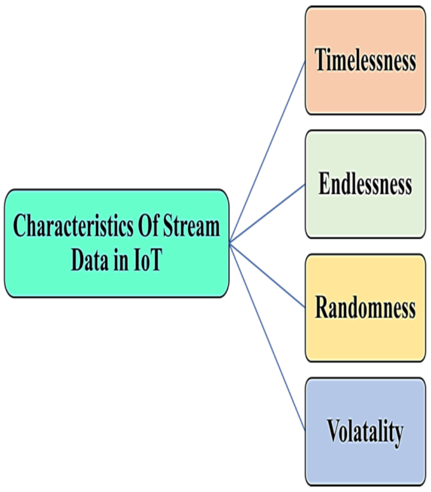


Figure 3.1: Characteristics of IoT stream Data [18]

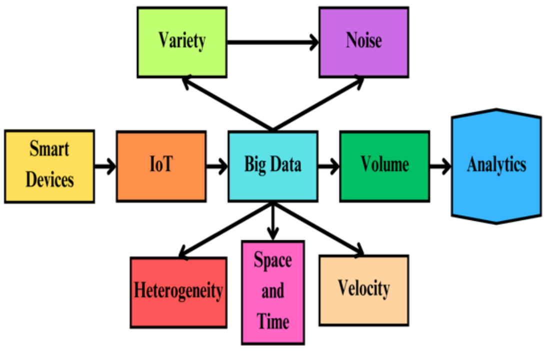
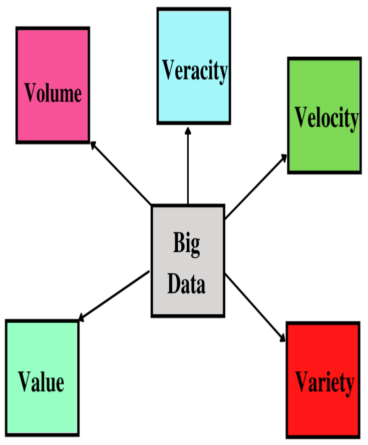


Figure 3.2: Relationship Between BDA and IoT [12]

## 3.3 IoT Big Data Characteristics

The Internet of Things (IoT) is considered the primary source of big data, utilizing a multitude of smart devices connected via the internet to regularly report on their behaviour in various environments. Figure 3.3 highlights the 5 Vs of Big Data. The fundamental use of BDA involves recognizing and extracting useful patterns from massive raw information, which enhances intuition for pattern forecasting and decision-making [92]. The insights and information extracted from big data are useful to several businesses, as it facilitates their profits.



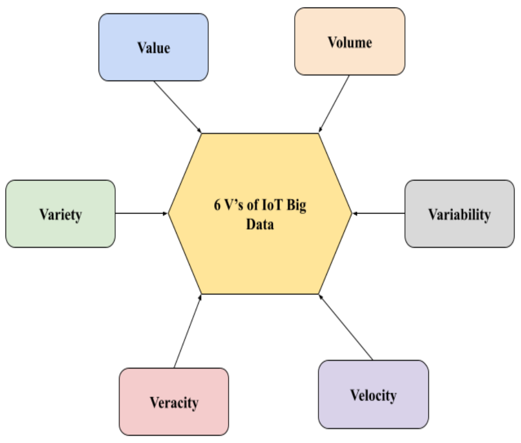


Figure 3.3: The 5V's of Big Data [18]    Figure 3.4: The 6V's of IoT Massive Data [14]

Figure 3.2, which adopts big data features, illustrates the relationship between big data and IoT. The Figure 3.4 illustrates the 6V's that characterize big data.

**1. Velocity:** There is enough big data in the IoT to process and improve output in real-time. As a result, enhanced analytics technologies and tools are required to work efficiently and at a high rate of data processing.

**2. Volume:** The amount of data in a database determines whether it is big data, traditional vast data or huge data. The volume of data created by IoT devices is significantly higher than it was previously.

**3. Variety:** Typically, big data can be available in diverse types and forms. It may be structured, unstructured or partially structured. Data types produced by IoT can have a broad variety, such as video, audio, sensory data, text data, etc.

**4. Veracity:** This attribute demonstrates the data's integrity, quality and consistency, resulting in efficient analytics. In IoT applications, especially those utilizing crowd-sensing data, the veracity factor deserves specific attention.

**5. Value:** This feature is concerned with the translation of big data into useful knowledge and intuitions that allow enterprises to earn profit. The value of the data is determined by the underlying services and processes, as well as the way the data is processed. For example, in one application, all sensor information may be required, whereas in another application, only random sensor samples may be required.

**6. Variability:** This characteristic indicates varying rates of information flow and load. Different information generating entities may have different data flows depending on the nature or kind of IoT applications. Furthermore, the information source may have different rates of data loading at different times. For example, during peak hours, a parking service application using IoT sensors or systems may have the highest information load.

Beyond the above-discussed features, researchers [93, 94] found that big data can be a footprint of IoT interplay or digital activity. Moreover, the scalability of such massive data systems is crucial, allowing for their extension to multiple databases. However, this attribute has the potential to increase the complexity of big data and introduce challenges such as transmitting and cleaning information. The BDA framework suggested to support the speed and volume of IoT data analytics, shows that SDA and integrating IoT big data are still unresolved issues that need more research [95]. Beyond BDA, IoT data concentrates on a novel form of analytics known as rapid and streaming data analytics (SDA), tailored to manage high-speed data stream applications that necessitate immediate responses. Without a doubt, fire prediction, health condition detection, automated monitoring devices and other applications require quick processing of incoming data and real-time decision-making to achieve their goals. Several researchers [96, 97] have developed frameworks and methodologies for quick SDA that take advantage of cloud services and infrastructures. However, with small-scale platforms like IoT, mist and edge computing, efficient analytics is required for these IoT applications. For example, with driverless vehicles, making quick judgments on driving activities such as speed or lane change is critical. Rapid analytics of likely multi-modal information from heterogeneous sources, including multiple vehicle sensors like cameras, LIDARs, radars, speedometers are necessary to enable such judgments. Traffic entities, such as traffic signs, traffic lights and communications or interactions between other vehicles play a crucial role in this process. In such conditions, transferring information to cloud systems for analysis and acquiring the response back may experience latency, which could lead to accidents or traffic violations. Recognizing pedestrians in those applications would be a critical condition. We desire precise recognition in real-time to prevent fatal accidents. As a result, these scenarios involve massive amounts of data for decision-making and prediction tasks, necessitating proper stream data handling mechanisms for appropriate operation. Streaming data is the term used to describe information that is generated in an ongoing manner by several sources. Sequential handling of such information and data necessitates the use of stream-processing methodologies. Data, often obtained or generated within a brief timeframe, necessitates rapid processing to extract immediate insights or make prompt assessments [98]. Processing streams, or complex event processing (CEP), refers to the analysis of a continuous data flow. Various IoT devices can continuously broadcast or collect the data stream they generate, serving as a general source of big data. Big data encompasses vast datasets that pose challenges in terms of storage, handling, management and analysis utilizing commonly used software and hardware platforms. These two procedures must be evaluated individually because they require different analytical approaches. While understanding and analyzing big data can take several days to develop, understanding and analyzing streaming data requires rapid activation within a few hundred milliseconds to seconds. For the development of widespread platforms relying on IoT data, it is crucial to prioritize information transmission and data fusion. Data fusion and information sharing are crucial in time-sensitive various IoT applications, where the quick integration of information is necessary to analyze and provide accurate and reliable actionable insights [99].

## 3.4 IoT Data Stream Handling Algorithms

This section presents algorithms utilized in the proposed IoT data stream handling and analysis models.

### 3.4.1 Compression and Decompression

Compression is a method of encoding input data with fewer bits than the original representation requires to reduce its size. We use decompression to recover the original data representation from compressed data [100]. Decompression involves reversing the compression process. Lossy and lossless compression are two classifications of data compression methods. Lossy compression methods decrease the data size by eliminating extraneous or unimportant information. This creates some data distortion. These methods are irreversible and use some inexact approximations to compress the data. These methods compromise the quality of the data, allowing only the reconstruction of an approximation of the original data. These methods achieve high compression ratios because of data loss. On the other hand, lossless compression methods reduce the data size without distorting it. Lossless compression does not affect the quality of the data. These techniques make it possible to fully recreate the original data from the compressed data. Because they are lossless, they only achieve a limited degree of compression.

- **Delta Encoding:** Delta encoding is an approach for storing or transmitting data in the form of delta. Most people use it for sequential data, not for complete files. It is also called delta compression. When nearby samples correlate, this reduces the variance of the data. The delta encoding technique converts the data set from fixed precision to integer, allowing only an integer format for transmission. We represented data elements with a 24-bit binary value and compressed them into 8 bits, with the choice of 8 or 24 bits depending on the application and other parameters.

  For the demonstration, consider the uncompressed data elements 225657 and 225698 represented in 24 bits or 3 bytes as $(225657)_{10} = (00110111000101111001)_2$ and $(225698)_{10} = (00110111000110100010)_2$ respectively. The compression algorithm sends two types of data: an uncompressed data element (which is the whole number as either 225657 or 225698) and a compressed data element, which is the difference between the current value and the next value. We send the first uncompressed element in 3 bytes, followed by the second element in 1 byte. Now, send the first data element, 225657, as an uncompressed data element and represented in 20 bits. We send the next data element, 225698 in decimal, as 41 (101001 in binary), representing the difference between the current and previous data elements. Therefore, the uncompressed data element, with a data length of 20 bits, represents a value in the range of $-2^{20}$ to $2^{20}$, while the difference between elements, with a compressed data length of 6 bits, represents a value in the range of $-2^{6}$ to $2^{6}$. We use the first bit of each byte as a flag bit to distinguish between the types of data we are sending. The flag value 0 represents the uncompressed data element and 1 represents compressed data. We use the second bit of the byte as a sign bit to represent a negative value. The value 0 represents positive data elements and 1 represents negative data elements. In the proposed work, we have also used two's complement representation to save this sign bit. Next, we exploit the linear property of the data elements, where the difference between two consecutive data elements is relatively smaller than the original data element. The proposed compression algorithm follows a similar approach, using a linear approach for data compression on IoT devices. In certain cases, the large difference between data elements may prevent the representation of an 8-bit signed integer. Then, we must transmit the uncompressed data elements as a 24-bit signed integer.
- **Fast Error Bounded Lossy Compression**: This work uses the SZ compression algorithm [101–103]. They are lossy and error-bounded compression algorithms. Simulation software has suggested this method to compress massive amounts of data. It can compress all data

types, including integers, floating-point numbers and doubles. However, this study considers floating-point data. For data compression, the SZ method relies on a prediction model. It searches for patterns and then uses those patterns to predict previously unknown data. It predicts new data points based on previously observed data. The algorithm's design enables the error to stay precisely within the defined range. This approach also enables a maximum compression ratio while keeping error bounds depending on the application. The absolute error bound value of 0.1 is considered in this research . We compute the absolute error using equation (3.1), which shows both the actual data value and the decompressed data value. As a result, any data value X after decompression should lie between X - 0.1 and X + 0.1. SZ algorithms are lossy error-bounded compression techniques. They use a prediction model to forecast new data based on previously observed data. Consequently, compression saves specific information about the prediction model rather than the original data. Using the prediction model, the algorithm can recreate an approximation to the initial data during decompression. The SZ compression method consists of three varieties: SZ 1.1 [104], SZ 1.4 [103] and SZ 2.1 [101]. The proposed work modifies the SZ 2.1 compression method to achieve improved compression.

$$\mathrm{Error} = |\mathrm{d_c} - \mathrm{d_o}| \tag{3.1}$$

- **Variable Length Encoding:** We discovered an uneven distribution in the quantization codes generated after quantization. We employ variable-length encoding to compress this non-uniform distribution. In variable-length encoding, we assigned a smaller code to more frequent symbols compared to less frequent ones. This method successfully reduces the overall length of the data. It is a lossless data compression method. For variable-length encoding, all three methods use Huffman encoding.
- **SZ 1.1 Compression:** Using several curve-fitting models, SZ 1.1 tries to predict new data based on previously observed data. The compression technique initially converts the multi-dimensional data array into a single-dimensional array. The method then uses the best fit curve fitting model for each data point, based on the previous three data points, to forecast the next data point. The method uses three models: the previous neighbour fitting model, the linear curve fitting model and the quadratic curve fitting model are the three models used by the method. In this model two-digit code replaces the predictable data. Equation (3.2) represents the preceding neighbour curve fitting model. Here, $\hat{p}_i^P$ denotes the $i^{th}$ predicted point and $\hat{p}_{i-1}$ denotes the final expected data point. The equation (3.3) represents the linear curve fitting model, where $\hat{p}_i^L$ represents the $i^{th}$ predicted point and $\hat{p}_{i-1}$ and $\hat{p}_{i-2}$

represents the last and the second last predicted data points and equation (3.4) represents the quadratic curve fitting model, where $\hat{p}_i^Q$ represents the $i^{th}$ predicted point and $\hat{p}_{i-1}$, $\hat{p}_{i-2}$ and $\hat{p}_{i-3}$ represents the last, the second last and the third last predicted data points.

$$\hat{p}_i^P = \hat{p}_{i-1} \qquad (3.2)$$

$$\hat{p}_i^L = 2 \times \hat{p}_{i-1} - \hat{p}_{i-2} \qquad (3.3)$$

$$\hat{p}_i^Q = 3 \times \hat{p}_{i-1} - 3 \times \hat{p}_{i-2} + \hat{p}_{i-3} \qquad (3.4)$$

- **SZ 1.4 Compression:** SZ 1.4 [102] employs a multidimensional prediction model to forecast new data based on previously observed data. When a data point exhibits predictability, the system saves some metadata, rather than the data point itself, for recovery after decompression. Equation (3.5) gives the general formula of the n-layer prediction model. We denote the number of dimensions as d, the initial value of the data point as g(i,j), and the predicted value for the point as f(i,j). Finally, we apply variable-length encoding to further enhance compression. Additionally, SZ 1.4 uses adaptive error-controlled quantization to keep the predicted points below a set error limit. This lets for the highest compression ratio while still ensuring an error bound based on the application. A higher n may lead to a more precise prediction, resulting in greater compression quality. A good choice will result in a good compression ratio, reduced compression error and faster compression.

$$f(a_1, \ldots, a_d) = \sum_{0 \le b_{1,\ldots,} b_d \le n}^{(b_1,\ldots,b_d) \neq (0,\ldots,0)} - \prod_{j=1}^{d} (-1)^{b_j} \binom{n}{b_j} . g(a_1\ b_1, \ldots, a_d - b_d) \qquad (3.5)$$

- **SZ 2.1 Compression :** SZ 2.1 [1] uses three different prediction models to predict the data points, namely the Lorenzo Prediction Model, the Mean Integrated Lorenzo Prediction Model [101] and the Linear Regression Model. In SZ 2.1, the algorithm divides the input data into blocks of equal size. Then, for each block, it tries to find out the best model for predicting the data. Each data block dynamically determines the best prediction model. We then use the prediction model to obtain the predicted values, followed by performing linear-scaling quantization and variable-length encoding. IEEE 754 binary representation analysis compresses the regression coefficients for blocks using the linear regression model [103]. In the end, the algorithm compresses the resulting stream using the zstd lossless compression algorithm .

- **Compression using 754 Binary Representation Analysis**: IEEE 754 binary representation analysis compresses the unpredictable data. This is a three-step process. First, we map all the unpredictable data to a smaller range by subtracting the range median value from all the values. We now refer to this data as normalized data. The data will be closer to 0 and it requires fewer mantissa bits to meet the specified precision. Secondly, we curtail the value by ignoring the insignificant mantissa part, in line with the user-specified error bounds. Third, we utilize the leading zero-based floating-point compression to further minimize the storage space. Essentially, we compress each point using a leading zero count and the remaining significant bits from the XOR of the subsequent normalized values. With the aid of three separate prediction models, SZ 2.1 forecasts data points using three distinct prediction models: the Lorenzo prediction model [104], the mean integrated Lorenzo prediction model and the linear regression model. The SZ 2.1 method splits the incoming data into equal-sized blocks. Then it attempts to identify the best model for predicting the data for each block. We dynamically determine the optimal model for prediction for each data block. We then use the prediction model to obtain the predicted values, followed by linear-scaling quantization and variable-length encoding. IEEE 754 binary representation analysis compresses the regression coefficients for blocks using the linear regression model. Finally, the method uses the zstd lossless compression algorithm [105] to compress the output stream. The Lorenzo model is a prediction model that uses adjacent points values to predict a points value. Equation (3.5) represents it as a special case of the multidimensional prediction model, with the value of n set to 1. For each block, it selects the best prediction model dynamically based on the cost function between the best Lorenzo predictor, L-predictor and linear regression predictor. The cost function for linear regression model, Lorenzo model, and mean integrated Lorenzo model is given by equation (3.6), (3.7) and (3.8) respectively, where $p_i$ is the predicted value, $o_i$ is the original value, Y is the sample size and E is the error bound and mean is the mean calculated for the Mean Integrated Lorenzo model.

$$E_r = \sum_{i \in S} |p_i - o_i| \tag{3.6}$$

$$E_r = \sum_{i \in S} |p_i - o_i| + Y * 1.22 * E \tag{3.7}$$

$$E_r = \sum_{i \in S} \min(|p_i - o_i| + Y * 1.22 * E, |mean - o_i|) \tag{3.8}$$

## 3.4.2 Encryption and Decryption

The process of encryption must encode data to prevent access by anyone other than the intended recipient. Decryption is the process of breaking an encryption to recover the original data. The plaintext is the original data, whereas the ciphertext is the encrypted data. For encrypting and decrypting data, a pseudo-random key is required, called the encryption key. To access the plaintext, only the authorized entity with a valid key may decode the ciphertext. The study employs the following algorithms to maintain security:

- **ChaCha20 Encryption Algorithm**: This algorithm is part of the family of stream ciphers specifically designed for the software platform. It supports the trade-offs between performance and security by allowing for different rounds, nonce, key and counter lengths. This stream cipher can design as a concentration of the Salsa20 stream cipher, ensuring security and high throughput without compromising performance on various software platforms.
- **Poly1305 Message Authentication Algorithm:** Daniel J. Bernstein developed the message authentication code Poly1305. It use to validate the message validity and integrity. The algorithm creates a MAC from the message. For key expansion, the Poly1305 architecture uses AES encryption. Poly1305-AES uses two 128-bit keys and a 128-bit nonce to calculate a 128-bit MAC from a variable-length message. The algorithm processes the message in 16-byte chunks.
- **ChaCha20-Poly1305 for IoT Data Encryption**: The encryption algorithm often pair ChaCha20 with the Poly1305 authentication algorithm to authenticate encrypted communications. People often pair ChaCha20 with the Poly1305 authentication algorithm. The ChaCha20 algorithm is used to encrypt communications and generate keys for the Poly1305 method. The Poly1305 algorithm generates the message authentication code. Their combination forms the ChaCha20-Poly1305 authenticated encryption with associated data, or AEAD algorithm. This work combines the Poly1305 message authentication technique with the 12-round ChaCha symmetric key stream cipher. This architecture guarantees the integrity, validity and confidentiality of data exchanged over the network.
- **Stream Cipher Encryption:** A stream cipher [106], as shown in Figure 3.5, is essentially a symmetric key cipher. The pseudo-random cipher digit stream combines the plain text digits. In this cipher technique, the equivalent digit from the key stream encrypts each plain text digit, generating a cipher text stream digit that uses the input as an initialization vector (IV) and key. Stream ciphers are typically faster than other cipher techniques, with certain

restrictions. As this cipher works only on a few bits at a time, it also indicates memory efficiency.

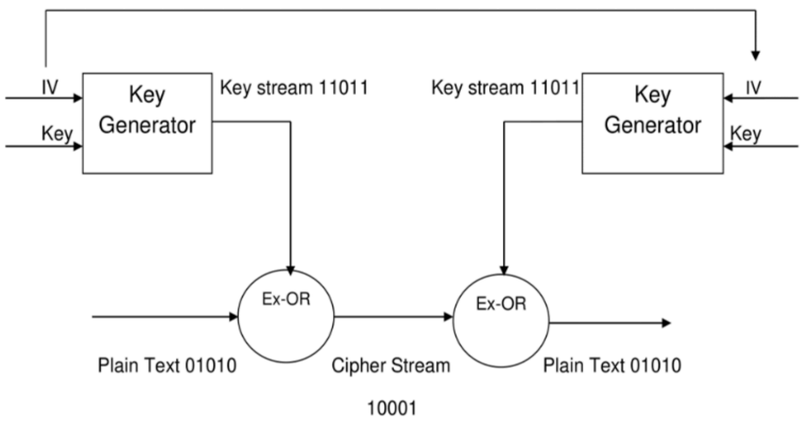


Figure 3.5: Stream Cipher Encryption

This technique works best when the amount of data is continuous, such as in network streams and the cost of implementing it is quite low compared to other cipher techniques. Stream ciphers are generally best for cases where the amount of data is either continuous or unknown, such as network streams. X. Fan et al. [107] applied lightweight stream ciphers such as WG-8 and Grain to devices with limited resources.

# CHAPTER 4

# IoT DATA STREAM HANDLING

Chapter 2 observes that traditional data stream handling systems face various issues, including data transmission, data loss, power consumption, and security. This chapter proposes IoT stream data handling systems using compression and encryption to ensure improved energy efficiency and data security while several IoT devices communicate with each other. The proposed models implement various stream data handling algorithms to ensure energy efficiency and security while transferring IoT data to the cloud. The experimental results, analysis, and performance evaluation show that our proposed model provides a feasible and reliable IoT environment. We have proposed the following IoT data-steam handling models.

## 4.1 Secure Energy-Efficient Novel SZ 2.1 Hybrid Model

The study investigates an improved and more efficient data compression method than the existing state-of-the-art methods to improve the compression ratio while maintaining data quality and reducing network load.

### 4.1.1 Overview

IoT devices generate massive amounts of sensitive data, which they transfer tremendously to the cloud for processing and decision-making. The most significant issues that need to be solved for IoT devices are improving energy efficiency and guaranteeing data security while several devices are connected. In this work, for edge computing, a hybrid algorithm is proposed that uses compression and encryption in the same manner to improve efficiency in terms of energy and data security on IoT devices. This hybrid architecture uses the authenticated encryption with associated data (AEAD) ChaCha12-Poly1305 algorithm and improved SZ 2.1 compression. We maintain confidentiality, integrity and authentication while sending the data to the edge. We conducted several experiments using the driver stress detection dataset [108,109] and the gas-sensor dataset [110] for home activity monitoring to validate this approach. We compare the performance of the proposed improved SZ 2.1 compression algorithm with five baseline algorithms: SZ 1.4, original SZ 2.1, selective compression, algorithm-based fault tolerance (ABFT) and digit rounding algorithms. The key parameters used in the experiment to measure the performance of the proposed algorithm are data reduction, compression ratio, power consumption, encryption time, total processing time and error. By combining the improved SZ 2.1 compression algorithm with the ChaCha12-Poly1305 AEAD algorithm, we enhance the device's battery life by approximately 10% and ensure the security of data disseminated to the Edge. For both datasets, the proposed secure hybrid model reduces both encryption and overall processing time by 95% and 98%, respectively. The motivation of this study is that IoT devices are yet to be used in common life because they are still lacking in battery and data management steps during the transmission of real-time data streams, which makes room to trade off data and battery usage over the IoT device. The proposed work employs a secure and energy-efficient algorithm to maintain a trade-off between security and performance in an IoT-based system. Therefore, the proposed work's novelty lies in managing both algorithms to ensure complete data transfer for any IoT-based system. Based on the motivation mentioned above and the novelty proposed in the improved SZ2.1, the study have the following contributions:

1. We conducted an analysis of the proposed compression method's speed for various input sizes. It investigates current data compression methods that may be implemented on IoT devices to decrease the quantity of data transmitted to the edge device, thus decreasing the consumption of energy on IoT devices.
2. For authenticated encryption with associated data (AEAD), the ChaCha12-Pol1305 encryption algorithm used intermediate-level security. As the number of rounds increases, so does the amount of time required by the algorithm. Hence, the ChaCha8-Poly1305 algorithm requires the least amount of time, ChaCha20-Poly1305 requires the maximum amount of time and ChaCha12-Poly1305 requires an intermediate amount of time. However, the greater the number of rounds, higher the algorithm's security. We maintain a trade-off between security and performance by selecting the 12-round variant.
3. The proposed work contributes to the trade-off between power and security steps involved in data transfer management on IoT devices.
4. The system employs compression and encryption algorithms to reduce data, which not only decreases transmission time but also boosts battery power consumption. On the other hand,

fewer security rounds result in shorter encryption timelines, which directly affect battery power consumption because of the simpler algorithm.

5. By reducing rounds in the chacha encryption algorithm to make the system more energy efficient, it can save the battery of an IoT device. The improved SZ2.1 algorithm for compression also saves data transfer time, which reduces battery life.
6. We looked at how different versions of the existing compression method performed in terms of data reduction, power consumption, encryption time, total processing time, and error. This was done to see what happened when we combined the compression and encryption algorithms and how that affected the device's energy use and network latency.

## 4.1.2 Proposed Work

IoT devices consume energy during data transmission, which also degrades the battery's life. Therefore, this section proposes the following work to improve both the battery life and security of IoT devices.

### 4.1.2.1 System Model

A hybrid SZ and ChaCha12-Poly1305-based secure energy-efficient model is presented in this study. We compare the work with current SZ 1.4, original SZ 2.1, selective compression, algorithm-based fault tolerance (ABFT), digit rounding compression algorithms and the version of the ChaCha-Poly1305 algorithm to enhance the security and energy efficiency of IoT devices. The performance of the three compression algorithms is compared in terms of compression and speed. Similarly, the ChaCha12-Poly1305 technique is compared to the inferior ChaCha8-Poly1305 and ChaCha20-Poly1305 algorithms, which are used more frequently. The block diagram of the procedure on the client device is shown in Figure 4.1. The client device first gathers the data and then applies the SZ compression method. The ChaCha12-Poly1305 algorithm uses the compressed data, any additional associated data and a nonce N. It generates both the ciphertext and the hash. The message payload transmits the ciphertext, hash, nonce and any other related data to the edge device. While Figure 4.2 depicts the process block diagram on the Edge device, We compute the hash after extracting the ciphertext, additional associated data and nonce N from the message payload. We authenticate the message and proceed if the generated hash matches the hash in the message payload. If there is a mismatch, we may deny the message and take precautionary measures. We use the ChaCha12 algorithm to decode the ciphertext into plaintext after authenticating the communication. The SZ method decompresses the encrypted data to retrieve the original data.

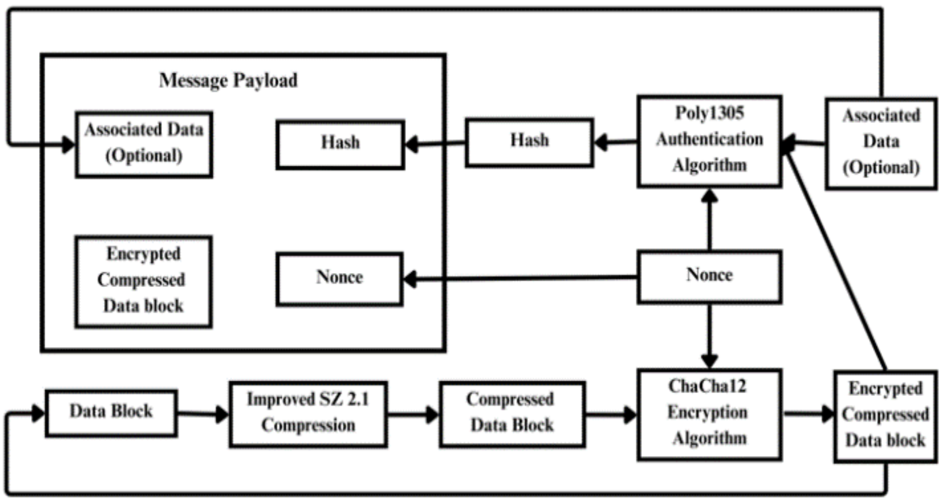

Message Payload
Associated Data (Optional)
Hash
Hash
Poly1305 Authentication Algorithm
Associated Data (Optional)
Encrypted Compressed Data block
Nonce
Nonce
Data Block
Improved SZ 2.1 Compression
Compressed Data Block
ChaCha12 Encryption Algorithm
Encrypted Compressed Data block

Figure 4.1: Client (IoT Device)

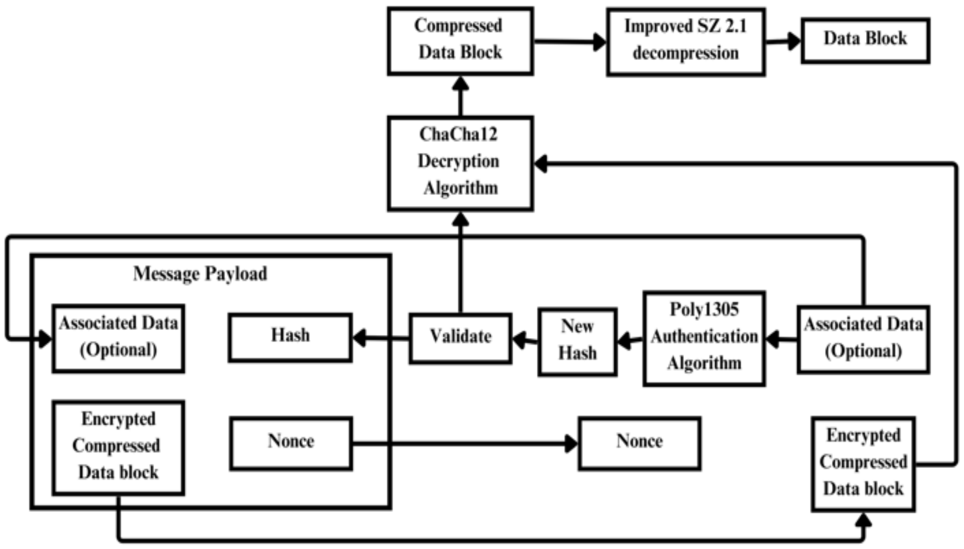

Compressed Data Block
Improved SZ 2.1 decompression
Data Block
ChaCha12 Decryption Algorithm
Message Payload
Associated Data (Optional)
Hash
Validate
New Hash
Poly1305 Authentication Algorithm
Associated Data (Optional)
Encrypted Compressed Data block
Nonce
Nonce
Encrypted Compressed Data block

Figure 4.2 : Server (IoT Edge Device)

Algorithm 4.1 describes the proposed hybrid algorithm. The algorithm's error bound E, key K, time period P and SZ compression algorithm version V are all inputs. The algorithm initially needs P days to collect data from sensors. Next, the algorithm levels the gathered data into a one-dimensional array and applies the SZ compression method. We do not need to protect or encrypt any additional associated data that we must send in addition to the gathered data. Furthermore, the system generates a nonce N. Each invocation of a given key must be distinct. Finally, we applied the ChaCha12-Poly1305 algorithm, which created the ciphertext and hash, to the compressed data CD, key K, nonce N and additional related data AAD. However, as the SZ technique uses lossy compression, the decompressed data will contain mistakes up to the set error limit. The edge device could examine the decompressed data before transmitting it to the cloud for archival and further processing.

**Algorithm 4.1: Proposed Hybrid Algorithm**

*Improved SZ2.1*

**Input:** Error_BoundE, Key K, Period P, SZ Compression Algorithm Version V
**Output:** Ciphertext, Hash

```
1. Begin
2.  while Energy > 0 and Sensor_Status = active do
3.  data = data collected for time period P
4.    input_data = flatten(data)
5.    CD = SZ_Compression (input_data, E, V)   // Collect ADD (additional associated
   data) to Generate Nonce N//
6.    Ciphertext, Hash = Chacha12_poly1305_AEAD (CD, K, N, AAD)
7.    transmit_data (Ciphertext, Hash, N, AAD)
8.  End
9. End
```

It is believed that the IoT device is constantly collecting data. After a P-second delay, the device transmits the data to the edge node. Before applying the SZ compression method, the device levels the collected data. Once compressed, we have used the ChaCha12-Poly1305 technique to protect the data. We first encrypt the data using the ChaCha12 encryption algorithm. The Poly1305 authentication technique, which creates the hash for the input, sends the encrypted data together with any optional related data. The edge node then receives the hash, encrypted data and any accompanying information. To restore the original data, the edge node can authenticate, decrypt and decompress it.

### 4.1.2.2 SZ Compression

This section describes the logic and algorithms of the proposed architecture. After lossy compression, SZ 2.1 compresses the data using the zstd lossless compression algorithm, achieving even more compression. The proposed improved SZ 2.1 replaces SZ 2.1 Burrows-wheeler transform (BWT) compression algorithm. Bzip2, a free and open-source lossless compression software, utilizes the BWT technique [111]. When compared to the LZW and

Deflate algorithms, bzip2 achieves a significantly greater compression ratio. To achieve high compression, bzip2 uses the burrows-Wheeler transform, run-length encoding, move-to-front converted and Huffman encoding techniques. By sorting the data, the Burrows-Wheeler transform helps with data compression. Run-length encoding and Huffman encoding can easily compress the series of symbols produced. Algorithm 4.2 represents the improved SZ compression algorithm. We calculate the densest location v and frequency f1 by sampling X points, where X is the total number of points in D. We determine the densest frequency, f2, by sampling 1% of the data. We select the mean integrated Lorenzo predictor as the optimal Lorenzo predictor for the data if f1 > f2. If not, we compute the mean value using the original Lorenzo predictor. Next, we compute the regression coefficients for each block. This method eventually divides the data into chunks. To generate a sample dataset S for the block, Y points are sampled. S determines the cost of both prediction models. Based on the cost function, it dynamically chooses the best prediction model among the best Lorenzo predictors, L-predictors and linear regression predictors. To generate predictions, the system uses the best-fit prediction model. The next step involves quantization through linear scaling and variable-length encoding. Finally, we further compress the data using bzip2. Algorithm 4.3 represents the bzip2 compression. It begins by dividing the data into blocks. Then it conducts a series of actions on each block to compress the data. The data is encoded using a run length. The most crucial step in bzip2 compression, the Burrows-wheeler transformation, comes next. Following this, apply run-length encoding, huffman and delta encoding to generate compressed data CD.

**Algorithm 4.2: SZ Compression**

*SZ_Compression*

**Input:** Data_D, Error_Bound E

**Output:** CD (Compressed Data)

```
1. Begin
2.        v,f_1= estimate_densest_position_and_frequency (D, E)
3.        f_2= estimate_densest_frequency_Lorenzo (D, E)
4.        Iff_1>f_2 then
5.            L-Predictor = Mean Integrated Lorenzo Prediction Model
6.            mean = compute_mean (D, v, E)
7.        Else
8.            L-Predictor = Lorenzo Prediction Model
```

```
9.          End
10.         coefficients= compute_linear_regression_coefficients(D)
11.         for each block in D do
12.             S = sample_points (D, Y)
13.             E_l= compute_cost (S, L-Predictor)
14.             E_r= compute_cost (S, Linear Regression Model)
15.             If E_r<E_l then
16.                 temp_block = compress (block, Linear Regression Model)
17.             Else
18.             temp_block = compress (block, L-Predictor)
19.             End
20.             temp += temp_block
21.          End
22.          temp += compress_IEEE_binary_representation_analysis (coefficients)
23.          CD = compress_bzip2(temp)
24.  End
```

**Algorithm 4.3: bzip2 Compression**

*bzip2_compression*

**Input:** Data_D
**Output:** Compressed_Data CD

```
1.  Begin
2.          for each block in D do
3.              temp_block = run_length_encode(block)
4.              temp_block = burrow_wheeler_transform(temp_block)
5.              temp_block = move_to_front(temp_block)
6.              temp_block = run_length_encode(temp_block)
7.              temp_block = huffman_encode(temp_block)
8.              temp_block = delta_encode(temp_block)
9.              temp_block += append_symbol_array(temp_block)
10.             CD+=temp_block
11.         End
12. End
```

### 4.1.2.3 Chacha Poly1305 Encryption

This sub-section analyzes the security features of the proposed architecture to address security threats. ChaCha12 is the twelve-round variant of Daniel J. Bernstein's popular ChaCha20 stream cipher. Instead of 20 rounds of computation, it takes 12 rounds. It is like the 20-round variant in all other aspects. Reducing the number of rounds speeds up the method. However, a smaller number of rounds implies degradation in terms of security.

**Algorithm 4.4: ChaCha12-Poly1305-AEAD-Algorithm**

*Chacha12_poly1305_AEAD*

**Input:** Plaintext P, Key K, Nonce N, Additional Associated Data (AAD)

**Output:** Ciphertext, Hash

```
1. Begin
2.         OTK= generate_one_time_key (K, N)
3.         Ciphertext = ChaCha12_Encrypt (P, K, N)
4.         M = AAD + pad16(AAD)
5.         M += Ciphertext + pad16(Ciphertext)
6.         M += AAD_length_in_bytes
7.         M += Ciphertext_length_in_bytes
8.         Hash = Poly_1305(M, OTK)
9. End
```

**Algorithm 4.5: ChaCha12-Encryption-Algorithm**

*ChaCha12-Encrypt*

**Input:** Plaintext P, Key K, Counter C=1, Nonce N

**Output:** Ciphertext

```
1. Begin
2.     constants = [0x61707865, 0x3320646e, 0x79622d32, 0x6b206574]
3.     roundSchedule = [(0, 4, 8, 12), (1, 5, 9, 13), (2, 6, 10, 14), (3, 7, 11, 15), (0, 5, 10,
   15), (1, 6, 11, 12), (2, 7, 8, 13), (3, 4, 9, 14)]
4.     Ciphertext = []
5.     for index, block in enumerate (P [index: index+64] for index in range (0,
   length(P), 64) do
```

```
6.              state = constants + K + [C] + N
7.              workingState = state
8.              for rounds=1…3 do
9.                  for a, b, c, d in roundSchedule do
10.                     x_a = workingState [a]
11.                     x_b = workingState [b]
12.                     x_c = workingState [c]
13.                     x_d = workingState [d]
14.                     x_a = (x_a + x_b) & 0xffffffff
15.                     x_d = x_d ^ x_a
16.                     x_d = ((x_d << 16) & 0xffffffff | (x_d >> 16))
17.                     x_c = (x_c + x_d) & 0xffffffff
18.                     x_b = x_b ^ x_c
19.                     x_b = ((x_b << 12) & 0xffffffff | (x_b >> 20))
20.                     x_a = (x_a + x_b) & 0xffffffff
21.                     x_d = x_d ^ x_a
22.                     x_d = ((x_d << 8) & 0xffffffff | (x_d >> 24))
23.                     x_c = (x_c + x_d) & 0xffffffff
24.                     x_b = x_b ^ x_c
25.                     x_b = ((x_b << 7) & 0xffffffff | (x_b >> 25))
26.                     workingState [a] = x_a
27.                     workingState [b] = x_b
28.                     workingState [c] = x_c
29.                     workingState [d] = x_d
30.                 End
31.             End
32.             keyStream = [(s + w) & 0xffffffff for s, w in zip (state, workingState)]
33.             Ciphertext += [x ^ y for x, y in zip (keyStream, block)]
34.         End
35. End
```

**Algorithm 4.6: Poly1305-Algorithm**

*poly1305_AEAD*

**Input:** Message M , Key K,

**Output:** Hash

```
1.  Begin
2.          P = 0x3fffffffffffffffffffffffffffffffb
3.          accumulator = 0
4.          set r to first 16 bytes of K and read as a little-endian number
5.          clamp r by setting r to r & 0x0ffffffc0ffffffc0ffffffc0fffffff
6.          set s to the last 16 bytes of K and read as a little-endian number
7.          for each 16-byte block in M do
8.              set n to block + b'\x01' and read as a little-endian number
9.              accumulator += n
10.             accumulator = (r * accumulator) % P
11.             accumulator += s
12.         End
13.         set Hash to the 16 least significant bytes from the accumulator read as a little-
    endian number
14. End
```

This work proposes the ChaCha12-Poly1305-AEAD algorithm. This algorithm combines the Poly1305 message authentication code and the ChaCha12 encryption algorithm. Algorithm 4.4 represents the ChaCha12-Poly1305 algorithm. Here, we first combine the key K and the nonce N to create a one-time key. We padded the AAD and ciphertext until they were multiples of 16 before concatenation. Finally, calculate the hash value using the Poly1305 algorithm. We selected these two algorithms due to their portability and potential applications on IoT devices. Particularly, the CPU-friendly Add-Rotate-XOR (ARX) operations used by the ChaCha12 encryption algorithm make it simple to operate on devices with limited resources. Algorithm 4.5 represents the ChaCha12 encryption. The ChaCha12 algorithm accepts the plaintext P, key K, nonce N and a counter with the default value of 1 as input. It begins by initializing some constants, then divides the input plaintext P into blocks of 64 bytes. The size of the last block can be less than 64 bytes. For each block, it creates a state variable from the constants, the key K, the counter C and the nonce N, then creates a copy of the state variable and performs three double rounds. In each double round, it performs some Add-Rotate-XOR operations. Finally, it constructs a keystream by combining the working state variable with the state variable. This keystream is XORed with the block to obtain the cipher text. Daniel J. Bernstein developed the message authentication code Poly 1305, represented by Algorithm 4.6. The message's validity and integrity are validated by Algorithm 4.6. It accepts the message M and the key K as inputs. It starts with initializing certain constants. After generating two variables, r and s, using the key, it adds b'x01' to each 16-byte block in M and reads it as a little-endian integer. This value is append this value to the accumulator variable and then use r and s to update the accumulator. Once we process all the blocks, we calculate the hash value by reading the 16 least significant bytes of the accumulator as a little-endian integer.

#### 4.1.2.4 Example of Chacha12-Poly1305 Algorithm

Instead of 20 rounds, the Poly1305 encryption Algorithm 4.6 is performed using just 12 rounds. Although a larger number of rounds indicated better cryptographic security, [110] provides research demonstrating an attack against seven-round versions of the ChaCha algorithm[112]. Despite the lack of a known attack on the ChaCha algorithm's 8-round version, the selection of the 12-round variant was based on its larger security gap. The 20-round version would improve security while also increasing computing complexity. Table 4.1 shows the results achieved while using the ChaCha12-Poly1305 method as an example.

Table 4.1 Results Achieved While using the ChaCha12-Poly1305

| **Variable** | **Value (in Hexadecimal)** |
|---|---|
| Key | *2c1f5718c0175025f56ce7622d8016607751f d1a988a2e163c0e6c90802ead32* |
| Nonce | *69cc92e6bda 9099bb1d4ccfc* |
| Plaintext | *74657374706c61696e74657874* |
| Initial State | *657870616e642033322d62797465206b2c1f5718c0175025f56ce7622d801660 7751fd1a988a2e163c0e6c90802ead320100000069cc92e6bda9099bb1d4ccfc* |
| Key Stream | *88ca722a8ebfe2f032432d55c5ea4f3103f97b3ae9de0bc4d20e1f8272dd3490 d05b4998700d336178fe79bf3d17f0144b5a351c799c47cee10df52716d66f7d* |
| Ciphertext | *fcaf015efed38 3995c37482db1* |
| One Time Key | *fd7185f331cb6a8239a21b56ebe94c0704061847703c94cd3687d29902df 5b13* |

| | |
|---|---|
| AAD | *74657374686561646 572* |
| Hash | *8a8db0c937371c28f43165fb0196b674* |

### 4.1.3 Experimental Analysis

This sub-section presents the details of the proposed model experimental setup, evaluates system performance, hardware and dataset and compares the proposed work with existing techniques.

We conducted the experiment to verify the effectiveness of SZ 1.4, the original SZ 2.1, selective compression, algorithm-based fault tolerance (ABFT), digit rounding algorithms and the improved SZ 2.1 compression algorithms in combination with the ChaCha12-Poly1305 algorithm. The Asus X541UA and Dell Inspiron 15R 5537 systems were used for various SZ compression and CHACHA8-POLY1305, CHACHA12-POLY1305 and CHACHA20-POLY1305 encryption algorithms, respectively, whose specifications are mentioned in Table 4.2. Experiments were performed on these datasets to study the application of the suggested protocol and its effects on IoT devices. The effect of larger input data sizes on SZ compression techniques and the effect of an increase in the size of input data on the ChaCha8-Poly1305, ChaCha12-Poly1305 and ChaCha20-Poly1305 algorithms. A client and server program for this purpose were implemented. Those experiments were conducted as mentioned in [113].

- **Client IoT Program:** We deployed this program on a Raspberry Pi. The Python programming language represents the IoT client program. The client IoT device deployed the existing compression algorithms and the improved SZ 2.1 with the AEAD algorithm ChaCha12-Poly1305.
- **Server IoT Program:** We deployed this program on the Asus-X541UA laptop running Ubuntu 18.04.03 LTS. We deployed the existing compression algorithms, the improved SZ 2.1 and the AEAD algorithm ChaCha12Poly1305 on the edge device. Table 4.2 lists the two devices specifications.

Table 4.2: Device Specification

| **Specification** | **Raspberry Pi 4B** | **Asus X541UA** |
|---|---|---|
| OS | Ubuntu-18.04.3 LTS | Ubuntu-18.04.3 LTS |
| CPU | ARM-CORTEX-A72 BCM2711 | Intel Core i3 7100U CPU 2.4 GHz |
| RAM | 4 GB | 4 GB |
| Storage | 64 GB | 64 GB |

The Raspberry Pi 4B with power (5V) and USB-C  processor ARM-CORTEX-A72 BCM2711 was used for the experiment as a client device. We assume that the client and server have previously established the shared secret key for the encryption algorithm through a secure key exchange mechanism. We break up the data into blocks of 2976,000 bytes in size. After one minute, the edge device receives and processes a data block. If the client device encrypts the data, the edge device must first authenticate the data using the tag or hash it received. Once authenticated, the edge device decrypts the data. If the client device compresses the data, the edge device will decompress it. The edge device will then proceed to process the data. Subsequently, we may repeat the data processing or upload it to the cloud. Four hours, or 241 intervals, of this process are repeated.

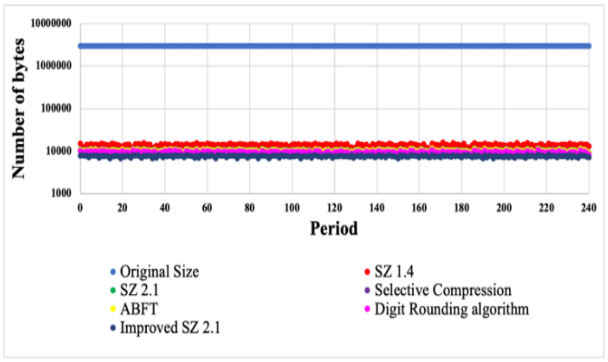
- **Datasets:** The experiment analysis uses two datasets: the drivers stress detection dataset [114,115] and the gas sensors for home activity monitoring dataset [110]. The first dataset contains a variety of signals collected from individuals while doing real-world driving activities. We collected the sample at a rate of 496 Hz. Vehicles in the Greater Boston region followed a predetermined path as they captured the data. In three segments—rest, city and highway—each driving job lasts at least 50 minutes. The term "rest" refers to the "low stress" intervals of rest at the start and end of a driving task. City is the term "city" refers to the time when the driving task takes place within a city, potentially exposing the driver to a variety of traffic referred to these as periods of "high stress." The highway periods encompass the duration of travel. We designated these as "moderate stress" periods. The drivers themselves validate this data labelling by using self-reporting questionnaires. We used five psychological signals to identify stress. The five psychological signals we used to identify stress were the ECG (electrocardiogram), HR (heart rate), GSR (galvanic skin response) of the hand, GSR (galvanic skin response) of the foot and RR (respiration rate). We store the data in the WFDB (waveform database) format, pre-processing it into a 32-bit binary file for further analysis. The second gas sensor home activity monitoring dataset consists of 100 recordings, which are an array of eight MOX gas sensors, a humidity sensor and a temperature sensor. We expose the sensor array to two different stimuli: wine and banana. The duration of each stimulation ranges from seven minutes to two hours, with an average of 42 minutes. There are 36 wine readings, 33 banana readings and 31 background activity readings. Wireless communication issues cause minor variations in the readings taken at one sample per second. We only use the temperature sensor readings from this dataset. We conducted the experiments using data blocks of sizes 100 KB, 200 KB, 400 KB, 800 KB, 1200 KB, 1600 KB, 3200 KB, 6400 KB, 12800 KB, 25600 KB and 51200 KB, created from both datasets.

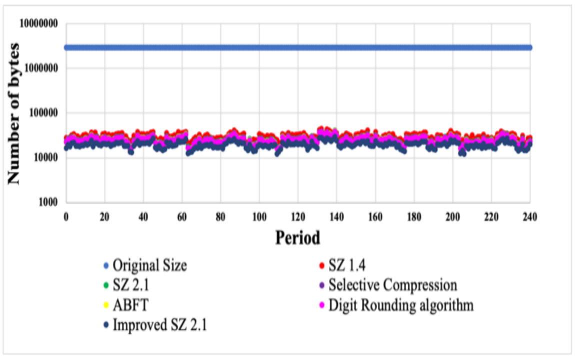
### 4.1.4 Result and Discussion

This section presents the results obtained from these experiments. During the experimental analysis, we applied the proposed protocol to observe data reduction, power consumption, encryption time, total processing time and error on IoT devices and we explain these findings as follows:

#### 4.1.4.1 Data Reduction

This subsection includes the data reduction results, i.e., the amount of data reduced due to compression and its effect on both datasets. Figure 4.3 shows the variation in data transmission to the edge device during each period when no compression algorithms are utilized and the same when the existing compression algorithms are used for two datasets. We obtain the transmitted data from the driver stress detection and home activity monitoring datasets. The vertical axis on a logarithmic scale shows the data delivered in bytes, while the horizontal axis represents the periods. We transmit 2976,000 bytes of data without compression. The SZ 1.4 compression algorithm increases the transmitted data from 19171 bytes to 44588 bytes and from 11388 bytes to 15926 bytes. Also, the maximum compression ratio obtained is 155 and 261 times, while the average compression ratio is 100 and 214 times, respectively, for the driver stress detection and home activity monitoring datasets. Figure 4.3 also presents the compression ratio of existing algorithms for both datasets. Using the SZ 2.1 compression algorithm, the transmitted data ranges from 16444 bytes to 40821 bytes and from 6723 bytes to 8291 bytes. Additionally, the maximum data reduction achieved is 180 times and 442 times, respectively. The average compression ratio achieved is 116 times and 392 times, respectively, for the driver stress detection and home activity monitoring datasets. Using the improved SZ 2.1 compression algorithm, the transmitted data ranges from 12296 to 29408 bytes and from 6723 to 8291 bytes .Additionally, the driver stress detection and home activity monitoring datasets can achieve a maximum data reduction of up to 242 and 442 times, respectively, with an average compression ratio of 154 and 392 times.





a. Driver Stress Data Detection Dataset b. Gas Sensor Data Dataset

Figure 4.3: Data Transmitted During Each Period

These compression algorithms showed variations in the amount of data transmitted. This is because some blocks are easy to compress, while others are harder to compress. The results demonstrated a significant reduction in transmitted data using the improved SZ 2.1. It also showed that the improved SZ 2.1 compression algorithm performed much better than the existing compression algorithms for driver stress detection and the home activity monitoring dataset. Figure 4.4 presents the comparison of both datasets using the existing compression algorithms with the improved SZ 2.1 and out of all the algorithms, the improved SZ 2.1 compression algorithm has the highest average compression ratio.

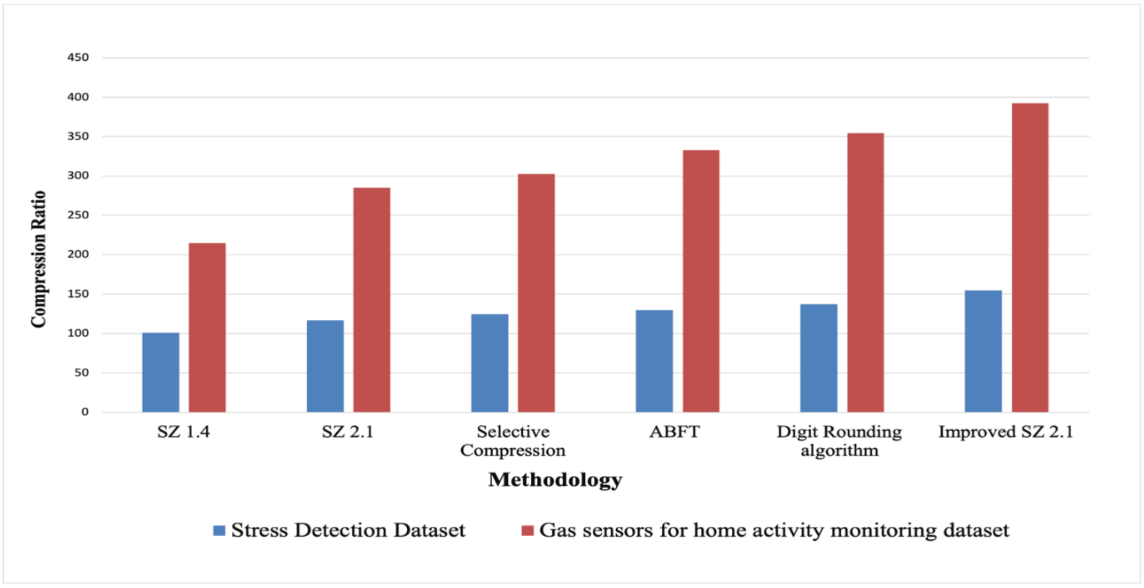
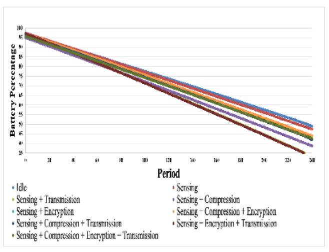


Figure 4.4: Comparison of Compression Ratio using the Compression Algorithms

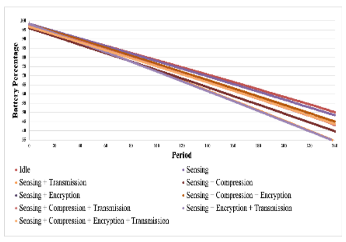
### 4.1.4.2 Power Consumption

This subsection includes details of the power consumption factor on two different datasets, i.e., the power consumed during each experiment. That experiment is conducted in different scenarios, such as: An inactive client, The client is using sensors to gather data. The client collects data and sends it to the edge at the end of each period. After each period, the client gathers data and encrypts it. After each period, the client collects data, encrypts it and sends it to the edge. After each period, the client gathers data, compresses it, and encrypts it. After each period, the client gathers data, compresses it, encrypts it and then sends it to the edge.

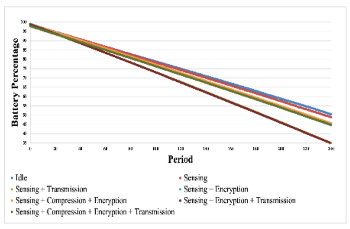
- **Driver Stress Detection Dataset :** These power consumption results shown by Figure 4.5 (a, b, c, d, e, and f) present the level of battery of the client IoT device over four hours during various scenarios when it is sending the data of the stress detection dataset and uses the proposed hybrid model (i.e., with all versions of the SZ compression algorithms, namely, SZ1.4, SZ2.1, selective compression, selective compression, ABFT, digit rounding algorithm and improved SZ2.1 algorithm) and ChaCha12-Poly1305 encryption algorithm over 241 periods in different scenarios as follows: The blue line indicates the reduction in battery level when the client is not in use. The battery level reduces to 51% using the proposed hybrid model. Redline: reduction of battery level when a client is sensing data. In this case, the battery level drops to 46%. The client is sensing and sending data to the edge, as indicated by the orange line. In this instance, the battery level drops to 37%. Black Line: Using the ChaCha12-Poly1305 algorithm, the client senses the data and encrypts it. In this instance, the battery level drops to 37%. Purple Line: The client detects the data, compresses it with the existing and proposed algorithms and encrypts it with the ChaCha12-Poly1305 technique. In this instance, the battery level drops to 45%. The client senses the data, encrypts it using the ChaCha12-Poly1305 algorithm and sends it to the edge. In this instance, the battery level drops to 37%. Yellow Line: The client is detecting the data, encrypting it with the ChaCha12-Poly1305 algorithm, compressing it with the existing and proposed algorithms and sending it to the edge. In this instance, the battery level drops to 45%. It shows in the green line that the data is being sensed by the client, encrypted with the ChaCha12-Poly1305 method and sent to the edge. In this instance, the battery level drops to 35%.



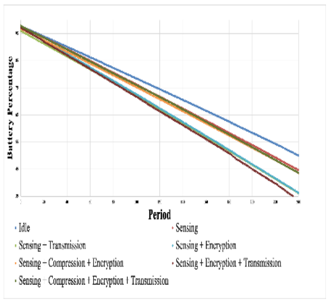
a. SZ 1.4

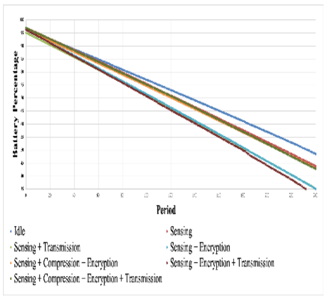


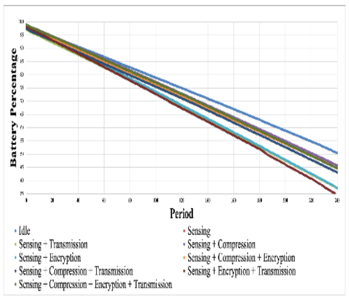
b. SZ2.1



c. Selective Compression



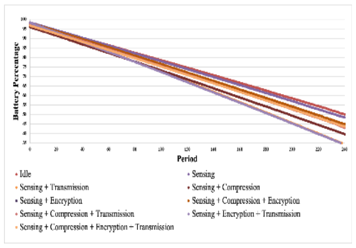


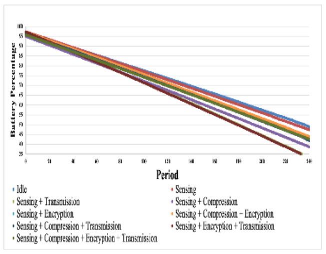


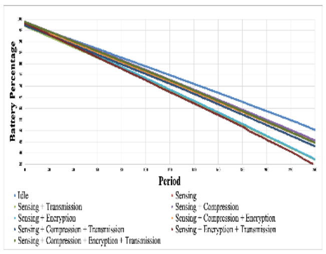
d. ABFT e. Digit Rounding f. Improved SZ2.1

Figure 4.5: Battery Level of the Client for Driver Stress Detection Dataset

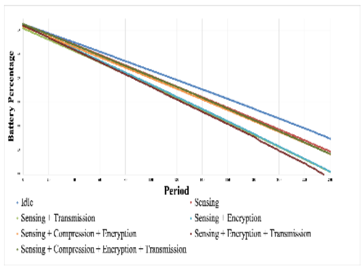
- **Gas Sensors for Home Activity Monitoring Dataset :** These power consumption results are shown by Figure 4.6 (a, b, c, d, e, and f) and depict the level of battery of the client over four hours during different scenarios when it is sending the data from the Gas Sensors for Home Activity Monitoring Dataset and using the proposed hybrid model (by using SZ 1.4, the original SZ 2.1, selective compression, algorithm-based fault tolerance (ABFT), digit rounding algorithms and the improved SZ 2.1 algorithm) and the ChaCha12-Poly1305 encryption algorithm over 241 periods. Figure 4.6 shows various scenarios, each accompanied by an explanation. blue line: an inactive client. In this case, the battery has dropped to 51%. Redline: The client is gathering information. In this instance, the battery level drops to 49%. Pink Line: The client senses the data and sends it to the edge. In this instance, the battery level drops to 36%. Black Line: The client senses the data and then encrypts it using the ChaCha12-Poly1305 method. In this instance, the battery level drops to 45%. The client senses the data, compresses it using the SZ 1.4 method and encrypts it using the ChaCha12-Poly1305 algorithm. In this instance, the battery level drops to 46%. The client senses the data, encrypts it using the ChaCha12-Poly1305 method and then sends it to the edge. In this instance, the battery level drops to 35%. Yellow Line: The client senses the data, compresses it using the SZ 1.4 algorithm, encrypts it using the ChaCha12-Poly1305 method and then sends it to the edge. In this instance, the battery level drops to 43%. Figure 4.6 (a) represents the client's battery level at SZ 1.4. Similarly, the remaining figures correspond to each of the mentioned compression algorithms, including the proposed one. The figures below demonstrate how the client senses the yellow line data, compresses it using an existing and proposed algorithm, encrypts it using the ChaCha12-Poly1305 method and then transmits it to the edge. In this instance, the battery level drops to 45%.

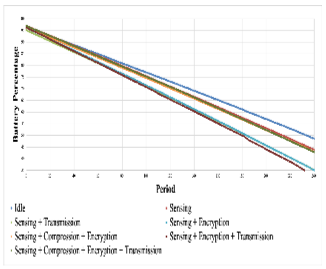


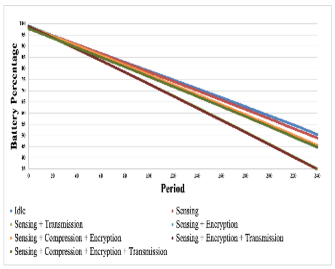




a. SZ 1.4 b. SZ2.1 c. Selective Compression







d. ABFT e. Digit Rounding f. Improved SZ 2.1

Figure 4.6: Battery Level of the Client with Home Activity Monitoring Dataset

All of these findings showed that the client uses the least battery when it is idle. When the client device detects data, it uses more battery power. The encryption technique further depleted the battery. The device consumes the maximum amount of energy when it encrypts data and then transmits it directly. However, compression significantly reduces energy consumption.

- **Percentage of Battery Saved:** Table 4.3 and 4.5 show the amount of battery saved when compression is used before encryption and transmission as compared to the case when the data is just encrypted and then transmitted. Using SZ1.4, 7.41% of the battery in the case of the driver stress detection dataset and 7.42% of the battery in the case of gas sensors for the home activity monitoring dataset were able to be saved. The compression algorithm SZ 2.1 increased battery life by up to 10.02% for the driver stress detection dataset and by 9.74% for the gas sensors in the home activity monitoring dataset. By using the selective compression algorithm, the percentage of battery saved in the case of the driver stress detection dataset was 8.89 and 8.98 in the case of the gas sensor dataset. For the driver stress detection dataset, the ABFT algorithm was able to save the battery by 9.57% and for the gas sensor dataset, it was able to save the battery by 8.99%. Next, the table displays the results of the digit rounding algorithm, which successfully saved 9.98% of the battery in the driver stress detection dataset and 9.45% in the gas sensors for the home activity monitoring dataset. In the end, the improved SZ 2.1 algorithm saved the most battery life compared to the others, 10.12% of battery life was saved for the driver detection dataset and 9.75% for the gas sensor dataset. This is because the original SZ2.1 used zstd lossless compression and the improved SZ2.1 used bzip2 compression. To achieve high compression, bzip2 uses the Burrows-wheeler transform, run-length encoding, move-to-front transform and Huffman encoding techniques. By sorting the data, the Burrows-Wheeler transform helps with data compression. Run-length encoding and Huffman encoding can easily compress the series of symbols this produces. Algorithm 4.2 represents the improved SZ compression algorithm.

Table 4.3: Percentage of Battery Saved for Drivers Stress Detection Dataset

| **Dataset** | Drivers Stress Detection Dataset | | | | | |
|---|---|---|---|---|---|---|
| **Compression Algorithm** | SZ 1.4 | SZ 2.1 | Selective Compression | ABFT | Digit Rounding algorithm | Improved SZ 2.1 |
| **% of Battery Saved** | 7.41 | 10.02 | 8.89 | 9.57 | 9.98 | 10.12 |

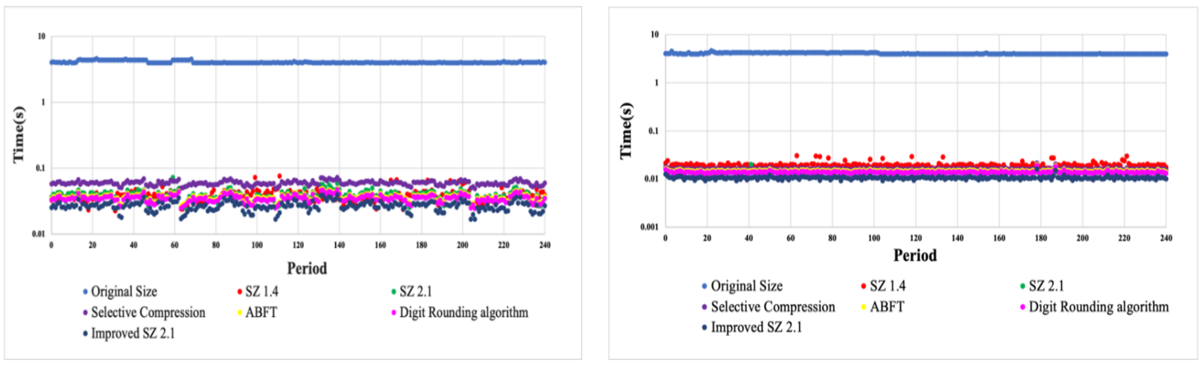
Table 4.4: Percentage of Battery Saved for Gas Sensors Dataset

| **Dataset** | Gas sensors for home activity monitoring dataset | | | | | |
|---|---|---|---|---|---|---|
| **Compression Algorithm** | SZ 1.4 | SZ 2.1 | Selective Compression | ABFT | Digit Rounding algorithm | Improved SZ 2.1 |
| **% of Battery Saved** | 7.42 | 9.74 | 8.98 | 8.99 | 9.45 | 9.75 |

Through the use of the proposed improved SZ 2.1, battery life could be increased by up to 10.12% in the case of the driver stress detection dataset and by 9.75% in the case of the gas sensors for home activity monitoring dataset. Therefore, the device's battery life continues to improve even after implementing the encryption algorithm. The results also showed that more energy is saved when using the improved SZ 2.1 algorithm as compared to other state-of-the-art techniques. This is because, for both datasets, the improved SZ 2.1 compression algorithm has a higher compression ratio than the others. A higher compression ratio implies lower encryption and transmission times, which leads to lower energy consumption. The amount of energy spent compressing the data is lower than the amount of energy saved during transmission.

### 4.1.4.3 Encryption Time

Encryption plays an important role in the security aspect. This subsection shows the result of the time taken to perform encryption. Figure 4.7(a,b) shows the time required by the ChaCha12-Poly1305 encryption algorithm to process every block of data for the two datasets over 241 periods. The vertical axis shows the time required on a logarithmic scale, while the horizontal axis shows the periods. The enhanced compression algorithm in SZ 2.1 reduced the time required by the ChaCha12-Poly1305 algorithm. The average encryption time is 0.026 for improved SZ 2.1 and 0.038 for both SZ 2.1 and the SZ 1.4 algorithm. Hence, the encryption time is the lowest for the improved SZ 2.1 algorithm. Figure 4.7 b shows the data from the gas sensors for the home activity monitoring dataset during 241 periods. The average encryption time is 0.010 for improved SZ 2.1, 0.014 for SZ 2.1 and 0.019 for the SZ 1.4 compression algorithm. Hence, once again, encryption time is the lowest for the improved SZ 2.1 algorithm.



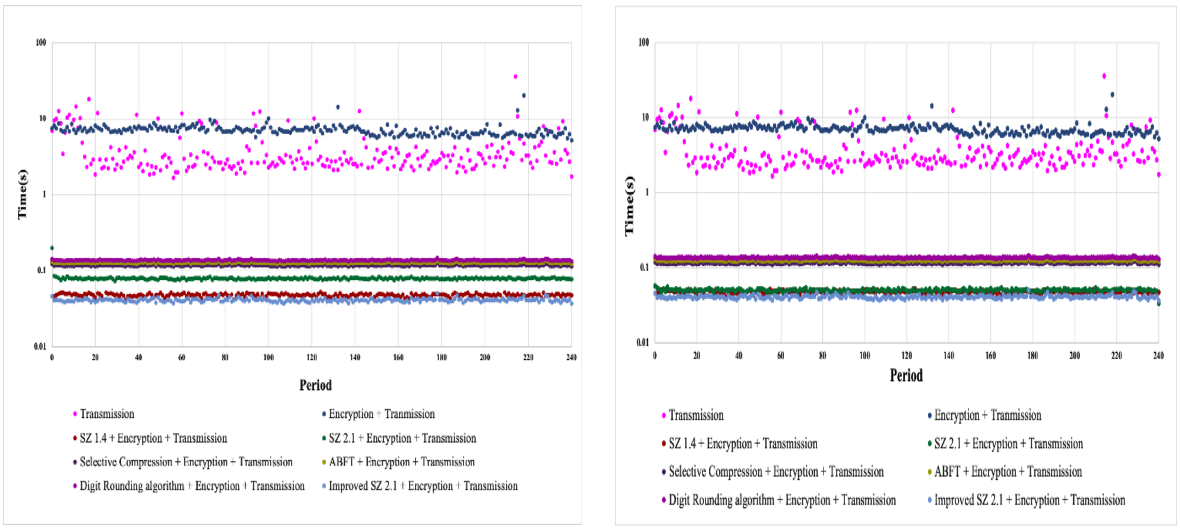
a. Driver Stress Detection Dataset

b. Gas Sensors for Home Activity Monitoring Dataset

Figure 4.7 : Encryption Time

### 4.1.4.4 Total Processing Time

This subsection includes information about processing time, i.e., the time taken to process a data block. Figure 4.8 (a, b) shows the overall processing time for each data block over 241 different periods for the driver stress detection dataset and gas sensors for home activity monitoring dataset. On a logarithmic scale, the horizontal axis represents the periods and the vertical axis represents the time. Using the ChaCha12-Poly1305 technique to secure data prior to transmission results in the highest total time required. The SZ 1.4, SZ 2.1 and improved SZ 2.1 compression algorithms reduce the processing time. According to the figure 4.8, the enhanced SZ 2.1 algorithm requires the least amount of processing time overall among the three compression techniques. The use of the compression algorithms SZ 1.4, SZ 2.1 and the improved SZ 2.1 reduces the time required for processing. Using improved SZ 2.1 compression and encryption before transmitting typically results in a significantly shorter total processing time compared to transmitting data without compression and encryption. This makes it clear that using compression and encryption together is possible without having a negative impact on the system. Even better, it reduces latency, enhancing the system.



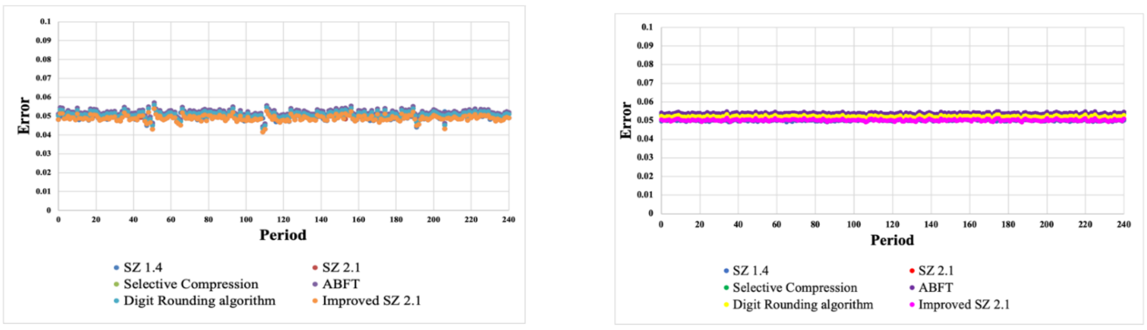
a. Driver Stress Detection Dataset b. Gas Sensors for Home Activity Monitoring Dataset

Figure 4.8: Total Processing Time

#### 4.1.4.5 Compression Error

This subsection includes the results of error parameters generated while working on the driver stress detection and gas sensor datasets, i.e., errors produced in the decompressed data due to lossy compression.

Figure 4.9 (a) shows the error produced by lossy compression in each data block over 241 periods. We obtained the data from the driver stress detection dataset. The vertical axis displays the absolute inaccuracy, while the horizontal axis shows the periods. We set the error bound at 0.1. However, for all three algorithms, the error remains around 0.05, well below the error bound. Figure 4.9 (b) shows the data obtained from the gas sensors for home activity monitoring dataset.

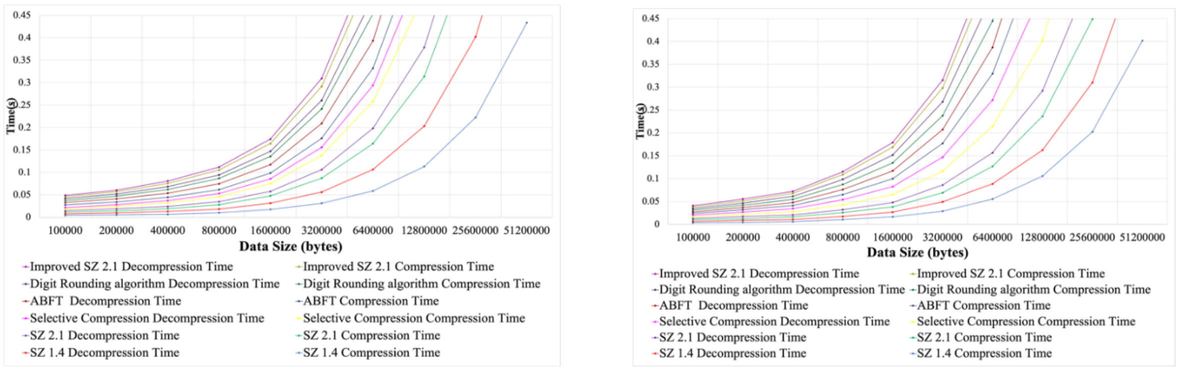


a. Driver Stress Detection Dataset b. Gas Sensors for Home Activity Monitoring Dataset

Figure 4.9: Error Produced due to Lossy Compression

#### 4.1.4.7 Compression and Decompression Time on Input Data Size

Figure 4.10 (a, b) shows that the time required for compression or decompression by all three algorithms increases linearly with size. Also, the time taken for decompression is always smaller than the time required for compression. This is because compression involves more complex calculations than decompression. Although the improved SZ 2.1 compression algorithm requires more time than the original SZ 2.1 algorithm for compression in the Gas Sensors for Home Activity Monitoring Dataset, it can achieve significantly higher compression compared to the original version. Therefore, the time required for encrypting and transmitting the data is less for the improved SZ 2.1 compression algorithm and it can effectively reduce the total processing time required.



a. Driver Stress Detection Dataset b. Gas Sensors for Home Activity Monitoring Dataset

Figure 4.10 : Compression and Decompression Time

#### 4.1.4.8 Encryption and Decryption Time for ChaCha12-Poly1305 Algorithm

Figure 4.11 depicts the time required by the ChaCha12-Poly1305 algorithm for processing blocks of different sizes. The encryption and decryption time required is approximately the same for all the data sizes. Hence, the points in the figure are overlapping. Additionally, it shows that the time complexity of both encryption and decryption is linear. The time utilized by the algorithm increases linearly as the data size increases, the algorithm's time usage increases linearly. The graph illustrates how long the ChaCha8-Poly1305, ChaCha12-Poly1305 and ChaCha20-Poly1305 algorithms take for varying data sizes. It increases as the data block size increases.

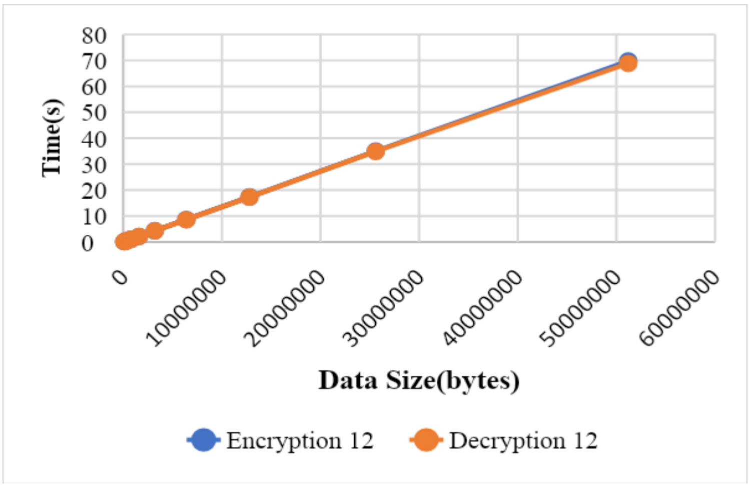


Figure 4.11: Encryption and Decryption Time for ChaCha12-Poly1305 algorithm

Also, as the number of rounds in the ChaCha algorithm increases, so does the time required by the algorithm. Thus, the ChaCha8-Poly1305 algorithm requires the least amount of time, the ChaCha20-Poly1305 algorithm requires the maximum amount of time and the ChaCha12-Poly1305 algorithm requires an intermediate amount of time. However, the greater the number of rounds, the higher the algorithm's security. The proposed model maintains a trade-off between security and performance by selecting the 12-round variant.

## 4.1.5 Summary

This work compares the proposed algorithm (improved sz2.1) with five existing state-of-the-art methodologies. The usage of the improved SZ 2.1 compression algorithm in combination with the ChaCha12-Poly1305 algorithm is presented for ensuring the security of data in IoT devices and it increases the energy efficiency of the edge computing architecture system. Both the ChaCha12-Poly1305 technique and the modified SZ 2.1 algorithm can be simply deployed for IoT devices on a real-time basis because of their simplicity. The experimental results show that this combination improves the battery life of IoT devices while also securing data. This combination improves security while preserving about 10% of the device's energy. Additionally, the latency has decreased, improving system performance. The improved SZ 2.1

compression algorithm achieves such high compression because it is a lossy compression algorithm. It also reduces the encryption time and total processing time required by 95% and 98%, respectively, for both datasets. This work also analyzes the impact of increasing input data size on the ChaCha12-Poly1305 algorithm. The ChaCha12-poly1305 algorithm's time increases linearly with size. We observe that the ChaCha8-Poly1305 algorithm outperforms the ChaCha12-Poly1305 algorithm due to its minimal security margin. On the other hand, the ChaCha20-Poly1305 algorithm is very slow and computationally expensive. Therefore, the ChaCha12-Poly1305 algorithm achieves a balance between security and energy trade-offs. In order to maintain a balance between power and security, this study prioritizes energy savings and employs a middle level of security. This is achieved by compressing data during transmission between IoT devices, which reduces computational complexity and costs associated with high levels of security.

## 4.2 IoT Streamed Data Handling Model using Delta Encoding

This work proposes a delta encoding model that uses a lightweight stream cipher encryption technique to facilitate delta-based compression.

### 4.2.1 Overview

The Internet of Things (IoT) is a trending internet connectivity extension that connects physical devices on heterogeneous networks to meet application requirements on various grounds, such as health, agriculture and industry. IoT devices generate a significant amount of streamed data, which they process either internally or through external resources like the cloud or fog, all while taking real-time constraints into account. When IoT devices generate a large amount of stream data, it travels through the network and requires processing on low-speed IoT infrastructure, posing a problem for real-time data transmission. Given this scenario, there is a significant need to design a model that relies on real-time stream data handling for speedy transmission, taking into account the real-time communication constraints of IoT devices. On the other hand, IoT devices are also very sensitive to energy consumption during streamed data transmission and low-battery devices become a barrier to real-time data handling. We propose a model to overcome these limitations by compressing and encrypting the data stream during IoT network transmission. For the experimentation, a test-bed setup is prepared utilizing a Raspberry Pi (IoT device), CPU and collectors to collect data streams. The proposed approach is compared with the baseline model, i.e., LDPC encoding on temperature sensor data sets. As a result, we compress the data up to 37.59% and reduce the average transmission time and average power consumption by 72.57% and 68.86%, respectively. The significant contributions of the proposed work are listed as follows:

- This work proposes a data stream handler utilizing the delta encoding method for encoding and compressing the data stream to speed up the transmission, maximize the compression ratio and minimize transmission time and power consumption with improved utilization of IoT devices.
- The proposed model has a high compression ratio for continuous time-series data, which can be beneficial in many use cases where measurements are taken at high frequency and the difference between consecutive data elements is smaller.
- The compression of the proposed method is dependent on the chosen value for the compressed word; therefore, it gives flexibility by choosing the size that is best suited to the given data stream.
- This model is computationally efficient as it is designed for IoT devices, utilizing low-cost operations like addition and bitwise operations for the data compression method. This way, it minimizes energy consumption during computation and saves the battery life of the IoT devices.
- Existing research ignored prior information on the compression percentage age; however, the proposed approach has taken into account how much compression is required for any input data set. This makes it a user-centred approach.
- The proposed model uses stream ciphers for encryption and decryption in order to ensure security during the data transmission from the client to the server side or vice versa. The stream cipher, which utilizes simple XOR operations and is reliable for continuous data transmission, is a faster and more secure technique for lightweight IoT devices. The same scenarios are accounted for in the proposed work.
- The proposed model is implemented on the real testbed of IoT devices on a publicly available data set [30] and compared with the baseline model .

### 4.2.2 Proposed Work

This section includes the details of the proposed model along with a demonstration of the algorithms. Sub-sections discuss the proposed data stream flow, data stream handler, details of the streamed data transmission from the IoT device to the IoT border-router, and procedures used for receiving the data stream at the collector.

- **Data Stream Flow:** This sub-section demonstrates the data flow from the IoT device or client to a data stream collector or server, as shown in Figure 4.12. From the IoT device, the data stream spreads to the data stream collector, passing through the IoT border router. IoT devices like IoT border routers, ZigBee, BT-LE and Z-wave, which are slow due to their low MTU, operate between the source and destination. Additionally, the router and data stream collector utilize Wi-Fi and Ethernet, both of which have high MTU and speed. The IoT device transmits data at regular intervals or in response to a request message [120]. Further, the client and server programs are created: The client program and server program are dedicated to IoT device data transmission and data stream collector, respectively, as mentioned in the respective subsections with detailed examples and algorithms.

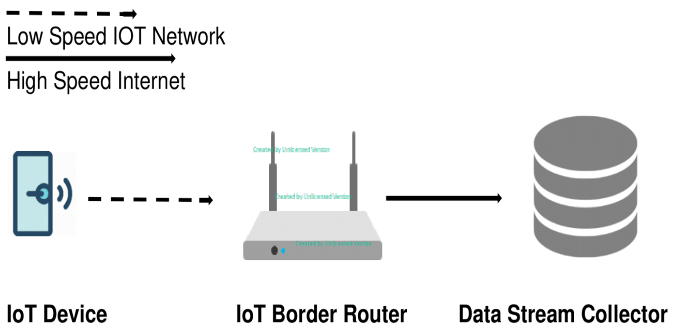


Figure 4.12: Flow of Data from IoT Device to Data Stream Collector

- **Data Stream Handler by Delta Encoding:** This sub-section describes the streamed data handler using an example. The proposed data stream handler utilizes delta encoding, transmitting the current element as the difference between the previous and current data elements [121]. To demonstrate the concept of delta encoding, the following example is considered: The sample data elements that are to be sent to the data collector are 26, 27, and 29. The proposed data stream handler calculates the difference between the second and first values, the third and second values, and so on. The first value is sent as 26 (same as the original element) in two's complement, stored in 6 bits and the difference values of the of the first and second data elements can be stored in 3 bits. The IoT device or client sent the values 26, 27 and 29 as '011010', '001' and '010' in binary to the data stream collector or server. Therefore, the second and third elements are combined into six bits of '001010' as one element. Afterwards, we encrypt these values and transmit them to the server. The server first decrypts and extracts the data, with the first element being 26. For the next two data elements, the next 6 bits are stored in a frame in two's complement form, i.e., '001010'. After two's complement, get the first 3 bits of data value '001' or 1 in decimal, then add it to the previously obtained value to get the element 27. Next, to get the third element, we use the rightmost three-bit data value, ‘010’ or 2 in decimal. Similarly, we add this value to the previously obtained element 27, yielding the third element, 29. Similarly, we can take another example of floating data: consider a data set with elements 26.2342, 27.3452, and 29.5345. These elements can be stored as either 262342, 273452 and 295345 (using lossless compression) or 26, 27 and 29 (using lossy compression). The proposed model considers lossy compression because it does not alter the temperature value result, as suggested in the base model [119]. If the value is negative, we have two options to represent the negative value: first, store it using the 2’s complement and second, store it using the sign bit representation. To store negative values, the proposed model uses 2's complement. In the data stream handler, delta encoding is used for compression, as shown in Algorithm 4.7. The model stores the input in data stream D and calculates the difference between the current and previous data elements for each subsequent input element. The final value stores this difference and the data element's 2's complement in the compressed output stream.

**Algorithm 4.7: Delta Encoding**

*delta_encoding*

**Input:** Data Stream D

**Output:** Compressed Data Stream C

```
1. Begin
2.        a = first data element in D
3.        append the first data element in C
4.        for i=2 , length(D) do
5.             diff = D [i]- D[i-1]
6.             final value = diff in M bits
7.             store final value in C
8.        end
9. End
```

- **Flow Chart of Data Stream Handler:** This sub-section explains the logical flow of the data stream from the client-side (IoT device) to the server-side (data stream collector) using delta encoding. Figure 4.13 shows the flow chart of data transmission from an IoT device using delta encoding, in which the input data stream is D. N bits are required to represent the original data and M bits are required to represent the output data. First, the data value is stored the same as the output stream. For the remaining elements, calculate the difference between the current data element and the previous one. If the calculated difference fits in M bits, send it using M bits; if not, send it using N bits. Repeat this process until you have processed all the data elements. The flow chart in Figure 4.14 shows the collection of data from the receiver side, followed by the collection and subsequent processing of the decompressed data. Compressed data stream C is taken as the input; all the elements in the data stream are processed one by one. The first data element is stored as it is in the output stream using N bits. For the rest of the elements, the value received is the difference between the current data element and the previous data element. Therefore, the original data element is obtained by adding the value of the differences to the previous data element. This process is repeated until all the data is processed.

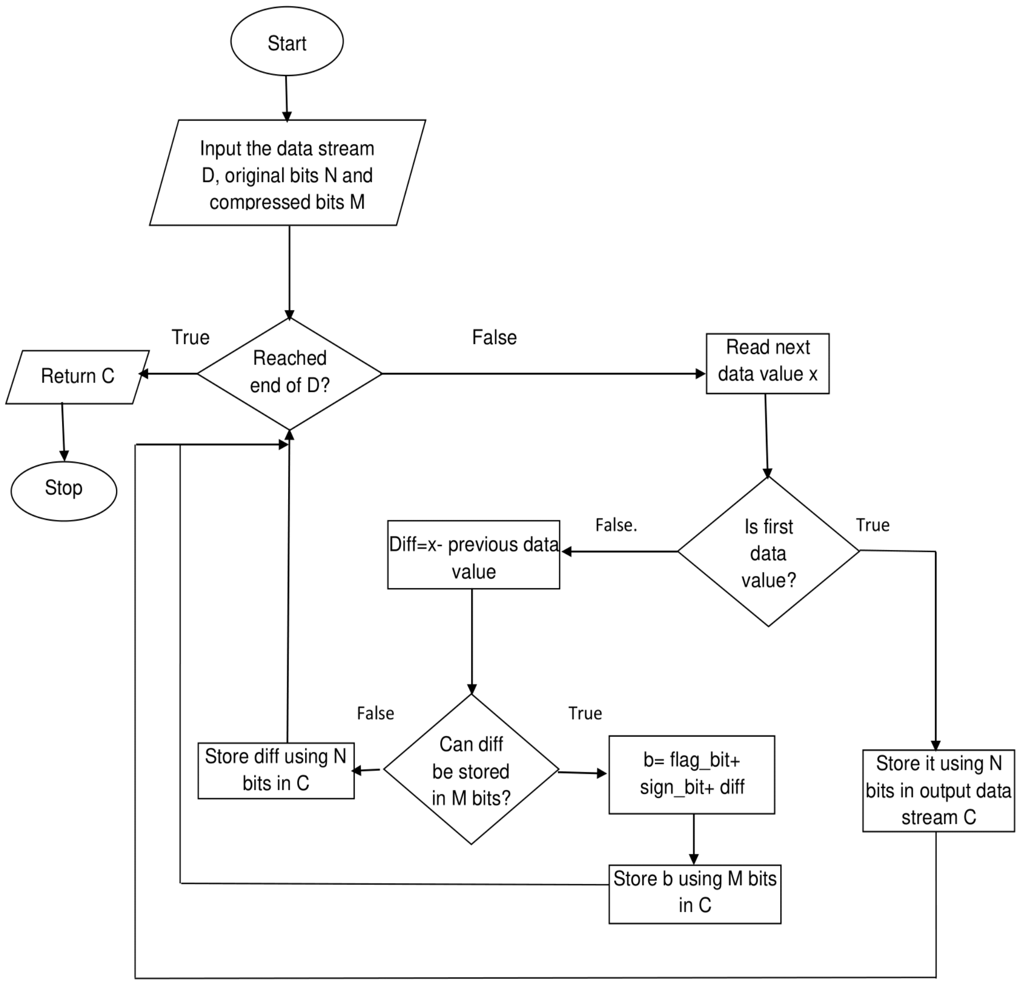

Start
Input the data stream D, original bits N and compressed bits M
True
Reached end of D?
False
Return C
Stop
Read next data value x
Is first data value?
False.
True
Diff=x- previous data value
Can diff be stored in M bits?
False
True
Store diff using N bits in C
b= flag_bit+ sign_bit+ diff
Store b using M bits in C
Store it using N bits in output data stream C

Figure 4.13: Data Transmission using Delta Encoding

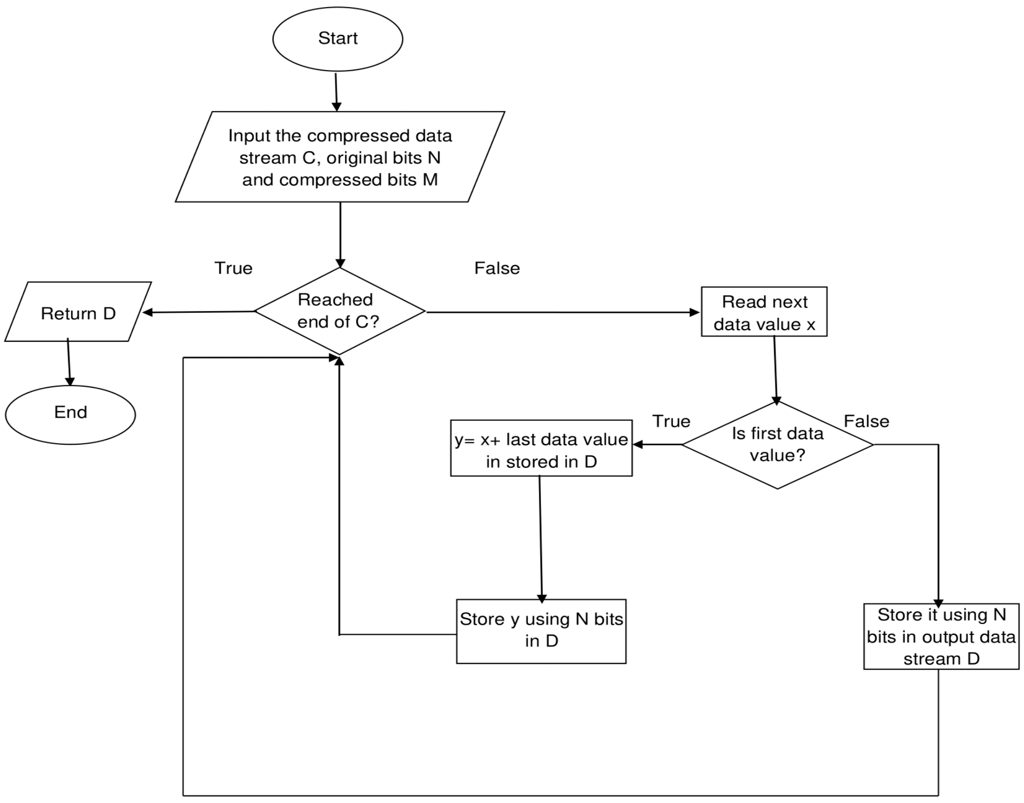

Start
Input the compressed data stream C, original bits N and compressed bits M
True
False
Reached end of C?
Return D
End
Read next data value x
True
False
Is first data value?
y= x+ last data value in stored in D
Store y using N bits in D
Store it using N bits in output data stream D

Figure 4.14: Data Receiving using Delta Decoding

- **Data Transmission in IoT Devices:** This sub-section provides a detailed description of the data transmission from the IoT device to the IoT border router. Figure 4.16 shows how stream data is transmitted to the IoT device through compression using delta encoding. For the encryption, stream cipher is used, as stream cipher has fast processing speed in cases of limited computing resources, as in the IoT [116]. Figure 4.15 shows how an IoT border router sends data. It uses key streams 010101, 011, 010 and 110 to compress and encrypt a set of numbers that add up to 25, 23, 24 and 23. The XOR operation performs the encryption, while the previously distributed seed generates the key streams. In this example, process the first element 23, as 23 (0101111), and the remaining elements 24, 23 and 25, as 1, -1 and 2 (001, 111 and 010 in binary). Further, these values are XORed with key streams, then encrypted to 00010, 010, 01 and 100 then sent to the collector or server. N stores the original uncompressed 6-bit data, while M stores the compressed 3-bit data. Therefore, the proposed model requires 37.59% of the memory for the compressed data, whereas the existing LDPC code model requires up to 50% of the memory for the same experimental data set [119]. Algorithm 4.8 represents the client-side data transmission in IoT devices. Here, first convert the input temperature data-set elements to integer format, taking into account the integer part of the temperature. Next, create an array, known as a diff dataset (initialized with a value of 0), where we store the differences between each set of two consecutive temperature elements. Different fractions of data in a different dataset array are sent to the server using an observation array. The experiment transmits data iteratively to the server 25 times for each observation factor based on the data set [118], and calculates the average transmission time for all the data set fractions. Algorithm 4.9 describes the transmit dataset function used in Algorithm 4.8. Binding the IP address and port number creates a socket. Next, the system calculates the handshake value using the first value of the input data set, stored in the 6 bits complement. It then converts this value into a decimal and stores it in the final variable. The system then initializes the message array with the final variable and traverses the diff dataset until it reaches the specified factor. The system stores the current and next elements in the data set in 3 bits complement, combines them into 6 bits and sends them as a single packet after encrypting them using a stream cipher. The system converts the encrypted value into a decimal value and stores it in the final variable, with each packet consisting of 6 bits for the experimental data set. The final variable is appended to the message array. Finally, the socket sends each element in the message array to the server.

-

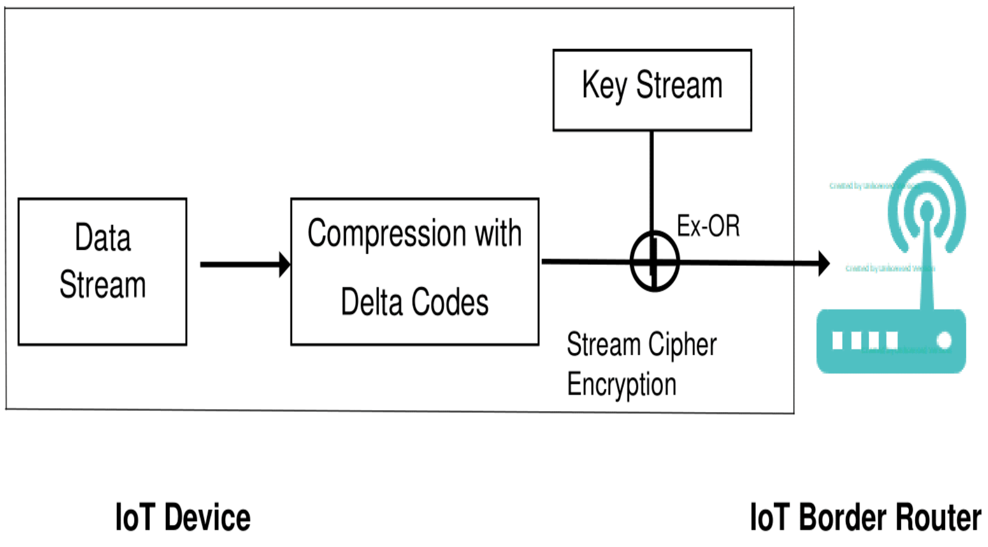


Figure 4.15: Data Transmission in IoT device to Border Router

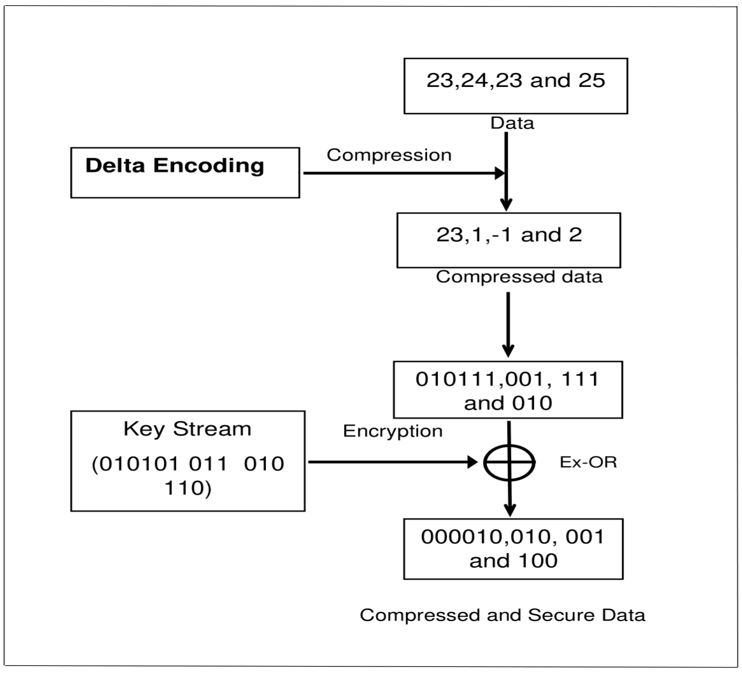


Figure 4.16: Data Transmission in IoT-Border Router: Example

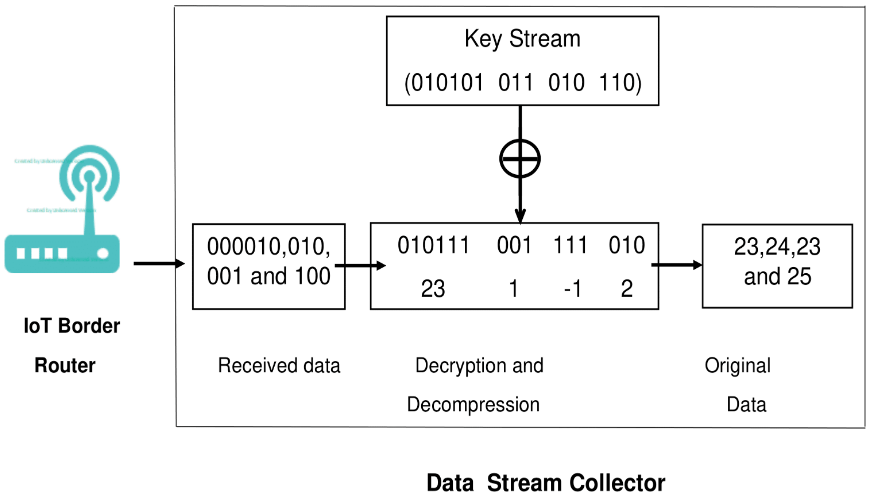


Figure 4.17 Receiving Data in Data-Stream Collector

- **Receiving Data in Data Stream Collector:** This sub-section discusses the process of receiving data from the IoT-Border router to the data stream collector. The data elements in the form of a stream are received, decrypted and decompressed using the methods discussed. Figure 4.17 shows the decompression mechanism at the data stream collector, which generates the original data sent by the IoT device or client prior to compression. The received data stream's values (considered in the previous example) are 000010, 010, 001 and 100. During the decryption process, the key stream is XORed with the received data to obtain the original data (010111, 001, 111, 010). With the encryption key, the original data stream can be decrypted. The data stream collector generates the decrypted elements, which represent 23, 1, -1 and 2. After using delta decoding for decompression, we generate the original data elements 23, 24, 23 and 25, using the first data element (23) as the current data element. We then perform an addition operation with the current and next-generated data to obtain the remaining data elements. Algorithm 4.10 presents the pseudo-code for the received data in the data stream collector at the server-side program. The server program function initializes an empty data dictionary using P and T for the packets and transmission times. Algorithm 4.11 presents the pseudo-code for the server program function, as called in Algorithm 4.10. Binding to the IP address and port number creates a socket, and the server patiently awaits the client's connection. The server continuously receives data from the client until it runs out of data to send. The server calculates the total transmission time by comparing the times of the first and last packets received. The server converts the received data from binary values into decimal elements before decrypting. When the first packet is received, the server records the current time in the variable start time and stores the received data in the final variable. Finally, the data is stored in the array org data. The experimental data set [118] fixes the packet size to 6 bits, containing a first byte and a second byte of 3 bits each to represent the two different values. Next, use masking and the right shift (>>) operation. The right shift by 3 bits is followed by the AND operation with the mask value, which gives the leftmost 3 bits. As mask represents 111 in binary, we apply the AND operation with mask (111) to get the leftmost three bits, i.e., the first byte. Similarly, by applying the AND operation with a mask, we can get the rightmost 3 bits as the second byte. Finally, we add the first byte to the last variable. The last variable is stored in the array org data. Similarly, the second byte is added to the last variable and stored in the array org data.

**Algorithm 4.8: Transmission in IoT Device**

*Transmission_in_IoT_device*

**Input:** Temperature Values T

**Output**: fraction of data-set in terms of i

```
1. Begin
2.         int data-set = Integer(T)
3.         diff data-set = [0]
```

```
4.          for i=2 , length(T) do
5.              diff data-set += [int data-set[i] - int data-set[i-1]]
6.          end
7.          observations = [ 1.0, 0.5, 0.25, 0.125, 0.0625]
8.          for i in observations do
9.              for j = 1 . . . 25 do
10.             Transmit_data_set(int data-set, diff data-set, i)
11.             end for
12.          end for
13. End
```

**Algorithm 4.9: Transmit Data-Set**

*Transmit_data_set*

**Input:** int data-set, diff data-set, Observation factor F

**Output:** Message array and each value in the message array is sent to the server using the socket

```
1.  Begin
2.          s = socket()
3.          s.bind(ip, port)
4.          count = 1
5.          handshake = twosComplement(6, int data-set[0])
6.          Convert handshake value to decimal and store it in byte format in final
7.          message array += [final]
8.          for i in range(1, length(int data-set) * factor, 2) do
9.              combined = twosComplement(3, diff data-set[i]) + twosComplement(3,
    diff data-set[i+1])
10.             handshake = encrypt(combined)
11.             Convert handshake value to decimal and store it in byte format in final
12.             message array += [final]
13.             count += 1
14.         end for
```

```
15.         for message in message array do
16.             send(s, message)
17.         end for
18. End
```

**Algorithm 4.10: Receiving in Data Stream Collector**

*Receiving_in_data_stream_collector*

**Input:** Data received from client: data

**Output:** Packets P, Transmission Times T 1: data = {}

```
1. Begin
2.          for i = 0 . . . 124 do
3.              P, T = Server_Program()
4.          if not data[P] then
5.              data[P] = [T]
6.          else
7.              data[P] += [T]
8.          end if
9.          end for
10. End
```

**Algorithm 4.11: Server Program**

*Server_Program*

**Input:** Data received from client: data

**Output:** Array of original data sent from client after decryption and decompression: org data

```
1. Begin
2.          s = socket()
3.          s.bind(ip, port)
4.          connection = s.accept()
5.          org data = []
```

```
6.      while true do
7.          data = receive data(connection)
8.          if not data then
9.              end time = current time
10.             close(connection)
11.             return count, end time – start time
12.         end if
13.             data = convert data to decimal from bytes
14.             data = decrypt(data)
15.             mask = 7
16.             if first packet of data received then
17.              start time = current time
18.              last = data
19.              org data.append(last)
20.              else
21.                 first byte = (data >> 3) & mask
22.                 second byte = (data & mask)
23.                 last += first byte
24.                 org data.append(last)
25.                 last += second byte
26.                 org data.append(last)
27.          end if
28.     end while
29. End
```

## 4.2.3 Experimental Study

This section outlines an experimental study that aims to analyse the performance behaviour of the proposed model through a series of experiments, experimental setup for the software and hardware devices used to conduct the set of experiments, different performance metrics and testing parameters used to carry out the experiments. Then provides a brief overview of the data set. Then, details the comparative experimental results with the baseline state-of-the-art model.

#### 4.2.3.1 Software and Hardware Specifications

Python libraries such as Pandas, NumPy, Matplotlib, SciPy and Socket helped implement the algorithms. The following are the details of the utilized H/W components.

- **IoT Device:** For the computation, the Raspberry Pi 4B was used as an IoT device, which uses the 6LoWPAN network. It utilizes Bluetooth Low Energy and the IoT SDK (software development kit) [122]. Table 4.5 provides a detailed specification of the devices used.
- **IoT Border Router:** The implementation of the delta encoding is done in the Python language; also, for the implementation of the IoT border router, the Raspberry Pi is used, as in [123]. We use IoT-SDK to support the 6LoWPAN network and BT-LE. Additionally, we establish an Ethernet-wired connection from the IoT-border router to the data-stream collector.
- **Data Stream Collector:** The implementation of a data stream collector is also done using a computer and operating system as listed in Table 4.5.

Table 4.5: Specifications for Raspberry-Pi 4B and Computer

| | Raspberry Pi 4B | Computer |
|---|---|---|
| OS | Raspbian Buster | Ubuntu 18.04.6 LTS |
| CPU | Quad-core 64bit BCM2711 | Intel i7-7700-HQ-4-Cores @ 2.8 GHz |
| RAM | 4 GB | 16 GB |
| Network | BLE (Bluetooth low energy), Wired Ethernet | Wired Ethernet |

#### 4.2.3.2 Parameter Settings

This sub-section presents the local and global parameter used for the implementation of the proposed model.

- **Local Parameter:** The experiment employed performance parameters to gauge the effectiveness of the proposed model. The Python script shows the variables used for data transmission and compression from the IoT device to the data collector.

- **Global Parameter:** Model conducted all experiments based on local parameters and evaluated the results using the following global parameters:

  ***Transmission Time***: Figure 4.19 shows the data transmission time from the IoT device to the collector or from the client to the server program. The transmission time for each data packet is much larger than the compression time. Even when we add the compression and transmission times, the obtained results remain unchanged. This proves the viability of this data stream handling method by reducing the total transmission time (TTT).

***Compression Time:*** The proposed model requires additional time to compress the data stream. The proposed approach can calculate the compression percentage in advance. Prior knowledge of the compression percentage is beneficial in deciding the need for compression. State-of-the-art research methods focused on packet header compression to reduce the gap in network performance between IoT and internet searchers [119]. On the other hand, the suggested method concentrated on compressing the payload, even when it contained encrypted data. If represent the data in n-bits and need to compress it in m-bits, meaning we can easily represent the difference between every pair of consecutive elements in m-bits, then we can calculate the compression ratio:

$$\text{Compression Ratio} = \frac{n}{m} \tag{4.1}$$

$$\text{Compression Ratio} = \frac{(100*n)}{\mu*\eta+(100-\mu)*m} \tag{4.2}$$

However, this may not always be the case in real-world scenarios, necessitating the modification of the compression ratio formula to incorporate a stochastic variable that indicates the likelihood of representing the difference using m-bits. The stochastic variable is called a collision rate and is denoted as μ. If there are 100 elements to be sent with a collision rate equal to μ, then μ data elements would require sending uncompressed data elements and 100-μ data elements could be compressed. To maximize the compression ratio, choose the optimal value of m for the given data set based on a sample. If you choose m too small, it will lead to an increase in the collision rate and a decrease in the compression ratio. If you choose m to be large, the collision rate will decrease, but the compression ratio will also decrease because of the inverse relationship between m and the compression ratio. For example, if we choose n = 24, m = 8 and μ = 3, then the compression ratio's corresponding value is 2.8301 and data compression is 38.48%.

***Power Consumption:*** An IoT network uses Bluetooth low energy (BT-LE), a slow network that affects the transmission time as the data size increases. This slow network also increases MTU and fragmentation, which affects the performance of the devices, especially those that are sensitive to power consumption. In order to minimize power consumption, it is necessary to reduce the size of the data through compression. Equation 4.3 provides the formula for calculating power consumption [119].

$$\text{Power Consumption(W)} = \text{Transmission Time x } 1.253 + \text{Compression Time x } 1.44 \tag{4.3}$$

The power consumption for the transmitted data stream and data compression were calculated with the help of a power meter. During the data stream transmission, the power consumption was 1.253 W and during the compression process, it was 1.44 W. The compression process takes very little time compared to transmission; however, the power consumption rate is a bit higher for compression as it utilizes more memory during the process.

#### 4.2.3.3 Dataset

In this work, have used a publicly available data set [118] using a set of gas sensor values from the home activity monitoring system. Home monitoring is one of the most widely practiced IoT applications. The data set consists of the recordings of eight gas sensor clusters made out of eight MOX temperature, gas and humidity sensors. The data set has different time series for three conditions: wine, banana and background activity. We recorded the responses to wine and bananas by placing the stimulus near sensors. The length of every simulation ranges from 7 minutes to 2 hours, with an average of 42 minutes. The data set is available in (124, 928992) dimensions, with a total of 12 columns and 928992 rows. We have only used temperature sensor data in our experiments. We present the temperature data values in ascending order. The total size of the data set is 90.5 MB.

#### 4.2.3.4 Experimental Results

The performance of the proposed model is measured in terms of transmission time, compression ratio and power consumption. The proposed approach uses delta encoding for compression, which decreases the data transmission time by compressing the data. For a data size of 1600 KB, the compression time is 0.64 seconds. Estimated the transmission time before and after data processing to calculate the compression time. As data size plays a major role in compression time, to verify the effectiveness of the proposed model on varied data sizes, the data set is divided into 100KB, 200KB, 400KB, 800KB and 1600KB to conduct a set of experiments.

- **Compression Time and Compression Ratio:** This sub-section presents the results of the compression time taken to compress the data, as shown in Figure 4.18. As the data size increases, the results show a graphical representation of the increase in compression time. The compression time increases linearly with respect to the increase in the data size. However, the compression time decreases even for larger data sizes. The compression times for different data sizes (100, 200, 400, 800 and 1600 KB) are 0.02, 0.03, 0.06, 0.12 and

0.25 seconds, respectively. The proposed model outperforms the baseline LDPC model due to its computational efficiency, which is tailored for IoT devices, as well as its use of low-cost operations such as bit-wise and addition. Furthermore, we observe that each data size's transmission time significantly exceeds its compression time. However, our analysis indicates that adding the compression time to the total transmission times of the compressed data does not affect the obtained results. According to the proposed model, the longest time for compression was 0.32 seconds and the longest time for transmission was 140 seconds for 1600 KB packets. These results can be seen in Figure 4.18 and Figure 4.19 (a). Additionally, the total processing time for 1600 KB of data is 140.32 seconds. This shows that compression time has little effect on the total processing time. The total processing time depends mainly on the transmission time. Using compression during processing doesn't harm the system. This shows the real-time viability of this data stream handling method by reducing the total transmission time for large volumes of data. Equation 4.1 and 4.2 allow us to calculate the compression ratio. In our experimentation, the maximum value for the temperature data set corresponds to 8 bits. Therefore, $n = 8$ is the uncompressed data size and $m = 3$ is the compressed data using the proposed approach, since it can be stored in 3 bits after the compression. The chosen collision rate ($\mu$) is ¡ 0.0005 for total '928992' temperature data values for the data set [118]. The value of compression ratio is obtained as 2.66 and data compression is 37.59%, which is better than the LDPC approach, which had compression up to 50% [119].

- **Transmission Time:** This sub-section presents the results collected after a set of experiments for the transmission time (the time taken for data transmission from the IoT device to the collector). Figure 4.19 (a) shows the data transmission time with and without the proposed data handler. We presented a comparative study of the results generated by the LDPC method and the proposed approach. The LDPC model has the highest transmission time without compression for data of varying sizes. The LDPC model with compression has an even lower transmission time. However, the proposed model with compression has the lowest transmission time. Therefore, the proposed method has achieved better performance in terms of transmission time because of a better compression ratio based on data compression using delta encoding. Figure 4.19 (b) shows the comparison of transmission time reduction rates for different data sizes. As the data size increases, the reduction rate varies. Our model's reduction rate is 67% for 100 KB, 81% for 200 KB, 74% for 400 KB, 69% for 800 KB and 70% for 1600 KB data sets, which is better than the LDPC model. Data compression directly reduces the transmission time, the proposed model sends data in the form of deltas rather than actual values and the data set's temperature values range from 26 to 54 Celsius [118]. To represent the temperature value in decimal form, 8 bits are required. We can store these temperature values in 3 bits after applying delta encoding, as we can store the difference in a maximum of 3 bits. As a result, we get data compression up to 37.59% for all stream data; however, in the LDPC encoding scheme, data compression was approximately 50% for maximum stream data [119]. This reduction in transmission time expedites the processing of the data stream. This way, it also conserves network resources.

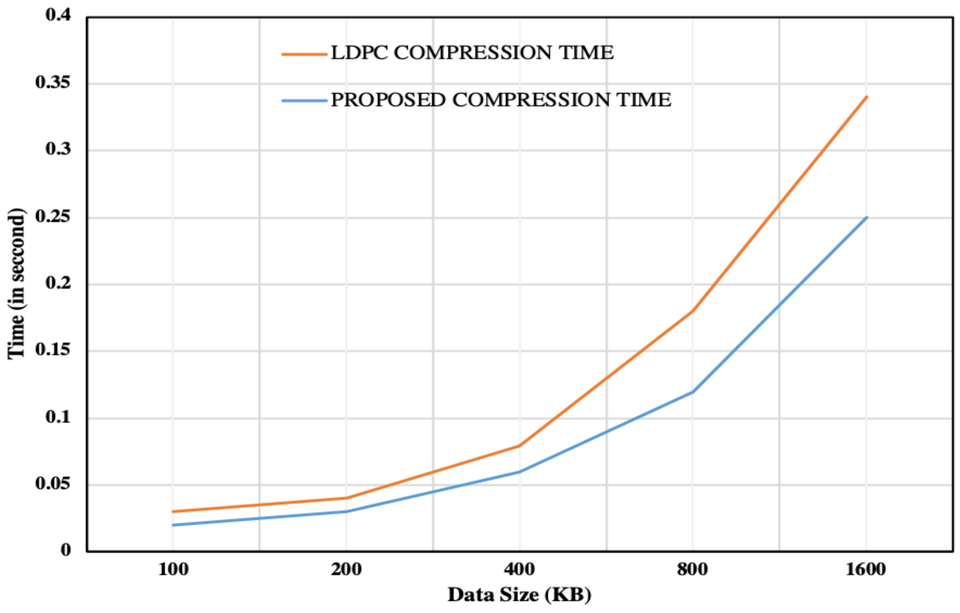
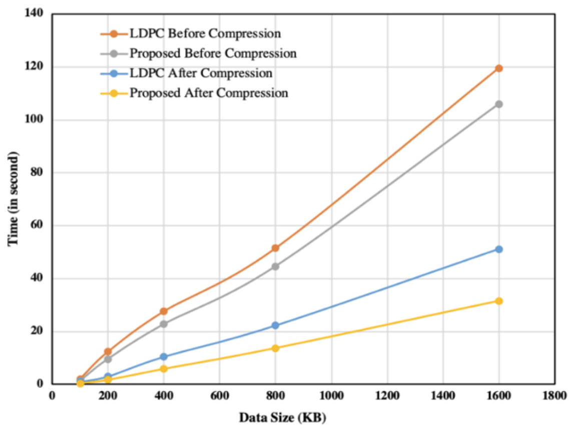
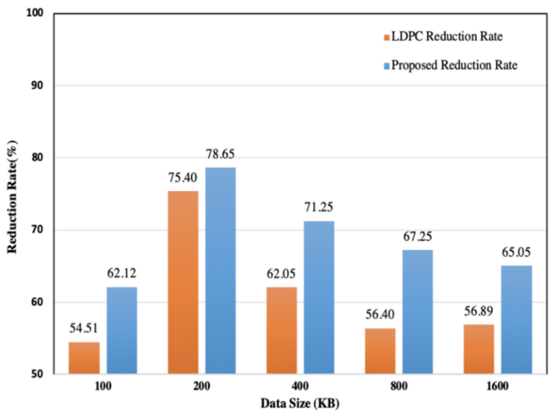


Figure 4.18: Compression Time

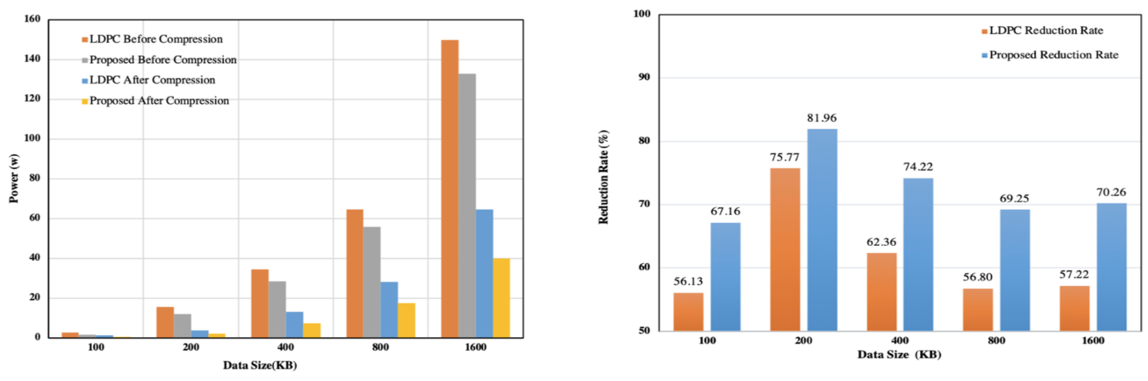




(a) Transmission Time (b) Transmission Time Reduction Rate

Figure 4.19: Transmission Time

- **Power Consumption:** IoT devices, operating in various scenarios without frequent battery charging, exhibit high sensitivity to power consumption. The power used (in watts) before and after compression with the LDPC and the proposed model is shown in Figure 4.20 (a). The data sizes are 100, 200, 400, 800 and 1600 KB. The proposed model has a lower power consumption before and after compression than the existing model. The reduction in transmitted data leads to a reduction in energy consumption, as data transmission is the primary energy consumer. By reducing the transmission time, we can save a lot of energy. Figure 4.20 (b) shows the power consumption reduction rate for the different data sizes. We obtain a reduced rate of 54% and 62% for 100 KB, 75% and 78% for 200 KB, 62% and 71% for 400 KB, 56% and 67% for 800 KB and 57% and 65% for 1600 KB for the LDPC and the proposed model, respectively. The proposed model's reduced power consumption is due to a reduction in transmission time and compression time, as power consumption is directly proportional to time.



a) Power Consumption        b) Power Consumption Reduction Rate

Figure 4.20: Power Consumption

### 4.2.4 Summary

This work proposes a data stream handler that reduces the data transmission time from IoT devices to a data stream collector. The proposed model reduces the data size, thereby bridging the performance gap between the IoT device and the Internet. The proposed model is implemented on temperature sensor data using the delta encoding method. Consequently, we reduced the average transmission time to 72.57% from the baseline model's 50.2%. The proposed model reduces data compression by up to 37.59% compared to the baseline model, which was 50% and also reduces the average power consumption by 68.86% [119]. It can be a candidate solution for the future era of IoT (industry 5.0) in order to minimize compression time, data transmission time and power consumption.

## 4.3 IoT Streamed Data Handling Model Using LDPC Coding

This work uses IoT stream data is handled by compression for fast transmission of the data on cloud. The technique used is LDPC coding for compressing the IoT data and transfer for further processing.

### 4.3.1 Overview

Internet of Things is an extension of internet connectivity to physical devices in heterogeneous networks. IoT network consists of the Internet and low- speed IoT. In IoT, with the help of the internet, many devices can communicate and interact with others. The devices can be monitored remotely. A large amount of data stream is transmitted from IoT devices to the internet. Therefore, it is observed that there is a difference in the speed and maximum transfer unit (MTU) of IoT which results in overhead. Overhead also depends on the data, as the data size increases the value of overhead increases. This is a major problem in IoT devices as they are sensitive towards power consumption also. They need to be handled in real-time. To solve this problem there is an approach that compresses the data stream. The approach is Low-density parity-check code (LDPC). This compression technique using LDPC can also be applied to encrypted data. Therefore this method is also beneficial in privacy and confidentiality. For the implementation, they have used Raspberry Pi, collectors and a computer. Raspberry Pi used to generate data stream, collector used to collect the data. We focused on specific data in different sizes and to calculate efficiency. Efficiency is calculated using transmission time and compression time. The dataset used for the experiment is temperature sensor data. As a result, it is observed that the data stream transmission time is decreased by 65%. As a result of this study, the data transmission time from IoT to the collector can be reduced after compression using the LDPC code.

### 4.3.2 Proposed Work

This section includes details of data stream handler to improve utilization of data stream. Section A includes data stream transmission from IoT device and Section B discuss the decompression of data stream received at collector.

- **Transmission of Data in IoT Device:** In Figure 4.21 the data with the value 00111011 (in binary) are compressed using LDPC codes and encrypted with key streams (11010001). The encryption is done by the XOR operation. In example, the data is read in binary and processed also in binary. Then input value is XOR with key streams 11010001.The above procedure is applied to all the data streams which perform zero padding if required. The applied algorithm for encryption is required as needed. And then using LDPC coding the encrypted data after Ex-OR with key stream compressed into 0111.

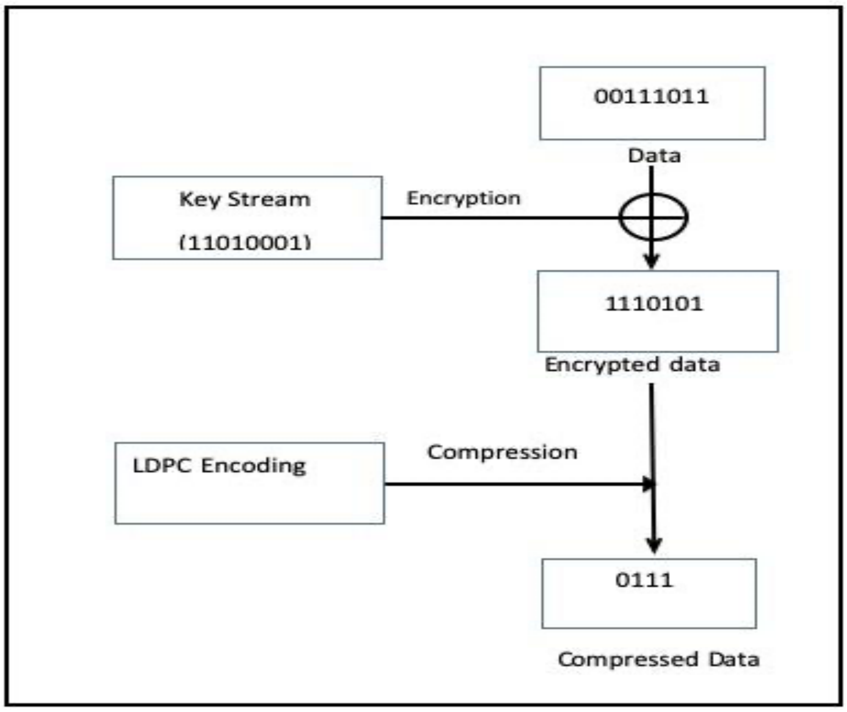


Figure 4.21: Data Transmission in IoT Device.

- **Receiving of Data in Data Stream Collector:** Figure 4.22 shows that after receiving the compressed data from IoT boarder router, data starts decompressing. The example shows the decompressing process for received 4-bit data to the original data. Each bit of the received data stream (0111) may have two outputs 01 and 10 for 1 and 00 and 11 for 0. Four bits are needed to decompress 1 byte and then combination of total 16 outputs are generated. The key stream XOR with the generated 16 outputs to perform decryption. With encryption key the data can be decrypted to correct value and then it decides the correct value by sensor characteristics of IoT device so the correct output is generated. In example the experiment used temperature value of data. Temperature range can be in 25 - 120◦C with SHT75 [125]. So, it is possible to remove the values which fall outside of the range.

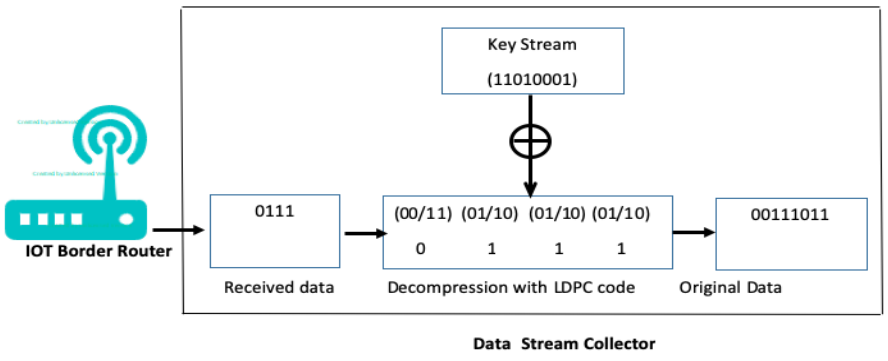


Figure 4.22: Receiving data in Data stream collector

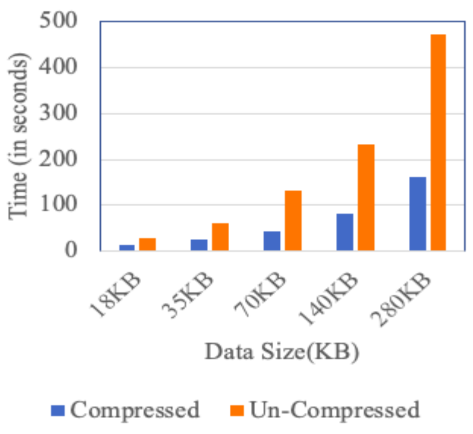
### 4.3.3. Implementation

IoT border router, data stream collector, were implemented using Raspberry Pi 4B, 6LoWPAN, BT-LE and IoT SDK. Python was used for sending and receiving programs and LDPC coding was used for compression. An Ethernet wired connection was established for the data stream collector. IoT SDK installed [126] for supporting BT-LE and 6LoWPAN.The dataset consists of eight gas, temperature and humidity sensors [127].

### 4.3.4 Result and Analysis

To prove the efficiency of LDPC code following parameters are used and analysed the result.

- **Transmission Time:** Figure 4.23 is the comparison between Transmission times for Compressed and Un-Compressed data. The transmission time calculated for compressed data does not include the time taken for the actual compression of the data.



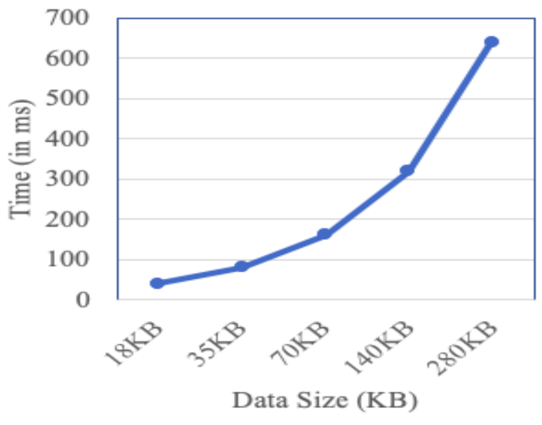


Figure 4.23: Comparison of Transmission Time Figure 4.24. Compression Time

- **Compression Time:** Figure 4.24 shows a graphical representation of the increase in compression time as the data size increases. As the dataset size increases, the compression time increases linearly. However, even for large amounts of data, the compression time is miniscule. The compression times for the smallest and largest data, 18 KB and 280 KB, take 0.05 and 0.64 seconds, respectively. Additionally, the transmission time for each data size significantly exceeds the compression time and adding the compression time to the transmission times of the compressed data yields the same results.

### 4.3.5 Summary

In this work, we proposed a handler for data streams that reduced the transmission time of data from an IoT device to a data stream collector. We employed LDPC coding to reduce the size of the actual payload and overcome the speed difference between the Internet and the IoT devices. The Raspberry Pi 4B and a laptop are used to implement an IoT device that can be used to generate stream data. We extracted the temperature data from a dataset created by the temperature sensor. Consequently, we have reduced the average transmission time by 65%. We examined the stream data of IoT devices, which can generate a range of values. The implementation of this method required prior knowledge of the device, the data set and their characteristics. The applications of IoT are diverse and the types of data collected are vastly different. This constrains the applicability of this study, as it is more appropriate for linear data sets. Future studies can solve this problem and make this study's results more general.

# CHAPTER 5

# IoT DATA STREAM ANALYSIS

This chapter represented the various methodology used in the IoT stream data analysis. The several methods implemented in the proposed IoT stream data analysis. For the purpose of predicting and analysing IoT stream data in real-time, we used the models and architecture. The performance study shows that the proposed strategy offers an IoT environment that is more effective.

## 5.1 Streamed Covid-19 Data Analysis using LSTM

We conducted this work during the global pandemic. Proposed the model to predict future COVID-19 cases using real-time IoT data from both government and non-government sources.

### 5.1.1 Overview

The architecture of long short-term memory (LSTM) is shown in Figure 5.1, which consists of three gates. We refer to these gates as the Input, Forget and Output Gates. The information entryway decides which new data to store in the drawn-out memory. It only processes data from current information and transient memory from the previous time step. Therefore, it must redirect the data away from these unhelpful factors. The Forget Gate determines whether to retain or discard data from the drawn-out memory. The Forget Gate completes this task by augmenting the approaching long-haul memory through an overlook vector, which it creates from the current information and the approaching transient memory. The current time step's yield can also be derived from this hidden state. LSTM uses feedback connections so that it can process the entire sequence of data with a single data point. People prefer LSTM over other techniques because it effectively handles continuous problems such as noise, distribution representation and values. Also, it provides more control. So, it is well suited for classifying and making predictions based on time series data. It models the system with many input variables and can control several parameters to make a prediction [128].

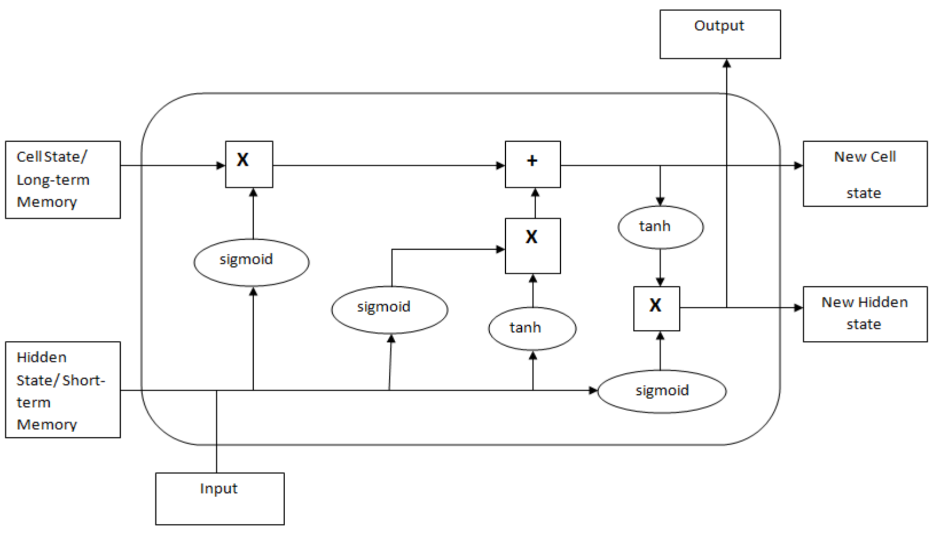


Figure 5.1: Architecture of LSTM

### 5.1.2 Proposed Model

This section includes a description of the proposed model, which includes the LSTM technique.

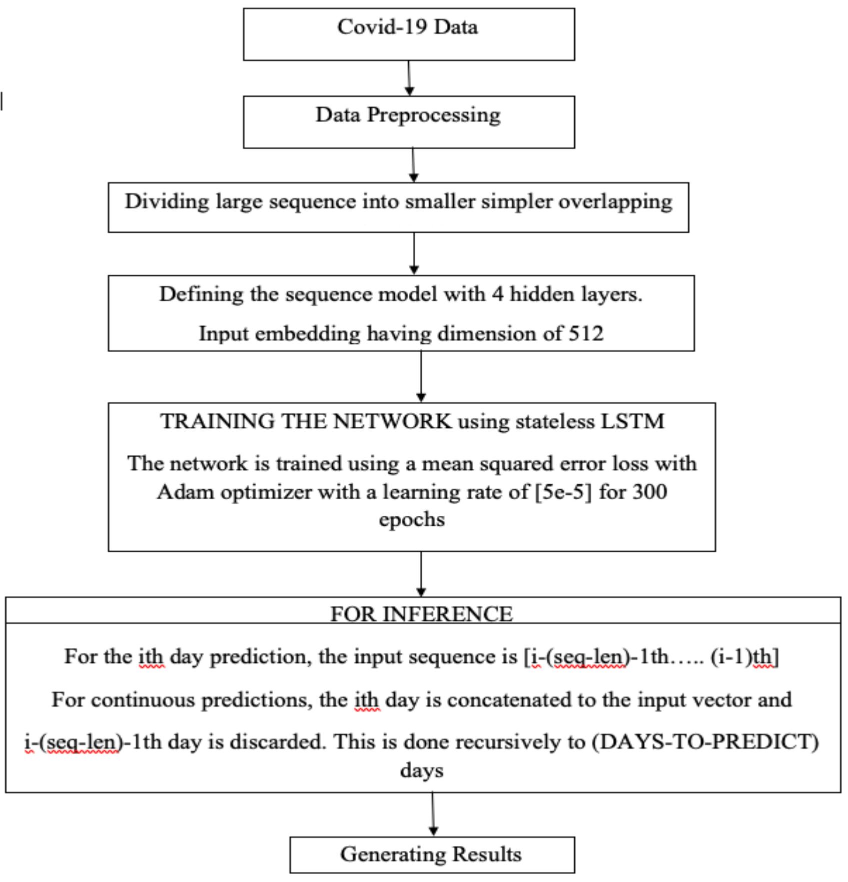

Covid-19 Data
Data Preprocessing
Dividing large sequence into smaller simpler overlapping
Defining the sequence model with 4 hidden layers.
Input embedding having dimension of 512
TRAINING THE NETWORK using stateless LSTM
The network is trained using a mean squared error loss with Adam optimizer with a learning rate of [5e-5] for 300 epochs
FOR INFERENCE
For the ith day prediction, the input sequence is [i-(seq-len)-1th….. (i-1)th]
For continuous predictions, the ith day is concatenated to the input vector and i-(seq-len)-1th day is discarded. This is done recursively to (DAYS-TO-PREDICT) days
Generating Results

Figure 5.2: Proposed Methodology

Figure 5.2 shows the flow of stream data through the proposed methodology. The first step in implementation is to prepare the data in a suitable format by pre-processing real stream data generated from a government source. Pre-processing also deals with null values. Subsequently, a sequence model with 4 hidden layers embeds 512-dimensional input. Finally, we can use stateless LSTM for training, training the network with mean squared error loss and an Adam optimizer. During the inference phase, we repeatedly feed the results to generate predictions.

### 5.1.3 Implementation

Implementation begins with preparing data suitable for training our model. Before proceeding, it's important to acknowledge two aspects of data exploration: the data may include unnecessary information, such as longitude and province latitude. We must first eliminate these unnecessary columns from the data. Additionally, because the data is in cumulative form, we need to reverse the accumulation process. Data should be checked for null or missing values. If the number of null values is low, we should remove the NA values. However, if the data contains a large number of NA values, we should fill it in with some dummy values. After handling null values, determine the total rows in the data and move on to the next step. The next step involves splitting the data into training and testing sets. The study utilized Scikit's Min Max scalar. To assess training speed and performance, learn to scale the data values for the training set into a range of 0 to 1. Lastly, begin the training and build the model.

**Constructor:** Used to initialize helper data to create the layers reset hidden state Stateless LSTM was used, therefore, we also need to reset the state. First, obtain the sequences and route them through the LSTM layer. Next, our linear layer processes the final time step output to derive the prediction. After training, we use the "fully trained" model to predict confirmed cases for approximately 12 days into the future. As before, we need to inverse the scaler transformation using the scalar function of Scikit Learn. After that, plot the desired results. Several parameters were taken for evaluation: Here, mean squared error is used to measure our training and test errors.

**Software used (Library/functions):** Python-3, Pandas, Matplotlib, NumPy, Cpython, Seabron ,Ipython ,Scikit learn ,Pylab.

### 5.1.4 Dataset

The cleaned dataset for India is used, which was proposed by John Hopkind. We obtained the data containing the number of confirmed cases, recovered cases and deaths worldwide from the repository of Johns Hopkins University. These three files contain time-series data for the real-time number of COVID-19 confirmed, recovered and death cases for all countries from January 22, 2020, to April 27, 2020. Every day, we update this repository with newly discovered cases. We selected only the data for India from these files. The corresponding figure shows the dashboard for the JHU dataset.

### 5.1.5 Results and Analysis

The proposed model can predict the following scenario for India during May 4 to 15, 2020.

#### 5.1.5.1 Confirmation of 20,000 Cured Cases in India

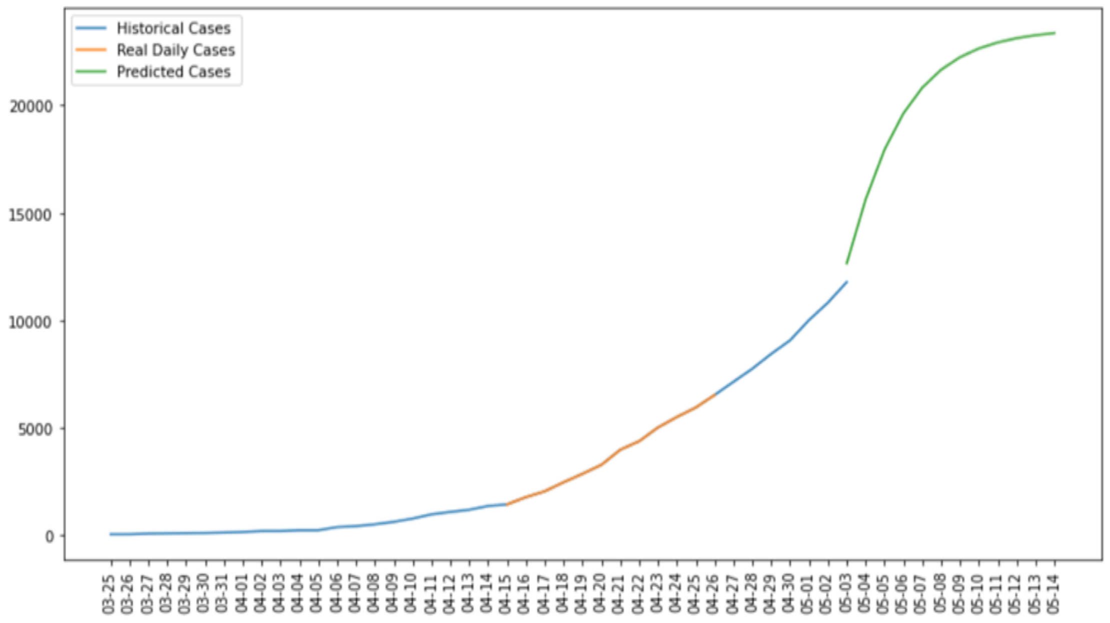


Figure 5.3: Confirmation of More than 20,000 Cured Cases in India

Figure 5.3 shows that till May 14, 2020, more than 20,000 cases will be cured in India. The corresponding graph shows the trend of recovered cases in India beyond May 6, 2020. Analysis of the result shows that the prediction of the period is almost close to the confirmed case.

#### 5.1.5.2 When Was the Earliest Case in India from 0 to 40,000 Cases?

Figure 5.4 shows that from 0 to 40000 cases, almost 100 days have passed. Within a span of 100 days, the number of death cases remains constant, while the number of confirmed cases significantly increases. There are fewer active cases compared to confirmed cases and fewer cured patients, but the number of deaths remains constant.

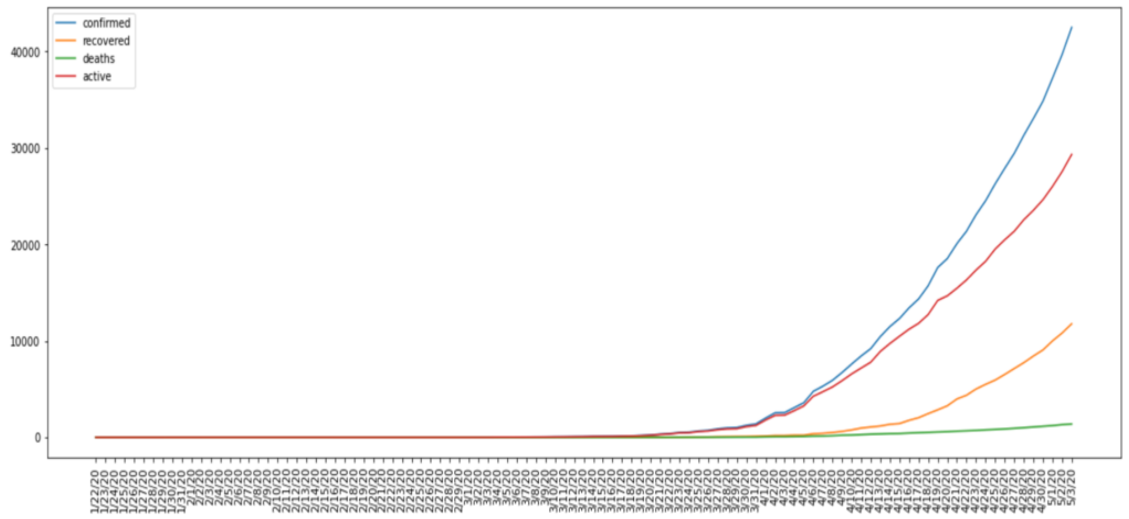


Figure 5.4 Earliest Case in India From 0 to 40000 Cases

### 5.1.5.3 Per Person Number of People are Infected:

The ratio R0 = $\boldsymbol{\beta}/\boldsymbol{\gamma}$ is known as the reproduction rate and it is necessary for a disease to have R0>1 such that it can spread in a large population otherwise it will die out. We have taken R0 as 2.5.

### 5.1.5.4 The Incubation Period of Virus

At that time, the average incubation period of the virus ranged from 1 to 14 days, most commonly around five days. During that period, also known as the "pre-suggestive" period, some tainted people can be infectious.

### 5.1.5.5 The Rank of State as Per Number of Confirmed Case

Figure 5.5 represents the rank of 20 states (to date) based on confirmed cases. The representation shows that Maharashtra has a maximum number of confirmed cases (more than 20,000) and Tripura, with a minimum number of cases, is at the end of the list. Death cases are almost constant in all cities, whereas they fluctuate.

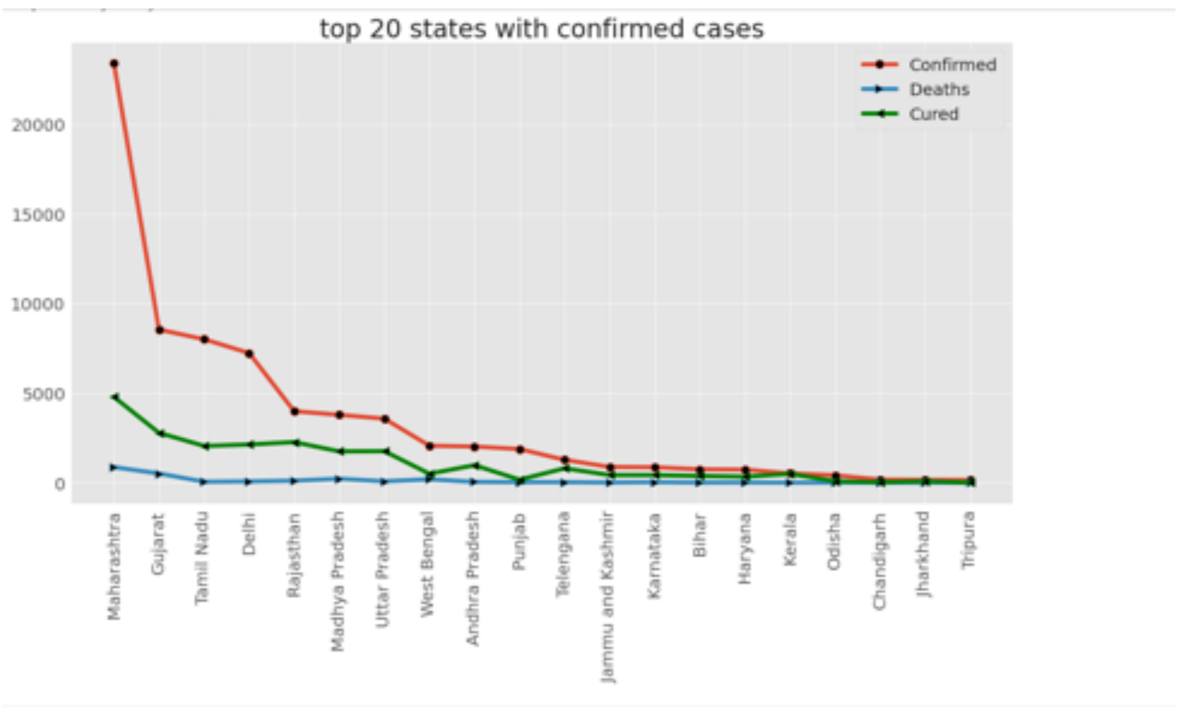


Figure 5.5: The Rank of State as Per Number of Confirmed Case

#### 5.1.5.6 The Gender-Wise and Age Group Distribution and Impact of COVID-19 in India

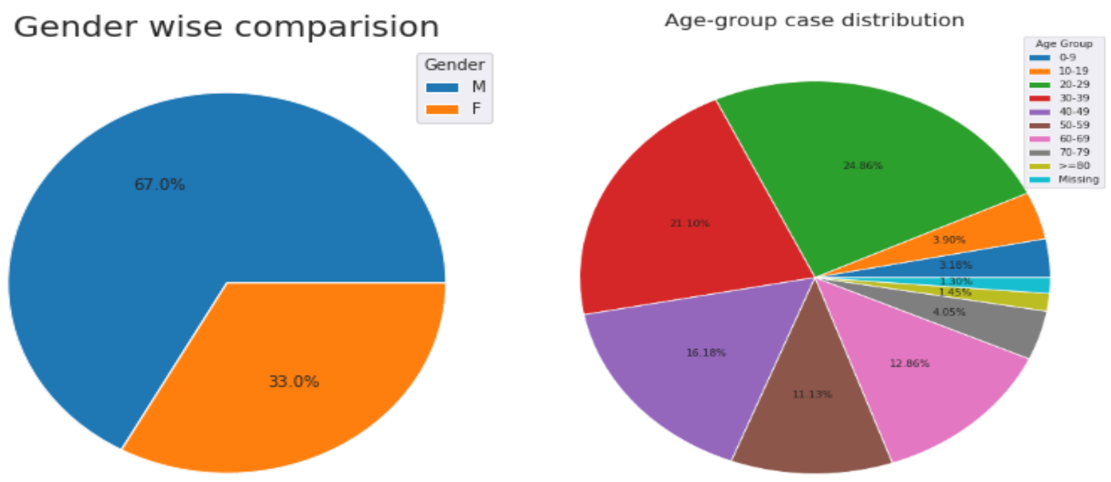
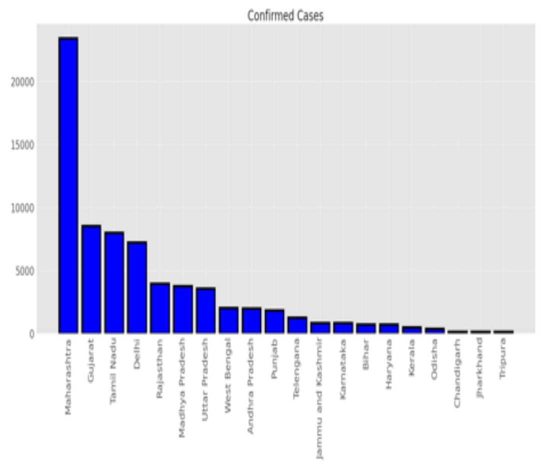


Figure 5.6: The Distribution of Cases Based on Gender and Age Group

Figure 5.6 shows the virus's gender-wise and age group impact in India. COVID-19 infected approximately 67% of males and 33% of females. The difference in infection rates can be attributed to the lifestyle choices of both genders. Reports pointed out that in India, males smoke more than females, which increases their infection rate. Additionally, due to genetic factors such as the X chromosome, women exhibit superior brain and immune system functions as compare to male. The results show that people in the age group 20–29 have a high risk of contracting the virus (24.86%)

### 5.1.5.7 Total Confirmed, Cured and Death Cases For 20 States In India



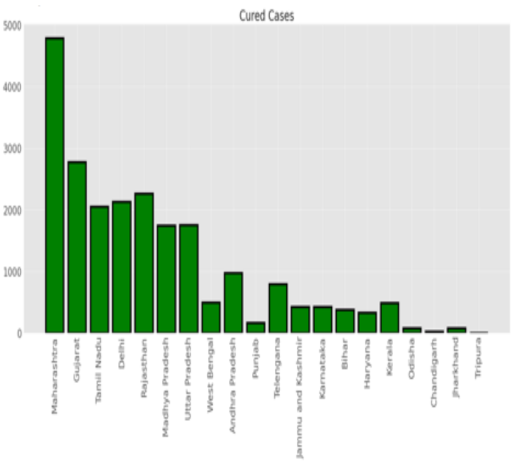


Figure 5.7: Total Confirmed Cases Figure 5.8: Total Cured Cases

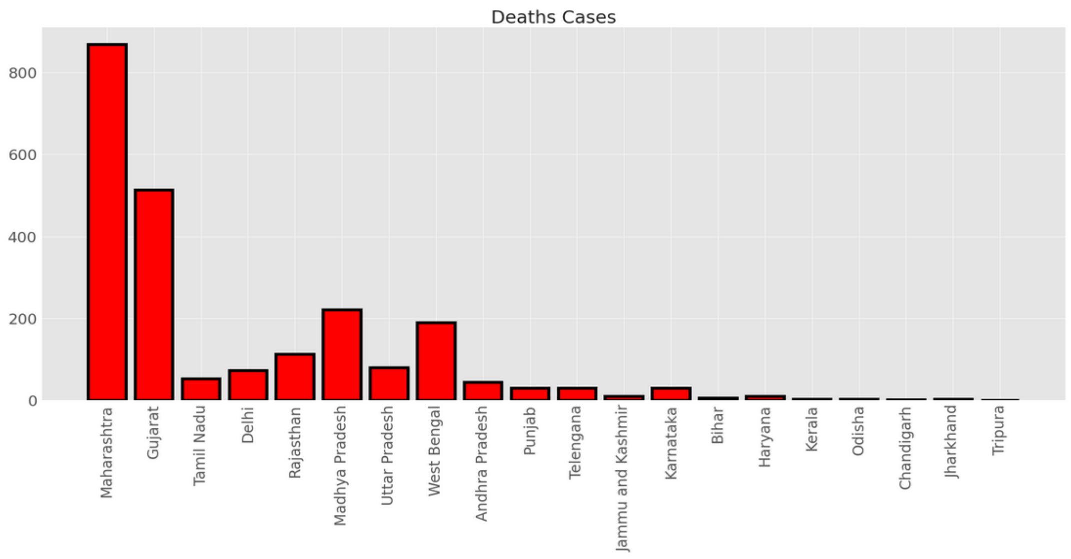


Figure 5.9 Total Death Cases

India's absolute caseload has become the world's second-biggest. Figures 5.7 and 5.8 reveal that five states - Maharashtra and Andhra Pradesh, followed by Tamil Nadu, Karnataka and Uttar Pradesh - originated and cured a significant portion of India's COVID-19 cases. Also, the active cases were high in those states and as given in Figure 5.9, the death rate was high in Maharashtra and Gujrat and trailed by Tripura, which matched the actual cases very nearly as predicted by our model for said duration.

### 5.1.6 Summary

COVID-19 is affecting a significant portion of the world's population and the vaccine is currently undergoing trials. However, the government has issued a guideline for precaution. The facts and figures show the impact of guidelines that have been followed by people [129] and how they affected the Indian people. This paper presents a prediction model that uses the Long Short Term Memory (LSTM) technique. This model is useful for predicting future COVID-19 cases based on the current situation. We can only address this pandemic situation through precautions until we discover a vaccine. In the proposed model, LSTM is well-suited for predictions because it is based on time series data. The experiments were conducted on the John Hopkins dataset for India, which is updated on a daily basis. The data for the month of January–April was taken and predicted for 12 days in May 2020. The results show that there was a very small error between the model and the curve of official data.

## 5.2 Grapes Apple Potato Leaf Disease Detection Network (GAP-LDDN)

In this work we have collected our own leaf disease dataset analysed and process to classify the category of potato, apple and grape plant disease using proposed model.

### 5.2.1 Overview

Leaves disease in various plants such as grapes, apple and potato significantly affect the production and therefore, early identification of the diseases is very important to prevent the crops from the negative side-effects. The complexities and learning requirements involved in plant disease diagnosis make this problem a prime candidate to utilize machine learning (ML) techniques for disease identification and classification. The focus of current research is on the early identification and treatment of plant leaf disease for IoT data stream. In this work, the grape, apple and potato leaf disease detection network (GAP-LDDN) is developed. It makes use of dual attention techniques to extract features, identify crop diseases and classify diseases. ResNet 101 with modified channel and spatial attention network is used to extract the features. Activation function of Tanh is used at every node of convolution neural network (CNN) with nonlinear bias function which overcome chances of over-fitting. The performance of proposed model is evaluated on grapes, apple and potato data-sets where, potato data stream is collected and prepared which contains images of the three different potato variety, lady rosetta, kufri chipsona-1 and kufri chipsona-3. This dataset is grown and collected from the village morta, district shajapur, Madhya Pradesh state of India in June 2022. Based on our experimentation, in comparison to ten state-of-the-art models, GAP-LDDN is able to detect diseases more accurately in grapes (esca, black-rot and isariopsis), apples (powdery mildew, ceder quince rust) and potatoes (brown rot and blackleg) disease dataset. As a result proposed GAP-LDDN model achieves 99.98%, 97.88% and 99.88% accuracy for grapes, apple and potato datasets respectively which is better than its competitors and also efficient in terms of detecting and classifying diseases from plant leaves. The significant contributions of proposed work are enlisted as:

- A new architecture for leaves disease detection is pro-posed using ResNet101 on the top of baseline ResNet architecture for feature extraction which concatenates with spatial and channel attention feature maps.
- A Dual-task model is introduced to detect and classify diseases of leaves of grapes, apple and potato plants. Dual task includes detection whether image of leaf contains diseases or not, if leaf contains diseases, then model recognizes the disease of particular plant. Conclusively, the proposed work performs two actions, hence, it is refereed as dual task model.
- The proposed model uses hyperbolic tangent function as activation function which helps in eliminating bias shifting problem exists in the state-of-the-art methods.
- The model's performance is evaluated on three data sets grapes, apple and potato where, datasets for grapes and apple are taken from plant village data-set and a data-set is separately prepared for potato which contains images of the three different potato variety," lady rosetta"," kufri chipsona-1" and" kufri chipsona-3", grown in the village morta, district shajapur, Madhya Pradesh state of India in June 2022.
- The proposed GAP-LDDN model achieves 99.98% accuracy for grapes dataset, 97.88% accuracy for apple dataset and 99.88% accuracy for potato dataset which is better than its competitors and also efficient in terms of detecting and classifying diseases from plant leaves as shown in Table 5.3, 5.4 and 5.5 respectively.

### 5.2.2 Proposed Work

This section of the paper elaborates the proposed apple, grape and potato plant leaves disease detection network model, which is developed over Faster R-CNN, multitask learning and multilevel extraction of features using spatial and channel-wise attention mechanisms. Figure

5.10 presents the workflow of proposed GAP-LDDN model. The components of proposed model as explained as follows:

### 5.2.2.1 Faster R-CNN

In comparison to R-CNN, faster R-CNN has been shown to be a well-performing network for object detection, notably, in terms of increasing detection speed and accuracy for small objects [130]. In grape, apple and potato data-set, there are several small objects which need to identify with high detection accuracy. As a result, our baseline detector is a Faster R-CNN implementation that is trained on ImageNet [131] and initialized with ResNet-50 weights. A two-stage detector called Faster R-CNN first extracts the common features from the input image before producing region proposals using a region proposal network (RPN). Finally, objects are detected from the region proposals which were cropped from the region of interest (ROI) pooling layer.

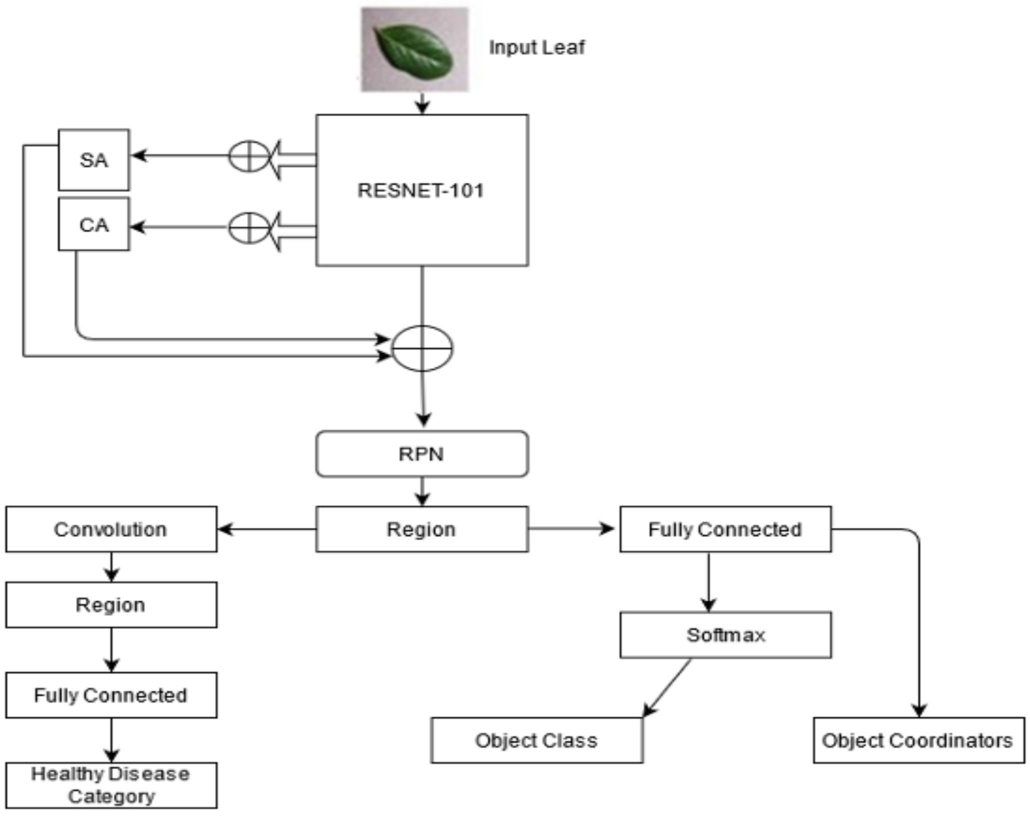


Figure 5.10 : Workflow of GAP-LDDN Model

### 5.2.2.2 Multilevel Features: Spatial Channel-Wise Attention

The research on effective region extraction and noise sup-pression for attention-based feature evaluation focuses on achieving the best performance. For additional regression and classification tasks, the proposed model computes spatial and channel attention weights using the most recent feature maps.

***Spatial Attention (SA):*** In contrast to other approaches, spatial-attention encompasses a number of positions over a sequence while at the same time assigning weights to the local feature map. Features at a deeper level contain additional edge data. Therefore, spatial attention is applied using lower levels of ResNet-101. SA weight assignment includes two stages, much like the research from [132], utilizing stacked convolutional layers, average pooling is used in the first stage to assess attention weights. This might help improve noise suppression and spatial aspects. The second stage involves multiplying the current feature map by the attention weights once the attention weights have been evaluated. Bi-linear interpolation is used to spatially down sample the output of Conv 2d by 3 and to channel-wise up sample it by 2 via convolution. Low level feature maps that have been improved for Conv 2d and Conv 3d are provided as input in the proposed work. Further, these are concatenated using element-wise addition to produce a spatial attention weight. Utilizing convolution, the channels of the concatenated map are compressed. After that, Tanh is used to add a non-linear activation to the concatenated feature map. In order to compute the spatial weight map μs, the output context is gated via the use of the sigmoid activation function.

$$\mu^s = \pi^s \times \mathrm{sigmoid}(w_1 \times \mathrm{TanH}(w_0 \times \pi^s + b_0) + b_1) \quad (5.1)$$

where, π is the reduced version of the map. This is accomplished by applying two deeper layers made up of $w_0$, $w_1$, $b_0$, $b_1$ respectively. The concatenated feature map π is readjusted using the attention weights in the following manner:

$$\delta^s = \pi^c_{1,1} \Theta \mu^s_{1,1} \dots \dots \pi^c_{w,h} \Theta \mu^s_{w,h} \quad (5.2)$$

where, $\delta^s$ is the re-calibrated feature map and w, g are the dimensions of the concatenated feature map in terms of width and height respectively, Θ is the re-calibration operator. The outcome re-calibrated map is additionally down-sampled spatial through 4x and up-sampled channel-wise to have ξ channels, where, ξ = 512 to retrieve the resulted feature map ϕ.

***Channel Attention (CA):*** High level traits are also necessary to distinguish between healthy and unhealthy leaves in the case of the detection of healthy leaves. High-level feature maps, however, may not effectively portray colour and shape information [133]. Therefore, in high-level feature maps, varying weights are given to different channels to preserve colour and shape information. The outcome of Conv 4b is spatially down-sampled by 2x using bi-linear interpolation and channel-wise up-sampled using 1×1 convolution. In this work, down-sampled improved Conv 4b and Conv 5b, which are added together using component-wise addition and used to calculate the weights of the channel-wise attention. Initially, average pooling is performed over spatial in the following manner as shown in equation 5.3.

$$\mu^{c} = \frac{1}{(w \times h) \sum_{h}^{w} \sum_{j=1}^{h} F_{i,j}^{c}} \qquad (5.3)$$

where, ”c” stands for a feature map that has been concatenated. By averaging across c in the spatial dimensions, c is achieved. The pooled map c overtaken by context gating and sigmoid activation function to derive the channel-wise attention weight map c, which is defined as equation 5.4.

$$\mu^{s} = \pi^{s} \times sigmoid(w_{2} \times \mathrm{TanH}(w_{3} \times \pi^{s} + b_{2}) + b_{3}) \qquad (5.4)$$

where, $w_2$, $w_3$, $b_2$, $b_3$ represent the weight value (w) and bias value (b) respectively and belong to fully connected layers of model. The integrated feature map $F^c$ is re-adjusted with the attention weights in the following manner:

$$\kappa^{s} = F_{1}^{c} \Theta \tau_{1}^{s} \dots \dots . F_{c}^{c} \Theta \tau \varsigma^{s} \qquad (5.5)$$

where, κ is the reconfigured matrix of feature and ς is the set of channels. The outcome feature matrix of ResNet-101, i.e., Conv 5d are added together with activated feature maps ϕ and κ to retrieve the feature map M, which is fed as input to the RP as shown in equation 5.5.

### 5.2.2.3 Multi-task Learning Based Diagnosis

Within the scope of this section, the identification and localization of leaves that as infected as distinct from leaves that are from healthy class is an objective. The input feature, which is designated by the letter F and is the same thing as the output feature of the feature map M, takes the form of n, w, h and c. In this case, n refers to the total number of regions of interest w and h refer to the width and height of the feature maps and c indicates the total number of channels. Unfold F as F = [F1, F2,..., $F_n$], where $F^i$ is the ith region of interest, then use average pooling to concatenate features that have been extracted from n different regions of interest. Where, Gc is the average value obtained by pooling the characteristics that were collected from the n different regions of interest shown in equation 5.6.

$$G^c = \frac{1}{N}\sum_{i=1}^{N} F_c^i \qquad (5.6)$$

In this case, the form of $G^c$ is denoted by 1whc. After that, a (3,3) kernel is used to leverage $G^c$ in order to convolution. In addition, average pooling is performed for each channel of $G^c$ and the resulting value, which is indicated by $V^c$ and may be stated in equation 5.7.

$$V^c = \frac{1}{w\,x\,h}\sum_{i=1}^{w}\sum_{j=1}^{h} G_{ij}^c \qquad (5.7)$$

Finally, $V^c$ is fed as input in 2 fully connected layers of the model and single SoftMax layer to categorise (using object coordinate regression) the health of the potato, apple and grape leaves. Given that the Faster R-CNN architecture includes infection detection, the function representing the losses in GAP-LDDN can be modelled as shown in equation 5.8.

$$Loss_{GAP}{-}LDDN = L_{RPN}(\{pi\}, \{fi\}) + LODN(w, u, t^u, v) + L_{NC}(m, n) \qquad (5.8)$$

We denote loss as the total loss of the proposed GAP-LDDN model that comprises LRPN (pi , fi) which represents loss of the region proposal network (LRPN), LODN(w, u, t, u , v) as the loss of the objects detection network (ODN), object detection block (ODB) and LNC(m, n) as loss network coverage (LNC).

Here, LRPN ($p_i$, $f_i$) and LODN (w, u, $t^u$, v) are almost the same to faster R-CNN architecture.

$$L_{RPN}(\{p_i\}, \{f_i\}) = \frac{1}{Ncls}\sum_i L_{cls}(pi, p_i^*) + \lambda_1\frac{1}{Nreg}\sum_i p_i^*\ Lreg(p_i, p_i^*)$$

$$L_{ODN}(w, u, t^u, v) = L_{cls}(w,u) + \lambda_2[\mu \geq 1]L_{reg}(t^u, v)$$

In this case, v represents the predicted object coordinate, whereas $t^u$ represents the actual object's coordinate and w represents the proposed category score value. In our system, $N_{cls}$ as the value of cls is considered to be 512 while the value of $N_{reg}$ is regarded to be 1024. In conclusion, the $L_{NC}$ value for the disease identification task may be written as shown in equation 5.9.

$$L_{NC}(m, n) = \lambda_3 \quad \sum_i m_i \log \frac{1}{n_i} \qquad (5.9)$$

In the $L_{NC}$, n and m denote the proposed and actual class score values respectively and $\lambda_1$, $\lambda_2$, and $\lambda_3$are balancing parameters used to normalize the loss expression LRPN and LODN.

The Algorithm 5.4 begins with iterating over the image dataset in line 1. In line 2, ResNet101 extracts feature maps MF by using anchors boxes. In line 3, attention channels are used to remove noise from feature maps, optimizing the next steps. Further, line 4 pass feature maps into RPN with extract regions where N the object is presented in the foreground region. After that,1ROI regularizes the object contained regions called CMF. After that, it passes these regions into a fully connected convolution In this case, the form of Gc is denoted by 1 w h c. After that, network to detect disease in leaf and represent its output and display healthy leaf if no infection is present on a leaf.

### 5.2.3 Experimental Study

To test the performance behaviour of the proposed model, be stated as follows: an experimental study is carried out in this section. This study briefs about the data-set, methodology and the comparative study of the proposed method along with base-line state of the art methods is also carried to verify its effectiveness.

- **Dataset Description:** This work is dedicated to perform the tests and experimental study on grape, apple and potato crops data-set. The experiments with grape images are carried out using a publicly available benchmark data-set, named Plant Village1 comprises 38 categories by species and disease. As shown in Figure 5.17 to 5.19, the dataset is divided into 4 classes: healthy, black-rot, esca and isariopsis, totaling around 4500 images. During the evaluation, the leaves images are divided at random into three datasets: the training dataset, the validation dataset and the test datasets.

**Algorithm 5.1: Comparable Result Computation**

*Comparable_result_computation*

**Input**: Dataset D, Annotations A
**Output:** count

```
1. Begin
2.     a = []
3.     c = []
4.     N = 0
5.     Load D & A  # Load is a built-in function to read data CSV files
6.     For i in D:
7.         c[i] = D[i]
```

```
8.    End
9.    For j in A:
10.       a[j] = A[j]
11.   End
12.   For i in D:
13.       N++
14.   End
15.       For i in D:
16.           For j in A:
17.               c[i] = a[j][0]  # storing names of files from JSON
18.               if a[j] == c[i]:
19.                   while count != cell(N - 1 / 2):
20.                       if "Train" directory exists:  # exists is a built-in function
   to check whether a folder exists or not
                              move(c[i], Train)
21.                           count++
22.                       else:
23.                           mkdir("Train")
24.                           move(c[i], Train)
25.                           count++
26.                       if count ≥ cell(N-1/2) and count ≤ N-1:
27.                               if Val directory exist:
28.                                   move(c[i],val)
29.                                   count++
30.                               else:
31.                                   mkdir("Val")
32.                                   move(c[i],Val)
33.                                   count++;
34.                               if count==0:
35.                                   return 0
36.                               End
37.                       End
38.                   End
39.               End
40.           End
41.    End
```

**Algorithm 5.2: Leaf Diseases Detection using Instance Base Segmentation**

*leaf diseases detection using instance base segmentation*

**Input:** Training images with annotation JSON file

**Output:** leaf disease classes

```
1.  Begin
2.      For each image i in os.listdir('Tr'):
3.          M = ResNet101(i)
4.          N++
5.          i = 0
6.          For Each i in M:
7.              M[i] = Sp[i] + Ch[i] + M[i]
8.          End
9.          RPN = tensorflow.keras.preprocessing.image()
10.         i = 0
11.         For i in M:
12.             M[i] = RPN(M[i])
13.             CMF[i] = M[i].Bi()
14.             count++
15.             If count == N:
16.               Begin
17.                   return Clsfy(CMF) = 0
18.               End
19.         End
20.     End
21. End
```

Generally, Kaggle datasets is pre-processed which may lead to drift the model. To avoid this, we collected datasets from real world fields. Figure 5.11, 5.12 and 5.13 present detailed understanding about datasets and their various classes. In the apple data-set, total number of diseased images are 3900 with two diseases: powdery mildew and ceder quince rust. Out of those, 2250 are used for training in which 1200 belongs to powdery mildew and 1050 images to ceder quince rust. Five hundred fifty are used for validation, out of these 300 images belong to powdery mildew and 250 images belong to ceder quince rust. Further, 1100 images are used for the test, out of these, 700 images belong to powdery mildew and 400 images belong to cedar quince rust. Figure 5.12 shows the sample images used for the apple data-set. Plant leaves images dataset is distributed randomly into train, validation and testing data-sets during the experimental evaluation.

**Algorithm 5.3: Test Scores Prediction**

*test scores prediction*

**Input**: Test folder path (Tt), L - list of files in current directory, N - count of files, P -location variable

**Output:** score

```
1.  Begin
2.      L = os.listdir(Tt)
3.      N = len(L)
4.      For i in os.listdir(Tt):
5.      p = Tt + '\' + i
6.      For j in N:
7.          cls = LDD(p)
8.          count++
9.      If count == N:
10.         For k in L:
11.             If k.endswith('json'):
12.                 Trb = pd.load(k)
13.                 score = mAP(Trb, cls)
14.             End
15.         End
16.     End
17.   Return score = 0
18. End
```

**Algorithm 5.4: GAP-LDDN Algorithm**

*GAP-LDDN algorithm*

**Input**: Image dataset

**Output:** leaf_health_status

```
1.  Begin
2.    For each Image i in Dataset:
3.      M = ExtractFeatureMapsUsingResNet101(i)
4.      M ← spatial +channelattention+M
```

```
5.        M = CombineFeatureMapsWithAttention(M)
6.        RPN_result = RunRPN(M)
7.        MF = SelectObjectsWithRPN(RPN_result)
8.        CMF = ROIALIGN(MF)
9.        leaf_health_status = ApplyConvolutionAndSoftmax (CMF)
10.       Return leaf_health_status
11.    End
12. End
```

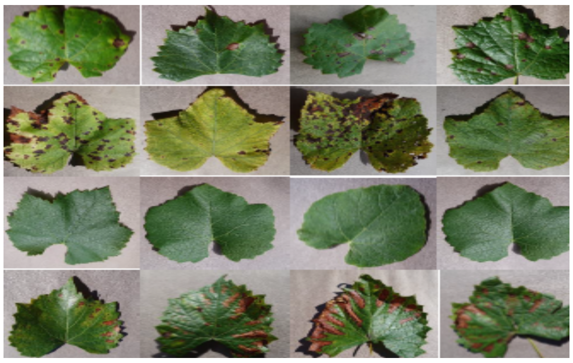


Figure 5.11: Samples Image of Grape Dataset [10].

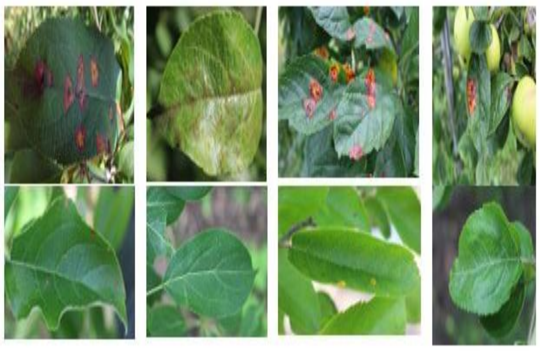


Figure 5.12: Samples Image of Apple Dataset

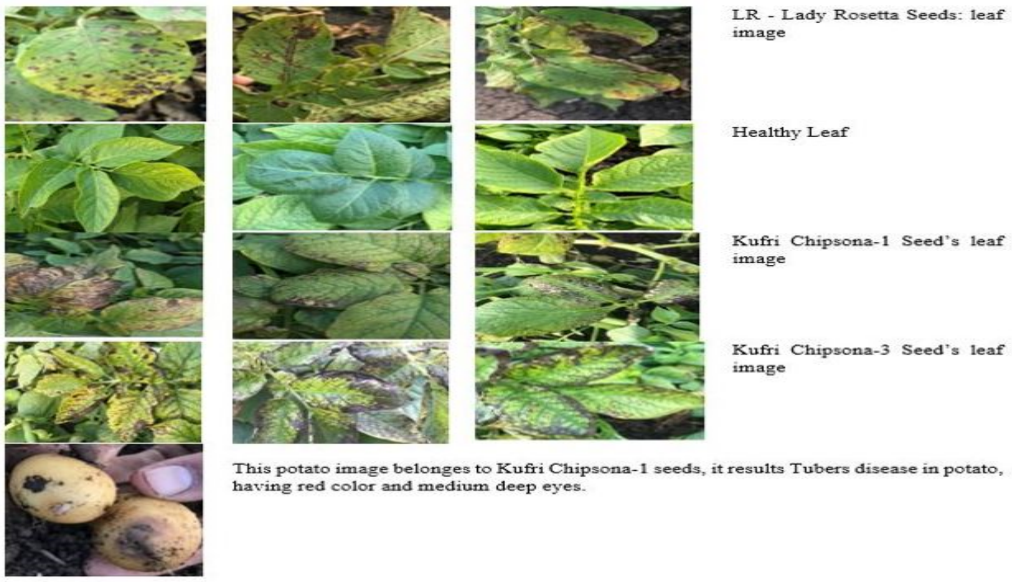
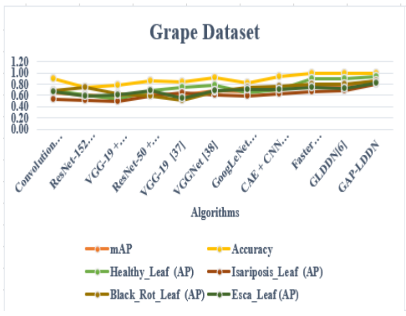
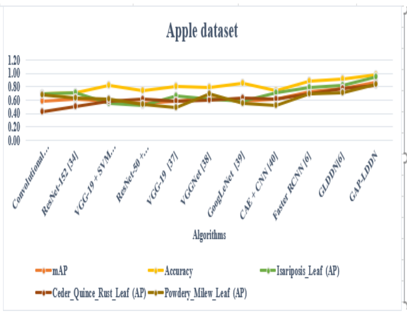

LR - Lady Rosetta Seeds: leaf image
Healthy Leaf
Kufri Chipsona-1 Seed's leaf image
Kufri Chipsona-3 Seed's leaf image
This potato image belonges to Kufri Chipsona-1 seeds, it results Tubers disease in potato, having red color and medium deep eyes.

Figure 5.13: Examples of Real Images of the Potato Dataset and Their Types in India

For potato, data is collected in Morta village, Shajapur district of MP state of India, considered three brand seeds of the potato dataset, such as LR (Lady Rosetta), Kufri Chipsona-1 and Kufri

Chipsona-3 as shown in Figure 5.15. details about first dataset, LR-Lady Rosetta is a specialist crisping variety, with high and low dry matters and reducing sugars. It is good for crisp quality output, either fresh or for short-term storage and has an early harvest maturity. It is low out-grades, moderate to high yields of round, uniform tubers and overall disease resistance. Information about Kufri Chipsona-1, the second dataset: Off-white, ovoid tubers have shallow eyes. Variety can be used to make French fries and chips since it is resistant to late blight. The yield ranges from 300 and 350 q/ha. Tubers are ovoid, crimson and have medium-deep eyes, according to Kufri Arun. The third dataset's specifics are as follows: Kufri Chipsona-3, north Indian plains 35 tonnes/ha of yield potential, resistant to late blight and good for French fries and chips. Medium-sized, round, white, quick eyes and flesh that is yellow. The crop matures in 90-110 days, it is generally found in grey colour in India. In the potato dataset, total diseases images are 5090 with two diseases: brown rot leaf (a fungal) disease and blackleg leaf (a fungus disease). Out of those, the 2880 images are used for training in which 2080 images belong to brown rot leaf and 800 images belong to lack leg leaf. Also, 570 images are used for validation in which 270 images belong to brown rot leaf and 300 images to blackleg leaf. The 1640 images are used for the test out of those 790 images belong to brown rot leaf and 850 images are used for blackleg leaf, a fungus. During the assessment, the leaf images are randomly split into training, validation and test data-sets as depicted in Figure 5.17 to 5.19. Each category's related target label has been marked. During the assessment, the leaf images are randomly split into training, validation and test data-sets.





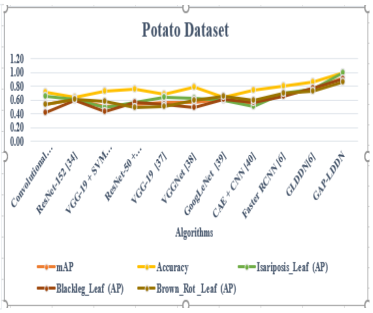


Figure 5.14: Distribution of Grape  Figure 5.15: Distribution of Apple  Figure 5.16: Distribution of Potato

### 5.2.4 Results Analysis

All the para-meters used by the model, in training, are reported in the Table 5.1, as the initial parameter epochs is considered as 0.001 because at the beginning, model requires more training epochs to make the changes in weights. However, to make faster training even on small training epochs, model tried multiple learning rates i.e., 0.01, 0.01 and 0.005. The significance of changing in learning rates is that as increase in learning rate, learning becomes faster and requires fewer epochs to make an update in any weight. Further, training step is defined as 60000 as ResNet101 model is used, as a fact, a deep network requires more steps to perform well in the training phase. As region of interest (ROI) algin is used after region proposal network (RPN), then model requires a short pixel side around 256 so that ROI align performs well in regularizing foreground regions taken out by RPN. The model is also tested on several weight-loss factors during training to analyse the training loss function at every back-propagation step and the accuracy is evaluated for the same. The proposed model uses weight loss factor 1.0 because the learning rate is low as 0.001 and it also uses 0.3 as the weight factor when using 0.005 as the learning rate. When the RPN is initially attempting to predict the positions of objects, the anchor boxes are responsible for providing a preset collection of bounding boxes in a variety of sizes and ratios that are used as a guide. These bounding boxes may be of any size. The model uses 1:4 scale ratio when intersection over union (IOU) ratio is around 30%, 1:3 scale ratio when IOU ratio is around 60% and 3:4 scale ratio when IOU is 75% which helps in object localization into the image. The model uses anchor scales based on anchor ratios when anchor ratio is 1:4 than scale is 8, when anchor ratio is 1:3 then scale is 16, and when anchor ratio is 3:4 then scale is 32. Pooling size depends upon feature maps, thus, the feature maps are extracted by ResNet101 using the pooling size 7.

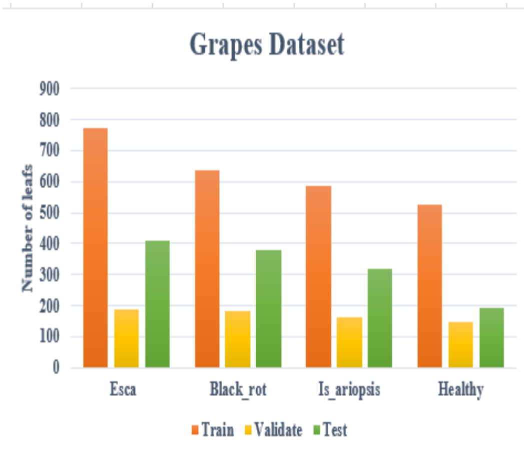
Table 5.2 shows accuracy versus weight-loss factor of Grapes, Apple and Potato. From this, it is observed that maximum accuracy is achieved at weight-loss factor 0.3, the accounted accuracy is 0.9991 for grapes dataset. The weight-loss factor affects the learning rate of the model as increase in the value of this factor increases the accuracy. Due to update in weights at every epoch, it becomes faster and the error starts decrease. It is also observed from the Table 5.2 that with the weight-loss 0.1 to 0.3, accuracy increases, however, when it reaches to 0.3 it starts decreasing because all the weights are already updated to its optimal values and further updating in weights leads to increase in error and decease in accuracy. Further, it is also observed that from 0.4 onward, accuracy starts decreasing. According to pattern, minimum accuracy is obtained at 1.0 weight-loss factor value that is 0.9943 and the difference between maximum and minimum accuracy is 0.0048. We observed this change in results from our reference work due to the environmental variables like graphics processing unit (GPU) capacity, clock speed of central processing unit (CPU), type of storage, as we are using cloud as service platform like google cloud platform (GCP) to access GPU service and high-speed CPU which make results to vary from reference work.

Table 5.1: Parameter Settings of Proposed GAP-LDDN Model

| | | |
|---|---|---|
| 1 | Initial learning rate | 0.001 |
| 2 | Learning rate | 0.001,.01.005 |
| 3 | Training step | 60000 |
| 4 | Shortest side pixel side | 256 |
| 5 | Loss weight factors | 1.0,0.3 |
| 6 | Anchor ratio | 1:4,1:3.3:4 |
| 7 | Anchor scales | 8,16,32 |
| 8 | Pooling size | 7 |

Table 5.2: Accuracy of GAP-LDDN Model for Weightloss Factor 3

| **Weightloss Factor 3** | **Grapes Dataset** | **Apple Dataset** | **Potato Dataset** |
|---|---|---|---|
| 0.1 | 0.9971 | 0.9823 | 0.9988 |
| 0.2 | 0.9982 | 0.9845 | 0.9989 |
| 0.3 | 0.9991 | 0.9887 | 0.9995 |
| 0.4 | 0.9981 | 0.9894 | 0.9997 |
| 0.5 | 0.9995 | 0.99 | 0.9999 |
| 0.6 | 0.9981 | 0.9899 | 0.9992 |
| 0.7 | 0.9961 | 0.989 | 0.9987 |
| 0.8 | 0.9958 | 0.9886 | 0.9986 |
| 0.9 | 0.9956 | 0.9881 | 0.9978 |
| 1 | 0.9943 | 0.9876 | 0.9964 |



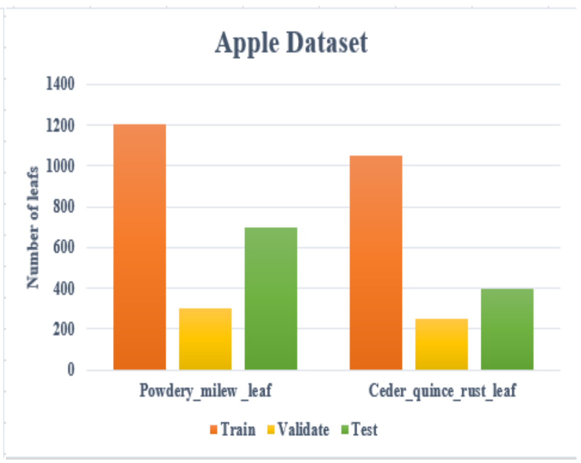


Figure 5.17: Graphical Representation of Grape Dataset

Figure 5.18: Graphical Representation of Apple Dataset

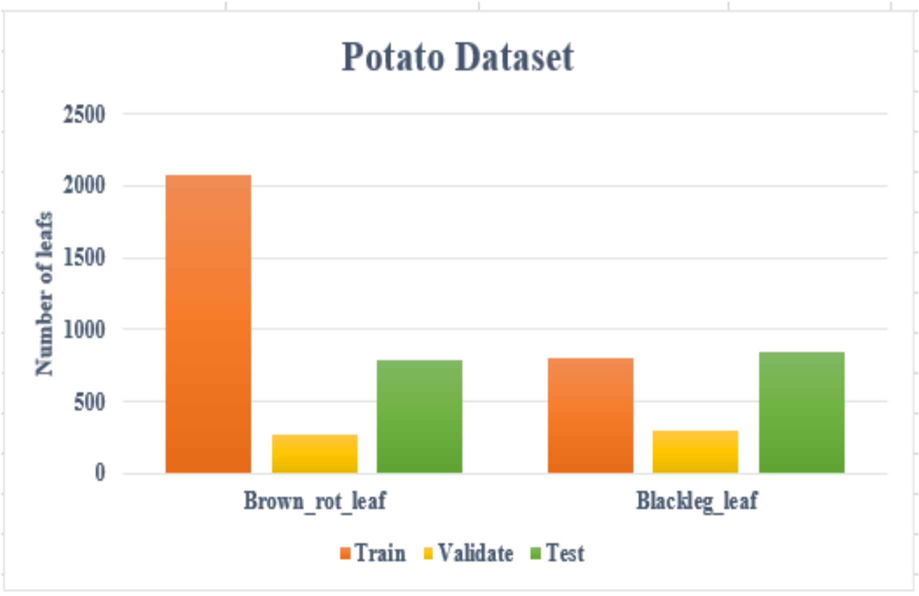


Figure 5.19: Graphical Representation of Potato Dataset

Figure 5.17, 5.18 and 5.19 show the graphical representation of grape, apple and potato data sets represent splitting of data-sets into training, validation and testing sets. The dataset is partitioned in parts, based on number of disease classes in that particular dataset. They presents the count of train, validation and testing in terms of the number of images for different datasets. Figure 5.14 to 5.16 helps in comparing mean average precision and accuracy value for different classes of diseases of leaves for the potato dataset; we observed that the GAP-LDDN model has a mean average precision value of 0.9144, which is 0.1582 higher than the GLDDN model and 0.2331 higher than Faster RCNN. We can also interpret that results of GAP-LDDN model are 17.3% better than GLDDN and 25.4% better than faster RCNN. Further, Accuracy is 0.9988 which is 0.1343 higher than GLDDN and 0.2002 higher than faster R-CNN. Figure 5.15 shows the comparative results corresponding to mean average precision and accuracy value for different classes of leaves for the apple data-set. Figure 5.16 compares mean average precision and accuracy value for different classes of leaves for the potato dataset. From the results, it is also observed that the proposed GAP-LDDN model is 10.7% better than GLDDN and 16.6% better than faster R-CNN in terms of mean average precision. From the results in Table 5.5, one can easily observe that the GAP-LDDN model outer-performs the comparative exiting methods for all three datasets. This can be a candidate solution for other similar applications.

The observation values of accuracies of various datasets first increases from weight loss factor 0.1 to 0.5 then these values starts decreasing, therefore, maximum values of accuracy for all datasets (grapes, apple and potato) achieves its peak value on 0.5, the reason behind is that as weight loss value increases learning moves faster towards optimization values but once optimal value is achieved, effect of weight loss on accuracy starts losing its strength which is also observed in the accuracy of all datasets. Thus, the maximum value of accuracy on grapes datasets is 0.9995, for apple, it is 0.99 and for potato, is 0.9999. All these values are achieved at weight loss factor of 0.5. Table 5.6 shows the comparative results of 10 existing state of art methods and proposed GAP-LDDN in terms of average precision (AP), mean average precision (mAP) and accuracy. The lowest average precision values is 0.54 and 0.51 for Esca leaf and Black Rot leaf class of diseases for VGG-19 method and 0.49 for Isariposis Leaf disease class for VGG-19 + SVM method. The proposed model because spatial and channel-wise attention is not concatenated with the feature maps extracted by the ResNet architecture of the model. Whereas, in the GLDDN model, these values are better than Faster RCNN but lower than the proposed GAP-LDDN model, this is due to the bias shifting problem in Relu function and low ResNet18 architecture used in the model. All issues are tackled in the GAP-LDDN model due to which accuracy and precision values improved in the proposed GAP-LDDN model. The average precision value of the proposed model is 0.8213 for esca leaf, 0.8564 for a black-rot leaf, 0.7945 for isariposis leaf and 0.9278 for a healthy leaf which is greater than 10 existing methods due its capabilities of handling false positives. Table 5.3 shows results of proposed GLDDN model and their comparison with existing 10 models in terms of AP, mAP and accuracy. The average precision value of our proposed model is 0.8312 for powdery mildew leaf,0.8256 for ceder quince rust leaf and 0.9476 for a healthy leaf. From the results, it is observed that apple dataset on GAP-LDDN gives around 9% of improvements in all evaluated metrics which shows that irrespective of images, the proposed model is able to perform well over existing ten models. Table 5.3, 5.4, 5.5 shows results proposed GAP-LDDN methods in terms of average precision (AP), mean average precision (mAP) and accuracy. Changes in the numeric values relatively to grapes and apple dataset are reflected due to a number of samples of images used in various splits of datasets and the dataset was taken from different climate zone and lighting from the different device as well, which create a difference in the image quality and dataset and affect the results. The average precision value of our proposed model is 0.8578 for brown rot leaf, 0.8987 for blackleg leaf and 0.9876 for a healthy leaf. The GAP-LDDN for potato dataset gives 7% higher scores in comparison to all existing techniques.

Table 5.3: Results on Grape Dataset

| **Methods** | **Esca Leaf (AP)** | **Black Rot Leaf (AP)** | **Isariposis Leaf (AP)** | **Healthy Leaf (AP)** | **mAP** | **Accuracy** |
|---|---|---|---|---|---|---|
| Convolutional Encoder Network [131] | 0.67 | 0.69 | 0.53 | 0.68 | 0.64 | 0.89 |
| ResNet-152 [133] | 0.59 | 0.75 | 0.51 | 0.61 | 0.61 | 0.74 |
| VGG-19 + SVM | 0.61 | 0.62 | 0.49 | 0.53 | 0.56 | 0.79 |
| ResNet-50 + SVM | 0.68 | 0.59 | 0.58 | 0.68 | 0.63 | 0.85 |
| VGG-19 | 0.54 | 0.51 | 0.64 | 0.75 | 0.61 | 0.83 |
| VGGNet | 0.69 | 0.67 | 0.61 | 0.78 | 0.68 | 0.91 |
| GoogLeNet | 0.70 | 0.74 | 0.59 | 0.64 | 0.66 | 0.82 |
| CAE+CNN | 0.71 | 0.77 | 0.62 | 0.71 | 0.70 | 0.93 |
| Faster R-CNN | 0.7325 | 0.7984 | 0.6709 | 0.8871 | 0.7717 | 0.9985 |
| GLDDN | 0.7295 | 0.8031 | 0.6823 | 0.8906 | 0.7754 | 0.9993 |
| **GAP-LDDN** | **0.8213** | **0.8564** | **0.7945** | **0.9278** | **0.8512** | **0.9998** |

Table 5.4: Results on Apple Dataset

| **Methods** | **Powdery Mildew Leaf (AP)** | **CederOuince Rust Leaf (AP)** | **Healthy Leaf (AP)** | **mAP** | **Accuracy** |
|---|---|---|---|---|---|
| Convolutional Encoder Network | 0.68 | 0.42 | 0.69 | 0.59 | 0.69 |
| ResNet-152 [133] | 0.63 | 0.51 | 0.71 | 0.61 | 0.71 |
| VGG-19 + SVM | 0.62 | 0.58 | 0.56 | 0.58 | 0.83 |

| | | | | | |
|---|---|---|---|---|---|
| ResNet-50 + SVM | 0.54 | 0.61 | 0.52 | 0.55 | 0.74 |
| VGG-19 | 0.49 | 0.59 | 0.67 | 0.59 | 0.81 |
| VGGNet | 0.69 | 0.60 | 0.62 | 0.63 | 0.79 |
| GoogLeNet | 0.56 | 0.64 | 0.59 | 0.59 | 0.86 |
| CAE+CNN | 0.52 | 0.62 | 0.72 | 0.62 | 0.75 |
| Faster R-CNN | 0.7011 | 0.6963 | 0.7889 | 0.7217 | 0.8923 |
| GLDDN | 0.7175 | 0.7823 | 0.829 | 0.7722 | 0.9211 |
| **GAP-LDDN** | **0.8312** | **0.8256** | **0.9476** | **0.8654** | **0.9788** |

Table 5.5: Results on Potato Dataset

| **Methods** | **Brown Rot Leaf (AP)** | **Blackleg Leaf (AP)** | **Healthy Leaf (AP)** | **mAP** | **Accuracy** |
|---|---|---|---|---|---|
| Convolutional Encoder Network [131] | 0.54 | 0.41 | 0.65 | 0.53 | 0.71 |
| ResNet-152 [133] | 0.61 | 0.59 | 0.61 | 0.60 | 0.64 |
| VGG-19 + SVM | 0.58 | 0.43 | 0.49 | 0.50 | 0.73 |
| ResNet-50 + SVM | 0.49 | 0.56 | 0.57 | 0.54 | 0.75 |
| VGG-19 | 0.51 | 0.54 | 0.64 | 0.56 | 0.69 |
| VGGNet | 0.59 | 0.49 | 0.62 | 0.57 | 0.78 |
| GoogLeNet | 0.65 | 0.61 | 0.59 | 0.62 | 0.64 |
| CAE+CNN | 0.60 | 0.56 | 0.51 | 0.56 | 0.74 |
| Faster R-CNN | 0.6987 | 0.6462 | 0.6988 | 0.6813 | 0.7986 |
| GLDDN | 0.7287 | 0.7762 | 0.7549 | 0.7562 | 0.8645 |
| **GAP-LDDN** | **0.8578** | **0.8987** | **0.9876** | **0.9144** | **0.9988** |

Table 5.6: Comparison of the Existing Models with Proposed GAP-LDDN Model

| **Algorithm** | **Plant Leaf** | **Disease** | **Accuracy** |
|---|---|---|---|
| Convolutional Encoder Network [131] | Grape | Multiple | 0.71 |
| Convolutional Encoder Network [131] | Apple | Multiple | 0.67 |
| Convolutional Encoder Network [131] | Potato | Multiple | 0.66 |
| ResNet-152 [133] | Grape | Multiple | 0.76 |
| ResNet-152 [133] | Apple | Multiple | 0.81 |
| ResNet-152 [133] | Potato | Multiple | 0.74 |

| | | | |
|---|---|---|---|
| VGG-19 + SVM | Grape | Multiple | 0.69 |
| VGG-19 + SVM | Apple | Multiple | 0.75 |
| VGG-19 + SVM | Potato | Multiple | 0.71 |
| ResNet-50 + SVM | Grape | Multiple | 0.86 |
| ResNet-50 + SVM | Apple | Multiple | 0.84 |
| ResNet-50 + SVM | Potato | Multiple | 0.86 |
| VGG-19 | Grape | Multiple | 0.79 |
| VGG-19 | Apple | Multiple | 0.81 |
| VGG-19 | Potato | Multiple | 0.78 |
| VGGNet | Grape | Multiple | 0.82 |
| VGGNet | Apple | Multiple | 0.69 |
| VGGNet | Potato | Multiple | 0.79 |
| GoogLeNet | Grape | Multiple | 0.76 |
| GoogLeNet | Apple | Multiple | 0.71 |
| GoogLeNet | Potato | Multiple | 0.72 |
| CAE + CNN | Grape | Multiple | 0.73 |
| CAE + CNN | Apple | Multiple | 0.76 |
| CAE + CNN | Potato | Multiple | 0.87 |
| Faster R-CNN | Grape | Multiple | 0.7986 |
| Faster R-CNN | Apple | Multiple | 0.8923 |
| Faster R-CNN | Potato | Multiple | 0.9985 |
| GLDDN | Grape | Multiple | 0.9993 |
| GLDDN | Apple | Multiple | 0.9211 |
| GLDDN | Potato | Multiple | 0.7986 |
| **GAP-LDDN** | **Grape** | **Multiple** | **0.9998** |
| **GAP-LDDN** | **Apple** | **Multiple** | **0.9788** |
| **GAP-LDDN** | **Potato** | **Multiple** | **0.9988** |

### 5.2.5 Summary

In this work, the proposed GAP-LDDN model suggests a novel method for the classification of grapes, apple and potato leaves as healthy or diseased using faster R-CNN as baseline architecture. The feature matrix retrieved from aggregated attention mechanism is given as input into RPN for multitask learning which categorizes the defected leaves to determine defected regions from these leaves. Multi-task learning involves pooling layers by cropping defected region proposal to detect the object and classify them to decide whether it belongs to infected or not. The study is centred on a unified architecture that includes multilevel feature matrix development based on attention, object recognition and classification. The results obtained through a set of experiments show that the method is effective as it obtains accuracy 99.98%, 97.88% and 99.88% for grapes, apple and potato datasets respectively.

## 5.3 Identification of Biofuel Producing Plants Using IoT and Machine Learning

This is the third work done under IoT data stream analysis subsection in which the work done for identification of biofuel producing plants using IoT and Machine Learning.

### 5.3.1 Overview

The fast-growing society needs energy which could be generated from plants like corn to get converted to ethanol and to be used as fuel. Due to the increasing use of fossil fuels in automobiles and industrial sectors, in the past few years, the world has started facing severe problems like environmental pollution, ozone layer depletion, global warming. Therefore, we implement a model on identification of biofuel plants using machine learning algorithm to help to make use of efficient green energy. Biofuels can replace the present energy crisis and further help in reducing the usage of fossil fuels and global warming. Utilization of biofuels results in the reduction of global warming and also helps in maintaining the demand and supply levels of fossil fuels. Our present work gives an idea to identify different types of plants which can be used to produce biofuels using IoT which consists of machine learning methods using Mask R-CNN (an improved version of Faster R-CNN), that helps in classifying different plants present in real-world that can be used as an alternative for fuel and classify them on the basis of procreated data set.

### 5.3.2 Proposed System

The Figure 5.20 explains the working of our system which ultimately helps to distinguish between those plants that can be used for the production of biofuel from those which cannot be utilized for the same. The process begins by taking a picture of the crop we want to test. This is given as an input to our model that works in two steps, the first being the identification of plant correctly using Mask R-CNN and the second being the checking done by system to verify weather that particular plant produces biofuel or not. This makes use of a database that contains information about plants that are used for production of biofuel.

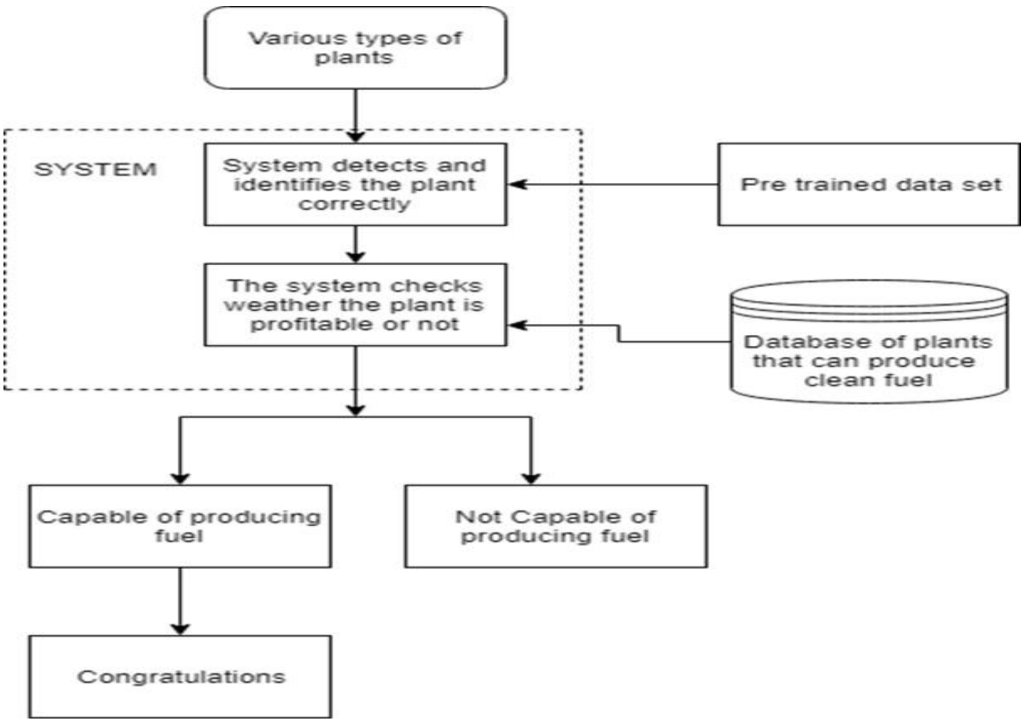

Various types of plants
SYSTEM
System detects and identifies the plant correctly
Pre trained data set
The system checks weather the plant is profitable or not
Database of plants that can produce clean fuel
Capable of producing fuel
Not Capable of producing fuel
Congratulations

Figure 5.20 : Flow of Proposed Approach

#### 5.3.2.1 Biofuel Process

A very successful example of using plants as a source of clean fuel is the production of ethanol from corn. Corn ethanol is ethanol generated from corn biomass and is the primary resource of ethanol fuel in the United States. Around 40.5% of the U.S. corn farms are utilized for ethanol manufacturing. The process starts from harvesting corn from farms in a surplus amount which then pass through the testing phase to check the quality of corn. The extracted corn passes through a hammer-mill than grinds the corn. The obtained product is transferred to slurry tank with the addition of yeast for fermentation. The process is followed by fermentation and distillation respectively which results in the formation of ethanol. The following Figure 5.21 gives a step-by-step representation of the whole process that is responsible for production of ethanol from the corn plant.

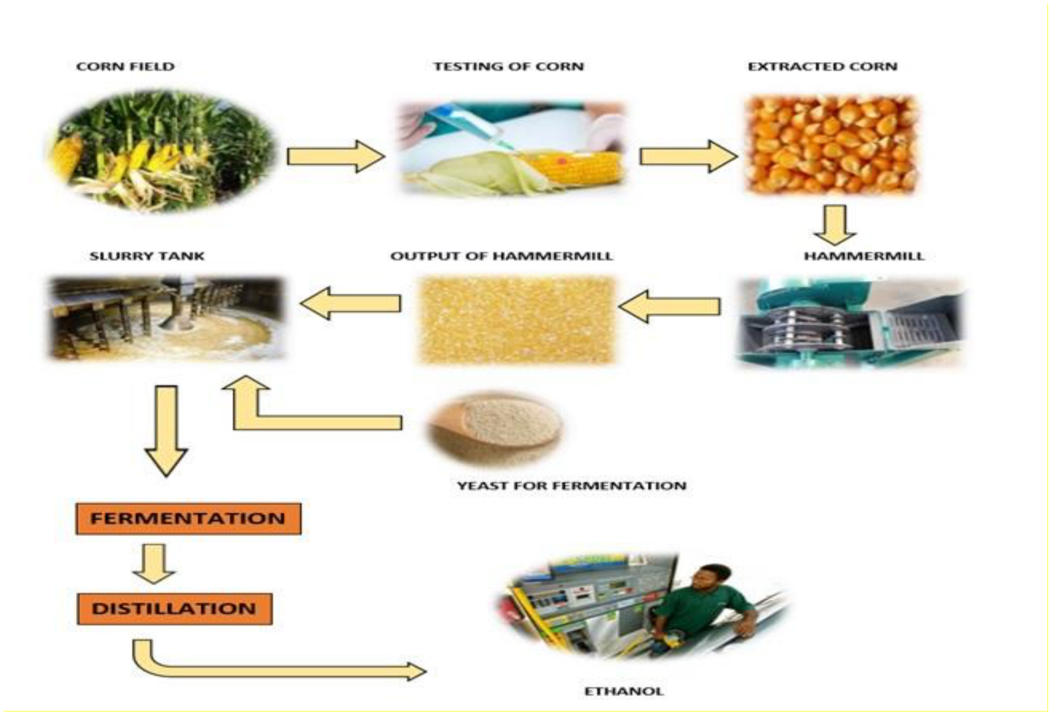
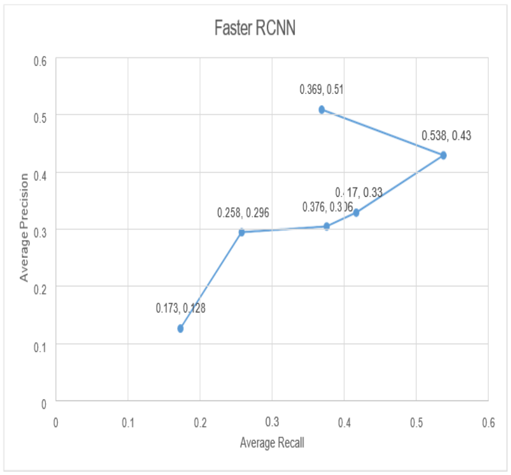


Figure 5.21: Conversion of Corn to Ethanol

### 5.3.2.2 Experimental Results

Mask R-CNN with ResNet 101 architecture to extract features from the images in Mask R-CNN as shown in Figure 5.22. Scatter view of graph between average precision & average recall for Mask RCNN and Faster RCNN shown in Figure 5.23.



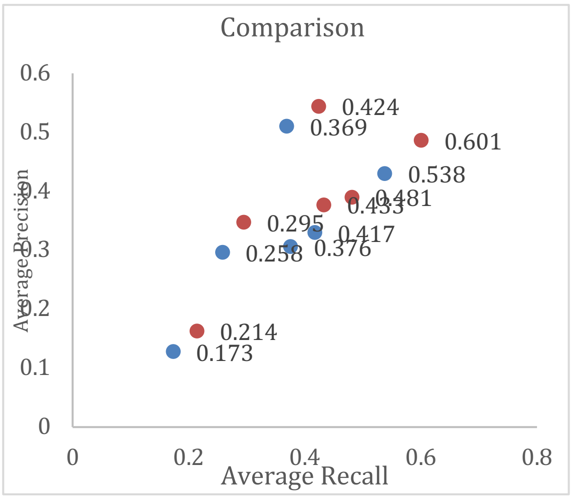


Figure 5.22 Average Precision & Average Recall Values for Faster RCNN

Figure 5.23 Average Precision & Recall for Mask RCNN and Faster RCNN

### 5.3.3 Summary

Second-generation of biofuels are amongst some of the low-carbon alternatives for road transportation which currently prove to be commercially possible or are within an initial commercialization phase. They hence are desirable options for reaching environmental goals. Concerning the case of Sweden, our team investigates cost-effective utilization of the cleaner fuels in inland transportation supporting the CO2 reducing objectives till 2050, moreover the outcomes of this implementation regarding purposes concerning an approximately conventional fuel-free road transportation area till 2030. Our team employs this bottom-up, optimization MARKAL Sweden design,  that includes the complete Swedish power system also concerning the transportation area. Towards $CO_2$ declines of about 80% till 2050 under the Swedish Energy Scheme as a sum. The consequences concerning the chief situation determine the annual growth scale concerning land transportation cleaner fuels of around 6% from 2010 till 2050, including biofuels considering to 78% of land transportation ultimate power practice during 2050. The selected cleaner fuel options hold methanol plus bio-methane. When proposing further conventional fuel diminishing strategies for land transportation (80% until 2030), an increase of about double the increase rate until 2030 is demanded.

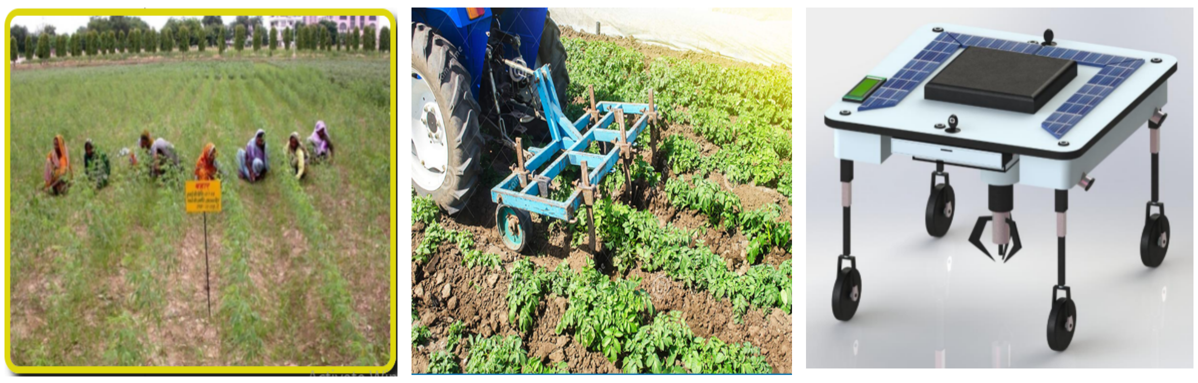
# CHAPTER 6

# APPLICATIONS OF PROPOSED WORK

This chapter represents the application of the IoT stream data handling and analysis titled "Smart autonomous weed detection and removal system". The proposed innovative work is patented in agriculture domains and in process to translate the research at commercialize stage with the help of collaboration with industry.

## 6.1 About the Invention

The invention relates to a smart autonomous weed detector and removal system which particularly to eliminate man power and use of herbicides in removal of weed in agriculture sector. The Figure 6.1(a, b) shows the existing weed detection and removal system in India and Figure 6.1(c) represents proposed AI and IoT based solution.



a) Manual System b) Mechanical Cultivator c) Proposed System

Figure 6.1: Existing V/s Proposed System

## 6.2 Overview

Weeds are small plants adjacent to crops. Their common types are grassy, broadleaf and small leaves which absorb nutrients from original crops and create hurdles in the growing phase of the original plant. Farmers need to remove these weeds to enhance crop productivity and they employ a variety of methods to do so, including manual labour to remove the weeds from the soil and the use of herbicides to kill them at their source. The current solutions lack efficiency in terms of cost, time and energy because they don't target weeds in their early stages. Once the weed reaches maturity, these methods become applicable, leading to a decrease in efficiency and an increase in effort. Therefore, there is a need for methods that depend upon renewable energy sources such as solar panels, which make them efficient in terms of energy and also having backup power such as lithium-ion batteries, which provide independence from power failure issues. The required method must use the latest technology, which makes it efficient in terms of time and cost and also be a technology-oriented method to remove and detect weed.

This smart weeder includes a multisensory information acquisition module that collects data from various sensors, such as ultrasonic sensors. Machines mount infrared sensors and utilize their data to initiate actions. The central processing unit, which is GPU-enabled processing hardware named NVIDIA Jetson Nano, performs weed detection tasks using the instance segmentation method. The system utilizes a robotic arm to remove weeds from the ground, while a suction tube collects the weeds from the robotic arm and deposits them into the cutting assembly. The system employs a weeder grass cutting blade drive arrangement to both cut and destroy the weeds. Weeder vehicle wheel drive device that manages the movement of the machine, hydraulic suspensions and excels that increase and decrease the height of the machine according to the type of weed. Sensors communicate with the processing unit to provide information about the robot's position and they also detect several barriers in the robot's path during its movement in the field. Smart weeders use the Mask R-CNN algorithm through the camera to detect and recognize plants in real time. To remove the weeds, a robotic arm extracts the weed from the root itself. Its dependency on solar power makes the robot completely autonomous in terms of energy. We used the Cranesbill weed dataset to test and train the model, dividing it into 60 and 40 ratios for training and testing, respectively (60% for training and 40% for testing). During the testing phase, it detects weed with an accuracy of 98.2%, which proves its efficiency in the field of weed. Despite this, the proposed model maintains a similar accuracy when trained on various types of weed.

## 6.3 Objectives of the Invention

- To detect and remove weed stuff automatically.
- To eliminate the use of herbicides in the weed-removing process.
- To diminish manpower efforts in the weed-removing process in congested or risk-filled fields.
- To avoid soil fertility degradation via the use of unwanted chemicals in the weed-removing process.
- To offer a low-cost and effective method for weed removal.
- To create a data repository of images of various weeds that can assist in further weed-related research analytics for practitioners.
- To offer an effective and weather-relevant condition-less weed removal solution.

## 6.4 Brief Description of the Drawings

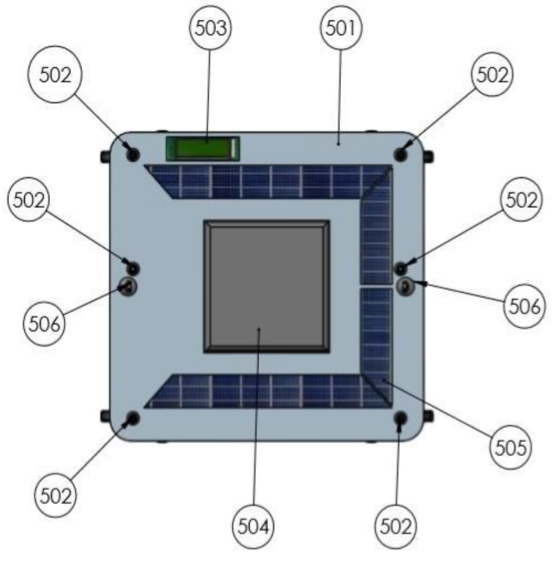


Figure 6.2: Top View of Smart Autonomous Weed Detection and Removal Machine

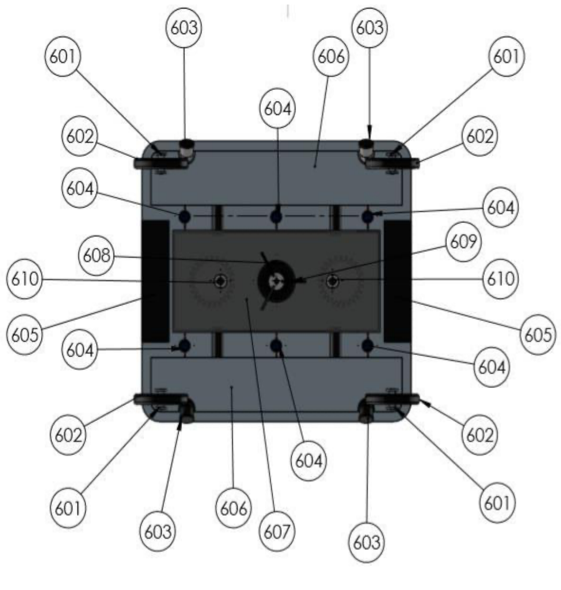


Figure 6.3: Bottom View of Smart Autonomous Weed Detection and Removal Machine

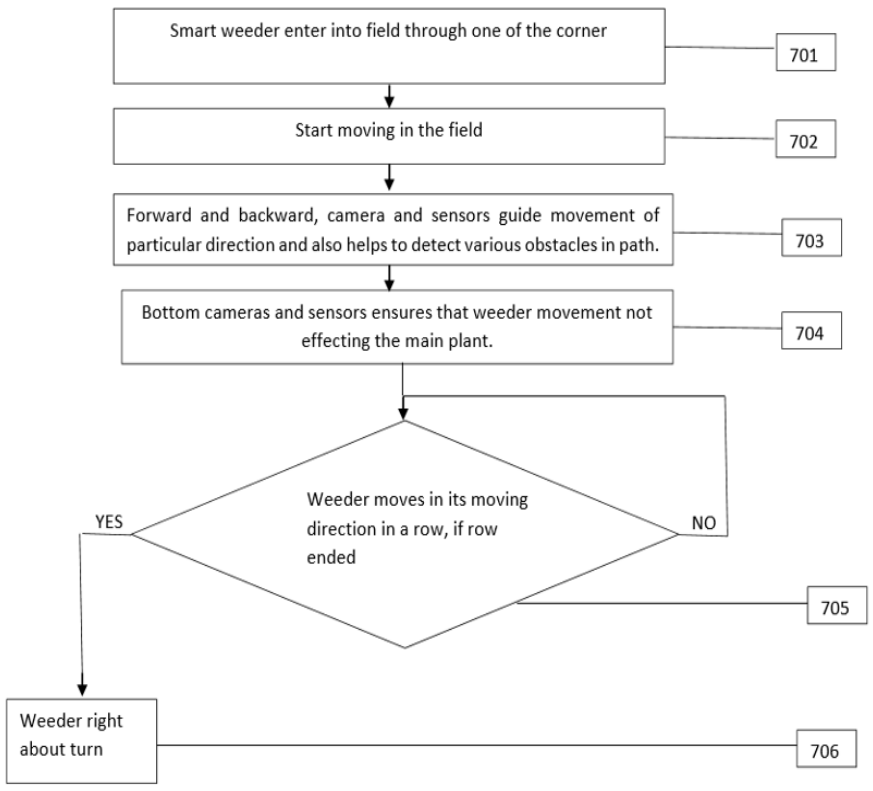

Smart weeder enter into field through one of the corner
701
Start moving in the field
702
Forward and backward, camera and sensors guide movement of particular direction and also helps to detect various obstacles in path.
703
Bottom cameras and sensors ensures that weeder movement not effecting the main plant.
704
Weeder moves in its moving direction in a row, if row ended
YES
NO
705
Weeder right about turn
706

Figure 6.4: Inside Field Working Flow of the Proposed Innovation

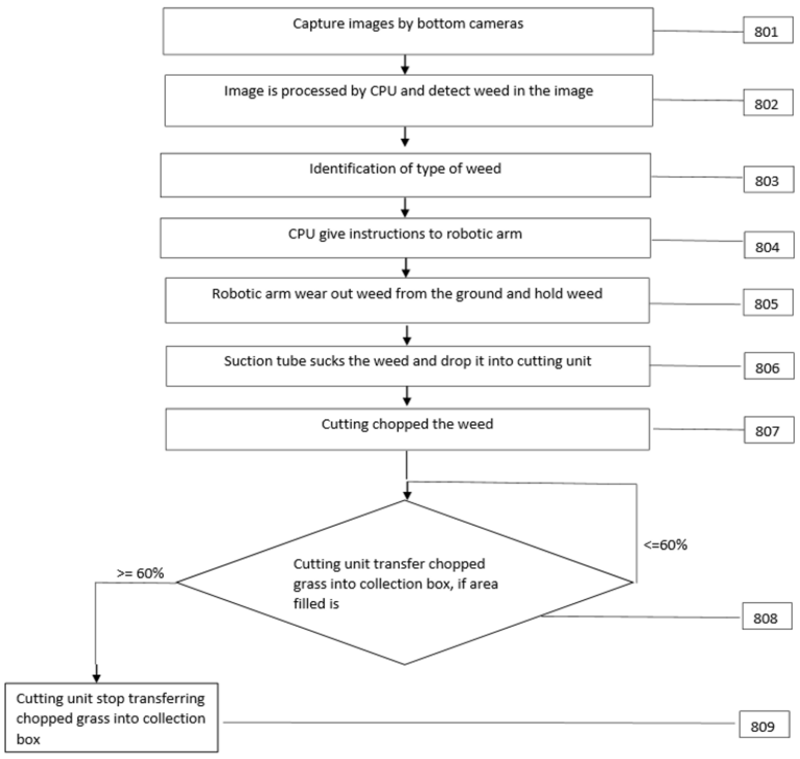

Capture images by bottom cameras
801
Image is processed by CPU and detect weed in the image
802
Identification of type of weed
803
CPU give instructions to robotic arm
804
Robotic arm wear out weed from the ground and hold weed
805
Suction tube sucks the weed and drop it into cutting unit
806
Cutting chopped the weed
807
Cutting unit transfer chopped grass into collection box, if area filled is
>= 60%
<=60%
808
Cutting unit stop transferring chopped grass into collection box
809

Figure 6.5: Outside Field Working Flow of the Proposed Innovation

So that the manner in which the above-recited features of the present invention can be understood in detail, a more particular to the description of the invention, briefly summarized. Above, may be had by reference to embodiments, few of which are represented in the appended drawings. It is to be noted, however, that the appended drawings illustrate only typical embodiments of this invention and are therefore not to be considered limiting of its scope, the invention may acknowledge to other equitably effective embodiments. These and other features, benefits and advantages of the present invention will become apparent by reference to the following text figure, with like reference numbers referring to like structures across the views, wherein Figure 6.2 illustrates top view of a smart autonomous weed detection and removal machine, to which various embodiments of the present invention may be implemented. Figure 6.3 illustrates bottom view of a smart autonomous weed detection and removal machine, to which various embodiments of the present invention may be implemented. Figure 6.1(c) illustrates 3D view of a smart autonomous weed detection and removal machine, to which various embodiments of the present invention may be implemented. Figure 6.4 illustrates flow of the smart autonomous weed detector and removal of movement of weeder, in accordance with an embodiment of the present invention and Figure 6.5 illustrates weed detection and removal flow of the smart autonomous weed detector and removal of movement of weeder, in accordance with an embodiment of the present invention.

## 6.5 Detail Description of the Drawings

In Figure 6.2, the top view of a smart weeder is shown. The base or platform (501) of a smart autonomous weed detector and remover can be used in different ways with this invention. It comprises a sensory component, which is a combination of ultrasonic and infrared sensors (502); an indication panel (503); a central processing unit (504); solar panel plates (505) and cameras (506). The sensory component, a combination of ultrasonic and infrared sensors, assists smart weeders in identifying obstacles in their path and in identifying uninteresting objects in crop fields, such as cattle. A LED indication panel that indicates various conditions of a smart weeder, like indication for battery life, indication for detection of large obstacles in the machine's path and indication of an emergency situation for the machine, such as a machine collapsing with any object. The central processing unit (CPU) of the smart weeder performs the core functions of the machine. It consists of an image processing unit, such as the NVDIA Jetson Nano, which captures real-time images from the bottom camera. Then it processes these images using an instance segmentation-based object detection algorithm MASK R-CNN to distinguish between weed and crops in the field. MASK R-CNN uses the ResNet101 architecture to extract features from the images and these features now apply a region proposal network (RPN) to the features extracted by ResNet 101. RPN predicts whether an object is present in that region or not. The RPN may yield regions with varying or random shapes, to which MASK RCNN applies a region of interest. This process transforms these regions into similar shapes, which it then feeds to fully connected networks that forecast class labels and bounding boxes. After this, MASK R-CNN generates a segmentation mask by using intersection over union (IOU). IOU is defined as the ratio of area of intersection to area of union; if IOU is greater than 0.5, then only area is considered for mask application. As shown in Figure 6.3, the solar plates serve as the energy source for the smart weeder, converting sunlight into electrical energy that continuously charges the lithium-ion batteries (605) located at the base of the platform. The two cameras (506) on the edge of the platform (501) capture real-time images of plants, assisting the CPU in achieving better results. These cameras also aid in detecting suspicious objects around the weeder, thereby enhancing the accuracy of the weeder's movements. Figure 6.3 shows the smart weeder's bottom view. It has a hydraulic rod (601) connected to a wheel, the smart weeder's wheels (602), an outlet for the chopped grass collector (603), ultrasonic sensors (604) for figuring out the level, lithium-ion batteries (605), a box for collecting chopped grass (606), a cutting unit (607), a suction tube (608), a robotic arm (609) and cameras (610). The hydraulic rods help smart weeders control their height at various ground levels and maintain good ground clearance while operating in off-road fields. Additionally, by simply adjusting the machine's height, the hydraulic rods assist the smart weeder in avoiding small obstacles in its path, enabling it to operate autonomously. The wheels regulate the movement of smart weeders; they are rim-based tubeless tires that excel in off-road conditions. The wheels are positioned slightly outside the body line to effectively handle small bumps and manage the waves generated during off-road weeder movement, ensuring that these waves do not negatively impact the various components on the tractor. The outlets facilitate the removal of chopped grass from the collection box, enabling farmers to prepare ready-made cattle feed. Ultrasonic sensors are designed to determine the level of chopped grass collected in the collection box. If the box becomes full, the sensor will alert the user. The intermediate-level sensors link to a grass-passing tube or pipe, with one end attached to the cutting unit and the other to the grass-cutting collection box. As the collection box approaches the intermediate level, the sensor begins to signal. Solar plates charge the lithium-ion batteries and allow direct electric charging. For solar panel systems, lithium-ion batteries are the most reliable and cost-effective choice due to their ability to withstand hundreds of changes and maintain stability, making them a dependable energy source. Their reduced selfdischarge rate aligns perfectly with the gradual recharge of solar plates. The collection boxes are designed to collect chopped grass that emerges from the cutting unit through a grass pipe. Each box can hold approximately 1-3 kg of chopped grass. To facilitate efficient retrieval of this chopped grass, each box features two outlets, allowing the user to easily remove it. The smart weeder's cutting unit connects to a robotic arm, which removes weed from the ground and raises it slightly. A suction pump then draws the weed through a suction tube, depositing it into the cutting unit, where circular blades grind it into chopped grass. The suction tube collects weed from the robotic arm and transports it to the cutting unit; it is equipped with a suction pump that operates within a range of 0-16" Hg (approximately). Only when a robotic arm holds weed near the tube edge does the suction tube start its suction pump, yielding satisfactory results. The robotic arm, which is connected to a suction tub, moves towards the ground only when the CPU detects weed on the ground. Its flexible mechanical arm then gently removes the weed from the ground with an accuracy of 1-2 cm (approx.) without causing any disturbance to the adjacent main crop. The bottom-facing cameras capture real-time plant images, which the CPU uses to process and make decisions.

Figures 6.4 and 6.5 shows the flow of the smart autonomous weed detector and the weeder's movement removal in accordance with an embodiment of the present invention. The flow commences when the smart weeder enters the field from any corner the user chooses (701), following the verification of its path by (502) and (506). Once verified, the weeder begins to move in the field (702). Forward and backward cameras and sensors guide movement in a particular direction and also help detect various obstacles in the path (703). The sensor components and sensors continuously monitor the surrounding area of the weeder, ensuring an obstacle-free path. The system combines one or more ultrasonic and infrared sensors. The ultrasonic sensors emit ultrasonic waves, which, upon encountering an object, collapse and return to the sensor, thereby confirming its presence. Infrared (IR) sensors use the changing temperature surrounding the weeder to detect suspicious objects nearby. The system captures images of uncommon field objects to detect their presence. Bottom cameras and sensors ensure that the movement of the weeder does not negatively impact the main plant (704). Ultrasonic sensors and camera regularly monitoring movement of weeder so that its movement not affecting the main plant just they perform their action just same as infrared sensors and camera. The weeder moves in its direction until it reaches the end of the row (705). If it doesn't, it continues to move in the same direction. If it reaches the end of the row, it moves in a different direction (706), which uses infrared sensors and camera to determine the row's end. Once the weeder reaches its end, it performs a right turn to change the row. One side ultrasonic sensor fixes its position first and its opposite sides ultrasonic sensors rotate at 180 degrees (approximately) to make the weeder move backward in the next row. The flow of finding and getting rid of weeds in the smart autonomous weed detector and the movement of the weeder in one embodiment of the present invention is shown in Figure 6.5. The process begins with capturing images of weed using camera (801), a device that captures real-time, high-resolution images of plants from the ground. Camera (610) transmits these images to CPU (504), which analyzes them and determines the presence of weed beneath the weeder (802). The central processing unit of the smart weeder, responsible for the smart weeder's core functions, carries out this detection. The CPU consists of an image processing unit called NVIDIA Jetson Nano, which takes real-time images from camera (610) and processes these images using an instance segmentation-based object detection algorithm MASK R-CNN to separate weeds from crops in the field. MASK R-CNN uses the ResNet 101 architecture to extract features from the images and these features now apply a region proposal network (RPN) to the features extracted by ResNet 101. The RPN predicts whether an object is present in that region or not. The RPN may yield regions with varying or arbitrary shapes, which MASK R-CNN then transforms into the same shapes by applying a region of interest. These regions then feed into fully connected networks that forecast class labels and bounding boxes. After this, MASK R-CNN generates segmentation masks by using intersection over union (IOU). IOU is defined as the ratio of the area of the intersection to the area of the union; if IOU is greater than 0.5, then only the area is considered for mask application. After processing the image, CPU (504) identify the type of weed, which then guides hydraulic rod (601) to adjust its size accordingly. After that, CPU (504) guides the robotic arm (609) to give instruction to robotic arm (804) which means that robotic arm (609) wears out the weed from the ground smoothly without affecting the main plant and robotic arm adapts its size according to the results of identification of type of weed. After identification, it holds the weed at the edge of suction tube (608), which is sucking the weed into suction tube and dropping the weed into the cutting unit. Suction tube uses a suction pump that is compatible with portable batteries, such as a 20-watt suction pump. After identification of types of weeds, it reaches to cutting unit (607), where chopping the grass, takes place (807). Cutting unit is equipped with stainless steel blades that rotate at a high speed of approximately 6000 rpm, allowing the grass to be chopped within seconds. Then transferring the chopped grass to chopped grass box (606) via ultrasonic sensor (808). Pipes connect chopped grass tub to ultrasonic sensor (604), transmitting the grass from cutting unit (607).

There are one or more ultrasonic sensors that measure the level of chopped grass in chopped grass box. If the level exceeds, halts the transmission of chopped grass into chopped grass.

## 6.6 List of Claim

- A smart weeder (Figure 6.1 c, Figure 6.2, Figure 6.3) which is autonomously detect the weed by using one or more cameras (610), GPU enabled CPU (504) and remove out the weed from ground by using robotic arm (609).
- Smart weeder is completely free from herbicides it uses camera (610) to wear out weed from ground, uses suction tube (608) to suck the weed from cameras (610) and drop in into cutting unit (607) which chopped the weed using one or more blades control by one or more motors.
- As smart weeder claimed in claim 2, where smart weeder uses one or blades to destroy the weed, it helps to avoid soil degradation due herbicides and improve soil fertility which ultimately lead to enhance in production.
- As smart weeder claimed in claim 1, wherein it uses one or more cameras (610) to capture real-time images of weed, these images helps to build dataset of images of real world weed images of various regions that further used for studies related to weeds. And also helps to classify weed as useable or not useable.
- A Smart weeder uses infrared sensor (502) that sense images of smart weeder's surrounding which is processed by processing unit (504) to detect objects in the captured images and camera (506) that uses ultrasonic and infrared waves to detect presence of any object in the surrounding of smart weeder they both combined help to detect suspicious objects such as cattle in the crop field that helps to avoid damage of crops from their action.
- As a smart weeder claimed in claim 1, it uses camera (610), processing unit (504), robotic arm (609) for autonomously detection and removal of weed it completely eliminate labour cost in weed removal process.
- A smart weeder uses box (606) for collecting chopped grass or weed, from which user can collect this chopped grass which a type of cattle feed, so smart weeder provides readymade cattle feed which is additional benefits of its users.
- As a smart weeder claimed in claim 9, it provide readymade cattle feed which is additional benefits of its users so, smart weeder also reduces labour and machine cost to first wear out grass from field than process into machine to make it cattle feed, it directly gives readymade

cattle feed. And physical dimensions of smart weeder is such as a bag of 10 kg which makes smart weeder completely portable.

## 6.7 Summary of the Invention

The present invention provide a smart autonomous weed detector and removal system. The smart autonomous weed detector and removal system consists of a GPU enabled NVIDIA Jetson Nano, which employs image segmentation techniques to identify weed in the crop field. It then transmits electronic signals to a circuit controller, which interprets these signals as commands to operate the mechanical part of the smart weeder. According to an embodiment of the present invention, the smart weeder consists of a circuit controller that receives electronic signals from sensors and processing units to guide the vehicle's movement, a weed cutting blade, and various other components. In accordance with an embodiment of the present invention, the smart weeder further comprises four cameras; two of them are in front and back, and two of them are at the bottom. The front and back cameras aid in path determination and enhance operational accuracy, while the bottom camera aids in weed detection and recognition. In accordance with an embodiment of the present invention, the smart weeder further comprises two ultra-sonic and two infrared sensors, which help the smart weeder detect barrier. According to an embodiment of the present invention, the smart weeder also includes wheels that are attached to a hydraulic suspension system. This system allows the weeder to adjust its height to accommodate varying weed heights and also enables it to move in the direction of the weeds. According to an embodiment of the present invention, the smart weeder incorporates a solar plate on the top of its platform, which provides energy for the weeder to perform its tasks. As a backup energy source, the smart weeder relies on lithium-ion batteries, which supply energy when the solar plate is not available. An embodiment of the present invention also includes indication LEDs on the top of the weeder platform, which provide various indications such as battery life, alarm condition and on/off status. In accordance with an embodiment of the present invention, the smart weeder further comprises a suction pump attached to a robotic arm. The suction pump from the robotic arm sucks weed into the cutting section, which pulls it out of the ground. In accordance with an embodiment of the present invention, the smart weeder further comprises a collection box at the bottom of the weeder, which collects weed cut by cutting sections through a valve.

# CHAPTER 7

# CONCLUSION AND FUTURE WORK

This chapter summarizes the work done in the thesis, presents, and highlights the major contribution of thesis with section and subsection of the chapters. Also discuss the future directions of the proposed work done in this research.

## 7.1 Summary of Work Done in the Thesis

**Firstly,** to start the research work by exploring existing research paper using a systematic literature review titled **“**IoT Data Stream Handling, Analysis, Communication and Security Issue: A Systematic Survey”. It navigates through the intricate landscape of IoT data streams, elucidating the challenges and opportunities that underpin this dynamic domain. The paper meticulously identifies the key characteristics of IoT data, including heterogeneity, massive streaming data, noise data and space-time correlation. It sheds light on the pressing challenges in managing and analysing IoT data, setting the stage for understanding the requisites for IoT big data analytics (BDA).

**Secondly**, to explore and proposed IoT stream data handling models, “IoT Data Stream Handling using Delta Encoding”, offers an innovative data handling model cantered around delta encoding, aimed at enhancing energy efficiency and security. The research accentuates the model's flexibility, allowing users to select compressed word sizes based on data stream characteristics. By utilizing low-cost operations and emphasizing computational efficiency, the proposed model significantly reduces transmission time and power consumption associated with data streams. Next **“**Secure and Energy-Efficient Edge Computing Improved SZ 2.1 Hybrid Algorithm for Handling IoT Data Stream" delves into the critical challenges faced by IoT devices, emphasizing the delicate balance between energy efficiency and data security. The paper introduces an enhanced SZ 2.1 compression algorithm coupled with ChaCha12-Poly1305 AEAD encryption, aiming to optimize data transfer, extend battery life and fortify data security. Addressing the limitations of prevailing transmission methodologies, the paper illuminates the trade-off between security and performance. It introduces a novel approach that ingeniously combines compression and encryption, significantly reducing data volume, transmission times and enhancing power efficiency.

**Thirdly,** to analyse IoT stream data processing, studies about the transmission process of COVID-19. Based on the research, we have proposed a new model for prediction which uses the technique of Long-short term memory (LSTM), a deep learning technique. LSTM is generally suited for real data predictions based on time series stream data. Next, " GAP-LDDN: Leaf Disease Detection Network based on Multi-task Learning and Attention Features for IoT Data Stream" ventures into the realm of agricultural technology, underscoring the significance of early detection and mitigation of plant diseases. This paper focuses on the economic losses associated with plant disorders and introduces GAP-LDDN, a novel architecture utilizing ResNet101 for detecting and classifying diseases in leaves. The model's success in achieving high accuracy rates across various plant datasets marks it as a robust tool for disease identification, spanning multiple crops and diseases. Later we implement a model on identification of biofuel plants using machine learning algorithm to help to make use of efficient green energy.

## 7.2 Future Work

There have been several approaches proposed for IoT data stream handling, analysis, data minimization and security enhancement in IoT applications. In order to maintain a balance between power and security, this study prioritizes energy savings and employs a middle level of security**,** a higher level of security may be incorporated in further studies**.** This is achieved by compressing data during transmission between IoT devices, which reduces computational complexity and costs associated with high levels of security. Therefore, we can further enhance security by reducing the complexity of the encryption algorithm. In the future, we will try to decrease computation time with 20 or higher rounds in the ChaCha Poly1305 encryption algorithm.

The proposed classification-based data analysis model (GAP-LDDN) is computationally efficient, simpler and works accurately, which could replace the onsite decease identification efforts of experienced experts. If it is integrated into the embedded system, it may also lessen the amount of human intervention that was required, laying the theoretical groundwork for a precise detecting robot for the infected leaves. Consequences propose that second-generation biofuels, simultaneously among the energy-effective transportation methods like plug-in hybrids, prove to be one of the significant portions regarding optimized processes.

## 7.3 Social Impact

As an social application of research work, we have worked on handling of real-time data generated from IoT devices which can help in ease and reducing the manpower, reducing time , as well as solve the problem in society for smart transport, smart farming etc. For this we invented a "AI and IoT based Smart Autonomous Weed Detection and Removal System" which help the farmer to removal weed from Indian field and replace manual and traditional cultivator system a mechanical device work with the help of tractor.

# Publications Related to the Thesis

## Papers Published in International Journals

- Sanjay Patidar, Rajni Jindal,Neetesh Kumar, IoT Data Stream Handling, Analysis, Communication and Security Issue: A Systematic Survey, *Wireless Personal Communication,* Springer, 2024, pp 1-50. **(SCIE, IF: 1.9)**
- Sanjay Patidar, Rajni Jindal and Neetesh Kumar, A Secure and Energy-efficient Edge Computing Improved SZ 2.1 Hybrid Algorithm for Handling IoT Data Stream. *Multimedia Tools Appl* , Springer , 2024.pp 1-32 **(SCIE, IF: 3.6)**
- Rajni Jindal, Neetesh Kumar, and Sanjay Patidar, An Energy Efficient, Secure Model for IoT Streamed Data Handling using Delta Encoding, *International Journal of Communication System,* Wiley, 2022. **(SCIE, IF: 1.82)**

## Papers Published in International Conferences

- Rajni Jindal, Neetesh Kumar and Sanjay Patidar, IoT Stream Data Compression Using LDPC Coding. *In book Evolution in Computational Intelligence, Frontiers in Intelligent Computing: Theory and Applications (FICTA 2020),* NIT Suratkal, Karnataka, 4-5th January, 2020, Volume 1 , pp. 455-466.
- Sanjay Patidar, Rajni Jindal and Neetesh Kumar, Streamed Covid-19 Data Analysis Using LSTM—A Deep Learning Technique. *In book Advances in Intelligent Systems and Computing : Soft Computing for Problem Solving (SoCP 2020),* IIT Indore, MP, 18-19th Dec 2020, Vol 1393. Pp. 493-504. **(Awarded best paper)**

## Paper Communicated in International Journal

- Sanjay Patidar, Rajni Jindal and Neetesh Kumar , GAP-LDDN : Leaf Disease Detection Network based on Multitask Learning and Attention Features for IoT Data, *European Journal of Plant Pathology ,* Springer Nature, 2024. **(Communicated).**

## Patent Published

- Sanjay Patidar, Rajni Jindal, Neetesh Kumar, Smart Autonomous Weed Detector and Removal , *Official Journal of The Patent Office, India* , Patent Id : 202011028931, Issue No. 34/2020.

## Project Received

- Received **Translation Research Project** for development and commercialization of real-time application in product from Technology Innovation Hub foundation, IIT Kharagpur.

# References

1. J. Jang ,I. Y. Jung ,J. H. Park (2018), An effective handling of secure data stream in IoT , Applied Soft Computing, Vol.68,pp.811–820.
2. P. Jhang ,M. Durresi,A. Durresi (2019), Multi-access edge computing aided mobility for privacy protection in Internet of Things, Springer Nature, Computing,vol. 101, pp. 729–742, https://doi.org/10.1007/s00607-018-0639-0
3. S. Messaoud ,A. Bradai ,E. Moulay (2020), Online GMM Clustering and MiniBatch Gradient Descent Based Optimization for Industrial IoT 4.0, IEEE Trans. Ind. Informatics, Vol. 16, No. 2, pp. 1427–1435.
4. V. Chang ,V. M. Munoz ,M. Ramachandran (2020), Emerging applications of internet of things, big data, security and complexity: special issue on collaboration opportunity for IoTBDS and COMPLEXIS ,Springer, Computing, vol. 102, pp. 1301–1304,https://doi.org/10.1007/s00607-020-00811-y.
5. A. Singh,S. Garg,S. Batra ,N. Kumar,J. J. P. C. Rodrigues (2020), Bloom filter based optimization scheme for massive data handling in IoT environment. Future Generation Computer Systems, Vol. 82, pp. 440-449

6. J. Azar, A. Makhoul, M. Barhamgi , R. Couturier (2019), An energy efficient IoT data compression approach for edge machine learning , Future Generation Computer Systems 96 , 168–175.
7. S. Schneider, M. Hirzel and B. Gedik (2013), Tutorial: Stream Processing Optimizations, ACM DEBS, pp. 249–58.
8. Chun-Wei Tsai, Chin-Feng Lai, Han-Chieh Chao, and Athanasios V. Vasilakos (2015), Big data analytics: A survey.J. Big Data 2, 1,1–32.
9. N. Kumar,D. P. Vidhyarthi (2017), An Energy Aware Cost Effective Scheduling Framework for Heterogeneous Cluster System, Future Generation Computer Systems", Vol. 71,73–88.
10. H. Li,G. Zhu ,C. Cui ,H. Tang ,Y. Dou and C. He (2016), Energy efficient migration and consolidation algorithm of virtual machines in data centers for cloud computing, Springer, Computing ,vol. 98, pp. 303317,https://doi.org/10.1007/s00607-015-0467-4.
11. W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu (2016), IEEE Internet of Things Journal 3 (5) , Edge computing: Vision and challenges. doi: 10.1109/JIOT.2016.2579198, 637–646.
12. L. Cui et al. (2022), Security and Privacy-Enhanced Federated Learning for Anomaly Detection in IoT Infrastructures, in IEEE Transactions on Industrial Informatics, doi: 10.1109/TII.2021.3107783, vol. 18, no. 5, pp. 3492-3500.
13. , L. Wang and R. Ranjan (2015), Processing distributed internet of things data in clouds, IEEE Cloud Computing, 2, 1, 76-80.
14. P. Gope, T. Hwang (2015), BSN-Care: A secure IoT-based modern healthcare system using body sensor network, IEEE sensors journal, 16, 5 , pp. 1368-1376.
15. J. Shah and B. Mishra (2016), IoT enabled environmental monitoring system for smart cities. IEEE International Conference on Internet of Things and Applications (IOTA), Pune, India, 2016, pp. 383-388.
16. M. Wei, S. H. Hong, and M. Alam (2016), An IoT-based energy-management platform for industrial facilities. Applied energy, 164, 607-619.
17. S. A. I. Quadri and P. Sathish (2017), IoT based home automation and surveillance system, IEEE International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2017, pp. 861-866.
18. I. Mohanraj, K. Ashokumar and J. Naren (2016), Field monitoring and automation using IOT in agriculture domain. Procedia Computer Science, 93, 931-939.

19. P. Saarika , K. Sandhya and T. Sudha (2017), Smart transportation system using IoT, IEEE, International Conference On Smart Technologies For Smart Nation (SmartTechCon), Bengaluru, India, 2017, pp. 1104-1107,
20. S. Lakshminarasimhan et al.( 2011), Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data, in Lecture Notes in Computer Science including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 6852 LNCS, no. PART 1, pp. 366–379, doi: 10.1007/978-3-642-23400-2_34.
21. Z. Chen, S. W. Son, W. Hendrix, A. Agrawal, W. K. Liao, and A. Choudhary (2015), NUMARCK: Machine Learning Algorithm for Resiliency and Checkpointing, International Conference for High Performance Computing, Networking, Storage and Analysis, SC, 2014, vol. -Janua, pp. 733–744, doi: 10.1109/SC.2014.65.
22. N. Sasaki, K. Sato, T. Endo and S. Matsuoka (2015), Exploration of Lossy Compression for Application-Level Checkpoint/Restart, Proceedings - 2015 IEEE 29th International Parallel and Distributed Processing Symposium, IPDPS 2015, pp. 914–922, doi: 10.1109/IPDPS.2015.67.
23. P. Lindstrom (2014), Fixed-rate compressed floating-point arrays, IEEE Transactions on Visualization and Computer Graphics, doi: 10.1109/TVCG.2014.2346458, vol. 20, no. 12, pp. 2674–2683.
24. J. A. Healey and R. W. Picard (2005), Detecting stress during real-world driving tasks using physiological sensors, IEEE Transactions on Intelligent Transportation Systems, vol. 6, no. 2, pp. 156–166, doi: 10.1109/TITS.2005.848368.
25. A. L. Goldberger et al. (2000), PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals., Circulation, doi: 10.1161/01.cir.101.23.e215, vol. 101, no. 23.
26. J. Azar, A. Makhoul, M. Barhamgi and R. Couturier (2019), An energy efficient IoT data compression approach for edge machine learning, Future Generation Computer Systems, doi: 10.1016/j.future.2019.02.005, vol. 96, pp. 168–175.
27. M. Burtscher and P. Ratnaprabha (2007), High throughput compression of double-precision floating-point data, Data Compression Conference Proceedings, doi: 10.1109/DCC.2007.44, pp. 293–302.
28. P. Lindstrom and M. Isenburg (2006), Fast and efficient compression of floating-point data, IEEE Transactions on Visualization and Computer Graphics, doi: 10.1109/TVCG.2006.143, vol. 12, no. 5, pp. 1245–1250.

29. K. L. Tsai, Y. L. Huang, F. Y. Leu, I. You, Y. L. Huang and C. H. Tsai (2018), AES-128 based secure low power communication for LoRaWAN IoT environments, IEEE Access, doi: 10.1109/ACCESS.2018.2852563, vol. 6, pp. 45325–45334.

30. K. L. Tsai, F. Y. Leu, I. You, S. W. Chang, S. J. Hu and H. Park (2019), Low-Power AES Data Encryption Architecture for a LoRaWAN, IEEE Access, doi: 10.1109/ACCESS.2019.2941972, vol. 7, pp. 146348–146357.

31. S. Roy, U. Rawat, and J. Karjee (2019), A Lightweight Cellular Automata Based Encryption Technique for IoT Applications, IEEE Access, doi: 10.1109/ACCESS.2019.2906326, vol. 7, pp. 39782–39793.

32. X. Guo, J. Hua, Y. Zhang, and D. Wang (2019), A complexity-reduced block encryption algorithm suitable for internet of things, IEEE Access, doi: 10.1109/ACCESS.2019.2912929, vol. 7, pp. 54760–54769.

33. S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park (2017), Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions, Journal of Ambient Intelligence and Humanized Computing, doi: 10.1007/s12652-017-0494-4, pp. 1–1.

34. D. A. F. Saraiva, V. R. Q. Leithardt, D. de Paula, A. S. Mendes, G. V. González, and P. Crocker (2019), PRISEC: Comparison of symmetric key algorithms for IoT devices, Sensors (Switzerland), doi: 10.3390/s19194312, vol. 19, no. 19.

35. B. J. Mohd and T. Hayajneh (2018), Lightweight block ciphers for IoT: Energy optimization and survivability techniques, IEEE Access, doi: 10.1109/ACCESS.2018.2848586, vol. 6, pp. 35966–35978.

36. A. A. Diro, N. Chilamkurti, and Y. Nam (2018), Analysis of Lightweight Encryption Scheme for Fog-to-Things Communication, IEEE Access, doi: 10.1109/ACCESS.2018.2822822, vol. 6, pp. 26820–26830.

37. A. O. Akmandor, H. Yin, and N. K. Jha (2018), Simultaneously ensuring smartness, security, and energy efficiency in Internet-of-Things sensors, IEEE Custom Integrated Circuits Conference, CICC, doi: 10.1109/CICC.2018.8357069, pp. 1–8.

38. A. Fragkiadakis, E. Tragos, L. Kovacevic, and P. Charalampidis (2016), A practical implementation of an adaptive Compressive Sensing encryption scheme, WoWMoM 17th International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2016, doi: 10.1109/WoWMoM.2016.7523561.

39. Y. Zhang, Y. Xiang, L. Y. Zhang, Y. Rong, and S. Guo (2019),Secure Wireless Communications Based on Compressive Sensing: A Survey, IEEE Communications Surveys and Tutorials, doi: 10.1109/COMST.2018.2878943,vol.21, no.2, pp.1093–1111.

40. J. Qi, X. Hu, Y. Ma, and Y. Sun (2015), A hybrid security and compressive sensing-based sensor data gathering scheme, IEEE Access, doi: 10.1109/ACCESS.2015.2439034, vol. 3, pp. 718–724,.

41. F. de Santis, A. Schauer, and G. Sigl (2017), ChaCha20-Poly1305 authenticated encryption for high-speed embedded IoT applications, Proceedings of the 2017 Design, Automation and Test in Europe, DATE 2017, doi: 10.23919/DATE.2017.7927078, pp. 692–697.

42. S. Luangoudom, T. Nguyen, D. Tran, and L. G. Nguyen (2019), End to end message encryption using Poly1305 and XSalsa20 in Low power and Lossy Networks, Proceedings of 2019 11th International Conference on Knowledge and Systems Engineering, KSE 2019, doi: 10.1109/KSE.2019.8919479.

43. S. Fong,R. Wong ,A. V. Vasilakos (2019) Accelerated PSO Swarm Search Feature Selection for Data Stream Mining Big Data, IEEE Transactions on Services Computing, Vol. 9, NO. 1,pp. 1-13.

44. L. Du,Y. Du ,Y. Li ,J. Su ,Y. C. Kuan ,C. C. Liu and M. C. F. Chang (2018 )A Reconfigurable Streaming Deep Convolutional Neural Network Accelerator for Internet of Things , IEEE transactions on circuit and systems I, Vol. 65, Issue: 1.

45. H. Jin ,F. Chen , S. Wu ,Y. Yao ,Z. Liu ,L. Gu ,Y. Zhou (2019) Towards Low Latency Batched Stream Processing by Pre-Scheduling. IEEE Transactions on Parallel and Distributed Systems", Vol. 30, No. 3,pp. 1-13.

46. B. J. Mohd ,T. Hayajneh (2019),Lightweight block ciphers for IoT: Energy optimization and survivability techniques", IEEE Access. Vol. 6,pp. 35966–35978(2019).

47. Y. Zhang ,Y. Xiang , L. Y. Zhang ,Y. Rong ,S. Guo (2019), Secure Wireless Communications Based on Compressive Sensing: A Survey, IEEE Commun. Surv. Tutorials. Vol. 21, pp. 1093–1111.

48. T. Lloyd ,K. Barton ,E. Tiotto ,J. N. Amaral (2018), Run-Length Base Delta Encoding for High-Speed Compression, 47th International Conference on Parallel Processing Companion, No. 29, pp. 1-9.

49. K. Tsai ,Y. L. Huang ,F. Y. Leu ,I. You ,Y. L. Huang ,C. H. Tsai (2018), AES-128 based secure low power communication for LoRaWAN IoT environments", IEEE Access. Vol.6, pp. 45325–45334.

50. S. Roy ,U. Rawat ,J. Karjee (2019) , A Lightweight Cellular Automata Based Encryption Technique for IoT Applications, IEEE Access. Vol. 7, pp. 39782–39793.
51. D. Puthal, R. Ranjan,J. Chen (2016), DLSeF : a dynamic key-length-based efficient real time security verification model for big data stream ,ACM Trans. Embed. Computer system 6 (2).
52. T. Gomes , F. Salgado, S. Pinto , J. Cabral, and A. Tavares (2018), A 6LoWPAN Accelerator for Internet of Things Endpoint Devices, IEEE Internet of ThingsJournal, VOL. 5, NO. 1, pp. 1-7.
53. C. S. Park and W. S. Park (2018), A Group-Oriented DTLS Handshake for Secure IoT Applications, IEEE Transactions on autonomous science and engineering, vol. 15, no. 4, pp. 1920-1929.
54. X. Liu ,F. Xiong, Z. Wang , and S. Liang (2018), Design of Binary LDPC Codes With Parallel Vector Message Passing, IEEE Transactions on communication, vol. 66,No.4.
55. R. Nomura, and T. S. Han (2014), Second-Order Slepian-Wolf Coding Theorems for Non-Mixed and Mixed Sources, IEEE Transactions on information theory vol.60, No. 9.
56. M. Johnson, P. Ishwar, V. Prabhakaran, D. Schonberg, K. Ramchandran (2004), on compressing encrypted data, EEE Trans. Signal Process. 52(10) 2992-3006.
57. P. Kishore, N. Nagendra, K.Reddy, V. Murthy (2012), Smoothing and optimal compression of encrypted gray scale images, Int. J. Eng. Res. Appl.(IJERA) 2(3) 23-28.
58. G. Zeng, X. Dong, J. Bornemann (2013), Reconfigurable Feedback Shift Register Based Stream Cipher for Wireless Sensor Networks, IEEE Wireless Communications Letters, VOL. 2, NO. 5, pp. 1-4.
59. R. C. Das (2020), Forecasting incidences of COVID-19 using Box-Jenkins method for the period July 12-Septembert 11: A study on highly affected countries, Chaos Solutions and Fractals, Volume 140, November 2020, 110248.
60. A.S. Fokas, J. Cuevas-Maraver, P.G. Kevrekidise (2020), A quantitative framework for exploring exit strategies from the COVID-19 lockdown, Chaos, Solitons & Fractals, volume 140.
61. N. Piovella (2020), Analytical solution of SEIR model describing the free spread of the COVID-19 pandemic, Chaos, Solitons & Fractals, Vol 140.
62. Y. Mohama, A. Halidou, P. T. Kapen (2020), A review of mathematical modeling, artificial intelligence and datasets used in the study, prediction and management of COVID-19, Applied intelligence, Volume 50, pages- 3913- 3925 (2020).

63. Y. Wang, H. Wang, and Z. Peng (2021), Rice diseases detection and classification using attention based neural network and bayesian optimization, Expert Systems with Applications, vol. 178, p. 114770.

64. Y. Lu, S. Yi, N. Zeng, Y. Liu, and Y. Zhang (2017), Identification of rice diseases using deep convolutional neural networks, Neurocomputing, vol. 267, pp. 378–384.

65. X. Zhang, Y. Qiao, F. Meng, C. Fan, and M. Zhang, Identification of maize leaf diseases using improved deep convolutional neural networks (2018), IEEE Access, vol. 6, pp. 30 370–30 377.

66. S. Uguzˇ and N. Uysal (2021), Classification of olive leaf diseases using deep convolutional neural networks," Neural Computing and Applications, vol. 33, no. 9, pp. 4133–4149.

67. S. Ramesh and D. Vydeki (2020), Recognition and classification of paddy leaf diseases using optimized deep neural network with jaya algorithm, Information processing in agriculture, vol. 7, no. 2, pp. 249–260.

68. N. Duong-Trung, L.-D. Quach, M.-H. Nguyen, and C.-N. Nguyen (2019), Classification of grain discoloration via transfer learning and convolutional neural networks, Proceedings of the 3rd International Conference on Machine Learning and Soft Computing, pp. 27–32.

69. X. Nie, L. Wang, H. Ding, and M. Xu (2019), Strawberry verticillium wilt detection network based on multi-task learning and attention, IEEE Access, vol.7, pp. 170003–170011.

70. R. Dwivedi, S. Dey, C. Chakraborty, and S. Tiwari (2021), Grape disease detection network based on multi-task learning and attention features, IEEE Sensors Journal vol. 21, no. 16, pp. 17573-17580.

71. Y. Sun, B. Xue, M. Zhang, and G. G. Yen (2019), Evolving deep convolutional neural networks for image classification, IEEE Transactions on Evolutionary Computation, vol. 24, no. 2, pp. 394–407.

72. F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang , Residual attention network for image classification, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 6450-6458.

73. F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang (2017), Residual attention network for image classification, Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3156–3164.

74. X. Zhang, T. Wang, J. Qi, H. Lu, and G. Wang (2018), Progressive attention guided recurrent network for salient object detection, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 714–722.

75. R. Girshick, J. Donahue, T. Darrell, and J. Malik (2014), Rich feature hierarchies for accurate object detection and semantic segmentation, in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 580–587.

76. X. Zhang, Y. Yuan, and Q. Wang (2018), Roi-wise reverse reweighting network for road marking detection, in BMVC p. 219.

77. J. Mao, T. Xiao, Y. Jiang, and Z. Cao (2017), What can help pedestrian detection? in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3127–3136.

78. E. C. Too, L. Yujian, S. Njuki, and L. Yingchun (2019), A comparative study of fine-tuning deep learning models for plant disease identification," Computers and Electronics in Agriculture, vol. 161, pp. 272–279.

79. S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard (2017), Sluice networks: Learning what to share between loosely related tasks, arXiv preprint arXiv:1705.08142, vol. 2.

80. S. Ruder (2017), An overview of multi-task learning in deep neural networks, arXiv preprint arXiv:1706.05098.

81. R. Ranjan, V. M. Patel, and R. Chellappa (2017), Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition," IEEE transactions on pattern analysis and machine intelligence, vol. 41, no. 1, pp. 121–135.

82. S. Ren, K. He, R. Girshick, and J. Sun (2016), Faster R-CNN: towards real-time object detection with region proposal networks, IEEE transactions on pattern analysis and machine intelligence, vol. 39, no. 6, pp. 1137–1149.

83. D. Xu, W. Ouyang, X. Wang, and N. Sebe (2018), Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 675–684.

84. H. Durmus¸, E. O. Gunes , and M. Kırcı (2017), Disease detection on the leaves of the tomato plants by using deep learning, 6th International Conference on Agro-Geoinformatics. IEEE, pp. 1–5.

85. R. J. Henry (2010), Evaluation of plant biomass resources available for replacement of fossil oil, Plant Biotechnology Journal.

86. J. Wäldchen, M. Rzanny, M. Seeland, P. Mäder (2018). Automated plant species identification—Trends and future directions", PLoS Computational Biology.
87. S. Nie, Z. Jiang, H. Zhang, B. Cai, Y. Yao (2018), Inshore Ship Detection Based on Mask R-CNN", IGARSS 2018, IEEE International Geoscience and Remote Sensing Symposium.
88. A. Wu, Q. Zhang, W. Fang, H. Deng, S. Jiang, Q. Liu and P. Xia (2018), Mask R-CNN Based Object Detection for Intelligent Wireless Power Transfer, IEEE Globecom Workshops ,9-13..
89. M. Bizjak, P. Peer, Ž. Emeršič (2019), Mask R-CNN for Ear Detection, 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 20-24 May 2019.
90. A. N. Ömeroğlu, N. Kumbasar, E. A. Oral, I. Y. Ozbek (2019), Mask R-CNN Hangar Detection with Mask R-CNN Algorithm, 27th Signal Processing and Communications Applications Conference, 2019.
91. B. Li, Y. Diao, and P. Shenoy (2015), Supporting scalable analytics with latency constraints. Proceedings of the VLDB Endowment, 8, 11, 1166-1177.
92. X. Liu, A. Dastjerdi, and R. Buyy (2016), Stream processing in IoT: Foundations, state-of-the-art, and future directions. Elsevier, City.
93. M. Hilbert (2016), Big data for development: A review of promises and challenges. Development Policy Review, 34, 1, 135-174.
94. H. Hu,Y. Wen, T. S. Chua and X. Li (2014), Toward scalable systems for big data analytics: A technology tutorial. IEEE access, 2 (2014), 652-687.
95. M. Strohbach, H. Ziekow,V. Gazis and N. Akiva (2015), Towards a big data analytics framework for IoT and smart city applications. Springer, City.
96. M. Zaharia,M. Chowdhury, M. Das, T. Dave, A. Ma J. Mccauley, M. Franklin, M. Shenker, S. and Stoica (2012), I. Fast and interactive analytics over Hadoop data with Spark. Usenix Login, 37, 4 , 45-51.
97. C. Engle, A. Lupher,R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica (2012), Shark: fast data analysis using coarse-grained distributed memory. City, 2012.
98. M. Mohammadi,A. Al-Fuqaha, S. Sorour and M. Guizani (2018), Deep learning for IoT big data and streaming analytics: A survey. IEEE Communications Surveys & Tutorials, 20, 4 , 2923-2960.
99. F. Alam, R. Mehmood,I. Katib, N. N. Albogami and A. Albeshri(2017), Data fusion and IoT for smart ubiquitous environments: A survey. IEEE Access, 5 (2017), 9533-9554.

100. E. G. Petrakis,S. Sotiriadis, T. Soultanopoulos, P.T. Renta, R. Buyya and N. Bessis (2018), Internet of things as a service (itaas): Challenges and solutions for management of sensor data on the cloud and the fog. Internet of Things, 3 (2018), 156-174.

101. W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu,(2016), Internet of Things Journal 3 (5) (2016), Edge computing: Vision and challenges. doi: 10.1109/JIOT.2016.2579198, 637–646.

102. P. Jiao, S. Di, J. Liu, X. Liang and F. Cappello (2023), Characterization and Detection of Artifacts for Error-Controlled Lossy Compressors, *IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, Goa, India, pp. 117-126, doi: 10.1109/HiPC58850.2023.00027.

103. Y. Nir and A. Langley (2015), ChaCha20 and Poly1305 for IETF Protocols.

104. Y. Collet ( 2018), Zstandard Compression and the application/zstd Media Type.

105. bzip2: Home. [Online]. Available: https://www.sourceware.org/bzip2/. [Accessed: 21-May-2020].

106. L . Nan ,X. Yang ,X. Zeng ,W. Li ,Y. Du ,Z. Dai , Chen L (2019), A VLIW Architecture Stream Cryptographic Processor for Information Security", China Communications, pp. 1-15.

107. X. Fan ,K. Mandal ,G. Gong (2013), WG-8: a lightweight stream cipher for resource-constrained smart devices, International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, LNICST, Vol. 115, pp. 617-632.

108. J. A. Healey and R. W. Picard ( 2005), Detecting stress during real-world driving tasks using physiological sensors, IEEE Transactions on Intelligent Transportation Systems, vol. 6, no. 2, pp. 156–166, doi: 10.1109/TITS.2005.848368.

109. A. L. Goldberger (2000),PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals., Circulation, doi: 10.1161/01.cir.101.23.e215, vol. 101, no. 23.

110. UCI Machine Learning Repository: Gas sensors for home activity monitoring Data Set [Online].Available:http://archive.ics.uci.edu/ml/datasets/Gas+sensors+for+home+activity+monitoring. [Accessed: 25-Apr-2020].

111. J. P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger (2008), New features of Latin dances: Analysis of Salsa, ChaCha, and Rumba, in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), doi: 10.1007/978-3-540-71039-4_30, vol. 5086 LNCS, pp. 470–488.

112. R. Huerta, T. S. Mosqueiro, J. Fonollosa, N. F. Rulkov, and I. Rodriguez-Lujan (Aug. 2016), Online Decorrelation of Humidity and Temperature in Chemical Sensors for Continuous Monitoring, Chemometrics and Intelligent Laboratory Systems, doi: 10.1016/j.chemolab.2016.07.004, vol. 157, pp. 169–176.

113. N. Niwa,H. Amano and M. Koibuchi (2022), Boosting the performance of interconnection networks by selective data compression. IEICE TRANSACTIONS on Information and Systems, 105(12), 2057-2065.

114. S. Li, S. Di, K. Zhao, X. Liang,Z. Chen and F. Cappello (2021), Resilient error-bounded lossy compressor for data transfer. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-14).

115. X. Delaunay, A. Courtois and F. Gouillon (2019), Evaluation of lossless and lossy algorithms for the compression of scientific datasets in netCDF-4 or HDF5 files. Geoscientific Model Development, 12(9), 4099-4113

116. L. Nan ,X. Yang ,X. Zeng ,W. Li ,Y. Du ,Z. Dai ,L. Chen (2019), A VLIW Architecture Stream Cryptographic Processor for Information Security", China Communications, pp. 1-15.

117. X. Fan ,K. Mandal ,G. Gong (2013), WG-8: a lightweight stream cipher for resource-constrained smart devices, International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, LNICST, Vol. 115, pp. 617-632.

118. R. Huerta ,T. Mosqueiro ,J. Fonollosa ,N. Rulkov ,I. R. Lujan (2016), Online Decorrelation of Humidity and Temperature in Chemical Sensors for Continuous Monitoring, Chemometrics and Intelligent Laboratory Systems, Vol. 157, pp. 169-176

119. J. Jang,I. Y. Jung ,J. H. Park (2018), An effective handling of secure data stream in IoT , Applied Soft Computing, Vol.68,pp.811–820.

120. N. Kumar ,D. P. Vidhyarthi (2017), An Energy Aware Cost Effective Scheduling Framework for Heterogeneous Cluster System,Future Generation Computer Systems", Vol. 71,73–88.

121. T. Lloyd ,K. Barton ,E. Tiotto ,J. N. Amaral (2018), Run-Length BaseDelta Encoding for High-Speed Compression, 47th International Conference on Parallel Processing Companion, No. 29, pp. 1-9.

122. D. G. Carrillo and R. M. Lopez (2018), Multihop Bootstrapping With EAP Through CoAP Intermediaries for IoT, IEEE Internet of Things Journal, Vol. 5, No. 5.

123. A. N. Toosi,J. Son ,R. Buyya (2018), CLOUDS-Pi: A Low-Cost RaspberryPi-Based Micro Datacenter for Software Defined Cloud Computing, IEEE Cloud Computing, Vol.5, No. 5, pp. 81-91.

124. H. Jin ,F. Chen ,S. Wu ,Y. Yao ,Z. Liu ,L. Gu ,Y. Zhou (2019), Towards LowLatency Batched Stream Processing by Pre-Scheduling. IEEE Transactions on Parallel and Distributed Systems", Vol. 30, No. 3,pp. 1-13.

125. SHT75 Datasheet, http://legacy.caricoos.org

126. N. Semiconductor,nRF5 SDK forIoT,https://www.nordicsemsemi.com/ eng/products /Bluetoth-low -energy/nRF5-SDK-for-Iot.

127. R. Huerta, T. Mosqueiro, J. Fonollosa, N. Rulkov, I. Rodriguez-Lujan (2016), Online Decorrelation of Humidity and Temperature in Chemical Sensors for Continuous Monitoring, Chemometrics and Intelligent Laboratory Systems.

128. R. C. Das (2020), Forecasting incidences of COVID-19 using Box-Jenkins method for the period July 12-Septembert 11: A study on highly affected countries", Chaos Solutions and Fractals, Volume 140, November 2020, 110248.

129. S. Kannan (2020),What India did -- and what it didn't -- in Covid-19 battle. Accessed: 8 April 2020. https://www.indiatoday.in/news-analysis/story/what-india-did-and-what-it-didn-t-in-covid19-battle-1663064-2020-04-03

130. S. Ren, K. He, R. Girshick, and J. Sun (2016), Faster r-cnn: towards real-time object detection with region proposal networks, IEEE transactions on pattern analysis and machine intelligence, vol. 39, no. 6, pp. 1137–1149.

131. A. Krizhevsky, I. Sutskever, and G. E. Hinton (2012), Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems, vol. 25, pp. 1097–1105.

132. F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang (2017), Residual attention network for image classification in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3156–3164.

133. T. Y. Lin, P. Dollar,´ R. Girshick, K. He, B. Hariharan, and S. Belongie (2017), Feature pyramid networks for object detection," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2117–2

134. R. Dwivedi, S. Dey, C. Chakraborty, and S. Tiwari (2021),Grape disease detection network based on multi-task learning and attention features," IEEE Sensors Journal.

135. M. Vassallo-Barco, L. Vives-Garnique, V. Tuesta-Monteza, H. I. Mej´ıa-Cabrera, and R. Y. Toledo (2017),Automatic detection of nutritional deficiencies in coffee tree leaves

through shape and texture descriptors. Journal of Digital Information Management, vol. 15, no. 1.

136. Z. Chen, R. Wu, Y. Lin, C. Li, S. Chen, Z. Yuan, S. Chen, and X. Zou (2022), Plant disease recognition model based on improved yolov5, Agronomy, vol. 12, no. 2, p. 365.

137. M. Börjesson, E. O. Ahlgren, R. Lundmark, D. Athanassiadis (2014),Bio-fuel futures in road transport – A modeling analysis for Sweden", Transportation Research Part D Transport and Environment .

**Sanja Patidar** is working as Assistant Professor at Department of Software Engineering, Delhi Technological University, Delhi. He has total 13 + years of academic and industry experience. He received his Bachelor of Engineering in Computer Science & Engineering from Rajiv Gandhi Proudhyogiki Vishwavidyalaya, Bhopal (MP) and Master of Engineering in Computer Engineering from Devi Ahilya Vishwavidyalaya, Indore (MP). He is pursuing Phd in Internet of Things from Department of Computer Science and Engineering, Delhi Technological University, Delhi, India. He has patent, book and research publications in field of IoT ,AI ,ML and Basic Programming. He is also acting as a PI of sponsored projects in the domain of IoT in agriculture. He is working in the area of IoT real time data handling, analysis, application of IoT in smart agriculture ,smart city and smart transportation. His research interests include Internet of Things, Machine Learning, Computation Theory and Operating System.