Heterogeneous Sensor Data Streaming for Federated Learning in Fog-Centric IoT Applications

A thesis submitted to the University of Hyderabad in partial fulfillment

for the degree of

Doctor of Philosophy

by

Srinivasa Raju Rudraraju

Reg. No. 18MCPC13

SCHOOL OF COMPUTER AND INFORMATION SCIENCES

UNIVERSITY OF HYDERABAD

HYDERABAD -500046

Telangana

India

October, 2023

CERTIFICATE

This is to certify that the thesis entitled **"Heterogeneous Sensor Data Streaming for Federated Learning in Fog-Centric IoT Applications"** submitted by **Srinivasa Raju Rudraraju** bearing **Reg. No. 18MCPC13** in partial fulfillment of the requirements for the award of **Doctor of Philosophy** in **Computer Science** is a bonafide work carried out by him under our supervision and guidance. This thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for award of any degree or diploma. Parts of this thesis have been published online in the following publications:

1 IEEE Sensors Journal (Chapters 4,7)
2 Proceedings of ICST-2019, IEEE, Sydney, Australia (Chapters 2,3)
3 Proceedings of SPICSCON-2019, IEEE, Dhaka, Bangladesh (Chapters 2,3)
4 Proceedings of FedCSIS-2020, IEEE, Sofia, Bulgaria (Chapter 3)
5 Proceedings of ICETE-2019, Springer, Hyderabad, India (Chapter3)
6 Book Chapter of Assistive Technology for the Elderly, 2020 (Chapter 3)
7 Book Chapter of Handbook of Big Data Analytics, IET, 2021 (Chapter 3)

Further, the student has passed the following courses towards fulfilment of coursework requirement for Ph.D:

S.No.	Course Code	Name	Credits	Pass/Fail
1	CS803	Data Structures and Programming Lab	2	Pass
2	CS804	Algorithms	4	Pass
3	CS832	Computer Networks	4	Pass
4	CS800	Research Methods in Computer Science	4	Pass

Co-Supervisor	Supervisor	Dean
Prof. Atul Negi	Dr. Nagender Kumar Suryadevara	Prof. Atul Negi
SCIS, UoH	SCIS, UoH	SCIS, UoH

DECLARATION

I, **Srinivasa Raju Rudraraju**, hereby declare that this thesis entitled **"Heterogeneous Sensor Data Streaming for Federated Learning in Fog-Centric IoT Applications"** submitted by me under the supervision of **Dr. Nagender Kumar Suryadevara** and **Prof. Atul Negi**, is a bonafide research work and is free from plagiarism. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be deposited in Shodganga/INFLIBNET.

A report on plagiarism statistics from the University Librarian is enclosed.

Date:

Signature of the Student

(Srinivasa Raju Rudraraju)

Reg. No.: 18MCPC13

//Countersigned//

Signature of the Co-Supervisor **Signature of the Supervisor**

(Prof. Atul Negi) (Dr. Nagender Kumar Suryadevara)

Dedicated To My Family & Teachers

Acknowledgements

I am deeply indebted to **Dr. Nagender Kumar Suryadevara** and **Prof. Atul Negi**, my thesis supervisors for their guidance, wisdom, and support during the course of my Ph.D. at University of Hyderabad. Their consistent encouragement and positive reinforcement have made it a gratifying experience. I have learned from them the virtues of rigor and professionalism. I am grateful to them for keeping me focused at times when it mattered a lot. I express deep respect and gratitude to my supervisors for astute guidance and constant inspiration, without which this work would never have been possible.

I thank the members of my doctoral review committee, **Prof. Rajeev Wankar** and **Dr. Avatharam Ganivada** for their encouragement and insightful comments that strengthened my knowledge. I express my gratitude to the faculty members of SCIS for serving as a constant source of inspiration throughout my PhD journey.

I sincerely thank **Prof. Atul Negi**, Dean, School of Computer and Information Sciences (SCIS), University of Hyderabad, Hyderabad, for his academic support during my research work.

I extend my heartfelt appreciation to my fellow co-scholars for their invaluable support. I am grateful to **School of Computer and Information Sciences, University of Hyderabad** for making it a memorable experience. Finally, I thank my family members for their unwavering support throughout my Ph.D. work.

Abstract

The Internet of Things (IoT) has undergone significant growth, impacting various sectors. However, the diversity inherent in the IoT presents formidable challenges in the processing of IoT data. Traditional cloud-centric IoT data processing faces issues of latency, bandwidth consumption, and privacy concerns. Edge Computing aims to address these challenges by focusing on localized data processing, but cannot handle complex tasks due to resource constraints. Fog computing can alleviate this problem by adopting distributed processing of tasks using cluster resources.

The adoptability of existing distributed machine learning frameworks in fog cluster environment has limitations due to their resource intensive nature. Federated learning (FL) emerges as a viable solution for resource-constrained edge devices, enabling the training of machine learning models without centralizing data sets. This research is dedicated to harnessing the potential of Fog Computing-Based Federated Learning (FL) to mitigate these challenges.

This Ph.D. thesis endeavors to devise a communication-efficient algorithm for processing heterogeneous streaming IoT data through FL in fog-centric IoT applications. The research strives to bridge gaps in existing solutions by distributing IoT streaming data among cluster nodes to achieve load balancing and optimize resource utilization. A novel capability-aware federated average (CAFedAvg) algorithm is proposed, that addresses the limitations of the state-of-the-art federated average algorithm by taking into account the capabilities of the devices participating in the federated learning process.

In summary, this thesis contributes to fog computing framework tailored for processing heterogeneous IoT streaming data. It emphasizes load distribution and device capability awareness in FL-based model training. These advancements effectively address challenges related to data heterogeneity and resource utilization within fog-centric IoT environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet of Things (IoT) is a network of interconnected devices equipped with sensors and technology to collect and share data on the Internet [1]. Its goal is to enable devices to communicate and collaborate seamlessly, improving automation, efficiency, and decision making in various domains such as smart homes, healthcare, industries and transportation [2]. The inherent heterogeneity of the IoT landscape, characterized by a diverse range of sensor types, communication protocols, and data formats, presents unprecedented opportunities and formidable challenges [3][4]. The effective handling and utilization of these diverse sensor data streams is essential to obtain valuable insights and actionable intelligence from IoT applications.

Traditionally, cloud-centric approaches have been the backbone of data processing in IoT environments [5]. Processing IoT data with Cloud Computing results in slow response times, increased network bandwidth usage, and potential privacy concerns due to data being sent to external servers [6]. Moreover, Cloud Computing might not work well for real-time applications with strict timing needs. These problems highlight the need to find better ways to handle and unlock the full potential of IoT data. Edge Computing and Fog Computing have emerged as alternative paradigms to address these limitations [7].

Edge computing involves processing data at or near the data source, which is often the edge of the network [6][8]. Unlike conventional Cloud Computing, where data is transmitted to centralized servers for analysis, Edge Computing emphasizes localized processing on the IoT device itself or nearby gateway devices. This approach offers several advantages, including reduced latency, minimized data transmission needs, and improved responsiveness [9]. By processing data

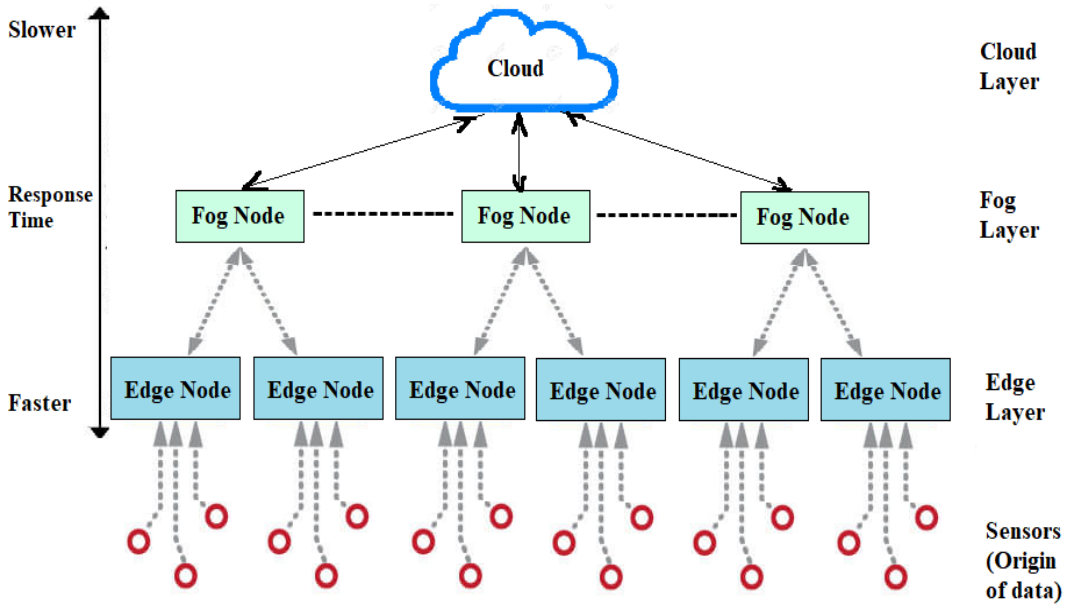


Figure 1.1: Edge Computing vs. Fog Computing

closer to where it is generated, Edge Computing is particularly well-suited for time-sensitive applications, such as industrial automation, autonomous vehicles, and real-time monitoring systems. Moreover, Edge Computing contributes to data privacy by keeping sensitive information localized and reducing the need for data to traverse the network [10].

Fog Computing takes this concept a step further by extending the processing capabilities to intermediate nodes in the network, such as routers and gateways [11], as shown in Fig. 1.1. This approach strikes a balance between Edge Computing's focus on immediate processing and traditional Cloud Computing's ability to handle more complex tasks. Fog Computing can handle more intricate data analysis while still offering reduced latency compared to centralized cloud processing [12]. Fog Computing finds applications in scenarios where intermediate-level processing is necessary, such as Smart Cities, Environmental Monitoring Networks, and Healthcare Systems. These distinctions differentiate between Edge Computing and Fog Computing [13], although these terms are sometimes used interchangeably in certain contexts.

The resource constraints of edge devices for processing IoT data can be mitigated by adopting distributed computing techniques within the edge cluster [14]. Distributed computing involves dividing complex tasks into smaller subtasks and

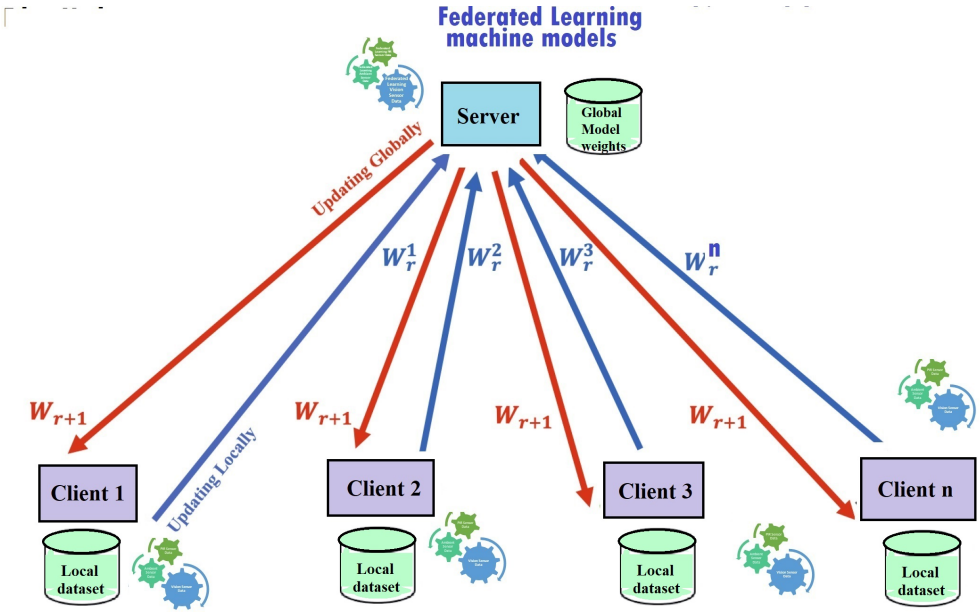


Figure 1.2: Model training using Federated Learning

distributing them across multiple devices within the cluster for simultaneous processing [15]. It optimizes the utilization of available resources within the edge cluster, enhancing the overall processing capability. However, the application of existing Distributed Machine Learning (DML) frameworks for IoT data processing has revealed limitations [16] [17]. Many conventional frameworks are resource intensive and require considerable memory and computational power, which may not be readily available on resource-constrained IoT devices [18]. This emphasizes the need for solutions that can be easily integrated into the IoT system.

Federated learning (FL), a form of Distributed Computing, emerges as a promising solution in this context [19] [20]. The federated learning is a decentralized learning concept capable of training a global machine learning (ML) model at the fog computing gateway without centralizing data sets, as illustrated in Figure 1.2 [21]. Instead of sending raw data to a central server, the model's weights are exchanged among devices, thereby circumventing the need for centralized data storage and processing. This not only addresses privacy concerns associated with transmitting sensitive data, but also leverages the inherent distribution of resources in the IoT environment. Federated learning thus aligns seamlessly with the principles of Edge and Fog Computing, offering a decentralized approach that optimally utilizes available resources and ensures real-time insights from IoT-generated data.

1.1 Challenges in Federated Learning for Processing IoT Streaming Data

Processing IoT streaming data through Federated learning presents various challenges that need to be addressed [22] [23]:

- **Heterogeneity of Data:** IoT devices generate diverse data types and formats. Federated learning must handle this variability and develop models that can learn effectively and generalize across different data sources.
- **Communication Efficiency:** Transmitting model updates between devices and a central server can consume significant bandwidth. In a streaming context, where data arrive in real-time, maintaining communication efficiency becomes even more critical to avoid delays and network congestion.
- **Dynamic Data Distribution:** IoT data streams can vary in volume, velocity, and distribution. Federated learning must adapt to these changes and ensure that the models remain accurate despite the dynamic nature of the data distribution.
- **Resource Constraints:** Many IoT devices have limited computational power and memory. Federated learning must account for these constraints and develop techniques that enable efficient model training and updates on resource-constrained devices.
- **Scalability:** Federated Learning needs to scale as the number of IoT devices increases. As the network grows, the management of communication, synchronization, and aggregation becomes more complex.

1.2 Objectives

The state of the art Federated Average (FedAvg) algorithm [24] [25] has challenges with respect to processing heterogeneous data in the IoT environment. To address this, there is a need to develop an efficient communication technique to process streaming data that can adapt to varying environments. To achieve this, we have formulated the following research objectives:

- Study existing DML frameworks such as Apache SparkML and experiment data processing in Fog-centric IoT Applications.
- Distribute data and computations to nodes and maintain shared parameters globally. Investigate the use of Fog Computing as a means of reducing communication costs in a federated computing environment.
- Process heterogeneous streaming data using novel communication-efficient algorithms in the FL environment that utilize client-side computation while minimizing the number of communication rounds required for convergence.
- Evaluate the performance of the proposed communication-efficient algorithms and computing techniques in terms of accuracy, speed, and scalability.

The focus of this research work is on 'Processing heterogeneous streaming data using Federated Learning in fog-centric IoT Applications'. The experimentation and validation done to meet the above objectives is done for a Smart-Home environment but can be deployable in applications Healthcare and Banking environments.

1.3 Thesis Organization

The organization of rest of the chapters in the thesis is as follows:

Chapter 2 provides a comprehensive review of the literature concerning the research focus, covering the IoT ecosystem, distributed machine learning, and federated learning for IoT data processing. It also presents the motivation behind our research and outlines the contributions.

Chapter 3 presents our research work related to fog computing-based smart home assistive living framework. Additionally, it delves into the study of distributed data processing in a fog environment and explores the adoptability of existing distributed machine learning frameworks in the IoT context.

Chapter 4 presents our research work on a federated learning framework based on fog computing to process heterogeneous sensor data on resource constraint computing devices. Experimental results related to FL based model training using heterogeneous IoT data are compared with the centralized model training.

Chapter 5 presents the research work related to load balance-aware federated learning to handle heterogeneous IoT streaming data in a fog computing-enabled smart home setting.

Chapter 6 presents the Capability-Aware Federated Averaging (CAFedAvg) algorithm that considers the capabilities of the clients participating in the federated learning process. Furthermore, Mobile-aware Neural Architecture Search (NAS) is explored for building machine learning models suitable for edge computing devices.

Chapter 7 presents the analysis of federated learning with distributed IoT data for the computation and storage of model parameters using the algorithms proposed in the fog computing framework.

Chapter 8 concludes the research work and suggests directions for future research.

In summary, this thesis aims to develop a comprehensive framework for heterogeneous sensor data streaming and efficient model training using federated learning in fog-centric IoT applications.

Chapter 2

Literature Review

This chapter provides a comprehensive literature review, covering the IoT ecosystem, the processing of IoT data using distributed machine learning, and federated learning. The review identifies the gaps in the existing literature for processing heterogeneous streaming data in IoT environments. Subsequently, the research contributions addressing these gaps are discussed.

2.1 Key Terms and Definitions

The definitions of the Cloud Computing, Fog Computing and Edge Computing are presented in this section, and the details about these concepts are elaborated further in subsequent sections.

- **Cloud Computing:** The National Institute of Standards and Technology (abbreviated NIST) provides a comprehensive definition of cloud computing [26], While it has not published specific definitions for fog and edge computing. They are often considered extensions of cloud computing.

  NIST characterizes cloud computing as *"a framework that facilitates widespread, user-friendly, and immediate network access to a collectively configurable set of computing assets (including networks, servers, storage, applications, and services) that can be swiftly allocated and relinquished with minimal administrative involvement or service provider engagement."*

- **Fog Computing:** The OpenFog Consortium published the *OpenFog Reference Architecture* that provides a comprehensive definition and framework

for fog computing [27]. It defines fog computing as *"a system-level architectural approach that disperses computing, storage, control, and networking functionalities closer to end-users along a continuum stretching from the cloud to IoT devices"*.

- **Edge Computing:** Edge computing positions applications, data, and processing at the logical peripheries of a network, rather than consolidating them centrally [27]. This approach, which places data and data-intensive applications at the edge, reduces the amount of data that needs to be transferred.

2.2 IoT Ecosystem

The Internet of Things ecosystem has experienced significant growth in recent years, revolutionizing various domains, including the smart home environment [1]. A key characteristic of IoT is the diverse nature of data generated by interconnected devices and sensors [3].

2.2.1 IoT Data Characteristics

In the IoT environment, data exhibit several characteristics, as illustrated in Figure 2.1 that pose challenges for seamless integration, processing, and analysis.

- **Heterogeneous Data Types:** In a smart home environment, data is collected from various sensors, such as temperature sensors, motion sensors, gas sensors, sound sensors, and cameras. Each sensor captures different types of data: numerical (ambient parameters), audio (voice commands), image (photos of objects), and video (surveillance). Handling such diverse data types in a smart home environment requires data preprocessing, transformation, and integration techniques to ensure seamless communication and meaningful insights across the different devices [4]. Integrating and fusing these diverse data types can be challenging due to variations in data quality, sampling rates, and measurement units. For example, temperature data may be measured in Celsius or Fahrenheit, while images and videos may have varying resolutions and formats.

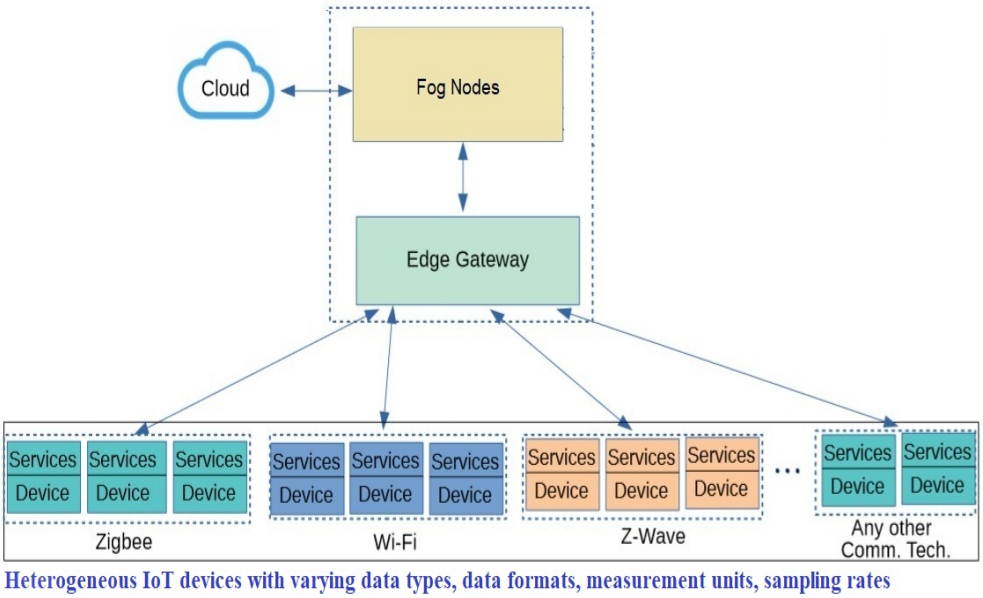


Figure 2.1: Fog Computing based Smart-Home Environment Architecture

- **Different Data Formats:** The data generated by various sensors in an IoT environment can vary in format due to differences in data transmission methods or communication protocols [4]. Different transmission methods utilize different physical media or technologies for data transfer. Examples of transmission methods include wired connections such as Ethernet cables, fiber optic cables, and serial connections, as well as wireless technologies such as Wi-Fi and Bluetooth. Communication protocols define the structure of data packets, error handling mechanisms, and the sequence of actions during data transmission. Communication protocols commonly used include Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Hypertext Transfer Protocol/Secure Hypertext Transfer Protocol (HTTP/HTTPS), Advanced Message Queuing Protocol (AMQP), Long Range Wide Area Network (LoRaWAN), and Zigbee, among others.

- **Variable Sampling Rates:** Each sensor has its own sampling rate, which determines how frequently it collects and updates its readings [28]. The temperature sensor updates its data every 30 seconds, whereas the motion sensor's data updates are event-driven, meaning it only reports data when

motion is detected. The light sensor updates its data every 1 minute, and the humidity sensor updates its data every 15 seconds.

As a result, the data collected by these sensors is not synchronized, and the rate at which new data is available from each sensor varies significantly. This can create challenges when analyzing and integrating sensor data to make informed decisions or trigger actions in the smart home environment. To address this issue, data processing techniques such as interpolation or data alignment may be used to normalize sensor readings, enabling synchronization of data points and creating a unified time series dataset. This allows for more effective data fusion and analysis.

- **Different Measurement Units:** Different manufacturers may manufacture devices that produce data in different formats [28]. For example, a temperature sensor from manufacturer 'A' may produce temperature readings in degrees Celsius, while that of manufacturer 'B' may produce temperature readings in Fahrenheit. Harmonizing data with different measurement units requires conversion techniques to ensure a consistent and meaningful representation of the data.

- **Heterogeneous Devices:** Various devices interconnected within the IoT ecosystem have different functionalities, processing power, communication protocols, and data handling capabilities [29]. These differences arise due to varying hardware specifications, software configurations, and intended use cases for each device. By leveraging the diverse capabilities of these devices, IoT applications can offer enhanced functionality, improved efficiency, and intelligent decision-making in various domains.

- **Communication Overhead:** In a smart home IoT ecosystem, numerous devices continuously exchange data, leading to communication overhead [29]. The constant data flow may result in network congestion and increased energy consumption. Managing communication overhead requires efficient communication protocols and mechanisms for data transmission and aggregation.

- **Data Distribution Imbalance:** Data distribution imbalance refers to the uneven distribution of data across the network of interconnected devices and sensors [30]. Some IoT devices may generate a significantly higher volume of data compared to others, leading to an imbalance in the data distribution.

Several factors contribute to the imbalance in data distribution in the IoT ecosystem. These include varying device capabilities, diverse sensor types, different data collection frequencies, and varying user behavior. For example, a high-resolution security camera may generate more data compared to a simple temperature sensor. Unbalanced data distribution can have implications for data processing and analytics. Devices generating large amounts of data may overload the network and central processing systems, leading to communication bottlenecks and increased latency. As a result, real-time data processing and analysis can be compromised. The imbalance in data distribution poses a challenge for machine learning algorithms used for predictive analytics and anomaly detection. Uneven data distribution can lead to biased training of machine learning models, affecting their accuracy and reliability.

- **Computational Complexity:** The computational complexity of IoT data processing refers to the level of computational resources and time required to process and analyze the huge amount of data generated by interconnected IoT devices [31]. The complexity arises due to the sheer volume, variety, and velocity of the data generated in real time. Processing data at a massive scale requires advanced computing techniques and scalable algorithms, necessitating the use of high-performance computing resources.

2.2.2 IoT Data Processing

Efficient IoT data processing paves the way for extracting valuable insights and actionable intelligence [5] [32].

2.2.2.1 Cloud Computing for IoT Data Processing

Traditionally, Cloud Computing solutions have been widely adopted for IoT data processing. Cloud computing provides scalable and cost-effective solutions to process large amounts of IoT data [33] [34]. IoT devices send their data to the Cloud, where it is stored, processed, and analyzed using various cloud-based services. Cloud computing offers substantial computing power and storage capacity, making it ideal for handling large datasets and complex machine learning algorithms. However, cloud computing faces challenges in managing real-time data processing due to potential delays caused by data transmission to remote servers and back.

2.2.2.2 Fog Computing: An Alternative Approach to IoT Data Processing

Fog computing is a distributed computing approach that moves computation, storage, and intelligence closer to the network's periphery, thereby reducing the necessity for data to travel to the cloud. This extension of cloud capabilities to the network edge allows for real-time data processing and mitigates communication overhead, as detailed in [35]. By deploying computing resources closer to IoT devices, fog computing addresses the latency and bandwidth challenges faced by cloud computing in handling real-time data. Fog nodes, acting as intermediaries between the cloud and edge devices, provide computational support for edge devices and facilitate efficient data processing and analysis.

2.2.2.3 Edge Computing vs. Fog Computing

While edge computing and fog computing are often used interchangeably, they are distinct concepts. Edge computing focuses on processing data locally on edge devices, closer to the data source, to achieve lower latency and reduce reliance on cloud servers. Fog computing, on the other hand, introduces an intermediate layer of fog nodes between edge devices and the cloud [9] [13]. Fog nodes act as distributed mini data centers that offer more processing and storage capabilities than edge devices but are still closer to the data source compared to the cloud. Fog computing provides a balance between edge computing's local processing and cloud computing's vast resources, making it well-suited for real-time and low-latency IoT data processing.

IoT data processing using cloud computing and machine learning has been widely adopted for its scalability and computational power. However, the limitations of cloud computing in handling real-time data have driven the emergence of fog computing. Fog computing brings computational resources closer to IoT devices, reducing communication overhead and enabling real-time data processing. Fog computing offers a promising alternative to cloud-centric IoT data processing, catering to the growing demand for low-latency and responsive IoT applications. Table 2.1 compares the features of cloud, fog, and edge computing scenarios.

Table 2.1: Features Comparison - Cloud vs. Fog vs. Edge Computing

	Cloud	**Fog**	**Edge**
Processing Capability	High	Limited	Low
Memory	High	Limited	Low
Latency	High	Low	Lowest
Scalability	High	Scalable within Network	Low
Distance	Far from edge	Close to the edge	At the edge
Interoperability	High	Medium	Low

2.3 Distributed Machine Learning for IoT Data Processing

The act of distributing machine learning applications across multiple nodes is called Distributed Machine Learning (DML). Distributed machine learning leverages the power of multiple computing nodes or devices to collaboratively process and analyze data, enabling efficient and scalable data processing for IoT applications [15]. Although fog computing already offers improved processing capabilities by bringing computation closer to IoT devices, there is an opportunity to further improve the performance of the IoT system through effective utilization of cluster resources. Leveraging the computational power of a cluster environment can lead to significant improvements in the execution of IoT tasks, optimizing data processing, and enabling more complex analytics. By efficiently distributing and balancing workloads across the cluster, the IoT system can achieve higher throughput, reduced latency, and enhanced scalability, thus unlocking the full potential of fog-centric IoT applications.

2.3.1 Features of Distributed Machine Learning

Here are some key aspects of the role of distributed machine learning in processing IoT data:

- Scalability: Distributed machine learning allows IoT data processing to scale easily with the increasing number of connected devices and data sources. As the IoT ecosystem grows, traditional centralized approaches

may become overwhelmed by the sheer volume of data, leading to processing bottlenecks. Distributed machine learning techniques distribute the computational load across multiple nodes, ensuring efficient processing even with large amounts of data.

- Real-Time Responsiveness: Many IoT applications require real-time or near-real-time data processing to enable timely decision-making and response. Distributed machine learning can perform data processing in parallel across distributed nodes, reducing latency and improving the responsiveness of IoT applications.
- Privacy: In many IoT scenarios, data privacy and security are paramount concerns. Centralized data processing may involve transmitting sensitive data to a remote server, raising potential privacy risks. Distributed machine learning allows data to be processed locally at the edge or fog nodes, reducing the need for data transmission and improving data privacy.
- Edge and Fog Computing Integration: Distributed machine learning seamlessly integrates with edge and fog computing paradigms in the IoT ecosystem. Edge devices can perform local data processing using distributed machine learning models, reducing the need for constant data transmission to the cloud or central servers. This integration optimizes data processing and minimizes communication overhead.
- Fault tolerance: In large-scale IoT deployments, failure of individual nodes or devices is inevitable. Distributed machine learning offers fault tolerance by distributing tasks across multiple nodes. If a node fails, processing can continue on other nodes, ensuring the overall stability and reliability of the system.
- Resource efficiency: Distributed machine learning optimizes resource utilization by distributing computational tasks among available nodes. This approach reduces the overall computational burden and efficiently utilizes computing resources, especially in resource-constrained IoT environments.

By leveraging collaborative model training, reduced communication overhead, and integration with edge and fog computing, distributed machine learning empowers resource-constrained devices to process and analyze data locally, preserving privacy, conserving energy, and supporting real-time data processing in the dynamic and interconnected world of the Internet of Things.

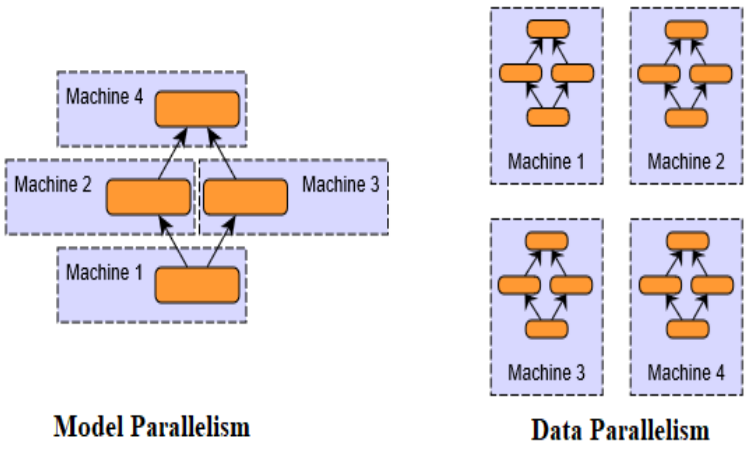


Figure 2.2: Distributed Machine Learning Approaches

2.3.2 Model Parallelism vs. Data Parallelism

Model parallelism and data parallelism are two different approaches used in distributed machine learning to train large models on multiple devices or nodes, as illustrated in Figure 2.2 [36]. They have distinct characteristics and are suitable for different scenarios.

Model Parallelism: In model parallelism, a large neural network model is partitioned into multiple segments, and each segment is assigned to different devices or nodes for training. Each device handles a specific part of the model, and during training, they communicate the intermediate results to synchronize the overall model updates. This approach is typically used when a single device does not have enough memory to store the entire model.

Advantages of Model Parallelism:

- Suitable for training large models that do not fit in the memory of a single device.
- Enables scaling up the model size by utilizing multiple devices efficiently.
- Allows flexibility in choosing devices with different computational capabilities for specific model segments.

Disadvantages of Model Parallelism:

- Requires frequent communication between devices, leading to increased communication overhead.

- Can be more challenging to implement and manage compared to data parallelism.

Data Parallelism: In data parallelism, the entire model is replicated on multiple devices, and each device is assigned a different subset of the training data. The devices independently compute gradients based on their assigned data and then communicate these gradients to update the model parameters. This approach is suitable when the model can fit in the memory of each device and the computational workload per device is manageable.

Advantages of Data Parallelism:

- Efficiently utilizes multiple devices to process different parts of the data simultaneously.
- Offers better scalability when dealing with large datasets.
- Generally simpler to implement and manage compared to model parallelism.

Disadvantages of Data Parallelism:

- May not be suitable for models that are too large to fit into the memory of a single device.
- Synchronization of model updates can become a bottleneck, especially when communication bandwidth is limited.

In summary, model parallelism and data parallelism are both important techniques in distributed machine learning, each with its strengths and limitations. The choice between these approaches depends on factors such as model size, data size, available computational resources, and communication capabilities of the devices involved. Some distributed machine learning frameworks, like TensorFlow, offer the flexibility to use a combination of both model parallelism and data parallelism to achieve the best performance for specific scenarios.

2.3.3 Parameter Averaging

Parameter averaging is conceptually the simplest approach to data parallelism [37] (illustrated in Figure 2.3. With parameter averaging, training proceeds as follows:

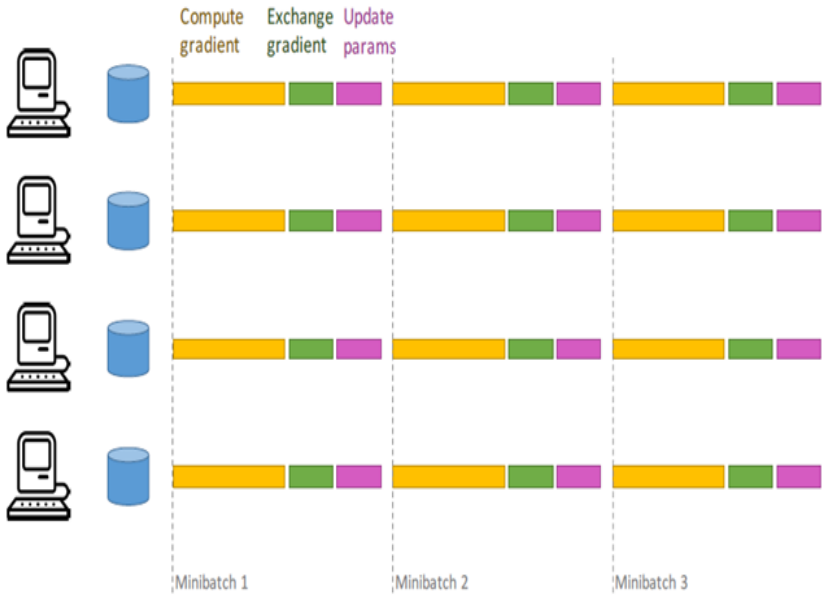


Figure 2.3: Basic Operation of Data Parallel SGD

1. Initialize the model parameters randomly based on the model configuration.
2. Share the current parameters with each worker.
3. Each worker processes a portion of the data for training.
4. Update the global parameters by averaging the parameters from all workers.
5. If there is more data to process, return to Step 2.

In distributed machine learning, there are two main paradigms for updating model parameters across multiple devices or nodes during the training process: Synchronous update and Asynchronous update [18]. These paradigms are variations of stochastic gradient descent (SGD) that are used to optimize the model parameters.

2.3.3.1 Synchronous Update (Synchronous Stochastic Gradient Descent)

In synchronous update, all devices or nodes wait for each other to complete their local gradient computations before synchronizing and updating the model parameters simultaneously [38]. During each training iteration, all devices

compute gradients using their local data, and then exchange these gradients with each other before applying the updates. This means that all devices are in sync with each other during the training process.

Advantages of Synchronous Update:
Better consistency: Since all devices update the model parameters simultaneously, they remain consistent with each other.
Convergence guarantees: Synchronous update ensures that the model converges to a similar solution across all devices.
Disadvantages of Synchronous Update:
Increased communication overhead: All devices need to communicate at each iteration, leading to higher communication costs and potential delays.
Straggler effect: The training process can be slowed down if one or more devices take longer to complete their computations.

2.3.3.2 Asynchronous Update (Asynchronous Stochastic Gradient Descent)

In asynchronous update, devices update the model parameters independently and at their own pace without waiting for other devices [38]. Each device computes gradients using its local data and directly updates the model parameters without synchronizing with other devices. This approach allows for more flexibility and does not require strict coordination among devices.
Advantages of Asynchronous Update:
Reduced communication overhead: Devices can update parameters independently, leading to lower communication costs and potential speed-ups.
Better utilization of resources: Devices are not idle waiting for others to finish their computations, leading to better resource utilization.
Disadvantages of Asynchronous Update:
Inconsistent Model States: Since devices update parameters independently, they may diverge from each other, leading to inconsistency in the model across devices.
Convergence challenges: Asynchronous updates can introduce noise and make it harder to achieve convergence, especially when the learning rate is not well-tuned.
The choice between synchronous and asynchronous update depends on various

factors, including the communication infrastructure, the scale of the distributed system, and the specific machine learning model being used. Asynchronous updates are often preferred when communication is a bottleneck and devices have varying computation speeds. However, synchronous updates are used more frequently when consistency and convergence guarantees are critical and communication overhead is manageable. Some hybrid approaches that combine the benefits of both paradigms also exist to strike a balance between communication efficiency and model convergence.

2.3.4 Existing Distributed Machine Learning Frameworks

This section reviews the features of various existing distributed machine learning frameworks.

2.3.4.1 Apache Spark

Spark is an open-source platform that defines a set of general-purpose APIs for Big Data processing. The key feature of Spark is its resilient distributed data sets (RDDs) [39], which can be represented by the nodes in a data processing graph. The edges represent the corresponding transformations that the RDDs will undergo. In spirit, the RDD-based programming model is quite similar to Hadoop Map-Reduce, and supports functional-style manipulations on RDDs via 'transformations', 'actions'. The difference between Spark and Hadoop lies in that Spark can cache the intermediate computation results in memory rather than dumping all results to the disk, which involves much slower disk I/O [40]. Since Spark can also dump the result to the disk, it is strictly a superset of Hadoop Map-Reduce, leading to much faster speed, especially after the first iteration. Spark is based on the data parallelism paradigm [41] with the following features:

- Low latency because of in-memory computation
- Speed: Fast for large scale data processing
- Polyglot: We can write applications in Java, Scala, Python, and R

Spark has two machine learning libraries—Spark MLlib and Spark ML—with very different APIs but similar algorithms [42]. These machine learning libraries inherit many of the performance considerations of the RDD and Dataset APIs they are based on, but also have their own considerations. MLlib is the first of

the two libraries and is entering a maintenance/bug-fix-only mode. Spark ML is the newer, scikit-learn inspired, machine learning library and is where new active development is taking place.

SparkML aims to provide a uniform set of high-level APIs that help users create and tune practical machine learning pipelines. Spark ML standardizes APIs for machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow.

- Distributed Computing:Apache SparkML is built on Apache Spark, a distributed computing framework. It allows parallel processing of data across multiple nodes, making it highly scalable and efficient for big data processing.
- MLlib Library: SparkML provides MLlib, a library with various machine learning algorithms for classification, regression, clustering, and collaborative filtering, among others.
- Ease of Use: SparkML offers a user-friendly API in Scala, Java, Python, and R, making it accessible to a wide range of developers and data scientists.
- Integration with the Spark Ecosystem: SparkML seamlessly integrates with other Spark components, such as Spark SQL and Spark Streaming, enabling end-to-end data processing and analytics.
- In-memory Processing: Apache SparkML leverages in-memory computing, which enhances performance by reducing data movement between disk and memory.

2.3.4.2 Microsoft's DMTK and CNTK

DMTK (Distributed Machine Learning Toolkit) is a platform designed for distributed machine learning [43]. In recent years, practices have demonstrated the trend that more training data and larger models tend to generate better accuracies in various applications. However, it remains a challenge for common machine learning researchers and practitioners to learn big models from the huge amount of data, because the task usually requires a large number of computation resources. To tackle this challenge, Microsoft has released DMTK, which contains both algorithmic and system innovations. These innovations make big data machine learning tasks highly scalable, efficient, and flexible. The algorithms

released in DMTK are mostly non-deep learning algorithms. For state-of-the-art deep learning tools, Microsoft has released CNTK (Cognitive Toolkit), which provides asynchronous parallel training functionalities [44]. CNTK describes neural networks as a series of computational steps through a directed graph. It allows the user to easily realize and combine popular model types such as feedforward deep neural networks (DNNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs/LSTMs). CNTK implements stochastic gradient descent (SGD, error backpropagation) learning with automatic differentiation and parallelization across multiple GPUs and servers.

2.3.4.3 Distributed TensorFlow

Distributed TensorFlow is an extension of the TensorFlow machine learning framework that enables the training and inference of machine learning models on multiple devices or across multiple machines [45]. It uses distributed computing techniques to accelerate the training process and efficiently handle large-scale datasets [46]. Some key features of distributed tensorflow are:

- **Distributed Training:** One of the primary features of Distributed TensorFlow is the ability to distribute the training process across multiple devices or machines. This is achieved through data parallelism or model parallelism, allowing the model to be trained on different subsets of data or different parts of the model to be trained on separate devices simultaneously. Parameter Servers: In a distributed setting, parameter servers are used to store and manage the model's parameters. These servers act as a centralized storage system for the model variables and are accessed by worker nodes responsible for the actual training.

- **Asynchronous Updates:** Distributed TensorFlow supports asynchronous training, where different worker nodes can update the model's parameters independently and asynchronously. This can lead to faster training as the model can continue training while waiting for updates from other nodes.

- **Synchronization:** To ensure that the model parameters remain consistent during training, Distributed TensorFlow provides mechanisms for synchronization between the parameter servers and the worker nodes. This synchronization ensures that the model's parameters are updated properly across all nodes.

- **TensorFlow's Estimator API:** TensorFlow provides an Estimator API that simplifies the process of building distributed models. The Estimator API abstracts away much of the complexity of distributed training, making it easier for developers to scale their models to multiple devices or machines.
- **Horovod Integration:** TensorFlow supports integration with Horovod, an open source distributed deep learning training framework. Horovod simplifies distributed TensorFlow training and allows seamless scaling of models to multiple GPUs and machines.
- **Fault Tolerance:** Distributed TensorFlow incorporates fault tolerance mechanisms to handle failures in the distributed system. It can recover from failures and continue training without losing progress.
- **TensorBoard for Visualization:** TensorBoard, TensorFlow's visualization toolkit, can be used to monitor the training progress of distributed models. It helps visualize performance metrics, network architectures, and other essential aspects of the training process.

Distributed TensorFlow is particularly useful for large-scale machine learning tasks that require significant computational resources, such as training deep neural networks on massive datasets. By leveraging the capabilities of distributed computing, it enables faster and more efficient training, ultimately leading to more powerful and accurate machine learning models.

2.3.4.4 Comparison of existing DML frameworks

The feature-wise comparison of Apache Spark MLlib, Microsoft DMTK, and distributed TensorFlow machine learning frameworks is given in Table 2.2. Each framework has its merits and demerits, and the choice of DML framework depends on the specific use case, data scale, and the type of machine learning or deep learning tasks we intend to tackle. Additionally, factors such as community support, documentation, and ease of use also influence the choice of a particular DML framework.

2.4 Federated Learning for IoT Data Processing

The Federated Learning (FL) computing approach is a distributed learning paradigm that can train a global ML model at the fog computing gateway for

Table 2.2: Comparison of Existing DML Frameworks

	Apache Spark MLlib	Microsoft DMTK	Distributed TensorFlow
Framework Type	Distributed machine learning library integrated with Apache Spark for big data processing	Distributed machine learning framework focused on deep learning tasks and neural networks	Extension of TensorFlow for distributed training and inference of deep learning models
Distributed Computing	Leverage the distributed computing capabilities of Apache Spark to process large-scale data	Supports distributed training of deep learning models across multiple GPUs and machines	Utilizes distributed computing techniques to train models across multiple devices or machines
Machine Learning Algorithms	Provides a rich set of machine learning algorithms for classification, regression, clustering, collaborative filtering, etc.	Focused on deep learning algorithms and neural networks, but may not offer the same breadth of traditional ML algorithms as Spark MLlib	Primarily used for deep learning tasks with a focus on training deep neural networks
Deep Learning Support	Offers some support for basic deep learning tasks but is more oriented towards traditional machine learning algorithms	Specialized for deep learning tasks, providing optimized algorithms for training neural networks	Designed for deep learning with extensive support for creating and training deep neural networks
Performance and Scalability	Efficiently scales to large datasets and clusters, suitable for big data applications	Optimized for performance on multi-GPU and multi-machine systems, enabling faster training on large datasets	Provides good performance in distributed setups, efficiently scaling across devices and machines
Flexibility	Offers flexibility in handling various data types and integration with other Spark components	Provides flexibility to experiment with new deep learning algorithms and architectures	Flexible in defining custom network architectures and training algorithms
Integration	Easily integrates with the broader Apache Spark ecosystem for data processing and analytics	Can be integrated with other Microsoft AI tools and services for end-to-end AI solutions	Seamlessly integrates with the TensorFlow ecosystem and its extensive features and tools
Use Case	Well-suited for big data analytics and traditional machine learning tasks on large datasets	Best suited for deep learning research and tasks involving complex neural network architectures	Ideal for large-scale deep learning tasks and training complex deep neural networks

many connected edge devices without centralizing the data sets [19] [21]. FL can improve the accuracy of the model in the context when a user does not have sufficient or related data. Moreover, FL is an apt solution in certain situations where data processing at the central server is undesirable due to privacy issues.

The design issues involved in federated learning based model training are given below [47]:

1. Handling of Non-Uniform Data: Federated learning faces challenges when clients generate non-uniform data due to varying data distributions across devices. To address this, techniques like weighted averaging can be employed during model aggregation to give more importance to clients with larger datasets or more representative data. Additionally, personalized federated learning approaches can adapt the model for individual clients to accommodate their specific data distributions.

2. Ensuring Convergence in Non-IID Data Scenario: In scenarios where clients have nonindependent and nonidentically distributed (Non-IID) data, convergence of the federated model can be challenging. Advanced optimization techniques, like federated averaging with momentum, can be used to handle the asynchrony in updates and promote convergence even in Non-IID settings.

3. Energy Efficiency: Energy-efficient federated learning is crucial for resource-constrained devices. Clients can form clusters to collectively train an aggregated model using their local data and then disseminate it to the central server. This hierarchical approach reduces energy consumption by limiting communication overhead while ensuring model accuracy.

4. Dealing with Drop-Out Clients: Federated learning may face issues when clients drop out during model training. To mitigate this, redundancy can be introduced by increasing the number of participating clients or implementing fault-tolerant mechanisms. Additionally, client re-joining strategies and model compression techniques can help maintain convergence speed and model accuracy.

5. Dealing with Mobility of Clients: In dynamic environments, clients can join or leave the federated learning process. Adaptive algorithms and dynamic client selection can handle client mobility. Furthermore, considering both

client mobility and their available bandwidth is essential to ensure efficient and effective communication during training.

6. Incentive Mechanisms for Client Participation: To encourage the transparent participation of clients, an incentive mechanism is required. Clients can be rewarded for their contributions to model training or for sharing their data for the federated learning process. Implementing privacy-preserving techniques can also build trust among clients and promote their willingness to participate in federated learning.

2.5 Distributed Machine Learning vs. Federated Learning

Federated Learning and Distributed Machine Learning are both approaches to train machine learning models across multiple devices or nodes, but they have some key similarities and differences:
Similarities:

1. Decentralized Training: Both Federated Learning and Distributed Machine Learning involve training machine learning models in a decentralized manner. Data is kept localized on individual devices or nodes, and model updates are computed and aggregated across multiple devices.

2. Scalability: Both approaches aim to address the scalability challenges of traditional centralized training. By distributing the computation across multiple devices, they can handle large datasets and complex models more efficiently.

3. Privacy Preservation: Both Federated Learning and Distributed Machine Learning are designed to preserve data privacy. Rather than transmitting raw data to a central server, the exchange involves model updates or gradients, preserving the privacy of sensitive data on the devices.

4. Communication Efficiency: Both approaches focus on reducing communication overhead by exchanging only model updates or gradients, rather than transmitting the entire dataset, which can be resource intensive.

Differences:

1. Data Distribution: The primary difference between Federated Learning and Distributed Machine Learning lies in how data is distributed. In Federated Learning, data is distributed across multiple devices or clients, often with different data distributions. In contrast, Distributed Machine Learning typically assumes a homogeneous distribution of data across the nodes.

2. Aggregation Mechanism: In Federated Learning, model updates from multiple devices are typically aggregated using methods like federated averaging or secure aggregation to form a global model. In Distributed Machine Learning, model updates from different nodes are combined using traditional distributed optimization techniques.

3. Heterogeneity: Federated learning is designed to handle heterogeneity in terms of data distribution, device capabilities, and network conditions. It can accommodate devices with varying computational resources and data characteristics. Distributed Machine Learning often assumes homogeneous computing resources and data distribution across nodes.

Use Cases: While both approaches can be applied to various domains, Federated learning is often used in scenarios where data is distributed among many user devices or edge nodes, such as mobile devices, IoT devices and edge servers. Distributed Machine Learning is commonly employed in distributed computing environments with a focus on parallelization and scaling across a cluster or data center.

In summary, Federated Learning and Distributed Machine Learning are similar in their decentralized and privacy-preserving nature but differ in the handling of data distribution, the aggregation mechanism, and the ability to accommodate heterogeneity. The choice between these approaches depends on the specific use case and the nature of the distributed data and computing environment.

2.6 Motivation

The rapid proliferation of Internet of Things (IoT) devices has led to an explosion of sensor-generated data, which has enormous potential for valuable insights and real-time decision-making. Traditional approaches for processing IoT data often involve centralized Cloud Computing, which may not be suitable for real-time

applications and can result in high communication overhead. Fog Computing, as an extension of Cloud Computing, has emerged as a promising paradigm to address these challenges by bringing computing closer to the edge devices.

However, existing work for processing IoT data in the fog-centric environment has not adequately addressed the heterogeneity aspect of data and devices. IoT data streams can vary significantly in terms of formats, sampling rates, measurement units, and data types, which presents a challenge for effective data fusion and analysis. Moreover, the IoT devices themselves come with diverse computational capabilities, ranging from resource-constrained devices to more powerful edge servers.

The primary problem identified in this research is the lack of comprehensive solutions that consider the heterogeneity of streaming IoT data and the diverse computing capabilities in fog-centric IoT applications. Existing work often overlooks the need for efficient data distribution and resource utilization in the fog cluster. This leads to suboptimal execution of machine learning tasks, hindering the potential of Federated Learning in IoT applications.

To address this problem, this research work aims to design and implement a novel approach that leverages Federated Learning techniques to process heterogeneous sensor data in a fog-centric IoT environment. By aggregating computing resources in the fog cluster, the proposed solution aims to achieve effective resource utilization and data distribution among nodes.

2.7 Research Contributions

This thesis contributes mainly to the design of "fog-based Federated Learning framework that processes heterogeneous streaming sensor data" in a Smart-Home environment.

- **Contribution 1:** Fog computing framework that assists the residents of a smart home environment by processing IoT data locally without sending it to the cloud.
- **Contribution 2:** Federated learning based system to distribute load among nodes in the fog cluster while processing streaming IoT data and maintaining shared parameters globally.

- **Contribution 3:** A novel capability-aware federated average algorithm that takes into account the heterogeneity of devices during FL-based model training. .

The part of literature review presented in this chapter is published in our following research publications:

- S. R. Rudraraju, N. K. Suryadevara and A. Negi, **Face Recognition in the Fog Cluster Computing**, *Proceedings of International Conference on Signal Processing, Information, Communication & Systems (SPICSCON), Dhaka, Bangladesh, November 28-30, 2019, IEEE*, Pages: 45-48, 2019, doi: 10.1109/SPICSCON48833.2019.9065100. Indexed in: Scopus. Status: Accepted and Published
  URL: https://ieeexplore.ieee.org/abstract/document/9065100

- S. R. Sahith, S. R. Rudraraju, A. Negi and N. K. Suryadevara, **Mesh WSN Data Aggregation and Face Identification in Fog Computing Framework**, *Proceedings of 13th International Conference on Sensing Technology, ICST 2019, Sydney, Australia, December 2-4, 2019, IEEE*, Pages: 1-6, 2019, doi: 10.1109/ICST46873.2019.9047708. Indexed in: DBLP, Scopus. Status: Accepted and Published
  URL: https://ieeexplore.ieee.org/abstract/document/9047708

2.8 Summary

In this chapter, a thorough literature review was conducted on the IoT ecosystem, data processing with distributed machine learning, and federated learning. It highlighted the challenges in processing heterogeneous streaming data in the IoT environment and identified the gaps in existing research work. The contributions of this study aim to optimize resource utilization to effectively process heterogeneous streaming data in the IoT environment.

Chapter 3

Distributed Machine Learning for IoT Data Processing

In this chapter, research works based on the fog computing framework aimed at supporting smart-home residents are presented. Additionally, the implementation aspects related to distributed storage and processing of datasets within the fog environment are examined. To conclude the chapter, we present insights derived from the experiments conducted in this research work.

3.1 A Smart-Home Assisted Living Framework

This section presents research works on fog-enabled smart home environments that help residents and enhance their living experience.

3.1.1 Audio and Lighting Stimulation based System to Assist Dementia People

This work introduces an assistive living framework aimed to support elderly individuals suffering from dementia. The approach uses a novel fog computing based recognition model that assists elderly people within a smart home environment to quickly recall and identify familiar visitors and household items. When a known visitor arrives the house, relevant audio and lighting stimuli associated with the visitor are given using smart speakers and hue lights. This leverages the principles of music and light therapy to trigger associative recall for the dementia people to

recognize the visitors quickly. Similarly, tailored auditory and lighting cues are employed to assist individuals in accurately recognizing domestic objects. The visitor is recognized using the Local Binary Point Histogram (LBPH) Classifier model [48] from OpenCV [49]. The system recognizes 85% of known visitors, and realizes the benefits of music and lighting therapy for everyday living.

3.1.1.1 Basic Concepts

Dementia is a persistent condition characterized by a gradual decline in cognitive functions, primarily affecting the elderly population. It has already impacted an estimated 50 million individuals globally and is projected to affect around 152 million people by the year 2050, as indicated in a study by [50]. One of the prominent symptoms of dementia is the loss of memory related to object placement, recent events, and even the recognition of individuals. In dementia care centers, sensory stimulation has gained recognition as a therapeutic approach, encompassing the activation of various senses such as sight, smell, hearing, touch, and taste as mentioned in [51]. Different forms of sensory stimulation offer unique advantages, with music and light therapy emerging as particularly effective methods, as discussed below.

Studies indicate that colors have a notable impact on memory, acting as a potent information channel for the human cognitive system and augmenting memory function [52] [53]. Audio stimulation also proves to be effective in improving mood, relaxation, and cognition [54]. Music, in particular, aids people with dementia in recalling their past experiences and mitigating feelings of anxiety and tension. The combined impact of music and light therapy helps people with dementia to recognize their surroundings more effectively and maintain their sense of identity.

The evolution of Internet of Things (IoT) technology has transformed the design of smart homes, making it a prominent research area. Smart home environments tailored for healthcare play a vital role in assisting older or disabled individuals to live independently. Modeling a smart home involves three core tasks: sensing, reasoning, and acting, as detailed in [55]. The sheer volume of data and the computational capabilities of devices situated between the cloud and data sources have given rise to the concept of fog computing. Unlike traditional cloud-centric approaches, fog computing involves distributing a portion of the computation closer to the data source. This paradigm offers several advantages, including faster response times, enhanced data security and privacy, and reduced bandwidth consumption, as highlighted in [6] and [11].

This work introduces a framework for an audio and lighting stimulation program tailored to the elderly within a smart home environment, employing the fog computing model. The model is trained to recognize familiar individuals and the classification process is conducted by fog devices. This approach yields quicker response times compared to conducting the training and classification in the cloud. When the model detects and classifies any person or object from the trained dataset, it activates audio and lighting stimulation. This, in turn, stimulates the associative recall mechanism, assisting the elderly in better identifying the person or object.

3.1.1.2 System Description

Figure 2.1 illustrates the general structure of our smart home environment based on fog computing. In the lower tier of this architecture, the sensor node gathers data from various sensors, conducts some initial processing, and establishes communication with other nodes in the network. Normally, all these sensor nodes establish communication with a gateway device, which has greater processing and storage capabilities in comparison to the sensor nodes.

The selection of communication protocols, such as Zigbee, Wi-Fi, Z-Wave, and others, is contingent on the particular gateway device and sensor nodes chosen according to the application's demands. The edge gateway accumulates data from diverse sensor nodes, which is subsequently refined by fog devices to align with the precise application requirements.

- **OpenCV and Fog Node:** Open Source Computer Vision (OpenCV) Library is a free library equipped with a range of built-in functions for implementing computer vision and machine learning algorithms [49]. These algorithms find applications in tasks like object identification and face detection, among others. In our setup, we employ the OpenCV Haar Cascade Classifier to identify facial features within images, while the LBPH algorithm is utilized for the visitor classification machine learning model. This model is trained on the fog node. The fog node then regulates the suitable audio-visual cues to aid residents with dementia in promptly recognizing visitors.

- **Audio-Lighting Stimuli and Sensing Units:** In the smart home environment, two distinct types of sensing units, also referred to as sensor nodes, are employed to efficiently manage data. Sensor unit type #1 is used to find

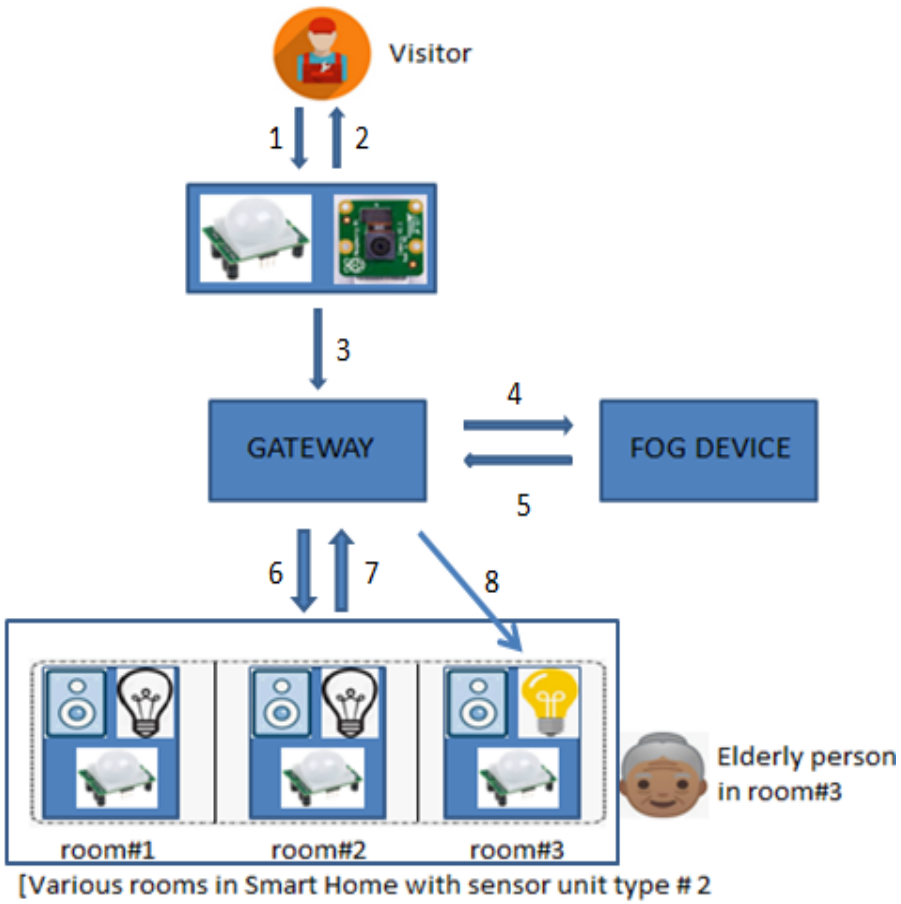
movement at the entrance of the house and capture visitor picture through the camera sensor. This image is then transmitted to the fog node for visitor identification. Sensor unit type #2 is associated with specific house-hold objects to determine the resident's interaction with them, enabling the provision of suitable audio and lighting cues. Furthermore, this unit can also identify the resident's location, indicating the room they are in, and relay this information to the fog node when a visitor arrives. This ensures that the relevant audio and lighting cues for the visitor are exclusively activated in that specific room.

- **Hue Lights and openHAB:** The open Home Automation Bus (openHAB) serves as a home automation platform with the capability to interface with a diverse array of devices and systems [56]. openHAB establishes electronic communication with devices within the smart home ecosystem and executes actions as per user-defined configurations. Within our smart home setup, openHAB wirelessly governs various parameters of hue lights, including color, brightness, and saturation. Figure 3.1 illustrates the functioning of the system implemented through our proposed framework.

3.1.1.3 Implementation Details

Experimental Setup: The implementation employed Raspberry Pi computing devices along with a diverse set of sensors and actuators, including PIR Sensors [57], RPi Camera [58], Philips Hue Lights [59], and Smart Speakers. Figure 3.2(a) illustrates the arrangement of various sensor nodes, fog devices, and the gateway in our experimental setup. In this configuration, two Raspberry Pi units were designated for the roles of the fog node and the gateway. The sensor nodes use Wi-Fi for communication among them.

The initial step involves training the visitor classification machine learning model using OpenCV on fog device, which utilizes a set of images featuring familiar individuals. Sensor unit type#1 is responsible for linking the RPi camera sensor with the PIR motion sensor through the RPi module. Positioned near the house's entrance, as depicted in Figure 3.2(a), sensor unit type#1 detects any motion using the PIR sensor, subsequently activating the camera sensor. Captured images are stored within the local database of sensor unit type#1. The



1 - PIR motion sensor identifies the person
2 - Camera sensor captures the photo
3 - Sensor unit type # 1 sends the photo to Gateway
4 - Sending the photo to Fog device for identification of person
5 - Selection of appropriate audio and lighting effect
6 & 7 - Identification of location (room) of elderly person in the home
8 - Triggering the selected audio & lighting effect in the appropriate room

Figure 3.1: Basic Operation of the Proposed Stimulation System

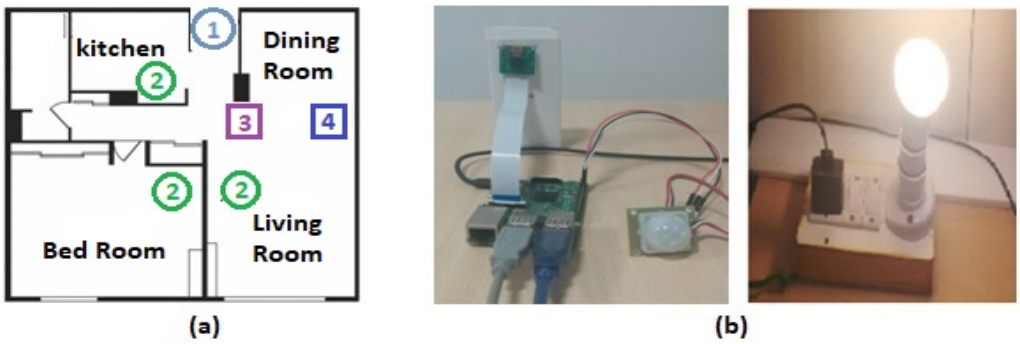


Figure 3.2: (a)Placement of Different Devices and Sensor Nodes (1: Sensor Unit Type#1, 2: Sensor Unit Type#2, 3: Gateway, 4: Fog Device) (b) RPi Connecting Camera and PIR Sensors, One Hue Light used in the Smart Home

fog node, equipped with its local database, retrieves these images for classification through database synchronization. For visitor classification, we utilized the LBPH classifier model from OpenCV.

The fog node is designed to activate the relevant audio and lighting cues within the room where the dementia person is located when a visitor arrives. As depicted in Figure 3.2(a), we have deployed three units of sensor type #2, each comprising a hue light, a speaker for playing music, and a PIR sensor, strategically positioned in the kitchen, bedroom, and living room. A PIR sensor is linked to a specific domestic object in each of these rooms (e.g., the sofa set in the living room, the bed in the bedroom, and the oven in the kitchen).

This PIR sensor serves a dual purpose: firstly, it detects the resident's proximity to the associated object, and secondly, it identifies the resident's location (the room) when a visitor arrives. Through the gateway device, the fog node then activates the relevant audio and lighting stimuli on the speaker and hue light located in the respective room whenever a visitor arrives. Figure 3.2(b) shows the sensing node at the entrance and one of the Philips hue lights used in our smart home setup.

3.1.1.4 Experimental Results

The visitor classification model, trained with 200 images of five known individuals, achieves an 85% accuracy rate in just 48 seconds. When any of these five people visits, the model identifies them based on training data. For demonstration, we

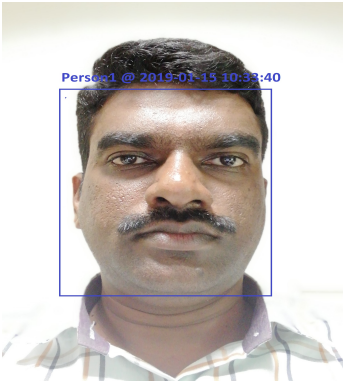


Figure 3.3: Visitor identification by fog node

```
MariaDB [SmartLab]> select * from PhilipsHueLight;
+-------+-------+-------+------------+------------+-------------+---------------------+
| sl_no | id    | color | saturation | brightness | temperature | entry_time          |
+-------+-------+-------+------------+------------+-------------+---------------------+
|    10 | Hue_1 |    45 |         59 |         72 |          65 | 2018-12-03 02:53:03 |
|    11 | Hue_2 |   221 |         11 |         40 |          18 | 2018-12-03 02:53:03 |
|    12 | Hue_1 |   219 |         61 |         59 |           0 | 2018-12-03 02:56:04 |
|    13 | Hue_2 |   143 |         29 |         40 |           2 | 2018-12-03 02:56:04 |
|    14 | Hue_1 |    80 |         16 |         27 |          29 | 2018-12-03 04:55:35 |
|    15 | Hue_2 |   291 |         85 |         66 |           8 | 2018-12-03 04:55:35 |
|    16 | Hue_1 |    80 |         16 |         27 |          29 | 2018-12-03 04:58:36 |
|    17 | Hue_2 |   291 |         85 |         66 |           8 | 2018-12-03 04:58:36 |
|    18 | Hue_1 |    80 |         16 |         27 |          29 | 2018-12-03 06:03:52 |
|    19 | Hue_2 |   291 |         85 |         66 |           8 | 2018-12-03 06:03:53 |
|    20 | Hue_1 |   166 |         42 |         59 |           0 | 2018-12-03 06:06:53 |
```

Figure 3.4: Portion of PhilipsHueLight table which stores hue light stimuli information

connected a display device to the fog node, showing the visitor's photo and label (as shown in Figure 3.3).

The implemented system was used continuously for a month, collecting sensor data, audio and lighting cue information, and storing in the database to verify the system functionality. Figure 3.4 shows a snapshot of the database table, that stores the hue light values. Table 3.1 displays the results of the classifier algorithm for one specific day. Out of the five individuals trained, the classifier correctly identified four, based on camera images captured by sensing unit type #1. However, it made an incorrect classification, mistaking person 1 for person 4 at the timestamp 2018-12-03 19:15:32.

Table 3.2 depicts the audio and lighting cues provided to sensing unit type#2 when the system identifies visitors and detects the resident's proximity to specific domestic objects. The system correctly administered audio and lighting stimula-

Table 3.1: Visitor classification based on images taken by sensing unit type#1

Person Arrived	Arrival Timestamp	Classifier Output
Person 4	2018-12-03 10:05:03	Person 4
Person 2	2018-12-03 10:20:15	Person 2
Person 4	2018-12-03 11:03:16	Person 4
Person 3	2018-12-03 11:30:21	Person 3
Person 5	2018-12-03 12:30:45	Person 5
Person 2	2018-12-03 15:02:32	Person 2
Person 4	2018-12-03 15:16:45	Person 4
Person 1	2018-12-03 19:15:32	Person 4
Person 3	2018-12-03 19:25:22	Person 3

tion for four individuals (2, 3, 4, 5). However, due to a wrong classification by the fog node, it failed to provide the correct stimulation for person 1, inadvertently triggering stimulation meant for person 4 when person 1 visited.

Additionally, during person 3's second visit, though accurately recognized by the fog gateway, the PIR motion sensors incorrectly identified the resident's location as the living room. Consequently, audio and lighting stimuli intended for person 3 was mistakenly delivered in the living room instead of bedroom. Nevertheless, the system effectively triggered audio and lighting stimuli for the sofa set, and bed when the dementia patient approached these objects.

Figure 3.5 displays an example of the configuration panel within openHAB, offering insights into various elements within our experimental arrangement. We can observe the changes in lighting stimuli for different visitors and objects in Figure 3.6. These outcomes affirm the proper functioning of the communication system among various components and demonstrate the utility of the proposed system in delivering appropriate audio and lighting stimulation based on the context.

In summary, fog computing expands on the principles of cloud computing, shifting data processing closer to the network edge. This makes it particularly well-suited for IoT applications that demand rapid response times. This research work (outlined in Section 3.1.1) introduced a fog computing framework for an audio and lighting stimulation program tailored to the dementia people in a smart home setting. The visitor classification model, trained and deployed on fog node,

Table 3.2: Audio and lighting stimuli for different visitors and objects

Exact location of the resident	Location identified by PIR sensor	Person/ House hold Object	Hue Light Parameters			Audio file played	Triggering action correctness
			C	S	B		
Kitchen	Kitchen	Person 4	143	29	40	File 4	Yes
Kitchen	Kitchen	Person 2	35	40	60	File 2	Yes
Living room	Living room	Sofa Set	55	62	59	File 6	Yes
Living room	Living room	Person 4	143	29	40	File 4	Yes
Kitchen	Kitchen	Person 3	45	59	72	File 3	Yes
Living room	Living room	Sofa Set	55	62	59	File 6	Yes
Living room	Living room	Person 5	219	61	59	File 5	Yes
Living room	Living room	Person 2	35	40	60	File 2	Yes
Living room	Living room	Person 4	143	29	40	File 4	Yes
Bed room	Bed room	Bed	95	25	35	File 7	Yes
Bed room	Bed room	Person 1	143	29	40	File 4	No
Bed room	Living room	Person 3	45	59	72	File 3	No

(C: Color, S: Saturation, B: Brightness)

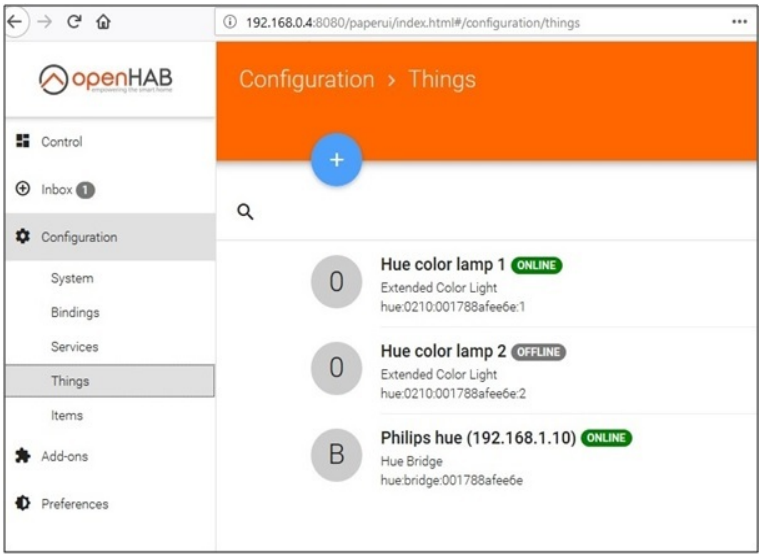


Figure 3.5: openHAB configuration panel instance

Figure 3.6: Visual stimuli given to dementia person for person 4, person 5, and for sofa set

classifies the visitors and enables triggering of appropriate stimuli to assist the dementia person.

3.1.2 Vistor Identification using Eigenfaces in Fog Computing

The research work on audio and lighting stimulated-based systems to assist dementia people, using Eigenfaces for visitor identification [60], is presented in this section. This work is similar to the research work presented in Section 3.1.1 in terms of providing audio and lighting stimulation, but differs in the way visitor classification is done.

3.1.2.1 Eigenfaces method for face recognition

Eigenfaces method is based on principal component analysis (PCA) [61]. An objective of PCA is to replace correlated vectors of large dimensions with uncorrelated vectors of smaller dimensions. Eigenfaces method consists of extracting the characteristic features of the face and representing it as a linear combination of eigenfaces obtained from the feature extraction process.

Let there be N face images in the training set and let the dimension of each image be $m \times n \times 3$ (the third dimension is for RGB channels).

- Convert each image in the training set into vector of length $(m \times n \times 3)$ elements i.e. $(m \times n \times 3) \times 1$ dimension. A training set of $(m \times n \times 3) \times N$ dimensions is created.

- Find the mean vector ($(m \times n \times 3) \times 1$ dimension) of all training image vectors and subtract it from all training image vectors. Let the matrix be A (whose dimension is $(m \times n \times 3) \times N$). The covariance matrix C is obtained by multiplying A and its transpose.
- Perform PCA to obtain principal components of the dataset calculated from the eigen vectors of the covariance matrix. Eigenfaces are obtained by reshaping these eigen vectors into images of dimension $m \times n \times 3$.
- The images in the training set are projected into eigenface space to represent the image in smaller subspace. The weight vector W_i for the image i in the training set (representation of image in smaller subspace) is obtained from k number of eigen vectors as follows (equation 3.1:

$$W_i = [W_{i1}, W_{i2}, ..., W_{ik}] \tag{3.1}$$

Where,
W_{ij} = ((image vector i)-(mean vector)).(eigen vector j), for $j = 1 \ldots k$

3.1.2.2 Basic Operation of the Proposed System

Initially, during the enrollment, principal component analysis is performed on the training images of various persons to obtain eigen vectors. The weight vectors for training images can be obtained using these eigen vectors. These weight vectors are used by the fog node for classification of visitor face image. The basic operation of the proposed system is shown in Figure 3.7. The camera module, attached to the edge sensor node at the entrance of the house, captures the visitor image whenever any motion is detected. The frontal face haar cascade classifier is used to detect the face portion of the captured visitor image [62] [63]. Eigen vectors are used to calculate the weight vector for the image of the face portion extracted from the visitor. The edge node sends this weight vector to the fog node for classifying the visitor.

The fog node classifies the visitor by calculating the distance between the visitor's face image weight vector and training images weight vectors. The visitor is classified as the person whose training image weight vector gives the least distance with the visitor image weight vector, and this difference is less than some threshold value (θ). If this lowest difference is greater than θ, then the

visitor is classified as an unknown person. There is no standard formula for setting this threshold value. One approach is to find the minimum distance of each image with all images in the training set and store that minimum distance in a vector V [64]. Then, threshold can be set using the following formula (given in equation 3.2):

$$Threshold(\theta) = 0.8 * max(V) \tag{3.2}$$

3.1.2.3 Implementation Details

The deployment of various sensor nodes in our experimental setup is shown in Figure 3.2(a). The sensor node at the entrance of the house comprises of raspberry pi integrated with pi camera and passive infrared (PIR) motion sensor. The arrival of the visitor is identified by detecting motion using PIR motion sensor, and photograph of the visitor is captured using pi camera. The facial portion of the visitor is extracted from the captured visitor image using *frontal face haar cascade classifier*. The weight vector for the visitor face image is computed on edge gateway node, as explained in Section 3.1.2.1. This weight vector is sent to the fog gateway node, which is a raspberry pi unit in our experiment, for visitor classification. The dotted line in the Figure 3.7 indicates the separation between edge node and fog node processing. The sensor nodes (labeled 2 in Figure 3.2 (a)) placed in the kitchen, living room, and bedroom are integrated with the PIR sensor, and each room consisted of Philips hue light and a smart speaker to trigger audio and lighting stimulation.

We considered 200 training facial images of 5 different people (40 images for each person with different lighting conditions and facial expressions). Figure 3.8(a) shows a sample set of face images for five different people from the training set. Figure 3.8 (b) shows a sample set of five facial images of a single person with different facial expressions. We have considered 100x100 color images in our training set.

Experimental Results: Each image in the training set is converted into a vector of length of 30,000 elements (that is, 100x100x3). In our experimentation, a training set of 30000x200 dimensions is created. Figure 3.9 shows a vector representation of five test images from the training set of 200 images. Here, each row represents one image (vector with 30000 elements). The program represents the pixel values in the range [0-1] by diving the original value by 255. The five

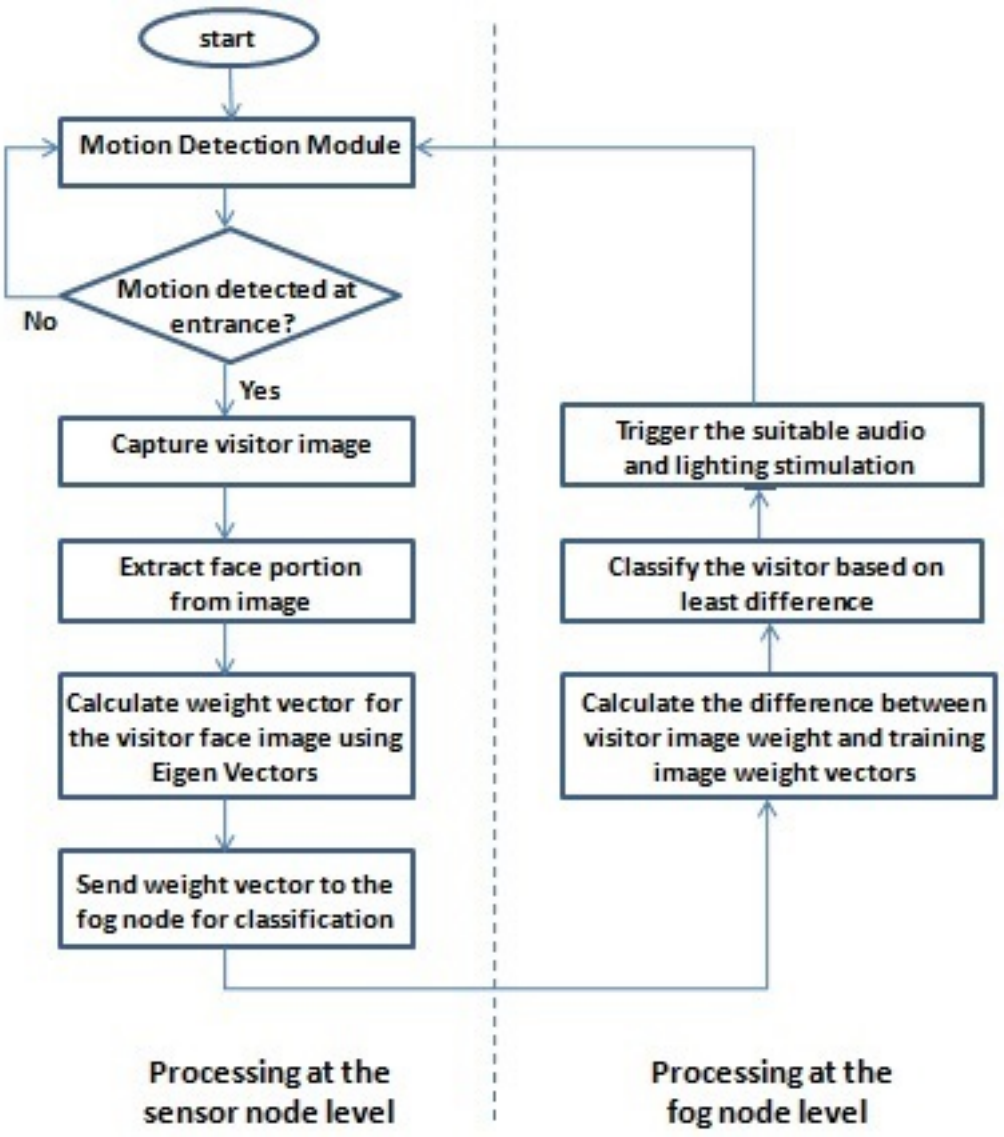


Figure 3.7: Flowchart indicating the basic operation of the proposed system

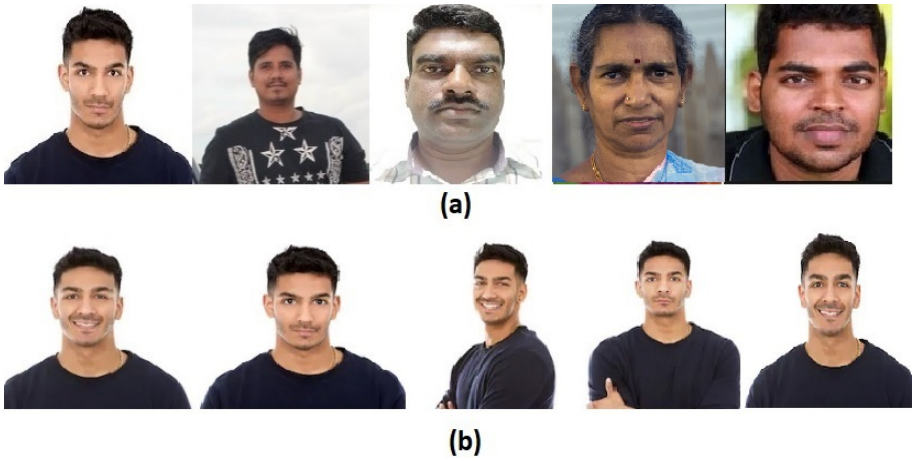


Figure 3.8: (a) Sample set of face images for five different persons (b) Sample set of face images of a single person with different facial expressions

rows indicate the image vectors for five test images. PCACompute() function in OpenCV is used to obtain the mean vector and eigen vectors from the training data set, and each of these vectors would have 30000 elements. We have chosen generation of 8 eigen vectors in our experimentation. The mean or average face is obtained from the mean vector by transforming it back into a 100x100x3 image. Similarly, the eigen faces can be obtained from eigen vectors by reshaping them into 100x100x3 images. The average face obtained from the test image data set is shown in Figure 3.10(a). The weight vectors are generated by the program, as explained in Section 3.1.2.1. for all the face images in the training set. Figure 3.11 shows the weight vectors generated for five sample images in the training set. The training of the eigenfaces-based face recognition model took 56 seconds of time, producing 82% accuracy.

Figure 3.10(b) shows the image captured by the edge node, when one of the five visitors arrives home. The edge node extracts the facial portion using the Haar cascade classifier, as shown in Figure 3.10(c). The weight vector (W) for the visitor face image is computed on the edge node using the eigen faces and average face of training images and sent to the fog node for classification. The Euclidean distance between W and each of the weight vectors for the training images (shown in Figure 3.8(a)) is given in Figure 3.12(b). The distance between the weight vector of the visitor face image (W) and the weight vector of the test image 3 is the least among all test images, and hence the system classifies the visitor as Person 3 and gives the audio and lighting stimulation corresponding to

```
[[1.  1.  0.  ...0.7843137 0.8039215 0.1960784]
 [1.  1.  1.  ...0.7058823 0.7254901 0.0980392]
 [1.  1.  0.  ...1.        1.        0.       ]
 [1.  1.  0.  ...0.7450980 0.7843137 0.2823529]
 [1.  1.  1.  ...0.7254901 0.6274509 0.1764705]]
```

Figure 3.9: Vector representation of five test images

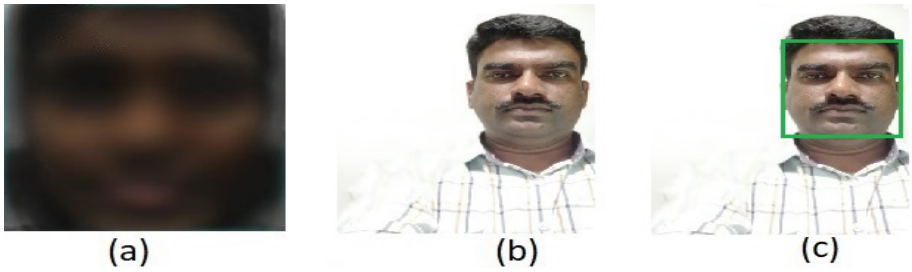


Figure 3.10: (a) Average face (b) Test image for classification (c) Detection of face in the test image

Person 3.

In summary, this research work presented an Eigenfaces-based visitor identification system to assist the dementia person in a smart-home environment. LBPH method for face recognition is relatively less computationally intensive when compared to the eigenfaces method and yields better performance. LBPH algorithm is robust to lighting conditions and expressions, compared to Eigenfaces, and is suitable for real-world scenarios with simple computation.

Test Image	Weight Vector
1	[0.8532147, -10.1284237, 13.5234421, 1.5663249, -8.3492324, 1.5066235, 9.5644321, -6.2334325]
2	[-0.7430422, 8.4322561, -5.2331054, 4.3364231, 10.3423305, -1.3349856, 5.3412246, -1.5323421]
3	[6.3530422, 10.2320454, -4.4231094, 1.3411045, 9.2426307, 1.5336182, -4.4322871, -2.4312564]
4	[4.2364122, 11.2654725, -3.5612453, -9.3534874, 0.8923155, 12.9722564, -10.2355871, -3.4232274]
5	[0.7425405, 12.4523449, 8.3423125, -7.2378105, 10.3453343, 2.1204342, 5.2355671, -4.5664523]

Figure 3.11: Weight Vectors

[7.1239425, 11.4323225, -2.3402565, 4.2335212, 9.4232374, 4.3439231, -4.2342524, -6.2342651]

(a)

Test Image	Distance
1	214.222687
2	95.431046
3	22.075933
4	70.925163
5	106.042313

(b)

Figure 3.12: (a) Weight vector (W) for the visitor face image (b) Euclidean distance between W and each of the weight vectors for training images

3.1.3 Face Mask Detection at the Fog Computing Gateway

This work proposes a fog computing-based face mask detection system to control the entry of a person into a facility. The proposed system uses fog nodes to process the video streams captured at various entrances into a facility. Haar-cascade classifiers are used to detect face portions in the video frames. Each fog node deploys two MobileNet models, where the first model deals with the dichotomy between mask and no-mask case. The second model deals with the dichotomy between proper mask wear and improper mask wear and is applied only if the first model detects mask in the facial image. This two-level classification allows people to enter a facility, only if they wear the mask properly. The proposed system offers performance benefits, such as improved response time and bandwidth consumption, as the processing of video stream is done locally at each fog gateway without relying on the Internet.

3.1.3.1 System Description

The proposed system employs an RPi fog node integrated with the Raspberry Pi camera and relay sensor at each entrance where entry control is required. The fog nodes are connected to the same Wi-Fi network. The basic architecture of the proposed system is shown in Figure 3.13. The frames in the video stream captured by Pi Camera are processed by the fog node. Whenever any face(s) is detected in

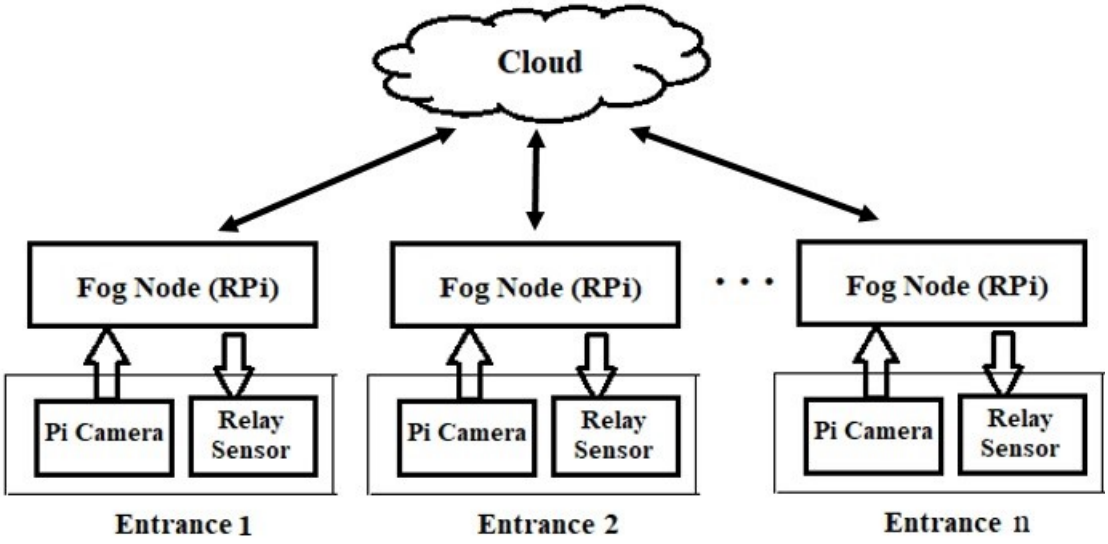


Figure 3.13: The basic architecture of the proposed system

the frame, the face mask detection model tries to identify whether the person(s) is wearing the mask or not. The RPi module sends a control signal to the relay to open the door, if the person wears the mask properly. The decision to open the door or not is taken completely on the fog node, and the event information can be sent to the Cloud optionally for further storage and processing. The proposed system uses the *frontal face haar cascade classifier* from OpenCV to detect faces in video frames. MobileNetV2 model is used in our experiment using Keras API, as it is a lightweight convolutional NN that reduces the inference cost on mobile and embedded devices [65].

3.1.3.2 Basic Operation of the Proposed System

The proposed system employs two MobileNetV2 models to classify whether a person is wearing the mask properly or not. The first model is a binary classifier that is trained using two classes of images – **mask** and **no mask**. The **mask** class contains facial images of people wearing a mask (including proper and improper mask wear). The second model is a binary classifier that is trained using images of **proper mask wear** and **improper mask wear** images. The mask is said to be properly put on if the nostrils and mouth are covered. If a nostril or mouth is detected even when the person wears a mask, the instance is classified as improper mask wear, and entry should be restricted. In our experimental setup, these two

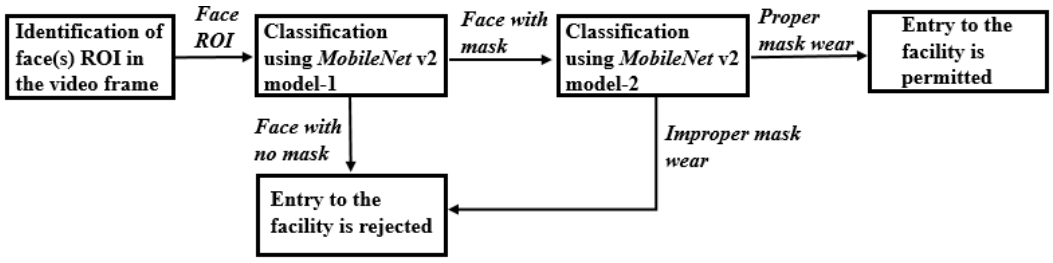


Figure 3.14: The basic operation of the proposed system

models are trained on a single RPi fog node and deployed in all fog gateways for inference purposes. The rationale behind choosing two binary classifiers instead of a single three-class classifier is to improve classification accuracy, especially between proper and improper mask wear classes. The choice provides a trade-off between classification accuracy and throughput.
Each fog node processes the video frames captured by Pi Camera and uses Frontal Face Haar Cascade classifier to detect the faces in those frames. When one or more faces are detected in a frame, level one binary classifier (mask vs. no mask) is applied on each face region of interest (ROI) in the video frame. If the person wears the mask, then level two classifier (proper vs. improper mask wear) is applied to identify whether the person is wearing the mask properly or not. This two-level classification restricts the entry of people with improper mask wear into the facility. The basic operation of the proposed system is shown in Figure 3.14.

3.1.3.3 Implementation Details

The system provides a proof-of-concept for face mask detection using fog computing gateway. The processing of the video frames is done at the source of the video capture. In reality, the frames in the video stream could be sent and processed using the resources in the fog gateway.

- **Face Mask Detection Model Training:**
  Datasets used for Model Training: As discussed in Section 3.1.3.1, the proposed system uses two binary classifiers based on the MobileNetV2 model. Classifier-1 (model-1) is trained using dataset-1 with a total of 770 facial images divided into two classes: with mask and without mask. The **with mask** class included images of faces with and without proper face mask wear. Classifier-2 is built using dataset-2 with a total of 500 facial images divided into two classes: proper mask wear and improper mask

wear. The average size of training images is around 5KB (image dimensions 250x160 with 96 dpi). Figure 3.15 and Figure 3.16 show a sample set of facial images from dataset-1 and dataset-2 respectively. Although there are few face mask datasets available online [66], we have prepared our own dataset as existing face mask datasets lack images for improper mask wear class.

Training the mask detection models: The various steps followed to train mask classification models are given below:

Step 1. Load images from the data set using the load_img() function from the Keras API. The loaded images are resized to a 224x224 format.

Step 2. The loaded images are normalized using the preprocess_input() function. The data (facial image) and label lists are updated with images in the dataset.

Step 3. Convert the data and labels to numpy arrays. One-hot encoding is performed on the labels to represent them as binary vectors.

Step 4. The data set is partitioned into training (80%) and testing (20%) sets using train_test_split() function.

Step 5. Instantiate MobileNetV2 model trained with ImageNet dataset. Load the model that does not include the classification layers at the top. Transfer learning is used in our experimental setup to transfer knowledge from the ImageNet dataset domain to our facial dataset domain [67].

Step 6. The weights of all the layers in the convolutional base are frozen to prevent updates during training. We added the classifier to this base model and trained the top-level classifier.

Step 7. The model is compiled using the Adam optimizer and the binary cross-entropy loss function.

Step 8. The model is trained and serialized to disk for use during the inference process.

- **Classification of Facial Images:**
  RPi 4 device integrated with Pi Camera is used as a fog node at each entrance to capture the video stream. The Pi Camera has 5MP resolution and can record 1080p videos at 30 frames per second. The face mask detection models trained in the previous step are loaded into each fog node for inference. The following are the various steps performed during the inference process on each video frame:

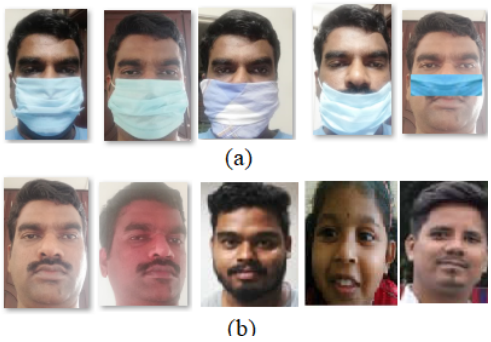


Figure 3.15: Sample set of facial images used for Model-1 training (a) With mask (b) Without the mask

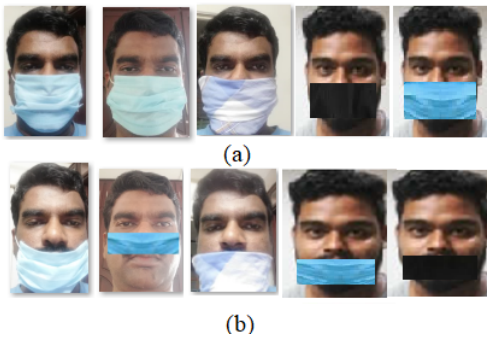


Figure 3.16: Sample set of facial images used for Model-2 training (a) With proper mask wear (b) Without proper mask wear

Step 1. Identification of face ROI in the frame using the OpenCV frontal face haar cascade classifier.
Step 2. If more than one face is detected in the frame, for each face do the following:

- Resize the face image to 224x224 RGB image.
- Convert the image to a numpy array and preprocess it.
- Predict the output class (mask vs. no mask) of the image using the MobileNetV2 model-1 loaded on the node.
- If the prediction class on the image is a mask, then model-2 is used for further classification.
- If the prediction class in model-2 is 'proper mask wear', then the fog node controls the relay to open the door. Otherwise, the entry is restricted.

Figure 3.17 shows the screenshots of the outcome of the face mask detection model for the proper and improper mask wear cases.

3.1.3.4 Results and Discussion

The face mask detection models (model-1 and model-2) are trained using different learning rates (LR) 0.001 and 0.0001 with two different numbers of epochs 10

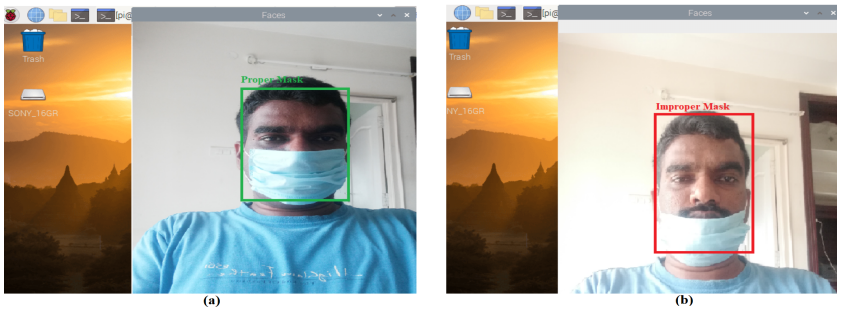


Figure 3.17: Face mask detection model prediction when (a) the person wears the mask properly (b) the person wears the mask improperly

and 20. The model-1 and model-2 training tasks have taken 34 and 20 minutes, respectively (with LR = 0.001 and #Epochs = 20) on RPi 4. The face detection and inference tasks for the given video frame have taken 0.3 seconds and 3.4 seconds, respectively, on average. Table 3.3 and Table 3.4 present the accuracy and loss values during the training and validation phases of model-1 and model-2, respectively (with LR = 0.001 and #Epochs = 20).

From the results in Table 3.3 and Table 3.4, we can observe that training, validation accuracy and loss values are improving during the models' training. After a few epochs (epoch #15 for model-1 training and epoch #10 for model-2 training approximately), we get fluctuations in the accuracy and loss values. The variations in the accuracy and loss values are due to model overfitting with more number of epochs. Overfitting of the model could occur with the selection of small data sets to train the model. As the RPi device is resource-constrained, we have chosen small datasets of facial images for model training. One solution to address this problem is early stopping using a callback mechanism.

While training model-1 on the Raspberry Pi node using dataset with 770 images, a warning message is received with respect to the allocation of memory exceeding 10% of the system memory. If we want to train the model with a large dataset to avoid model overfitting, we can train it on a high-end machine and deploy the model on the fog node for inference. Alternatively, we can distribute the model training to several nodes in the fog cluster. Figure 3.18 presents performance metrics related to model-1 and model-2 training.

Table 3.3: Accuracy and Loss Values during Model-1 Training

Epoch	Train_Loss	Train_Acc	Val_Loss	Val_Acc
1	0.64	0.69	0.49	0.71
2	0.39	0.82	0.39	0.82
3	0.36	0.88	0.48	0.74
4	0.35	0.88	0.47	0.74
5	0.32	0.89	0.45	0.75
6	0.24	0.9	0.47	0.74
7	0.22	0.91	0.46	0.78
8	0.21	0.92	0.44	0.79
9	0.21	0.91	0.43	0.79
10	0.19	0.93	0.44	0.78
11	0.19	0.92	0.41	0.8
12	0.18	0.93	0.34	0.82
13	0.17	0.94	0.4	0.78
14	0.17	0.93	0.35	0.83
15	0.17	0.94	0.42	0.81
16	0.18	0.93	0.47	0.77
17	0.17	0.93	0.56	0.75
18	0.16	0.94	0.57	0.74
19	0.17	0.93	0.54	0.76
20	0.16	0.94	0.55	0.75

	precision	recall	f1-score
with_mask	0.87	0.99	0.92
without_mask	0.98	0.73	0.83
accuracy			0.90

	precision	recall	f1-score
improper_mask_wear	0.92	0.82	0.87
proper_mask_wear	0.89	0.95	0.92
accuracy			0.90

Figure 3.18: Performance metrics related to Model-1 and Model-2 training

Table 3.4: Accuracy and Loss Values during Model-2 Training

Epoch	Train_Loss	Train_Acc	Val_Loss	Val_Acc
1	0.71	0.66	0.44	0.77
2	0.5	0.72	0.34	0.79
3	0.42	0.79	0.32	0.8
4	0.39	0.8	0.28	0.85
5	0.35	0.88	0.29	0.84
6	0.32	0.89	0.29	0.9
7	0.22	0.91	0.27	0.88
8	0.22	0.9	0.26	0.9
9	0.2	0.92	0.24	0.91
10	0.29	0.85	0.25	0.9
11	0.28	0.89	0.28	0.88
12	0.25	0.9	0.29	0.88
13	0.24	0.92	0.3	0.85
14	0.25	0.9	0.32	0.84
15	0.22	0.92	0.35	0.82
16	0.21	0.92	0.38	0.78
17	0.22	0.93	0.39	0.76
18	0.21	0.91	0.38	0.77
19	0.19	0.92	0.37	0.8
20	0.19	0.92	0.38	0.78

In summary, the research work in this Section 3.1.3 presented a proof-of-concept fog computing-based face mask detection system for automatic entry and access control into a facility. The fog gateway processes the video stream captured at the entrance to recognize whether a person is wearing a mask or not. Two MobileNet models are deployed in each fog node located at each entrance to the facility. The first model deals with the dichotomy between mask and no mask case. The fog node uses the second model to detect whether the person wears the mask properly or not, in case the first model detects the person wearing the mask. The proposed system allows entry into the facility, only if the person wears the mask properly. The results of the classification are encouraging with a model accuracy value around 90%.

As the processing of video stream is done locally at each fog node, the proposed system offers performance benefits such as improved response time and bandwidth consumption. The proposed system could be integrated with Cloud, optionally, for further storage and processing of video stream. In the future, the system could be extended to distribute model training among several fog nodes in the network.

3.1.4 Face Recognition in the Fog Cluster Computing

This work proposed a cluster computing-based facial recognition system in a fog computing environment under the Internet of Things theme. The proposed system uses Haar Cascades to detect faces in a test image and uses the Local Binary Pattern Histogram algorithm to recognize them in parallel by making use of the computing power of several Raspberry Pi nodes in the cluster. The cluster uses Simple Linux Utility for Resource Management scheduler to schedule various jobs among the nodes and the Message Passing Interface to facilitate communication among various processes running in the cluster. The system uses the aggregated computation of the cluster. The facial recognition tasks were executed in parallel to obtain better performance in terms of execution time. The proposed system can be extended to detect and recognize multiple objects in an image, which has several benefits in other applications.

3.1.4.1 System Description

The proposed system contains a cluster of four Raspberry Pi nodes that are connected to the same Wi-Fi network, as shown in Figure 3.19. Out of four nodes,

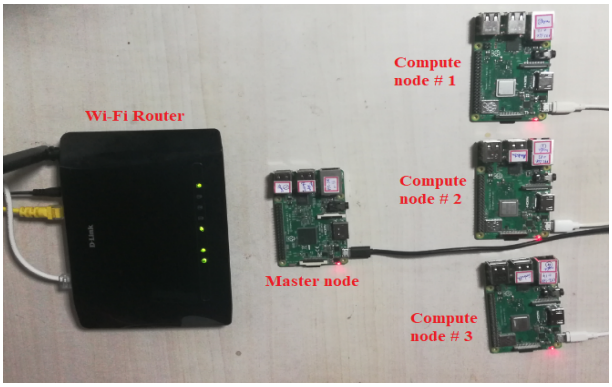


Figure 3.19: Experimental setup of Raspberry Pi cluster

one node acts as a master node and the other three nodes act as worker/compute nodes. Various software used in the system design are listed below:

- **Simple Linux Utility for Resource Management (SLURM) scheduler:** SLURM is a free and open-source cluster management and job scheduling system for Linux clusters [68]. SLURM performs various functions like allocating access to resources to users, provides a framework for monitoring the work, and arbitrates contention for resources by the pending jobs. The proposed system uses the SLURM scheduler to manage the jobs in the cluster. It runs as a daemon process on the master node that executes the parallel job on the set of nodes in the cluster.

- **OpenMPI:** Message Passing Interface (MPI) facilitates communication among various processes running in the cluster [69]. Several MPI APIs like OpenMPI, MPICH, Intel MPI etc. are available in market. OpenMPI, MPICH, are freely available, and Intel MPI comes with a license. The proposed system uses OpenMPI functions MPI_Send() and MPI_Recv() for communication among processes.

- **OpenCV:** The proposed system uses the frontal face haar cascade classifier and LBPH algorithm (from the OpenCV library) to detect and recognize faces in the captured image.

Basic Operation of the Proposed System: Initially the face recognition model is trained using training data of multiple facial images of several persons. This trained model is used by the nodes in the cluster to classify the person. When a test image that contains multiple human faces is given to the master

node for classification, the node detects the coordinates of the faces and sends these coordinates to different nodes in the cluster for parallel classification. Each worker node, upon receiving the facial coordinates, classifies that face in the test image using the pre-trained model shared with all the nodes in the cluster. In this way, the performance of the multifacial classification task can be improved using the computing power of several nodes in the cluster compared to the classification using a single node.

Algorithm 3.1 Multi_Facial_Classification_Serial(testImage)

Result: Class Labels
Load the LBPH Classifier model
Find the facial coordinates in the test image using *Haar* Classifier
for each *face* in the *list of faces* **do**
  Pass the *face* coordinates to the Classifier model
  Display the label
end for

Algorithm 3.1 and Algorithm 3.2 shows the high-level logic of the multi-facial classification task using serial and parallel versions. In Algorithm 3.2, when 'n' number of processes are spawned, the rank of those processes will be 0,1,. . . ,(n-1). These processes will be scheduled at different nodes in the cluster by the SLURM scheduler.

3.1.4.2 Implementation Details

Setting up the Raspberry Pi Cluster:
Raspberry Pi cluster is set up with four nodes (Raspberry Pi 3 B+ model) in our experimentation [70] [71]. The nodes in the cluster are given hostnames (node1, node2, node3, and node4) by editing the information in the **/etc/hosts** file, as SLURM scheduler uses these names. A shared folder is created on the master node by using Network File System (NFS) and mounted onto the worker nodes, as the nodes in the cluster need access to the same files. SLURM controller packages are installed on the master node and the controlled information is edited in the **slurm.conf** file to add the worker/compute nodes to the cluster. The compute nodes in the cluster are configured by installing the SLURM client. OpenMPI is installed on the nodes in the cluster to facilitate communication among various processes running on them. The Python package **mpi4py** is

Algorithm 3.2 Multi_Facial_Classification_Parallel(testImage)

```
Result: Class Labels
Load the LBPH Classifier model
Find the rank of the process
if myRank = 0 then
    Find the facial coordinates in the test image using Haar Classifier
    i = 1
    for each face in the list of faces do
        data = coordinates of the face
        MPI_Send(data, node=i)
        i = i+1
    end for
else
    MPI_Recv(data, node=0)
    Classify the face using classifier model
    Display the label
end if
```

installed to interface with OpenMPI in Python.

Training the Face Recognition model:
As discussed in Section 3.1.4.1, LBPH algorithm is used by the master node to train the face recognition model using the images of ten different persons (class labels [0-9]). Each class contains around fifty images of one person. The training method takes the list of extracted face portions (in gray scale) of training images and their corresponding label, and returns the trained model in the YAML file format. This model is saved by the master node in the shared folder, so that all worker nodes can access it for subsequent classification of test images. Figure 3.20(a) shows a sample set of face images for five different persons from the training set of 10 different persons. The class labels for these images are [0-4], respectively. Figure 3.20 (b) shows a sample set of five face images of a single person with different facial expressions.

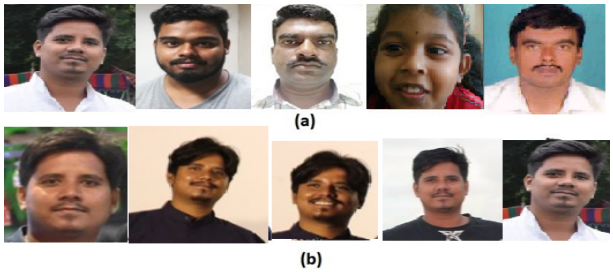


Figure 3.20: (a) Sample set of face images for five different persons (b) Sample set of face images of a single person with different facial expressions

Figure 3.21: Test image of two different persons with class label two and one

3.1.4.3 Results and Discussion

As discussed in Section 3.1.4.2, the face recognition model is trained using 485 images of 10 different persons. The average size of training images is around 5KB (image dimensions 250x160 with 96 dpi). The training process has taken around 150 seconds to build the model. After the model is built, it is used by all nodes in the cluster for facial classification.

The developed system is tested by giving multi-facial images as input. The proposed system performs better when compared to classifying the same multi-facial image using serial execution. Figure 3.21 shows one of the input test images given to the system (persons with class labels two and one). The output screen-shots of the classification program using serial version (Algorithm 3.1) and parallel version (Algorithm 3.2 using MPI) are shown in Figure 3.22 and Figure 3.23 respectively. As can be seen in Figure 3.22 and Figure 3.23, the serial version of the classification program (on a single node) to classify the test image with two persons has taken 13 seconds and the parallel version (the proposed system using MPI on the cluster of four nodes) to classify the same has taken 9 seconds. The proposed system can gain performance benefits compared to the serial version when the test image contains more faces for classification.

The processing times of several tasks like training, face detection are shown

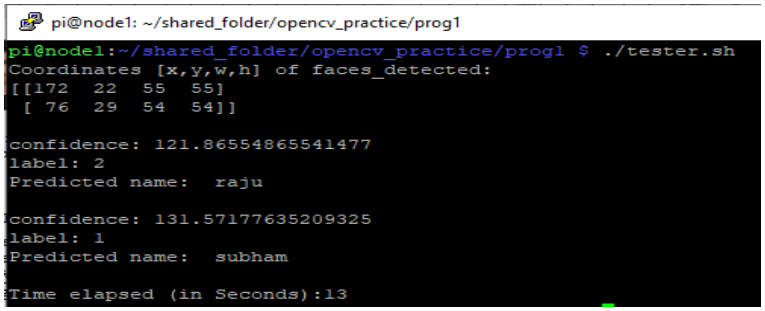


Figure 3.22: Output screenshot of serial version of classification program

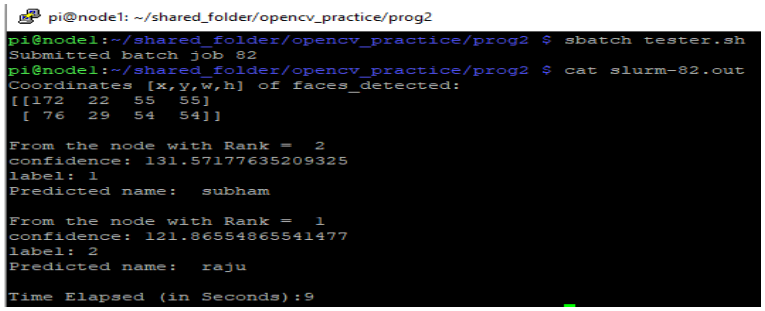


Figure 3.23: Output screenshot of parallel version of classification program

in Table 3.5. The face recognition model training process with 485 images took 144 seconds with a Raspberry Pi. The time taken for face detection is around 6 seconds. The variation in the processing time of face detection is not high with different number of faces in the test image. The time taken for the classification of various test images is shown in Table 3.6. Algorithm 3.2 gains performance benefits compared to Algorithm 3.1, when the test image contains multiple faces. For the test image with only one face, Algorithm 3.2 may not perform well compared to Algorithm 3.1, due to involvement of MPI message exchanges between processes with ranks 0 and 1.

Table 3.5: Processing Times of Training and Face Detection Tasks

Task	#images	Processing time (in Seconds)
Training	485	144
Face detection	1	6

Table 3.6: Performance Comparison of Algorithms - Serial and Parallel Version

Test image with number of faces	Time taken for execution (in seconds)	
	Algorithm (Serial Version)	Algorithm (Parallel Version)
1	8	9
2	13	9
3	16	10

3.2 Distributed Data Processing using Cluster Management

This section presents research work related to the distributed storage and processing of data sets in the fog computing cluster environment. The fog environmental setup consists of resource-constrained devices with minimal computational and storage capabilities. A fog computing framework augmented with cluster management can take the limitations of resource-constrained devices for effective big data processing. The implementation details of the fog computing framework using Apache Spark for Big Data applications in a resource-constrained environment are given. The results related to big data processing, modeling, and prediction in a resource-constraint fog computing framework are presented by considering the evaluation of case studies using the e-Commerce customer dataset and Bank loan credit risk datasets. Two case studies portraying how the proposed thought works are described for big data applications using a fog cluster.

3.2.1 Distributed Data Processing

Figure 3.24 depicts the taxonomy of distributed data processing considered in this subsection. Cluster computing supports parallel processing of a large task by making use of several computational nodes in the cluster [72] [73]. Utility computing provides computing resources to the customer based on demand and uses the pay-per-use billing method [74]. Peer-to-Peer (P2P) network consists of nodes where the communication and data sharing is carried on directly between nodes, rather than being arbitrated by an intermediary node. Each node in the

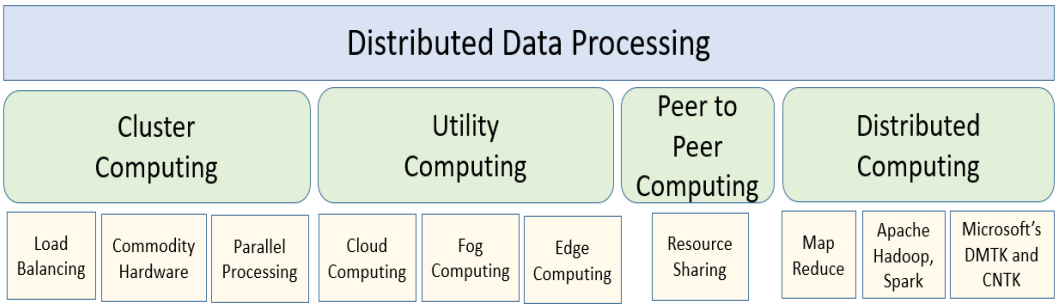


Figure 3.24: Taxonomy of Distributed Data Processing

Peer-to-Peer network acts as both a client and a server. The computing power of the P2P network helps solve complex problems that require powerful computers using commodity hardware [75]. P2P computing uses a distributed application architecture that divides the workload among peers in the network. Several distributed computing frameworks such as Apache Hadoop, Apache Spark, Microsoft's DMTK, and CNTK are available for processing Big Data. The research work in this section 3.2 aims at:

- Proposing fog cluster environment that has distributed storage and processing capabilities of IoT data using commodity hardware nodes in the cluster.
- Augmenting fog computing with cluster management to address the effective big data processing on resource-constrained devices.

3.2.2 System Description

The proposed system contains a cluster of three Raspberry Pi fog nodes that are connected using a router, as shown in Figure 3.25. Out of these three nodes, one node acts as the master node and the other two nodes act as workers. Apache Hadoop is installed on all nodes in the cluster. Hadoop is composed of the Hadoop Distributed File System (HDFS) that handles data scalability and redundancy across nodes [76] and Hadoop YARN [77], a framework for job scheduling that executes data processing tasks on all nodes. Spark has been installed on top of the Hadoop Yarn cluster for scheduling the spark jobs submitted to the cluster. The generic architecture of the proposed system is shown in Figure 3.26. At the lower level of the hierarchy, IoT devices generate large amounts of data. Traditionally

Figure 3.25: Experimental Setup of Fog Cluster with One Master Node and Two Worker Nodes

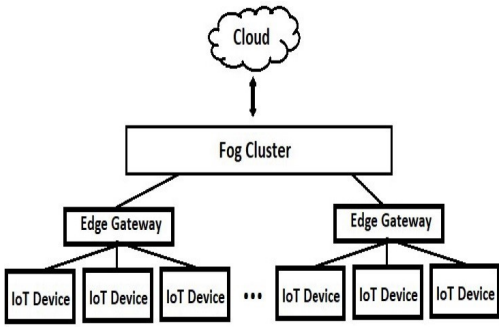


Figure 3.26: Generic Architecture of Proposed System

Big Data Analytics is performed in the Cloud, which has few disadvantages such as more bandwidth consumption, delayed response, and data security concerns. Fog Computing helps in addressing the problems related to Cloud Big Data Processing. As the resource capabilities (processing, memory, etc.) of fog nodes are limited in nature, our research work proposed a Fog Cluster architecture for processing Big Data in IoT setup. In the proposed architecture, the gateway devices aggregate data from several IoT devices and send them to the fog cluster for processing. These data could be processed using the distributed processing and storage capabilities of the nodes in the fog cluster. The cluster manager distributes the load among several nodes in the cluster for effective resource utilization and better response times. The summary information from the fog cluster could optionally be sent to the cloud for further storage and analysis. The distributed storage and processing capabilities of fog cluster lend themselves well to handle voluminous data generated in the IoT environment and offer various benefits such as improved response time, scalability, and data security.

3.2.3 Implementation Details

3.2.3.1 Using Resource Constraint Device (Raspberry Pi)

Initially, the Raspberry Pi cluster is setup with three nodes (Raspberry Pi 4B with 4GB RAM) in our experimentation [70] [71]. The nodes in the cluster are

given hostnames (`node1, node2`, and `node3`) by editing the information in the `/etc/hosts` file. This facilitates communication among nodes by using names. As the nodes use Secure Shell (SSH) connection with key-pair authentication to connect with other nodes, each node generates keys (private key, public key) and the public keys of all nodes are shared among the nodes. Care should be taken about the suitable Java version for the successful installation of Spark in the cluster. Hadoop binaries (version 3.2.0) are downloaded from the Hadoop project page [78] and environment variables such as PATH and HADOOP_HOME are set properly in all nodes in the cluster.

The configuration files `core-site.xml, hdfs-site.xml, mapred-site.xml` and `yarn-site.xml` are edited appropriately to properly configure the Hadoop Distributed File System (HDFS) [79]. The values set for important Hadoop configuration parameters are given in Table 3.7. The parameter `dfs.replication` specifies the replication factor of the data stored in HDFS. The remaining rows in Table 3.7 specify memory allocation (in MB) for YARN containers, mapper tasks, and reducer tasks. The `yarn.app.mapreduce.am.resource.mb` parameter specifies memory allocation to the map-reduce application manager. The `mapreduce.reduce.memory.mb` specifies the memory limits for the map and reduce processes, respectively. The resource manager allocates memory to containers in increments of the parameter value `yarn.scheduler.minimum-allocation-mb` and will not exceed `yarn.scheduler.maximum-allocation-mb` parameter value.

In our experimental setup, `node1` is the master node (runs the daemons HDFS NameNode and YARN ResourceManager) and `node2, node3` are the worker nodes (runs the daemons HDFS DataNode and YARN NodeManager). After the Hadoop cluster is established successfully, Spark has been installed on top of Hadoop. Spark binaries are downloaded to the master node from the Apache Spark download page [80] and the value of the SPARK_HOME environment variable is set properly. Spark driver program declares transformations and actions on RDD and these requests are submitted to the master. The worker nodes execute these tasks (called executors). The information about these Spark memory settings is specified (in MB) in the `spark-defaults.conf` file. The parameter `spark.executor.cores` specifies the number of cores used by the driver program.

Table 3.7: Hadoop Configuration Parameters

Configuration File	Parameter Name	Value
hdfs-site.xml	dfs.replication	2*
mapred-site.xml	yarn.app.mapreduce.am.resource.mb	1024 MB
mapred-site.xml	mapreduce.map.memory.mb	512 MB
mapred-site.xml	mapreduce.reduce.memory.mb	512 MB
yarn-site.xml	yarn.nodemanager.resource.memory-mb	3072 MB
yarn-site.xml	yarn.scheduler.maximum-allocation-mb	3072 MB
yarn-site.xml	yarn.scheduler.minimum-allocation-mb	256 MB

**2 indicates the replication factor. By default, the HDFS replication factor is 3, but in the experimentation the factor was set to 2, because there is one master node and two worker nodes*

Table 3.8: SPARK Configuration Parameters

Parameter Name	Value
spark.driver.memory	1024 MB
spark.yarn.am.memory	1024 MB
spark.executor.memory	1024 MB
spark.executor.cores	2*

**2 indicates the number of concurrent tasks an executor can run*

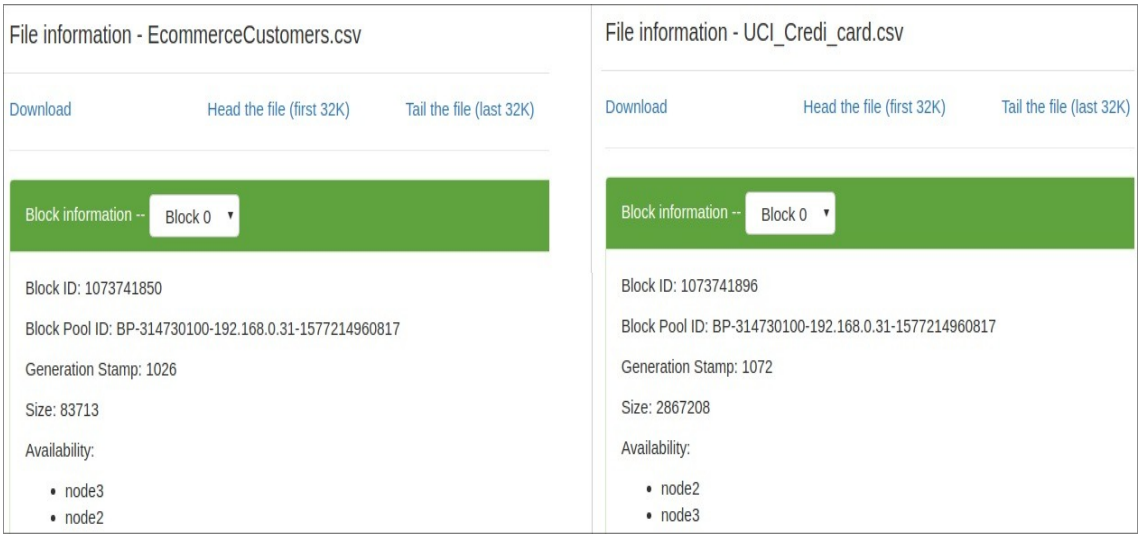


Figure 3.27: Storage of datasets in HDFS with a replication factor of 2

Table 3.8 gives information about the parameter values specified in our experimental setup (with 4GB RAM at each node).
The master node maintains knowledge about the distributed file system and schedules resource allocation. It hosts two daemons:

- The `NameNode` manages the distributed file system and knows where stored data blocks inside the cluster are.
- The `ResourceManager` manages the YARN jobs and takes care of scheduling and running processes on the worker nodes.

Worker nodes store the data and provide processing power to run the jobs, and will host two daemons:

- The `DataNode` manages the physical data stored on the node.
- The `NodeManager` manages the execution of tasks on the node.

3.2.3.2 SPARK Fog Cluster Evaluation

The functionality of the Spark fog cluster has been tested by running linear regression and logistic regression machine learning algorithms using the API from `spark.ml` library. Scala programming language is used to develop applications.

eCommerce customer dataset [81] and the bank loan credit risk dataset from the UCI machine learning repository [82] are used. Figure 3.27 shows the storage of data set files in HDFS with a replication factor of 2.

eCommerce customer dataset contains 8 attributes (such as Avg. Session Length, Time on App, Time on Website, Length of Membership, Yearly Amount Spent) with 500 instances. Linear Regression algorithm is used on this dataset to predict the value of Yearly amount spent attribute (dependent variable). Bank loan credit risk dataset contains 24 attributes (such as limit balance, bill amount, payment amount) with 30,000 instances. Here, logistic regression is used for the binary classification task to determine whether payment will be done for next month or not (dependent variable – `default_payment_next_month`).
Various steps followed to build the linear regression model in the data set of eCommerce customers in the group established in Section 3.2.2 are given below:

- Construct data frame by reading the eCommerce customer data set (.csv file downloaded from kaggle.com [81]).
- Add feature vector as a column to the data frame using VectorAssembler that allows the machine learning algorithm to use the features.
- Add label column to the data frame with the values of Yearly Amount Spent column.
- Dataset is split into training (80%) and test (20%) datasets by using randomSplit() function.
- Create an object on the LinearRegression( ) class and train the model using function fit( ) on the training data set.
- Run the model on a test data set to get the predictions. The trained model is also used to predict a new set of data.

Various steps followed to build a logistic regression classifier on bank loan credit risk data set on the cluster established in Section 3.2.2 are given below:

- Construct data frame by reading the bank loan credit risk data set [82].
- Add feature vector as a column to the data frame using VectorAssembler, that allows the machine learning algorithm to use the features.
- Add label column to the data frame with the values of Creditability column.

Table 3.9: Performance Metric Values for Linear Regression Model

Metric Name	Value
Root mean square error	9.9232567
Mean square error	98.4710252
R-squared	0.9843155

**2 indicates the number of concurrent tasks an executor can run*

- Dataset is split into training (70%) and test (30%) datasets by using randomSplit() function with a seed value of 5043.
- Create an object on LogisticRegression() class by setting the max iteration value to 100 and train the model using the fit() function on the training data set.
- Run the model on test data set to get the predictions. The prediction data frame is constructed using the transform() function on the logistic regression model.
- An object on BinaryClassificationEvaluator() class is created using the metric areaUnderROC to obtain the accuracy of the logistic regression model.

3.2.4 Results and Discussion

The proposed system makes use of the distributed storage and processing capabilities of Spark cluster to execute machine learning algorithms on large datasets in the resource-constrained environment augmented with cluster management. The performance metrics such as root mean square error, mean square error, and R-squared for the linear regression model on the e-Commerce customer data set are given in Table 3.9.

The prediction results of the model on a new set of data (given in Table 3.10) are shown in Figure 3.28. In this result, the first column of the table is the actual value of the dependent variable (yearly amount spent), the second column (features) is the feature vector prepared with the independent variable values. The

Table 3.10: New data items used for testing linear regression model on Ecommerce customer dataset

Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
40.49727	12.65565	39.57767	3.082621	687.9511
31.92627	20.10946	37.26896	2.664034	492.2049
33.00091	11.33028	17.1106	4.104543	487.5475

```
+-----------------+--------------------+-----------------+
|            label|            features|       prediction|
+-----------------+--------------------+-----------------+
|687.951053968401|[40.4972677251123...|687.5675908115015|
|492.204933444326|[31.9262720263602...|728.7461891763403|
|487.547504867472|[33.0009147556427...|496.4653551642202|
+-----------------+--------------------+-----------------+
```

Figure 3.28: Prediction result of the model given by the master node for a new set of data (given in Table 3.10)

prediction model uses this feature vector. The last column (prediction) indicates the prediction made by the model.

The execution of spark application on Ecommerce customer dataset created three containers in the cluster (two on node2 and one on node3), shown in Figure 3.29. The fog computing cluster can handle processing large datasets efficiently because of performing operations on RDDs using executors on several worker nodes in the cluster.

Figure 3.30 shows information on the execution of tasks by the Executors in the fog cluster environment. An instance of the creation of Executors to do

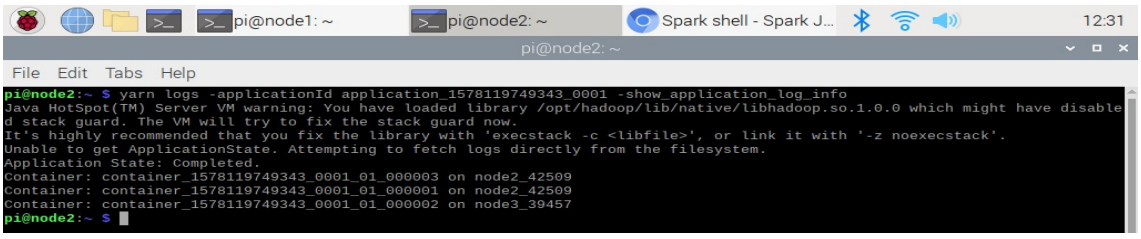


Figure 3.29: Containers created as part of the execution of spark application on Ecommerce customer dataset

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)
driver	node1:45651	Active	0	0.0 B / 366.1 MiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)
1	node3:36947	Active	0	0.0 B / 366.1 MiB	0.0 B	2	0	0	5	5	18 s (0.6 s)
2	node2:44355	Active	0	0.0 B / 366.1 MiB	0.0 B	2	0	0	8	8	20 s (0.5 s)

Figure 3.30: Execution of various tasks by Executors in the fog cluster environment

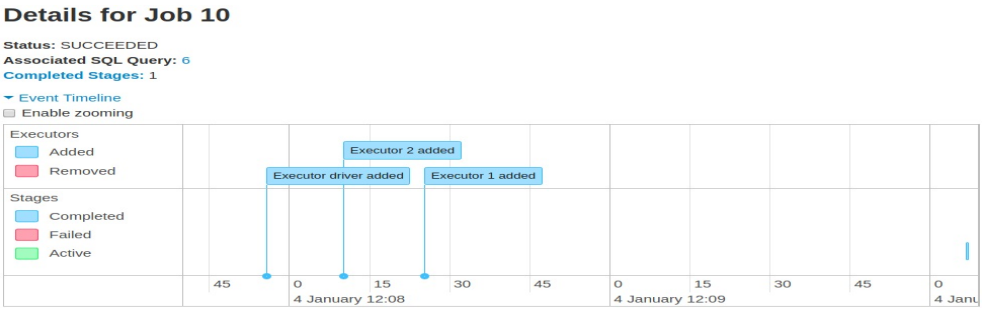


Figure 3.31: Executors information for a particular job execution in the cluster

the job submitted to the cluster is shown in Figure 3.31. As fog devices are typically resource-constrained, the configuration of parameters related to memory and CPU is very important for successfully establishing the cluster and making use of resources in the cluster for big data processing tasks. The classification accuracy of the binary classifier on the credit risk dataset is 82% using the metric area under the ROC curve (AUC) (given in Table 3.11).

Table 3.11: Performance metric values for Logistic Regression model

Metric Name	Value
Accuracy	0.8204
Area under ROC Curve (AUC)	0.6332

3.3 Experimental Findings

In smart-home environment, a diverse range of heterogeneous data is generated, with varying rates of data generation. While fog computing has shown promising advantages in terms of improved response time and privacy, the limitations of individual fog nodes become apparent when resource-intensive applications are run. This was apparent during training of the face mask detection model (model-1 training presented in Section 3.1.3.3). During this model training with 770 images in the dataset, the warning message ***"allocation of memory exceeding 10% of system memory"*** is received, due to resource limitations. We can deal with this situation by either using high-end machines or using the cluster resources for model training.

Distributed processing of IoT data, using cluster resources in the fog environment, provides a viable solution to process complex tasks. Existing distributed machine learning frameworks are not ideal choices for running machine learning applications in fog cluster environments due to their resource-intensive nature. Moreover, these frameworks cannot deal with heterogeneity of IoT data. This was evident from the experimentation carried out to set up Spark Cluster (outlined in Section 3.2.3) using resource constraint devices. The Spark Cluster could not be setup successfully due to memory limitations when RPi 3B + (with 1GB RAM) units are used.

In order to address these challenges, our research focused on exploring suitable frameworks that can run efficiently on diverse resource-constrained devices while effectively handling the heterogeneity of data. The concept of federated learning emerges as a promising solution to effectively tackle these issues. The subsequent chapters of this thesis center around the application of federated learning for processing IoT data.

The research work presented in this chapter is published in our following research publications:

- S. R. Rudraraju, N. K. Suryadevara and A. Negi, **Face Mask Detection at the Fog Computing Gateway**, *Proceedings of 15th Conference on Computer Science and Information Systems (FedCSIS), Sofia, Bulgaria, 2020, IEEE*, Pages: 521-524, 2020, doi: 10.15439/2020F143. Indexed in: DBLP, Scopus. Status: Accepted and Published
  URL: https://ieeexplore.ieee.org/abstract/document/9222988

- N. K. Suryadevara, A. Negi and S. R. Rudraraju **A Smart Home Assistive Living Framework Using Fog Computing for Audio and Lighting Stimulation**, *Proceedings of Advances in Decision Sciences, Image Processing, Security and Computer Vision: International Conference on Emerging Trends in Engineering (ICETE), Hyderabad, India 2019, Springer*, Pages: 366-375, doi: 10.1007/978-3-030-24322-747. Status: Accepted and Published
URL: https://link.springer.com/chapter/10.1007/978-3-030-24322-7$_4$7

- S. R. Rudraraju, N. K. Suryadevara and A. Negi, **Edge computing for visitor identification using eigenfaces in an assisted living environment**, Editors: Nagender Kumar Suryadevara, Subhas Chandra Mukhopadhyay, *Assistive Technology for the Elderly*, Academic Press, 2020, Pages 235-248, ISBN 9780128185469, Indexed in: Scopus, Status: Accepted and Published
URL: https://doi.org/10.1016/B978-0-12-818546-9.00008-7

- S. R. Rudraraju, N. K. Suryadevara and A. Negi, **Fog computing framework for Big Data processing using cluster management in a resource-constraint environment**, Editors: Ravi Vadlamani, Aswani Kumar Cherukuri, *Handbook of Big Data Analytics: Methodologies*, IET, 2021, Pages 317-334, Indexed in: Scopus, Status: Accepted and Published
URL: https://digital-library.theiet.org/content/books/10.1049/pbpc037f$_c$h9

- S. R. Rudraraju, N. K. Suryadevara and A. Negi, **Face Recognition in the Fog Cluster Computing**, *Proceedings of International Conference on Signal Processing, Information, Communication & Systems (SPICSCON), Dhaka, Bangladesh, November 28-30, 2019, IEEE*, Pages: 45-48, 2019, doi: 10.1109/SPICSCON48833.2019.9065100. Indexed in: Scopus. Status: Accepted and Published
URL: https://ieeexplore.ieee.org/abstract/document/9065100

- S. R. Sahith, S. R. Rudraraju, A. Negi and N. K. Suryadevara, **Mesh WSN Data Aggregation and Face Identification in Fog Computing Framework**, *Proceedings of 13th International Conference on Sensing Technology, ICST 2019, Sydney, Australia, December 2-4, 2019, IEEE*, Pages: 1-6, 2019, doi: 10.1109/ICST46873.2019.9047708. Indexed in: DBLP, Scopus. Status: Accepted and Published
URL: https://ieeexplore.ieee.org/abstract/document/9047708

3.4 Summary

In this chapter, we presented our research work aimed at supporting residents of a fog-enabled smart home environment. We introduced an audio and lighting stimulation-based system tailored to help people with dementia recognize visitors using LBPH and Eigenfaces methods. Furthermore, a two-level face mask detection system is discussed at the fog computing gateway and a face recognition system that utilizes a fog cluster is discussed. We also explored the adoptability of existing DML frameworks for processing IoT data by setting up Apache Spark in the fog cluster. To conclude the chapter, we collected and analyzed observations from the experiments carried out during this research that are significant for conducting research work in subsequent chapters.

Chapter 4

Federated Learning for Processing Heterogeneous Sensor Data in IoT Environment

This chapter presents the research work related to processing heterogeneous sensor data using federated learning in fog-enabled smart home environment,

4.1 Introduction

Data collected from diverse sensor types, such as temperature, humidity, motion, vision, and pressure sensors, constitute heterogeneous sensor data. In this research, we employ a distributed approach to fuse data from vision sensors, digital ambient sensors, and passive infrared sensors. Processing of these data is done through federated learning on Raspberry Pi edge nodes. The system trains machine learning models on these edge nodes using federated learning, eliminating the need to transmit data to a centralized server. Instead, results from training on multiple edge nodes are aggregated at a central node to produce a final machine learning model. Each edge node deploys the aggregated model for subject recognition in the smart home environment, triggering alerts upon detecting unknown subjects. Additionally, linear regression models for temperature prediction and logistic regression models for humidity prediction, using ambient parameters, are trained in a federated manner on the same edge nodes. The Flower federated learning framework [83] is employed for model training. This system is well-

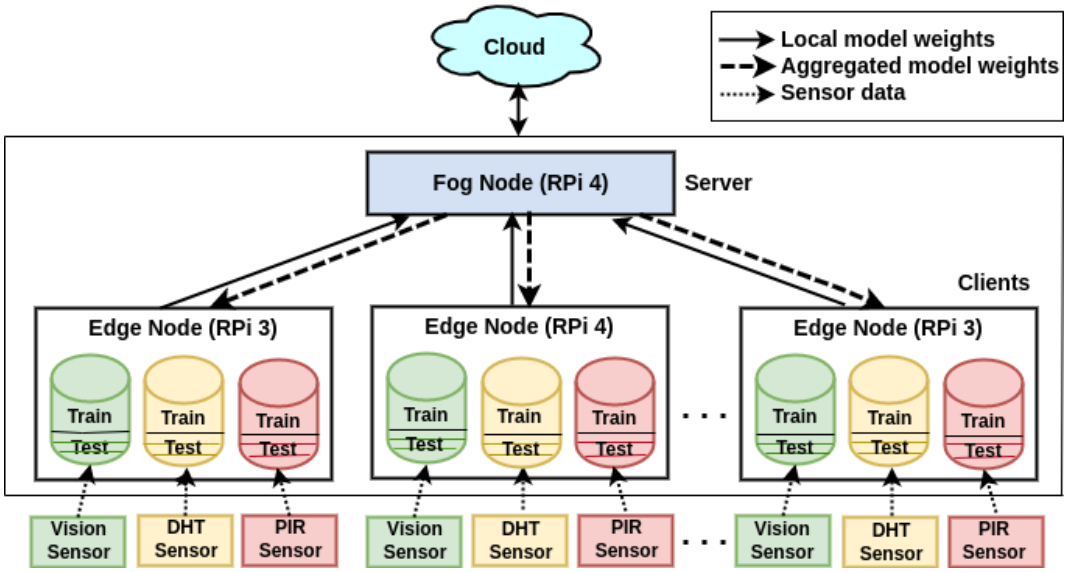


Figure 4.1: The basic architecture of the proposed system

suited for environments with limited communication bandwidth or data privacy concerns.

4.2 System Description

The basic architecture of the proposed system is illustrated in Figure 4.1. The federated learning application is architected with the help of a fog node that acts as a server. The edge nodes encompass vision sensor (RPi camera), passive infrared (PIR) sensor, and digital humidity and temperature (DHT) sensor. Typically, the edge nodes are strategically deployed at different locations in a Smart Home or Smart Building, with a single fog node serving as the aggregator/server. The network consists of nodes with varying specifications, including the RPi camera for image data, the PIR sensor for digital data, and the DHT sensor for converted analog-to-digital values. Collectively, they facilitate the transmission of sensed information to a sink node responsible for data collection and aggregation, constituting a wireless heterogeneous sensor network.

The system deploys heterogeneous sensors distributed across different rooms within a smart home, each connected to a Raspberry Pi (RPi) edge node. When the Passive Infrared (PIR) sensor detects movement in a room, it triggers the RPi camera to capture photos. These edge nodes gather and store ambient parameters

Algorithm 4.1 Face Recognition Model Training using FL

Server: initialize w_0	**Client(*i*):** *//Run on client i*
for each round **r** = 0,1,... **do**	Apply transformations – rotation, resize, shear – on each image in the dataset
for each client c_i **do in parallel**	Construct the CNN for face recognition
$m_p(r) \leftarrow$ Get updated weights from client *i*	**for** each round ***r*** = 0,1,... **do**
$m_p(r) \leftarrow \sum_{i=1}^{n} \frac{n_i}{n} m_p(r)$	**Receive** the aggregated weights from server
Send the aggregated weights to each client	**Split** the local dataset into batches of size *B*
	for each *local epoch e=0...E-1* **do**
	for each minibatch *b* of size *B* **do**
	$m_p(r) = m_p(r) - \eta * L_{(n)}(m_p(r, b))$
	return m_p to Server

(temperature and humidity) and images of individuals locally. This collected data serves as input for model training using federated learning, utilizing both the edge nodes and the fog node. The proposed system trains three ML models: facial recognition model training on face images, temperature and humidity prediction models training on ambient parameters in a federated fashion. One of the edge nodes also serves the purpose of testing the system, in addition to collecting sensor data and model training. The steps involved in training the facial recognition model are detailed in Algorithm 4.1.

Each round of federated learning comprises 3 steps: (i) receiving the aggregated weights from the server by each edge client, (ii) local model training by each client using the aggregated weights and the respective local dataset, and (iii) sharing the local updated weights from each client to the server for aggregating the weights. Initially, the server program is started, specifying the number of FL rounds and awaiting client connections to participate in the FL process. At the beginning of each round, the server node samples the corresponding clients $C = \{c_1, c_2, \ldots, c_n\}$. The dataset $(d_1, d_2, \ldots, d_n)$ comprises heterogeneous sensor data distributed across the 'n' edge clients, featuring unbalanced items. Each client c_i updates the model parameters (m_p) using the local dataset d_i. The training dataset consists of input and expected output (labelled data items).

The client program initially constructs various layers in the CNN model. In the IoT environment, data pre-processing plays a crucial role, as not all nodes get sufficient data sets to train effective models, which can lead to overfitting. To address this, data augmentation techniques, such as image rotation, resizing, shearing, and normalization, are applied to the local dataset (d_i) prior to its

utilization in model training. Within each round, model training occurs with the local dataset divided into batches of size 'B' for a specified number of epochs. The nodel training uses a learning rate denoted as η, a loss function represented by $l_e()$, and global aggregated weights (m_p) received from the server. In our experimental setup, we used a default learning rate of $\eta = 0.001$, and the categorical cross-entropy loss function during the face recognition model training. The aggregated loss, measured locally in the edge client, is defined by Equation 4.1, in which $l_e()$ corresponds to the local loss in the respective edge client, and the accuracy of the local model training falls within the range $0 \leq acc \leq 1$.

$$L_{(n)} = \frac{1}{\mid d_n \mid} \sum_{e \epsilon d_k} l_e(m_p) \tag{4.1}$$

Every edge client (n) updates the model parameters (m_p) as given by equation 4.2 where, $r = 1, 2, 3, \ldots, rounds$.

$$m_p(r) = m_p(r) - \eta * L_{(n)}(m_p(r, batch)) \tag{4.2}$$

After each round, the client program shares the updated weights back with the server, which performs weighted average using the FedAvg algorithm, and sends back the global model weights to each client. Alongside the training of the face recognition model, each node trains a linear regression model for temperature prediction and logistic regression model for humidity prediction using ambient parameters in a federated manner (similarly to the way discussed in Algorithm 4.1). The various steps in training the ambient parameter prediction model are outlined in Algorithms 4.2 and 4.3. The temperature and humidity prediction models training differs from the face recognition model training in terms of the machine learning model used and the dataset used for the training purposes. The model accuracy and loss values are calculated using the metrics R^2 score and mean squared error (MSE), respectively. The general formulas for calculating R^2 Score and MSE are provided in Equations 4.3 and 4.4 respectively.

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y}_i)^2} \tag{4.3}$$

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{4.4}$$

The R^2 score is derived by comparing the total sum of squares of residuals generated by the regression model against the total sum of squares of errors computed

Algorithm 4.2 Temperature Prediction Model Training using FL

Server:	**Client(*i*):** *//Run on client i*

for each round **r** = 0,1,... **do**
  for each client c_i **do in parallel**
  $m_p(r) \leftarrow$ Get updated weights from client *i*
  $m_p(r) \leftarrow \sum_{i=1}^{n} \frac{n_i}{n} m_p(r)$
Send the aggregated weights $m_p(r)$ to each client
	Split the dataset into train set and test set
	X ← Independent variables min temp, and humidity
	Y ← dependent variable max temp
	Initialize the linear regression model ***L*** with random weights
	for each round **r** = 0,1,... do
	  Receive the aggregated weights $m_p(r)$ from server
	  For each minibatch b of local dataset (***X,Y***)
	  $m_p(r) = m_p(r) - \eta * L_{(n)}(m_p(r,b))$
	  Return the updated model weights to the server

Algorithm 4.3 Humidity Prediction Model Training using FL

Server:	**Client(*i*):** *//Run on client i*

for each round **r** = 0,1,... **do**
  for each client c_i **do in parallel**
  $m_p(r) \leftarrow$ Get updated weights from client *i*
  $m_p(r) \leftarrow \sum_{i=1}^{n} \frac{n_i}{n} m_p(r)$
Send the aggregated weights $m_p(r)$ to each client
	Split the dataset into train set and test set
	X ← Independent variables temperature, and humidity
	Y ← dependent variable humidity *tomorrow*
	Initialize the logistic regression model ***L*** with random weights
	for each round **r** = 0,1,... do
	  Receive the aggregated weights $m_p(r)$ from server
	  For each minibatch b of local dataset (***X,Y***)
	  $m_p(r) = m_p(r) - \eta * L_{(n)}(m_p(r,b))$
	  Return the updated model weights to the server

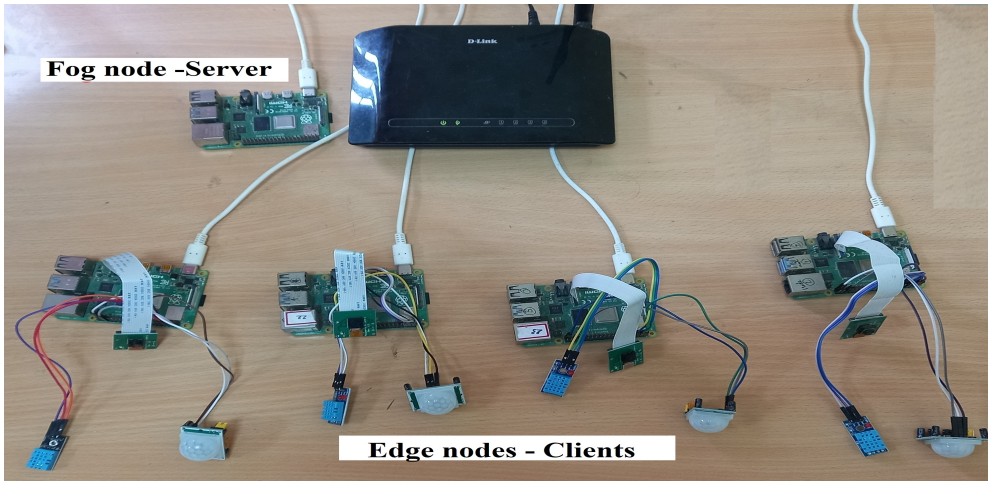


Figure 4.2: Experimental setup of the proposed system

from the average model, followed by subtracting this value from 1. MSE evaluates the average of the squared differences between observed and predicted values. The trained ML models are used at each edge node for inference of subject identification, temperature prediction, and humidity prediction.

4.3 Implementation Details

The experimental configuration of the proposed system is depicted in Figure 4.2. The system provides a proof of concept for processing heterogeneous sensor data on resource constrained fog devices using federated learning. The implementation comprises 5 Raspberry Pi (RPi) devices, each equipped with 4GB RAM and a 32GB SD Card, featuring a mix of RPi3 and RPi4 models with 32-bit and 64-bit processor architectures. Among these five units, 4 units of RPis are deployed each in one room. These are used as client/worker nodes (edge nodes), and the fifth unit is used as the aggregator/server node (fog node) for federated learning. Each edge node is equipped with an RPi camera, a DHT sensor, and a PIR sensor for data collection within the application. The RPis establish communication through Wi-Fi to facilitate network connectivity.

The experimental setup employs the Flower framework [83], which is ML framework agnostic, to implement federated learning. Communication between clients and the server during model training is facilitated through the gRPC mechanism [84]. Tensorflow-2.12.0 is used for face recognition model training

[85] [86] [87], and Scikit-learn-1.2.2 [88] is used for temperature and humidity prediction models training.

4.3.1 Face Recognition Model Training using Vision Sensor

The Face Recognition Convolutional Neural Network (CNN) architecture is composed of an input layer, four hidden layers for convolution, four hidden layers for max pooling, one flattening layer, one hidden Artificial Neural Network (ANN) layer, and one output layer [89]. With 31 classes of images in the dataset, the output layer is equipped with 31 neurons. To prevent overfitting and enhance the model's generalization ability, early stopping is incorporated during training. This mechanism terminates the training process if the model's performance on the validation set stagnates or deteriorates. Validation loss is monitored, and early stopping is triggered if no improvement is observed for five consecutive epochs within a specific round. The CNN is trained on a face recognition dataset [90] comprising 31 distinct classes, encompassing a total of 2562 unique facial images, each with a resolution of 160x160 pixels. The dataset is divided into four equal partitions class-wise, with one partition allocated per client, facilitating federated machine learning. In reality, the model training uses photos of subjects captured in each room through the vision sensor (RPi Camera), eliminating the need for manual intervention.

4.3.2 Temperature, Humidity Prediction Models Training using Ambient Sensors

Linear and logistic regression models for maximum temperature prediction and humidity prediction are developed with the Scikit-learn package [88]. These models are trained in a federated fashion using locally collected datasets from ambient sensors on each client. Performance evaluation uses a Kaggle dataset [91], which is divided into four equal segments, with each segment assigned to one client for training purposes. The construction of these models uses attributes such as temperature, humidity, precipitation, pressure, wind speed, and cloud cover. Data preprocessing and model training are carried out using Pandas and Scikit-learn packages. Model performance is evaluated using training and validation accuracy values.

4.4 Results and Discussion

Experiments are carried out by varying hyperparameter values such as the number of rounds (#rounds) and the epochs per round (#epochs), to assess the performance of the proposed system.

4.4.1 Performance Analysis of Face Recognition Model Training

The accuracy and loss metrics for the training of the face recognition model (with # rounds = 1, # epochs = 10) conducted in a federated manner without any data pre-processing or normalization are depicted in Figure 4.3. The lines in the graph represent the performance of different clients participating in the federated learning process. The training accuracy values show improvement from 0.05 (at the end of epoch#1) to 0.96 (at the end of epoch#10), measured on a scale of [0-1]. The validation accuracy values for face recognition model training vary across different clients from epoch#1 to epoch#10. This variability can be attributed to potential overfitting due to the relatively small size of the dataset. Similarly, the training loss values for all clients fall within the range of 400 to 600 at the start of model training (end of epoch#1), and then reduced to a smaller range of 1 to 3 by the end of epoch#2, gradually reducing further in subsequent epochs.

The training and validation accuracy values of face recognition model training in a federated manner, without any data preprocessing or normalization using 3 rounds, with 10 epochs per round, are illustrated in Figure 4.4. At the end of each round, the server collects local model weights from each client, aggregates them, and then distributes the values of global model parameters to all clients. This ensures that each client has a global view of the model weights. The accuracy of the model is improved through multiple rounds of training, with the data being fed into the training algorithm. The first column in Figure 4.4 shows the model training accuracy values for each client during the three rounds. In particular, we observe an improvement in the initial values of model training accuracy (at epoch#1) from round-1 to round-3, as clients receive updated global model weights from the server at the end of each round. The output screenshot of the sample execution of FL based face recognition model training is shown in Figure 4.5.

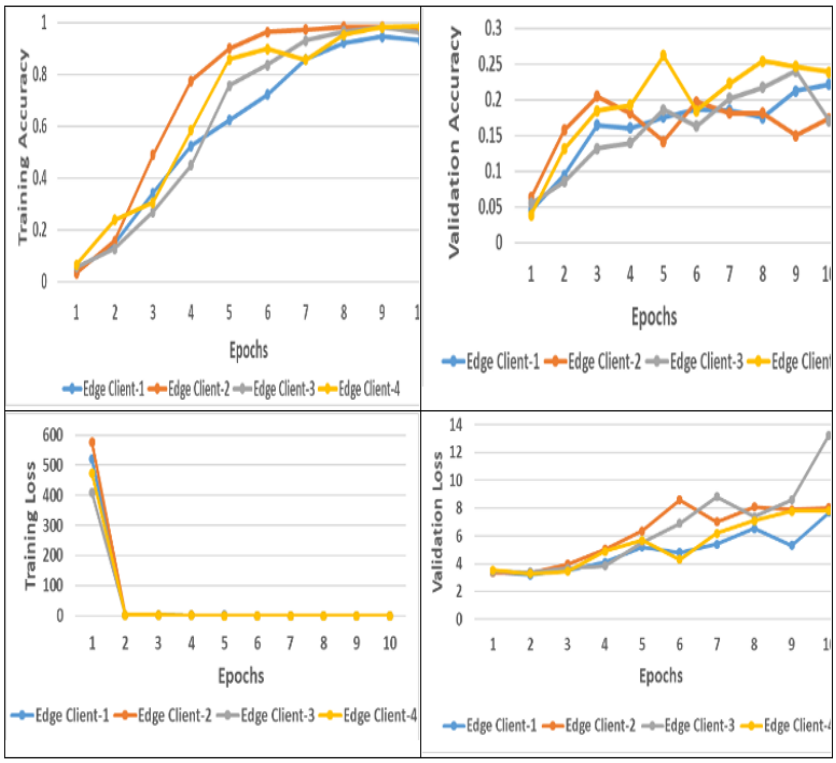


Figure 4.3: Face recognition model training - accuracy, loss values: Federated learning with no data preprocessing and no normalization (#rounds=1, #epochs=10).

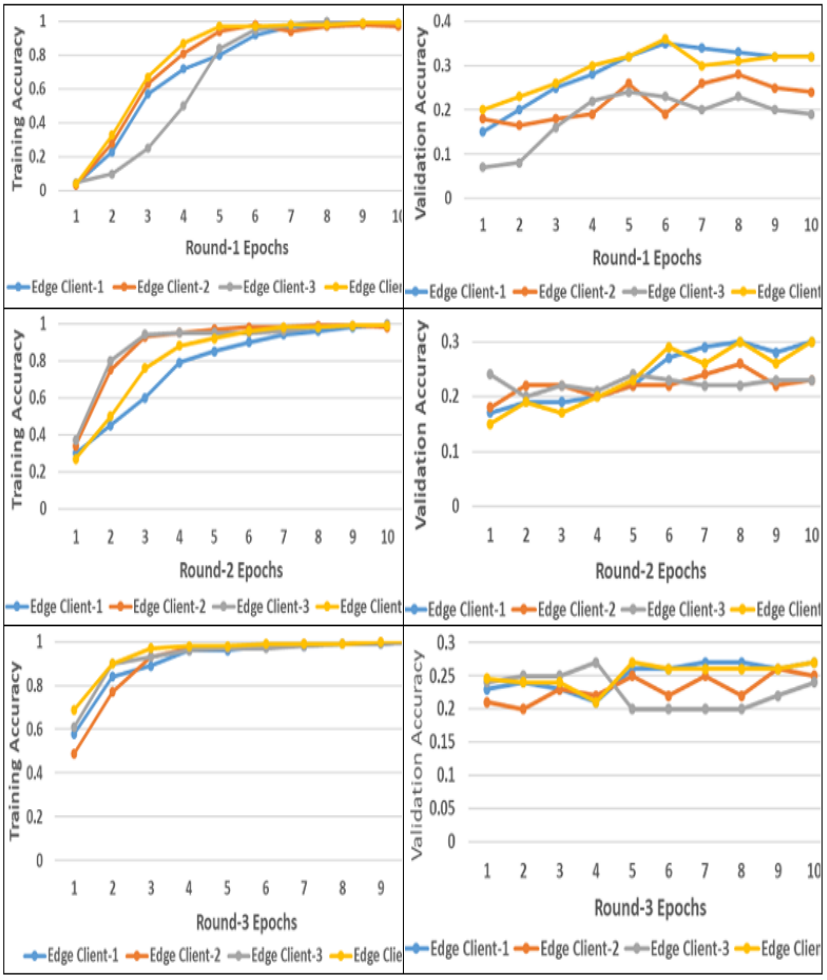


Figure 4.4: Face recognition model training - accuracy, loss values: Federated learning with no data preprocessing and no normalization (#rounds=3, #epochs=10)

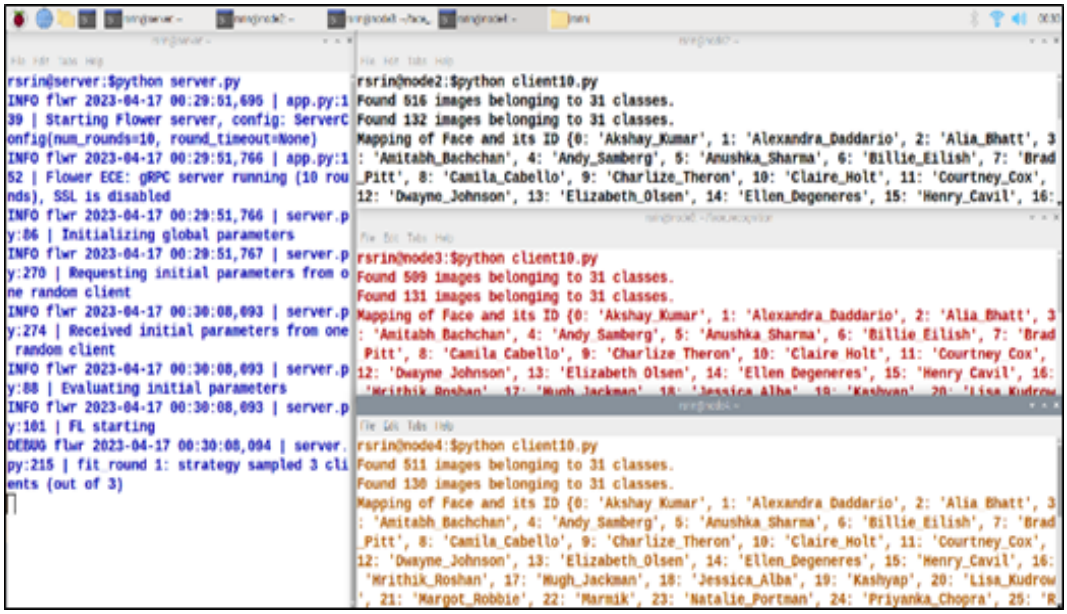
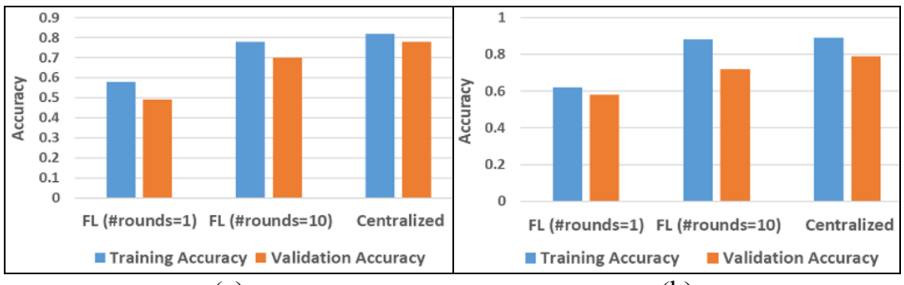


Figure 4.5: Output screenshot of face recognition model training using FL in edge/fog environment



(a) (b)

Figure 4.6: (a) Temperature prediction model accuracy values (b) Humidity prediction model accuracy values

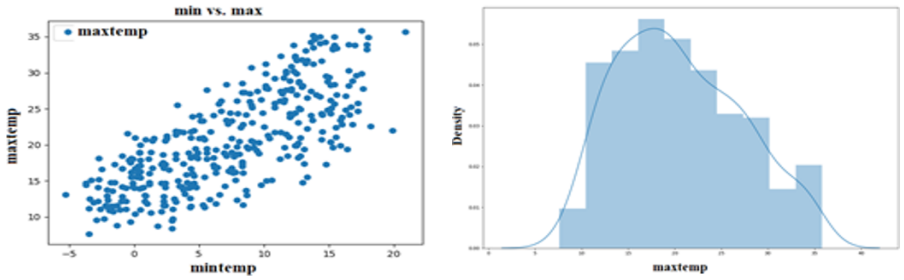


Figure 4.7: (a) Plot showing the correlation between min temperature and maximum temperature (b) Plot showing the density of maximum temperature values

4.4.2 Performance Analysis of Temperature Prediction, Humidity Prediction Models Training

The performance analysis of the linear regression model for maximum temperature prediction, and the logistic regression model for humidity prediction are given in Figure 4.6. The experimentation is done in three scenarios: (i) FL using one round (ii) FL using 10 rounds, and (iii) centralized training. In the FL setup, the model performance is improved considerably with the number of rounds. There is not much difference in model performance between FL with 10 rounds and centralized model training. Visualization of the ambiance parameter values is also done using matplotlib package [92], in addition to the model training. A visualization of some values of the ambient parameters is presented in Figure 4.7.

The accuracy of the FL based models improves with more number of rounds, as the clients see model weights obtained from training local datasets from other clients involved in the FL process. Furthermore, proper data augmentation and normalization techniques are very important to improve the accuracy of the FL model, as not all clients produce the same amount of data. The impact of the number of rounds and normalization techniques on model training performance is further analyzed in Chapter 7.

The research work presented in this chapter is published in our following research publication:

S. R. Rudraraju, N. K. Suryadevara and A. Negi, "Heterogeneous Sensor Data Acquisition and Federated Learning for Resource Constrained IoT Devices—A Validation," in *IEEE Sensors Journal*, vol. 23, no. 15, pp. 17602-17610, 1 Aug.1, 2023, doi: 10.1109/JSEN.2023.3287580. Indexed in: Scopus, Web of Science

(SCIE).
URL: https://ieeexplore.ieee.org/abstract/document/10161724

4.5 Summary

In this chapter, a FL framework based on fog computing is presented to process heterogeneous sensor data using resource constraint devices. This work presents the complete cycle of federated learning on Raspberry Pi devices (including edge clients and server) using heterogeneous sensor data. The data gathered from various sensors, such as vision sensor and ambient sensors, is used to train various ML models in a federated fashion. Convolutional Neural Network (CNN) is trained using facial images collected at each edge node using federated learning. The ambient data collected in the smart home environment is used to train linear regression and logistic regression models for maximum temperature prediction and humidity prediction, respectively. Experiments are conducted in a real-time IoT environment, to evaluate the performance of the federated learning-based model training with that of centralized training on resource constraint devices. The experimental results demonstrate that deep learning-based model training can be done on resource constrained devices using federated learning. The performance of models trained using federated learning is closer to that of models trained using centralized training with proper data preprocessing and normalization techniques.

Chapter 5

Load Balancing for Processing Streaming Data

This chapter presents the research work related to load balance-aware federated learning for handling heterogeneous IoT streaming data in a fog-enabled smart home environment.

5.1 Introduction

The advent of the Internet of Things (IoT) has contributed to a significant surge in the generation of heterogeneous streaming data. Integration and analysis of these data can be challenging, as IoT environments employ diverse sensors that generate data with different formats, sampling rates, and other characteristics. To work with these heterogeneous sensor data, it is often necessary to develop methods to normalize the data, extract relevant features, and fuse the data [93]. These methods can help transform sensor data into a uniform format that can be easily analyzed.

The primary focus of this research work is to implement load balance-aware federated learning to handle heterogeneous IoT streaming data in a smart home environment enabled by fog computing. A novel approach is proposed to integrate and process vision sensors and digital ambient sensors streaming data from heterogeneous edge nodes based on the load metric is proposed. The system uses the load metric of the edge nodes in the cluster, governed by the fog server, to distribute the streaming IoT data load among several edge nodes. The load metric of each node is calculated based on its memory and processor utilization and

is shared with all edge nodes on the client to prepare their local datasets. The proposed approach ensures efficient resource allocation and load balancing in the edge cluster environment.

5.1.1 Contribution

This research work presents a practical implementation of load balance-aware FL to process heterogeneous streaming data in IoT sensor network:

- An efficient load balancing scheme to distribute the heterogeneous streaming data evenly across the resource-constrained edge nodes in the cluster based on the resource utilization.
- Layered architecture to process the heterogeneous sensor data using the Flower FL framework in the resource-constrained edge/fog cluster environment.

5.2 System Description

The underlying structure of the proposed system is depicted in Figure 5.1. The fog node acts as a server to architect the FL application, while each node is integrated with the vision sensor (RPi camera [58]/ IMX219-77 Camera [94]), a Passive Infrared (PIR) Sensor [57], and a Digital Humidity and Temperature (DHT) sensor [95]. In the Smart Home/Smart Building context, edge nodes are typically distributed across various locations, with one fog node functioning as the central gateway. The PIR sensor activates the vision sensor to capture video when there is motion in any room. The edge nodes gather and reposit ambient parameters data and video data, which will serve as input for FL based model training with edge nodes acting as clients and fog node as the server.

5.2.1 Heterogeneous Streaming Data Collection

The cluster environment comprises 'n' edge nodes/clients, each of which is equipped with a vision sensor, a PIR sensor, and a DHT sensor. The PIR sensor triggers video data streaming in a particular room, while the data from the DHT sensor is collected once every minute. Apache Kafka software [96] [97] is installed on each node in the cluster for stream processing. Each client node in the cluster

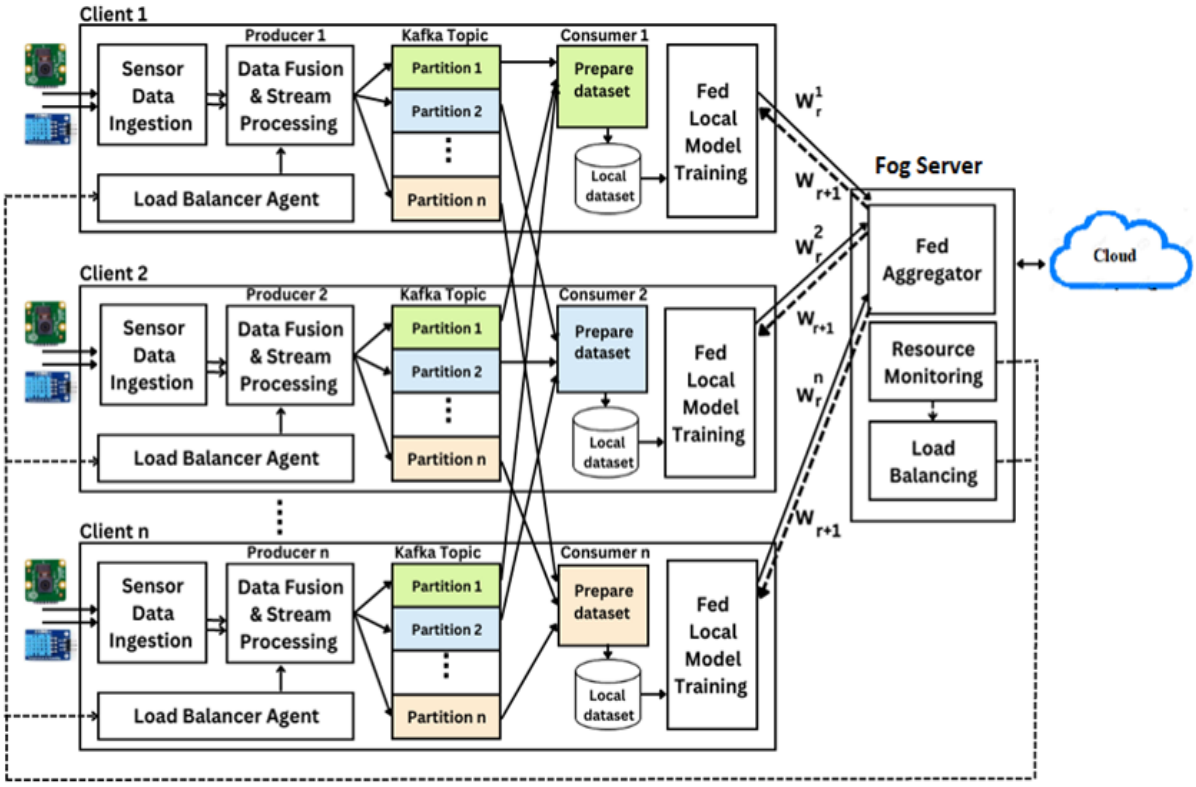


Figure 5.1: Underlying Structure of the Proposed System

Algorithm 5.1 Producer and Consumer Processes in each Client (k)

```
//Producer
while True:
    LM ← getLoadMetric()
    i ← selectNode(LM)
    data ← readSensorData()
    write Tki(data)
```

```
//Consumer
// ds – local dataset
while True:
    for each node i in the cluster:
        while True:
            msg = poll(Tik)
            if len(msg) != 0:
                for each item in the msg:
                    ds ← ds.append(item)
            else:
                break
```

creates a Kafka topic for every type of sensor data, such as video data or ambient parameters. These topics are divided into n partitions, each partition being assigned to a specific client node in the cluster. This guarantees that there is a dedicated partition for each node in the cluster, allowing each node to process the incoming data independently and without any conflicts.

The resource monitoring module in the fog server provides the load balancing module with resource utilization statistics, including processor utilization, memory usage, disk usage, and bandwidth usage, as well as the processor characteristics of the client machines. The load balancing agent then shares this information with the load balancing agents on each client node, which determines which node should process the incoming streaming data. The client node producer process writes the input stream into the corresponding client node's partition. This method ensures that the data load is distributed uniformly among the various edge nodes in the cluster when streaming data is generated.

The consumer process on each client node reads the streaming data corresponding to its partition from each node in the cluster in a circular fashion and prepares the local dataset for the application of FL. The high-level operations performed by the producer and consumer processes in each client node 'k' is presented in Algorithm 5.1. The producer process in each client 'k' gets the load metric values of several clients in the cluster, and selects the client 'i' with least load metric value. The streaming data received from the client 'k' is then written into the topic partition 'i' of the source machine 'k' i.e. T_{ki}. Similarly, the consumer process in each client 'k' polls the data from each client 'i' in the cluster from its corresponding topic partition 'k' i.e. T_{ik}.

Various layers in the software stack of the proposed system are given in Table 5.2. Raspbian OS (Debian GNU/Linux 11 (bullseye)) and JetPack OS (which is based

	Fog Server/Coordinator	Edge Client
Federated Learning Layer	Fed Aggregator Service	Fed Local Model Training Service
	ML Framework - TensorFlow 2.12.0	
	Federated Learning Framework - Flower 1.3.0	
Stream Processing Layer	Stream Processing Service	
Device Management Layer	Resource Monitoring & Load Balancing Service	Load Balancer Service
Data Ingestion Layer	Data from Heterogeneous Sensors: Vision Sensor, DHT Sensor, PIR Sensor	
	Apache Kafka 2.12-2.3.0	
OS Layer	Raspbian OS/JetPack OS	

Figure 5.2: Various layers in the architecture of the proposed system

on Ubuntu Linux) are installed on RPi devices and Jetson Nano kit, respectively. JetPack includes the required drivers and software libraries to support NVIDIA's GPU (Graphics Processing Unit) for AI and machine learning tasks.

Apache Kafka is used as a message broker in the data ingestion layer to handle large volumes of incoming streaming data from various sensors. It can buffer and store the incoming data for a short time until it is ready to be processed by downstream systems. The device management layer provides resource monitoring and load balancing services in the cluster. The stream processing layer in each node helps to process the data from the respective partition of Kafka topic from several clients while preparing the local dataset for applying federated machine learning. The Flower framework is used in the federated learning layer to provide fed aggregated service and local model training service on fog server and edge clients, respectively.

5.2.2 Processing the Video Streaming Data

The process of preparing a local dataset for federated learning from video streaming data involves several key steps, as illustrated in Figure 5.3. In each node 'k', the consumer process reads video frames from partition 'k' of the 'VIDEODATA' topic across all nodes in the cluster. Subsequently, it performs face detection on each frame, identifying the coordinates of any detected faces. To classify the person in the frame, the consumer process utilizes a local face recognition model. If

face coordinates are found, the corresponding facial image is stored locally within the appropriate class of the dataset, which is intended for federated learning. In the event that the classification accuracy falls below 90%, the consumer process prompts the user to label the face. If the facial label corresponds to any pre-existing class in the local dataset, the image is added in that particular class. However, if the label belongs to a new class, the consumer process creates a new class within the dataset and includes the image in it.

5.2.3 Federated Machine Learning Model Training

The proposed system employs federated learning approach to train three machine learning models. First, the facial recognition model is trained on facial images. Furthermore, temperature and humidity prediction models are trained on ambient parameters. Within the system, one of the edge nodes serves the purpose of testing the developed system, as well as collecting sensor data and training models. The various steps in facial recognition model training are outlined in Algorithm 4.1, while Algorithm 4.2 and Algorithm 4.3 facilitate the federated training of maximum temperature and humidity prediction models (as discussed in Chapter 4).

5.3 Implementation Details

The setup used for the experimentation of the proposed system is shown in Figure 5.4. It serves as a proof-of-concept for utilizing federated learning on resource-constrained edge/fog nodes for processing the heterogeneous sensor data. The implementation used a mix of heterogeneous devices, including three Raspberry Pi (RPi) units comprising both RPi3 and RPi4 models (32-bit/64-bit processor architectures with 4GB RAM and 32GB SD Card each), along with one Jetson Nano 4GB model (Jetson Nano Developer Kit). Each room is equipped with a RPi/Jetson Nano device, which functions as client or worker nodes (referred to as edge nodes), while the third RPi unit serves as the server node (acting as the fog node) responsible for coordinating FL tasks. Additionally, each edge node is equipped with a camera (RPi Camera/ IMX219-77 Camera), PIR and DHT Sensors for collecting the data in the application. Wi-Fi connectivity is utilized to establish communication among devices within the network.

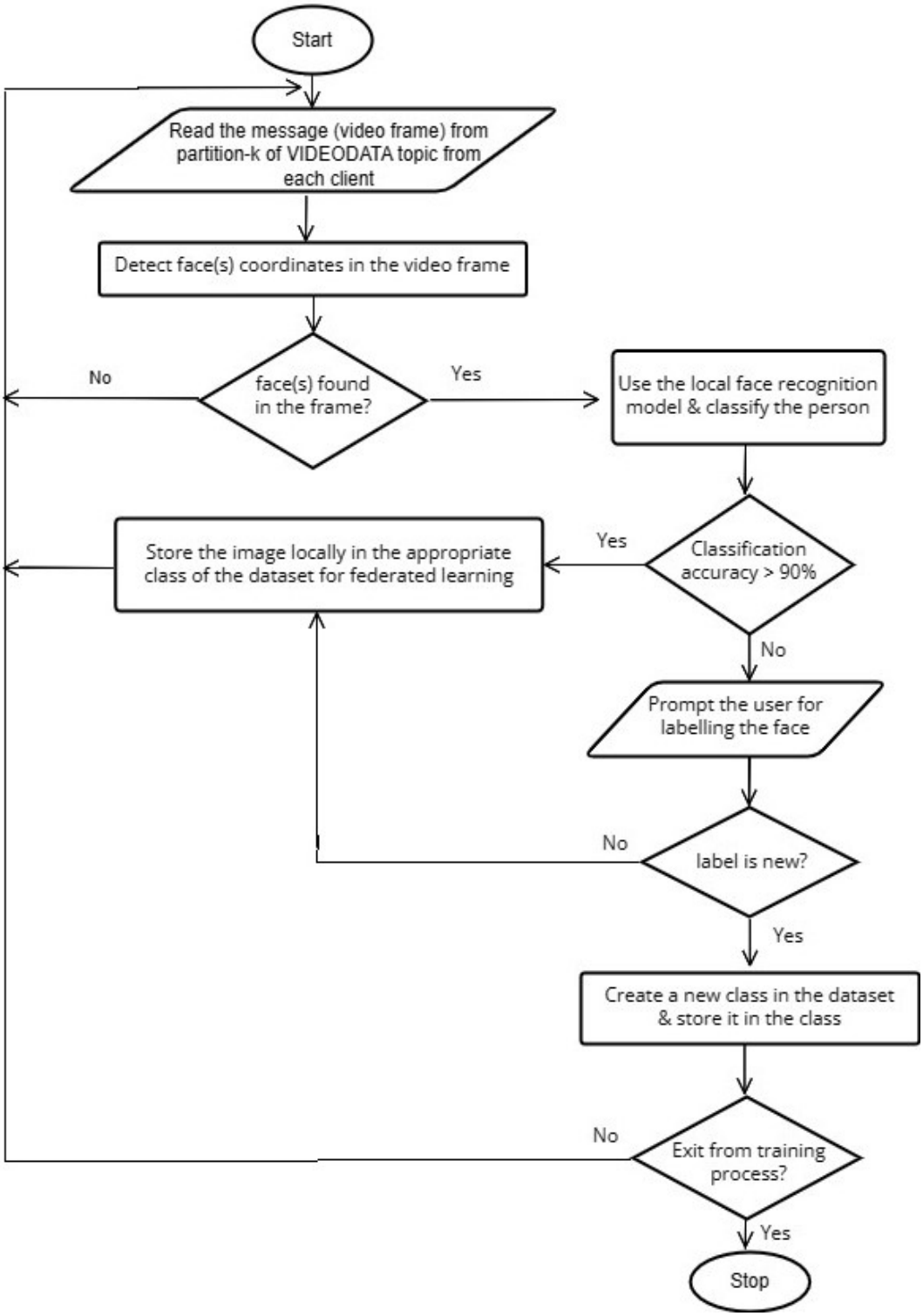


Figure 5.3: Consumer process at node-k for preparation of dataset using video data

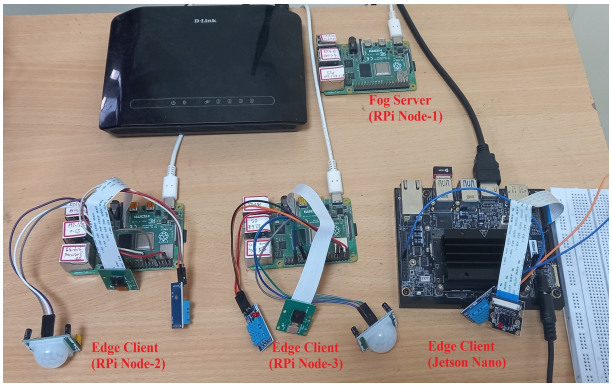


Figure 5.4: Experimental setup of the proposed system

Apache Kafka-2.12 and ZooKeeper-3.4.14 [98], a combination for streaming data collection, is used in our experiment. Kafka efficiently handles high-volume data ingestion and distribution, while ZooKeeper provides a distributed coordination service. The Flower framework is used to implement federated learning in our experimentation, as it is not dependent on any ML framework. The face recognition model training is performed using the Tensorflow-2.12.0 ML package, while the Scikit-learn-1.2.2 package is used for training maximum temperature and humidity prediction models.

5.3.1 Heterogeneous Streaming Data Collection

The server's resource monitoring module uses the **paramiko** Python module [99] to establish SSH connections with various clients in the cluster. These connections are established to retrieve statistics on processor and main memory utilization. Subsequently, the load balancing module in the server calculates the load metric of each client using the resource utilization statistics and shares the information with all the clients in the cluster, by writing into 'loadmetric' kafka topic in the server. The load balancer module in each client reads this information to obtain the load metric value of all clients in the cluster.

5.3.1.1 Video Streaming Data Collection

Video streaming data is captured for one minute duration when there is a trigger by the PIR sensor in any particular room. OpenCV module is used to capture the video stream and extract the frames from the streaming video. The load

balancing act is done by the kafka producer process, by writing these frames (at the source node) in the kafka 'VIDEODATA' topic partition that corresponds to the node with the lowest load metric value. The consumer process in the client machine 'k' reads the frames from partition k of the topic 'VIDEODATA' from all the client nodes as shown in Figure 5.3, and prepares the local facial image dataset for applying federated learning.

5.3.1.2 Ambient Parameters Data Collection

The humidity and temperature values in each room are sampled at every one minute from the DHT sensor attached to each client machine, and written (at the source) to the 'ambientparameters' kafka topic's partition that corresponds to the node with the least load metric value. Similar to the video streaming data collection, the consumer process in client machine 'k' reads the ambient parameters data from partition-k of the 'ambientparameters' topic from all the client nodes and prepares the local dataset for federated learning.

5.3.2 Face Recognition Model Training

The initial training of the federated face recognition model begins by constructing a convolutional neural network (CNN) using a face recognition dataset [90]. This dataset consists of 2562 facial images with a resolution of 160×160 pixels, categorized into 31 different classes. To facilitate federated machine learning, the data set is distributed evenly among the three clients manually. The CNN architecture used in the construction of face recognition model is detailed in Section 4.3.1. The output layer consists of 31 neurons that correspond to the 31 classes present in the initial dataset. To avoid overfitting and improve generalization, early stopping is used during model training. This technique involves monitoring the validation loss and stopping training if the loss fails to improve consecutively for five epochs within a specific round.

When real-time video data is streamed, the number of classes in the local dataset may change when a new subject's facial image is detected on a client machine. To accommodate these dataset changes, the output layer in the CNN is adjusted, and the neural network is trained by loading the pre-trained model obtained from the previous iteration in the federated learning process. This allows the model to adapt and incorporate the new class information while leveraging the knowledge learned from previous iterations. Consider a scenario to better

understand the model's adoptability when an image from a new class is received. Assume that a person's photo (from a new class not part of the existing dataset) is captured in one of the rooms in the smart home. This would create new class in the dataset, and further update the global model weights in the subsequent training process. As the other clients also see this updated weights, reflecting the new class, the new person could be recognized in other rooms as well, with proper training in subsequent rounds using images from this new class.

5.3.3 Models Training for the Prediction of Ambient Parameters

The ambient sensor data is utilized to train linear regression model and logistic regression model to predict maximum temperature and maximum humidity, respectively (as explained in Section 4.3.2). The models are trained using local datasets collected from each client in a federated fashion. The performance of the developed ML models is evaluated using the Kaggle dataset [91]. This dataset is divided into three equal-size groups, with each group loaded into a different client for training the models. The attributes such as temperature, humidity, pressure, precipitation, cloud cover, and wind speed are utilized in the construction of machine learning models to predict maximum temperature and maximum humidity. For preprocessing the data and training the models, the Pandas and Scikit-learn packages are employed.

5.4 Results and Discussion

To evaluate the efficiency of load balancing in distributing the streaming data among the 3 edge nodes, the load metric of each node in the cluster is measured. The load metric of node 'i' is defined as the weighted sum of its CPU utilization percentage and RAM utilization percentage divided by 100, with equal weights given to each criteria, as given in equation 5.1:

$$load\_metric(i) = (0.5 * CPU\_Util\_Perc(i) + 0.5 * Mem\_Util\_Perc(i))/100 \quad (5.1)$$

During the experimentation, a video stream of one-minute duration is continuously processed, whenever an event is triggered by PIR Sensor. Each frame of the video stream is written into the appropriate partition of the Kafka topic in the source. The load metric is calculated for each node while the load balancing

Table 5.1: Load metric value of various nodes in the cluster

Streaming data generation	Load metric value with load balance				Load metric value without load balance			
	Node1	Node2	Node3	Standard Deviation	Node1	Node2	Node3	Standard Deviation
At node 1 (Scenario 1)	0.62	0.54	0.58	0.04	0.72	0.56	0.48	0.12
At node 1, 2 (Scenario 2)	0.68	0.68	0.64	0.02	0.74	0.76	0.52	0.13
At node 1, 2, 3 (Scenario 3)	0.74	0.73	0.72	0.01	0.72	0.74	0.72	0.01

module is enabled. The average load across all the nodes is calculated, and the load deviation is analyzed by calculating the standard deviation of the load metric across all nodes in various scenarios:

Scenario 1: The video streaming data is generated in only one node, leaving the remaining two nodes without generation of video streaming data, with all the three nodes running local federated machine learning models, program that collects ambient parameters every one minute.

Scenario 2: The video streaming data is generated by two nodes, leaving one node without generation of video streaming data, with all the three nodes running local federated machine learning models, program that collects ambient parameters every one minute.

Scenario 3: The video streaming data is generated by all the three nodes, with all the three nodes running local federated machine learning models, program that collects ambient parameters every minute.

The results of Table 5.1 demonstrate the effectiveness of the load balancing module in distributing the data stream among the edge nodes, ensuring an even distribution of the workload, preventing any single node from being overwhelmed. With load balancing enabled, the standard deviation of load metric values of several nodes is less compared to load balancing disabled, indicating a relatively balanced distribution of the streaming data among the nodes. In scenario 3, the standard deviation values are same with and without using the data load distribution, since all nodes are receiving the streaming data, which does not initiate the streaming data distribution between the nodes in the cluster.

In a realistic IoT environment, the generation of event trigger-based video streaming data is nonuniform across several nodes, and there by the resource uti-

lization of various nodes is nonuniform. When a node is continuously operating at its maximum capacity for an extended period, it consumes more energy and generates more heat. This can result in increased power consumption and potentially cause thermal issues, leading to performance degradation or even hardware failures. Even distribution of streaming data among various nodes in the cluster can have a positive impact on energy consumption by preventing any single node from being completely drained. The resource-constrained edge devices would benefit from the developed system while processing the streaming data.

5.5 Summary

In this chapter, load balance-aware federated learning was introduced to handle heterogeneous IoT streaming data in a fog-enabled smart home environment. The research work introduced an innovative approach for collecting and processing data from vision and digital ambient sensors across a range of edge nodes, guided by a load metric. This load metric, derived from memory and processor utilization, plays a pivotal role in distributing the IoT data load across multiple edge nodes. The system ensures efficient resource allocation and load balancing within the edge cluster environment, effectively preventing overload or underload scenarios on specific nodes. This, in turn, has implications on the energy utilization of IoT devices. Empirical results demonstrate 0.10 enhancement in the standard deviation of load metric values (on a sclae of 0-1) across nodes when load balancing is applied. The system is ideally suited for managing IoT streaming data in resource-constrained settings.

Chapter 6

Capability-aware Federated Average Algorithm for Processing IoT Data

This chapter presents the research work related to Capability-aware Federated Learning that considers the capabilities of the clients involved during model training.

6.1 Introduction

The federated learning model training discussed in Chapter 4 utilized the default federated averaging algorithm on the server to aggregate parameters received from participating clients. This algorithm calculates a weighted average of parameters based on the number of examples used during model training locally by each client. However, the default federated averaging algorithm does not account for the heterogeneous capabilities of the clients during the aggregation process. The assignment of greater weights to devices with higher capabilities during the aggregation process enables more adaptive parameter aggregation. To address this limitation, we proposed a novel Capability-Aware Federated Averaging (CAFedAvg) algorithm.

Furthermore, mobile-aware Neural Architecture Search (NAS) [100] [101] is explored to build machine learning models suitable for edge computing devices. The goal of mobile-aware NAS is to create efficient and lightweight models that

can run effectively on mobile and IoT devices at the network edge. This work tries to leverage neural architecture search techniques, which automatically discover model architectures that strike a balance between accuracy and resource efficiency, making them well-suited for edge computing applications.

6.2 System Description

The underlying structure of the proposed system is similar to the structure presented in Section 5.2 (depicted in Figure 5.1). The fog node acts as a server that aggregates the local model weights received from each client in the FL application. Each node is integrated with a vision sensor, a PIR sensor, and a DHT sensor. The PIR sensor activates the vision sensor to capture video when there is motion in any room. The edge nodes gather and reposit ambient parameters data and video data, which will serve as input for FL based model training with edge nodes acting as clients and fog node as the server.

The basic idea of the CAFedAvg algorithm is presented in Algorithm 6.1. The updates to the default FedAvg algorithm to obtain the CAFedAvg algorithm are indicated using comment lines in the Algorithm 6.1. The algorithm takes the parameters *server_round*, *results*, *failures*, and *client_properties* as input, and produces output parameters *parameters_aggregated*, and *metrics_aggregated*. It begins by calculating the total RAM and CPU cores available across all clients. This calculation is performed by iterating through the input parameter *client_properties* (a dictionary of client properties).

For each client, the algorithm calculates the *capability score* based on the relative share of RAM and CPU cores that the client contributes to the total pool. This score represents the client's contribution to the federated learning process. The score is computed as a weighted average of RAM and CPU core contributions. The algorithm considers the *capability score* of each client during the parameter aggregation process. If custom metrics are provided (*fit_metrics_aggregation_fn* is not empty), the algorithm also considers client capabilities during the aggregation of these metrics.

In the CAFedAvg algorithm, devices with higher capabilities or resources are assigned higher weights or more significant roles during the aggregation process. Additionally, we take into account the number of examples used by each client during the parameter aggregation process. This means that devices with higher capabilities not only contribute more to the aggregation based on their resources,

but also consider local dataset size. Consequently, their model updates exert a stronger influence on the final aggregated model compared to devices with lower capabilities. By incorporating this capability-aware approach, the CAFedAvg algorithm enables more adaptive parameter aggregation, improving the overall performance and effectiveness of federated learning.

6.3 Implementation Details

6.3.1 CAFedAvg Algorithm

The experimental setup used for the experimentation consisted of heterogeneous computing devices and sensor units. The same experimental setup, which is presented in Section 5.3 (depicted in FIgure 5.4), is utilized for the capability-aware FL based model training. The CAFedAvg algorithm (Algorithm 6.1) is implemented by extending the Server Aggregator class in the Flower framework. Each client participating in the model training shares its capability information (number of processor cores and RAM size) with the server. This information, in addition to the number of samples used by each client for it's local model training, is used by the server/aggregator to adjust the weight values for the weighted average of the parameter aggregation.

6.3.2 Mobile-aware Neural Architecture Search for Face Recognition

Given the computational limitations, the neural architecture search algorithm (NAS) is developed to create a lightweight model specifically for face recognition. It optimizes hyper-parameters and adjusts the search space to align with the computational capabilities of edge devices [100]. The Python autokeras-1.0.20 module [102] is used in the NAS design, with a maximum of 10 trials conducted. To enhance model building, a Bayesian tuner is employed as an optimization function.

The algorithm performs a neural architecture search with the objective of finding an improved model using validation accuracy as the guiding metric. The batch size is set to 16, in view of resource limitations, and the model is trained for 10 epochs. To prevent overfitting and optimize training, early stopping is implemented with a patience value of 5. This means that if there is no improvement

Algorithm 6.1: CAFedAvg – Capability-aware fed average algorithm

Input:
server_round: The current round of the federated learning process on the server.
results: List of tuples containing the FitRes results from participating clients.
failures: List of tuples containing failed results or exceptions.
client_properties: Dictionary of client properties including processor cores and RAM size, with ip address of the client as key.
Output:
parameters_aggregated: Aggregated model parameters.
metrics_aggregated: Aggregated custom metrics.

```
if results is empty:
    return None, {}

ram_total ← 0
cores_total ← 0
//Obtaining the capabilities of each client by using the client IP as key
for each key in client_properties do
    ram_total ← ram_total + client_properties[key]['ram']
    cores_total ← cores_total + client_properties[key]['cores']

capability_score ← empty dictionary
//Calculating the capability score of each client
for each key in client_properties do:
    client_capacity ← client_properties[key]
    ram_score ← client_capacity['ram'] / ram_total
    cores_score ← client_capacity['cores'] / cores_total
    score ← 0.5 * ram_score + 0.5 * cores_score
    capability_score[key] ← score

adjusted_weights_results ← empty list
//Considering the client capabilities during parameter aggregation
for each (cp, fit_res) in results do:
    parameters ← parameters_to_ndarrays(fit_res.parameters)
    weight ← fit_res.num_examples * capability_score[cp.cid]
    adjusted_weights_results.append((parameters, weight))

parameters_aggregated ← ndarrays_to_parameters(aggregate(adjusted_weights_results))
metrics_aggregated ← empty dictionary
if fit_metrics_aggregation_fn is provided:
    fit_metrics ← empty list
    //Considering the client capabilities during metric aggregation
    for each (cp, res) in results do:
        weight ← res.num_examples * capability_score[cp.cid]
        metrics ← res.metrics
        fit_metrics.append((weight, metrics))
    metrics_aggregated ← fit_metrics_aggregation_fn(fit_metrics)

return parameters_aggregated, metrics_aggregated
```

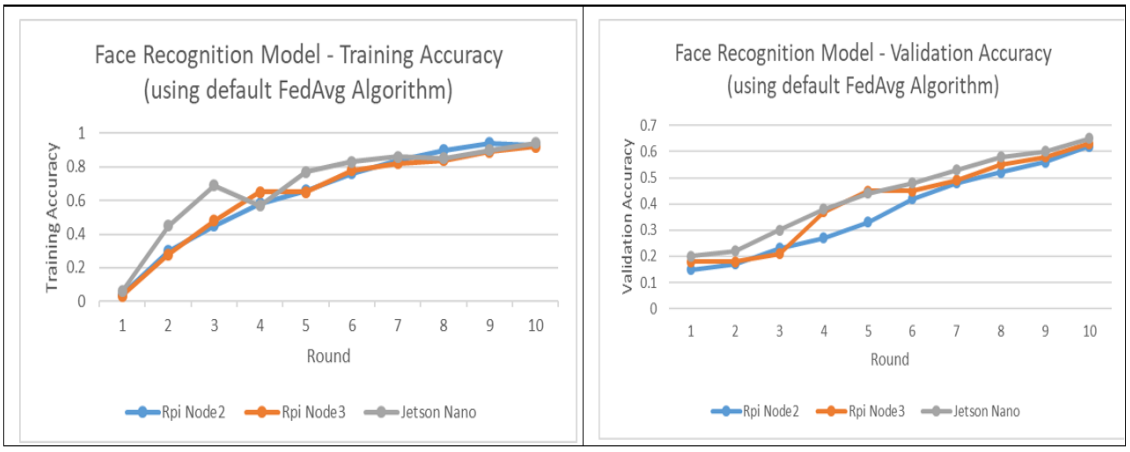


Figure 6.1: FL based facial recognition ML model training and validation accuracy using default FedAvg algorithm (#rounds=10, #epochs=10)

Figure 6.2: FL based facial recognition ML model training and validation accuracy using CAFedAvg algorithm (#rounds=10, #epochs=10)

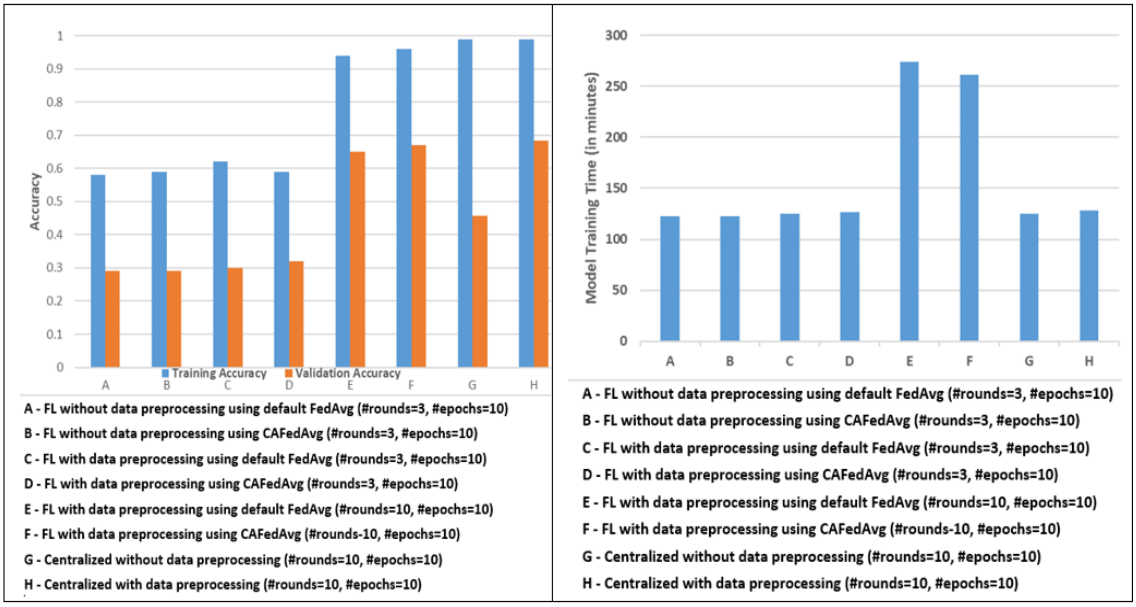
Figure 6.3: Facial recognition ML model training and validation accuracy using default FedAvg algorithm, CAFedAvg algorithm and Centralized training (#rounds=10, #epochs=10)

in validation accuracy for five consecutive epochs, the current machine learning model training is halted. To leverage the multicore capabilities of the devices, the Python multiprocessing module is utilized during neural architecture search, provided that such capabilities are available. This allows for more efficient utilization of resources and expedite search processes.

6.4 Results and Discussion

6.4.1 Performance Analysis of Model Training using CAFedAvg Algorithm

The performance of the ML models developed is evaluated by comparing three scenarios: utilizing the default federated average algorithm, employing the capability-aware federated average algorithm and running model training centralized setup. Experimentation is carried out by changing hyperparameter values in each scenario. The performance of the FL algorithms, including the default federated average algorithm and the capability-aware federated average algorithm, is compared with model training in a centralized fashion. For centralized ML model training, Intel Core i5-7500 CPU (3.40GHz clock speed with quad-core processor and 8GB RAM) is used.



(a) **(b)**

Figure 6.4: (a) Facial recognition ML model training – training accuracy vs. validation accuracy (b) Facial recognition model training time in various scenarios

The Jetson Nano node demonstrates better model training and validation accuracy compared to the Raspberry Pi nodes, utilizing both the default federated average and the proposed capability-aware federated average algorithm (as depicted in Figure 6.1 and Figure 6.2). The proposed capability-aware federated average algorithm improves the accuracy of the facial recognition model in multiple rounds. For the facial recognition model, the overall training accuracy values achieved using the default federated learning algorithm, CAFedAvg algorithm, and Centralized algorithm are 0.94, 0.96, and 0.99, respectively. Similarly, the validation accuracy values for these algorithms are 0.65, 0.67, and 0.68, respectively.

Comparatively, the accuracy values of the Centralized model training are significantly higher than those achieved through the federated learning approach (as illustrated in Figure 6.3) in the initial round, primarily due to training the model on the entire dataset. However, the accuracy values of the FL-based model training (using both the default federated average and CAFedAvg algorithms) gradually improve with each round and approach the accuracy values of the centralized approach by the end of the tenth round. In Figure 6.4(a), the combined training

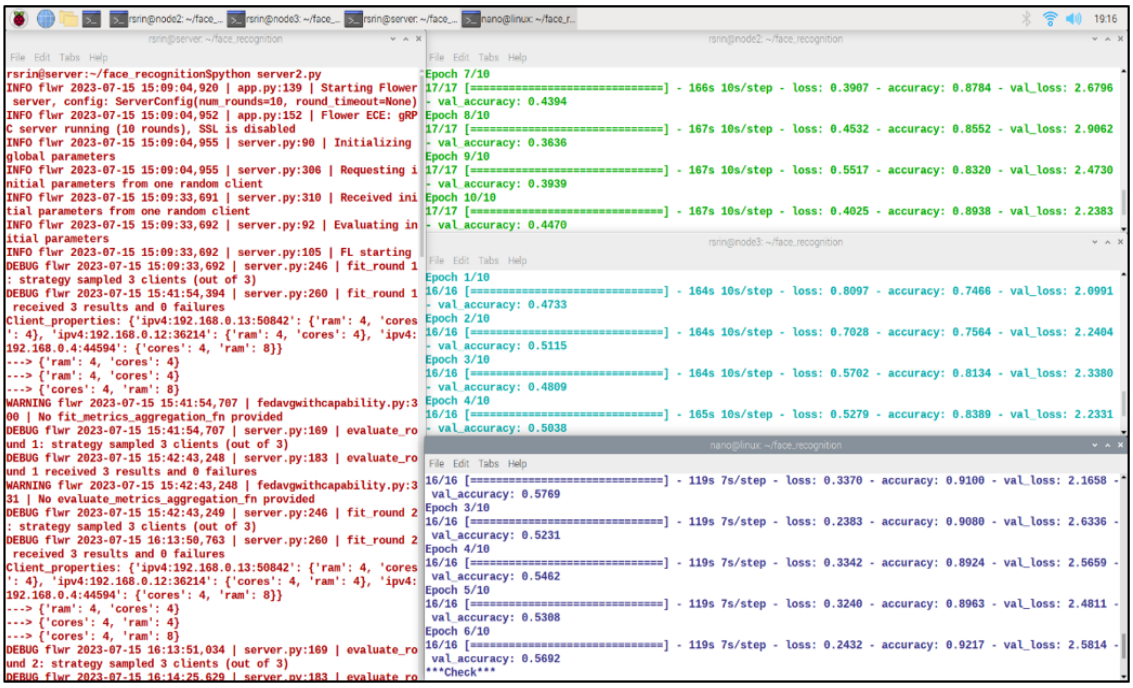
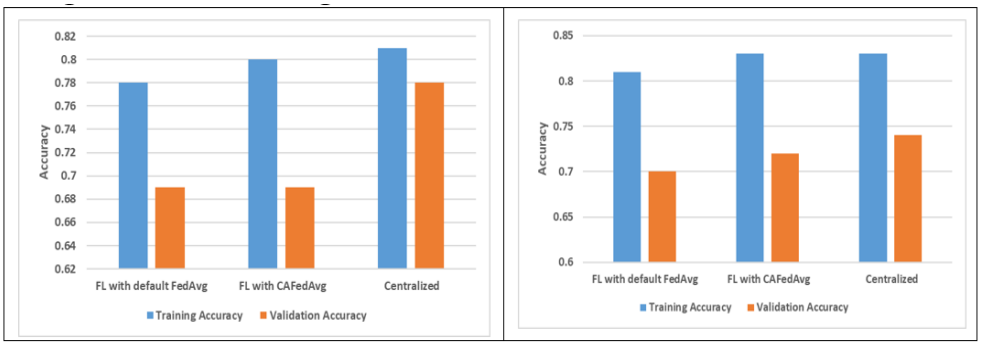


Figure 6.5: Facial recognition model training using CAFedAvg based FL approach – Output screenshot

accuracies, validation accuracies of multiple face recognition ML model training experiments are presented for both federated, centralized scenarios.

These experiments emphasize the significance of preprocessing the data while training the model, particularly in the context of the IoT environment, where not every client can provide substantial amount of data for training. Effective data processing and normalization techniques play a vital role in enhancing the accuracy of the model. Figure 6.4(b) shows the facial recognition model training time in different scenarios, incorporating early stopping with patience parameter set to 5 epochs. The screenshot depicting the output from trial run of FL based facial recognition model training using CAFedAvg algorithm is given in Figure 6.5.

The accuracy values of the maximum temperature and humidity prediction model training are presented in Figure 6.6. The training is carried out in the following scenarios using 10 rounds: (i) FL with default FedAvg algorithm (ii) FL with CAFedAvg algorithm (iii) Centralized model training.



(a) (b)

Figure 6.6: (a) Accuracy values of Temperature prediction ML model training (b) Accuracy values of Humidity prediction ML model training.

```
Search: Running Trial #3

Value             |Best Value So Far |Hyperparameter
False             |False             |image_block_1/normalize
False             |False             |image_block_1/augment
xception          |resnet            |image_block_1/block_type
global_avg        |flatten           |classification_head_1/spatial_reduction_1/reduction_type
0.25              |0.5               |classification_head_1/dropout
sgd               |adam              |optimizer
0.001             |0.0001            |learning_rate
False             |None              |image_block_1/xception_block_1/pretrained
False             |None              |image_block_1/xception_block_1/imagenet_size

Epoch 1/10
1/1 - 50s - loss: 3.4790 - accuracy: 0.0000e+00 - val_loss: 3.4340 - val_accuracy: 0.0000e+00 - 50s/epoch - 50s/step
Epoch 2/10
1/1 - 28s - loss: 3.4581 - accuracy: 0.0625 - val_loss: 3.4339 - val_accuracy: 0.1250 - 28s/epoch - 28s/step
Epoch 3/10
1/1 - 20s - loss: 3.3577 - accuracy: 0.0625 - val_loss: 3.4339 - val_accuracy: 0.1250 - 20s/epoch - 20s/step
Epoch 4/10
1/1 - 21s - loss: 3.3359 - accuracy: 0.0625 - val_loss: 3.4339 - val_accuracy: 0.1250 - 21s/epoch - 21s/step
Epoch 5/10
1/1 - 20s - loss: 3.3103 - accuracy: 0.0625 - val_loss: 3.4339 - val_accuracy: 0.1250 - 20s/epoch - 20s/step
Epoch 6/10
1/1 - 22s - loss: 3.2898 - accuracy: 0.0000e+00 - val_loss: 3.4338 - val_accuracy: 0.1250 - 22s/epoch - 22s/step
Epoch 7/10
1/1 - 22s - loss: 3.2228 - accuracy: 0.1250 - val_loss: 3.4338 - val_accuracy: 0.1250 - 22s/epoch - 22s/step
Epoch 8/10
1/1 - 20s - loss: 3.1977 - accuracy: 0.1250 - val_loss: 3.4338 - val_accuracy: 0.1250 - 20s/epoch - 20s/step
Epoch 9/10
1/1 - 20s - loss: 3.1029 - accuracy: 0.3125 - val_loss: 3.4338 - val_accuracy: 0.1250 - 20s/epoch - 20s/step
Epoch 10/10
1/1 - 20s - loss: 3.0941 - accuracy: 0.3750 - val_loss: 3.4337 - val_accuracy: 0.0000e+00 - 20s/epoch - 20s/step

Trial 3 Complete [00h 04m 55s]
val_accuracy: 0.125

Best val_accuracy So Far: 0.125
Total elapsed time: 00h 12m 03s
```

Figure 6.7: Output screenshot depicting the construction of the facial recognition model using NAS.

6.4.2 Analysis of Mobile-aware NAS Model Training

The construction of a lightweight model for face recognition using NAS is carried out through experimentation on edge devices. The experimentation consisted of 10 trials, considering different model architectures. Out of these trials, the best training accuracy value achieved is 0.24, and the process took a total of 25 minutes. To further enhance the model's accuracy, additional trials could be performed through NAS experimentation. As the edge devices are resource constrained, the experimentation could not be conducted with more number of trials due to memory limitations. Figure 6.7 presents a screenshot of the output depicting the construction of the facial recognition model using NAS.

6.5 Summary

This chapter presented a novel Capability-aware federated averaging algorithm the considers the capabilities of the client devices participating in the federated learning process. The performance of the proposed algorithm is compared with that of the default federated averaging algorithm. Federated learning is employed to train the ML models from the data collected by each edge client using the CAFedAvg algorithm and default FedAvg algorithm. The results of model training in federated learning setting and centralized setting are presented. Also, the experimental findings related to mobile-aware NAS model training on edge devices is presented.

Chapter 7

Analysis of Federated Learning with Distributed IoT Data for Model Parameter Computation and Storage

This chapter presents the analysis of federated learning for the computation and storage of model parameters. The performance of the proposed communication-efficient algorithms and computing techniques is evaluated in terms of accuracy, speed, and scalability.

7.1 Performance Analysis of Federated Learning for ML Model Training

7.1.1 Face Recognition Model Training

Data preprocessing and normalization techniques play a vital role during model training in IoT environment, as mentioned in Chapter 4. In the IoT environment, not all devices generated the same amount of data, hence model updates from certain clients may not be as informative as the updates from other clients, leading to inefficiencies in FL process. Data augmentation and normalization techniques help to address this challenge to certain extent. Figure 7.1 depicts the federated

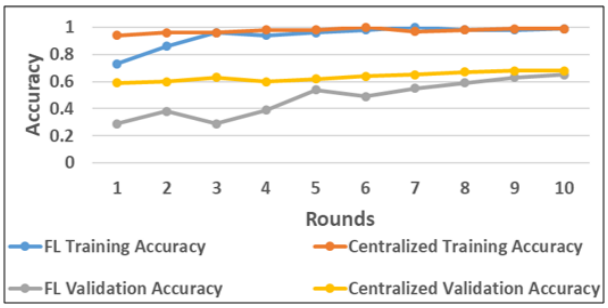


Figure 7.1: FL vs. Centralized model training accuracy

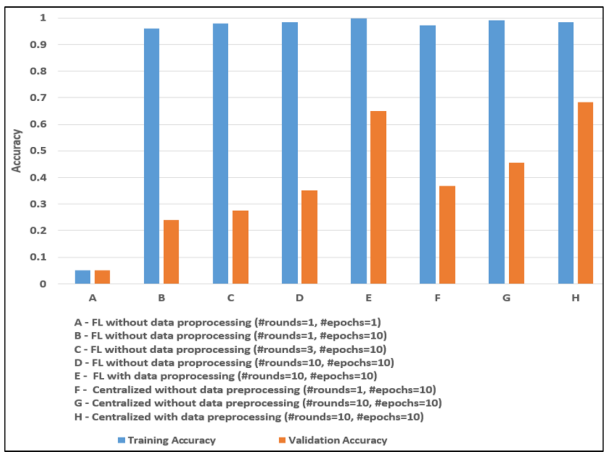


Figure 7.2: Face recognition model training – training accuracy vs. validation accuracy

and centralized model training accuracy values for 10 rounds and 10 epochs in each round (with data pre-processing and normalization).

For centralized execution of the model training on a machine with Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz four-core processor (with Intel HD 630 integrated GPU) having 8GB RAM), there is considerable improvement in the model accuracy values for each round in the case of FL, whereas there is no considerable improvement in further rounds in the case of centralized training, as the model is already trained using the whole set of images in the dataset by the end of round-1. Figure 7.2 presents the overall training and validation accuracies of several experiments of face recognition model training in federated and centralized setups for different hyper parameter values. The experiments highlight the fact that data preprocessing and normalization are very important during model training in the IoT environment.

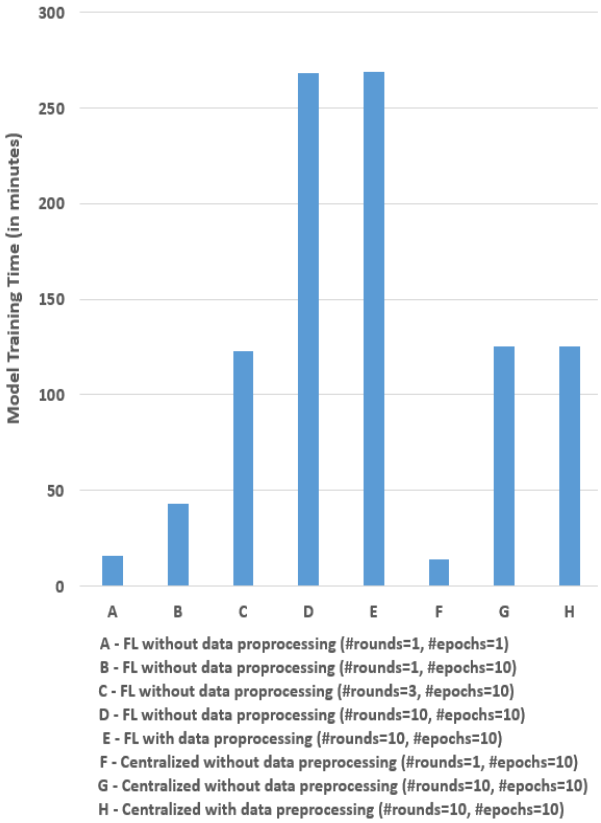


Figure 7.3: Time taken for face recognition model training in various scenarios

Proper data processing and normalization techniques help improve model accuracy. The training and validation accuracy values of the FL face recognition model (with data preprocessing and normalization) with 10 rounds, and 10 epochs in each round are 0.999 and 0.65 respectively. These values are closer to the centralized training values of 0.999 and 0.68 respectively. The federated, centralized model training (without early stopping) has taken 269 minutes and 125 minutes respectively (#rounds=10, #epochs=10) for the dataset size of 2562 images. The time taken for face recognition model training in various scenarios is given in Figure 7.3.

With early stopping, the FL model training (#rounds=10, #epochs=10) has taken 231 minutes, which is a significant improvement in training time, for the same training and validation accuracy values. During the FL experimentation with 10 rounds, the early stopping of model training occurred in edge client 2 and edge client 3 in round 7. The face recognition model is tested on edge

Table 7.1: Data Transfer between Clients and the Server

Device	Data transfer per round (in MB)	
	Sent	Received
Client 1	13.14	13.14
Client 2	13.14	13.14
Client 3	13.14	13.14
Client 4	13.14	13.14
Server	52.56	52.56

node 4 (Test node) using out-of-domain face images (heterogeneous facial images in different context which are not part of the dataset), and obtained a testing accuracy value of 64%.

The layer-wise number of parameters in the constructed face recognition model is shown in Figure 7.4. There are a total of 3,445,599 parameters in the CNN constructed for face recognition. Keras uses 32-bit floating-point precision for weights (4 bytes) by default. Therefore, each exchange of parameters between the client and server requires data transmission of 3,445,599 x 4 = 13,782,396 bytes (i.e. 13.14 MB), excluding the control data. The amount of data transfer for parameter exchange in face recognition model training per round (excluding control data) is presented in Table 7.1. Federated learning coupled with early stopping saves the parameters exchange between the clients and server, thereby reducing the communication overhead. In our experimentation, early stopping occurred in Client 2 and Client 4 in round 7, thereby avoiding the parameter exchange (of size 13.14 MB in both ways) for those clients for 3 rounds.

Analysis of network-related parameters such as bandwidth consumption, packets per second (PPS), and packet loss during federated learning is done using the Netdata tool [103], which is shown in Figure 7.5 for the server involved in federated learning. As is evident, the bandwidth consumption and packet transfer are more at the end of each round due to the exchange of model weights between clients and server.

7.1.2 Temperature Prediction, Humidity Prediction Models Training

The sizes of the temperature prediction model and humidity prediction model are 462 bytes, 474 bytes respectively, which are very small compared to the deep

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 156, 156, 32)      2432

 max_pooling2d (MaxPooling2D  (None, 78, 78, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 74, 74, 64)        51264

 max_pooling2d_1 (MaxPooling  (None, 37, 37, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 33, 33, 128)       204928

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 12, 12, 256)       819456

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 256)        0
 2D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 256)               2359552

 dense_1 (Dense)             (None, 31)                7967

=================================================================
Total params: 3,445,599
Trainable params: 3,445,599
Non-trainable params: 0
```

Figure 7.4: Layer-wise parameters in the face recognition CNN

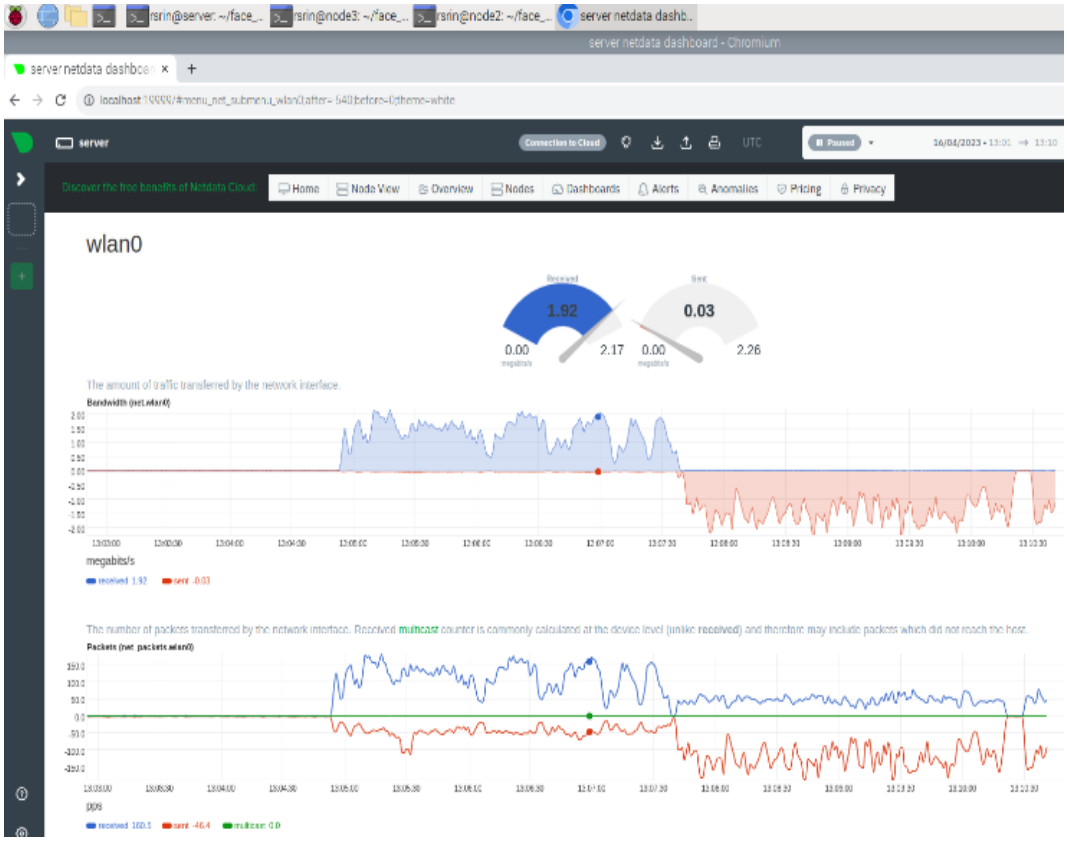


Figure 7.5: Analysis of network-related parameters during model training

learning-based face recognition model. These require very fewer amounts of data transfer between clients and the server for parameter exchange at the end of each round. The temperature prediction model training has taken 400 milliseconds and 1800 milliseconds in FL with one round and 10 rounds, respectively. The humidity prediction model training has taken 500 milliseconds and 2200 milliseconds in FL with one round and 10 rounds, respectively.

7.2 Scalability Analysis of Load Balance-Aware Federated Learning Algorithm

Evenly distributing the streaming data among various nodes in the cluster can have a positive impact on energy consumption by preventing any single node from being completely drained. The resource-constrained edge devices would benefit from the developed system while processing the streaming data. The maximum number of clients that can participate in our proposed load balance-aware federated learning process is limited by the available resources in each client. The upper bound for number of nodes in the Cluster (to take part in the FL process) is given by Equation 7.1:

$$max\_nodes = min\{min\{\lfloor\frac{d_i}{k \times s}\rfloor\}, min\{\lfloor\frac{m_i}{k \times b}\rfloor\}, \lfloor\frac{4000}{k}\rfloor\} \ \ where \ \ 1 \leq i \leq n \quad (7.1)$$

Here,
Number of nodes in the cluster = Number of partitions per topic in each node = n
Number of topics in each node = k
Segment size per partition (in bytes) = s (default – 1GB)
Buffer size per partition (in bytes) = b (default – 32 MB)
Disk size (in bytes) for node (i) = d_i
Memory size (in byes) for node (i) = m_i
In our proposed load balance-aware FL algorithm, we dedicate one partition per topic, for all clients in the cluster at each client machine. The expression $\lfloor\frac{d_i}{k \times s}\rfloor$ gives the number of partitions that can be allocated per topic in client node (i) without exceeding its disk space. Similarly, the expression $\lfloor\frac{m_i}{k \times b}\rfloor$ gives the number of partitions that can be allocated per topic in client node (i) without exceeding its memory space. The maximum number of partitions supported by the Kafka broker is 4000. As the number of clients is equal to the number of partitions per

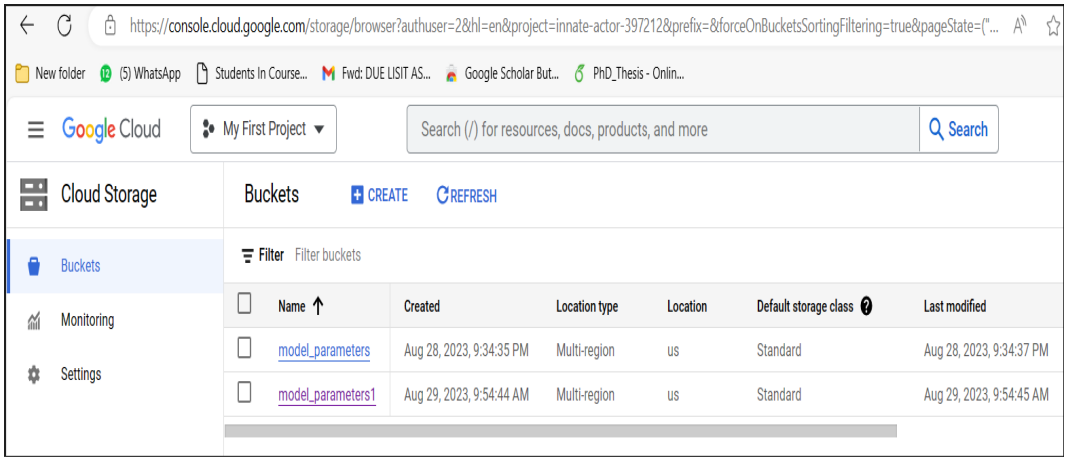


Figure 7.6: Global model parameter values stored in GCP Bucket

topic in the proposed algorithm, the upper bound for the maximum number of clients that can take part in the FL process can be obtained using the equation 7.1.

7.3 Analysis of Model Parameter Storage in the Cloud

Google Cloud Platform (GCP) [104] is used to store the obtained global ML model parameter values at the end of each round. This is useful as these parameters might be necessary for integrating analogous applications operating in diverse locations. The convolutional neural network (CNN) designed for facial recognition comprises a total of 3,445,599 parameters distributed over 31 classes of facial images. By default, Keras employs 32-bit floating-point precision for weights (4 bytes). Consequently, every parameter exchange between the client and the server requires a data transfer of 3,445,599 x 4 = 13,782,396 bytes (equivalent to 13.14 MB), excluding control data. As clients contribute these parameters, the server aggregates them, resulting in global model parameters of size 13.14 MB. These parameter values are stored in the form of a binary file and then uploaded to a cloud bucket for further reference (shown in Figure 7.6). The process of transferring the global model parameters from the FL server (fog node) to the Cloud takes an average duration of 3.5 seconds.

Transferring the dataset of 2562 facial images to the cloud for model training would consume approximately 187.5 MB of bandwidth, in addition to the privacy issues. Transferring just the model parameters would consume 13.14 MB of bandwidth. The impact on bandwidth consumption is significant when transferring images, which is approximately 14 times larger compared to transferring only the model parameters. This difference in bandwidth consumption highlights one of the advantages of federated learning: By transmitting model parameters instead of raw data, we can significantly reduce the amount of data that needs to be transferred over the network. This is especially important when dealing with resource-constrained devices, limited network bandwidth, or privacy concerns, as it minimizes the data sent over the network while still enabling model updates and improvements.

The research work presented in this chapter is published in our following research publication:

S. R. Rudraraju, N. K. Suryadevara and A. Negi, "Heterogeneous Sensor Data Acquisition and Federated Learning for Resource Constrained IoT Devices—A Validation," in *IEEE Sensors Journal*, vol. 23, no. 15, pp. 17602-17610, 1 Aug.1, 2023, doi: 10.1109/JSEN.2023.3287580. Indexed in: Scopus, Web of Science (SCIE).
URL: https://ieeexplore.ieee.org/abstract/document/10161724

7.4 Summary

This chapter presented the analysis of federated learning for the computation and storage of model parameters. The performance metrics, accuracy and training time, related to face recognition model training and ambient parameter models training are presented. Empirical results demonstrate that the accuracy values of the FL based model training are close to the centralized training value with proper preprocessing and normalization techniques. Federated learning facilitates deep learning-based model training without centralizing datasets, hence it is ideal for IoT applications with decentralized datasets without compromising the privacy. Furthermore, the results suggest that the amount of data exchange between the clients and server is reduced substantially, as they don't exchange the raw data between them. Furthermore, this chapter provided the scalability analysis of the proposed load balance-aware federated learning mechanism in the fog computing framework.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this research, we have developed a fog computing-based federated learning framework to process heterogeneous streaming data in IoT environment. Our research experimentation and validation are focused towards deploying fog based federated computing environment for a smart home environment, but can be deployable in: Ambient assisted living environment, Healthcare and Banking environments.

Firstly, we have designed and implemented fog computing frameworks tailored to enhance the quality of life for smart home residents. These frameworks facilitate efficient data processing and analysis within the fog environment, leading to reduced latency and improved responsiveness. Additionally, we have introduced a novel framework for big data processing in fog clusters utilizing Apache Spark, a distributed data processing technology. This paves the way for more streamlined data processing within fog clusters, a crucial component of fog computing.

Moreover, we have contributed to the field of federated learning by developing a robust framework for processing heterogeneous streaming data in fog-centric IoT applications. Our framework not only addresses data heterogeneity challenges, but also incorporates a system for distributing streaming data load efficiently among the cluster nodes. This development ensures that resource-constrained IoT devices can participate effectively in the federated learning process.

Furthermore, we have introduced a novel capability-aware fed average algorithm for processing heterogeneous streaming data. The proposed algorithm has

been implemented and validated using heterogeneous devices within the IoT environment. It addresses challenges related to the impact of device heterogeneity on model training performance.

8.2 Directions for Future Research

Our research has laid a strong foundation for further exploration and development in the field of fog computing and federated learning. The following directions offer promising avenues for future research:

1. Parameter Aggregation Methods for Diverse Neural Architectures: As the use of diverse neural architecture search-based models becomes more prevalent in the IoT context, developing efficient parameter aggregation methods for these models is essential. Future research can focus on devising innovative techniques that allow for seamless integration of various neural architectures in Federated Learning settings.

2. Enhanced Security and Privacy Protocols: Security and privacy concerns are paramount in IoT and Federated Learning environments. Future research should explore advanced encryption and privacy-preserving techniques to protect sensitive IoT data during the federated learning process.

3. Applying the proposed algorithms in fog computing-based federated learning from the smart home context to other case studies, thereby broadening their scope and applicability.

In conclusion, our research has contributed to the development of federated learning framework that effectively processes heterogeneous IoT streaming data in fog environment. The identified future research directions promise to further enhance the efficiency, scalability, and applicability of these technologies, ultimately leading to more advanced and practical solutions in the IoT landscape.

List of Publications

Journal

- S. R. Rudraraju, N. K. Suryadevara and A. Negi, "Heterogeneous Sensor Data Acquisition and Federated Learning for Resource Constrained IoT Devices—A Validation," in *IEEE Sensors Journal*, vol. 23, no. 15, pp. 17602-17610, 1 Aug.1, 2023, doi: 10.1109/JSEN.2023.3287580. Indexed in: Scopus, Web of Science (SCIE).
  URL: https://ieeexplore.ieee.org/abstract/document/10161724

- **Capability-Aware Fed Average for Processing IoT Streaming**, Communicated to ACM Transactions on Internet of Things.

Conference Proceedings

1. S. R. Sahith, S. R. Rudraraju, A. Negi and N. K. Suryadevara, **Mesh WSN Data Aggregation and Face Identification in Fog Computing Framework**, *Proceedings of 13th International Conference on Sensing Technology, ICST 2019, Sydney, Australia, December 2-4, 2019, IEEE*, Pages: 1-6, 2019, doi: 10.1109/ICST46873.2019.9047708. Indexed in: DBLP, Scopus. Status: Accepted and Published
   URL: https://ieeexplore.ieee.org/abstract/document/9047708

2. S. R. Rudraraju, N. K. Suryadevara and A. Negi, **Face Recognition in the Fog Cluster Computing**, *Proceedings of International Conference on Signal Processing, Information, Communication & Systems (SPICSCON), Dhaka, Bangladesh, November 28-30, 2019, IEEE*, Pages: 45-48, 2019, doi:

10.1109/SPICSCON48833.2019.9065100. Indexed in: Scopus. Status: Accepted and Published
URL: https://ieeexplore.ieee.org/abstract/document/9065100

3. S. R. Rudraraju, N. K. Suryadevara and A. Negi, **Face Mask Detection at the Fog Computing Gateway**, *Proceedings of 15th Conference on Computer Science and Information Systems (FedCSIS), Sofia, Bulgaria, 2020, IEEE*, Pages: 521-524, 2020, doi: 10.15439/2020F143. Indexed in: DBLP, Scopus. Status: Accepted and Published
URL: https://ieeexplore.ieee.org/abstract/document/9222988

4. N. K. Suryadevara, A. Negi and S. R. Rudraraju **A Smart Home Assistive Living Framework Using Fog Computing for Audio and Lighting Stimulation**, *Proceedings of Advances in Decision Sciences, Image Processing, Security and Computer Vision: International Conference on Emerging Trends in Engineering (ICETE), Hyderabad, India 2019, Springer*, Pages: 366-375, doi: 10.1007/978-3-030-24322-747. Status: Accepted and Published
URL: https://link.springer.com/chapter/10.1007/978-3-030-24322-7$_4$7

Book Chapters

1. S. R. Rudraraju, N. K. Suryadevara and A. Negi, **Edge computing for visitor identification using eigenfaces in an assisted living environment**, Editors: Nagender Kumar Suryadevara, Subhas Chandra Mukhopadhyay, *Assistive Technology for the Elderly*, Academic Press, 2020, Pages 235-248, ISBN 9780128185469, Indexed in: Scopus, Status: Accepted and Published
URL: https://doi.org/10.1016/B978-0-12-818546-9.00008-7

2. S. R. Rudraraju, N. K. Suryadevara and A. Negi, **Fog computing framework for Big Data processing using cluster management in a resource-constraint environment**, Editors: Ravi Vadlamani, Aswani Kumar Cherukuri, *Handbook of Big Data Analytics: Methodologies*, IET, 2021, Pages 317-334, Indexed in: Scopus, Status: Accepted and Published
URL: https://digital-library.theiet.org/content/books/10.1049/pbpc037f$_c$h9

References

[1] S. Marusica J. Gubbi, R. Buyya and M. Palaniswami. **Internet of Things (IoT): A vision, architectural elements, and future directions**. *Future Generation Computer Systems*, **29**(7):1645–1660, 2013.

[2] K. H. K. Reddy D. S. Roy, R. K. Behera and R. Buyya. **A Context-Aware, Fog Enabled Scheme for Real-Time, Cross-Vertical IoT Applications**. *IEEE Internet of Things*, **6**(2):2400–2412, 2019.

[3] Yogesh L. Simmhan. **The diversity of IoT data and its potential value makes it important to understand its characteristics**. Semantic Scholar, 2019.

[4] M. Hayajneh M. A. Elkheir and N. A. Ali. **Data Management for the Internet of Things: Design Primitives and Solution**. *MDPI Sensors*, **13**(11):15582–15612, 2013.

[5] L. Jiang H. Cai, B. Xu and A. V. Vasilakos. **IoT-Based Big Data Storage Systems in Cloud Computing: Perspectives and Challenges**. *IEEE Internet of Things*, **4**(1):75–87, Feb 2017.

[6] Q. Zhang Y. Li W. Shi, J. Cao and L. Xu. **Edge Computing: Vision and Challenges**. *IEEE Internet of Things Journal*, **3**(5):637–646, 2016.

[7] R. Abulibdeh A .Al-Qamash, I. Soliman and M. Saleh. **Cloud, Fog, and Edge Computing: A Software Engineering Perspective**. In *2018 International Conference on Computer and Applications (ICCA)*, pages 276–284. IEEE, 2018.

[8] B. Javier L. Ramon P. Jorge, D. Jessica and G. Angel. **Edge Computing**. *Computing, Springer*, **104**:2711–2747, 2022.

[9] M. D. Francesco G. Premsankar and T. Taleb. **Edge Computing for the Internet of Things: A Case Study**. *IEEE Internet of Things*, **5**(2), 2018.

[10] M. Abdallah J. Schneider A. Alwarafy, K. A. Thelaya and M. Hamdi. **A Survey on Security and Privacy Issues in Edge-Computing-Assisted Internet of Things**. *IEEE Internet of Things Journal*, **8**(6):4004–4022, 2021.

[11] S. Zeadally M. Aazam and K. A. Harrass. **Fog Computing Architecture, Evaluation and Future Research Directions**. *IEEE Communications Magazine*, **56**:46–52, 2018.

[12] H. Sabireen H and V. Neelanarayanan. **A Review on Fog Computing: Architecture, Fog with IoT, Algorithms and Research Challenges**. *ICT Express*, **7**(2):162–176, 2021.

[13] H. Moungla M. A. Cherif H. Afifi M. Laroui, B. Nour and M. Guizani. **Edge and fog computing for IoT: A survey on current research activities future directions**. *Computer Communications*, **180**:210–231, 2021.

[14] L. Yang Y. Sahni, J. Cao and S. Wang. **Distributed resource scheduling in edge computing: Problems, solutions, and opportunities**. *Computer Networks, Elsevier*, **219**:109430, 2022.

[15] D. Alistarh. **A Brief Tutorial on Distributed and Concurrent Machine Learning**. *In Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 487–488, 2018.

[16] Aluizio F. Rocha Neto, Flavia C. Delicato, Thais V. Batista, and Paulo F. Pires. *Distributed Machine Learning for IoT Applications in the Fog, IEEE*, pages 309–345. 2020.

[17] Fabrizio Marozzo, Alessio Orsino, Domenico Talia, and Paolo Trunfio. **Edge Computing Solutions for Distributed Machine**

Learning. In *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 1–8, 2022.

[18] P. Xie E.P.Xing, Q. Ho and D. Wei. **Strategies and Principles of Distributed Machine Learning on Big Data**. *Engineering, Elsevier ScienceDirect*, **2**(2):179–195, 2016.

[19] H. Ould-Slimane A. Mourad C. Talhi S. A. Rahman, H. Tout and M. Guizani. **A Survey on Federated Learning: The Journey from Centralized to Distributed On-Site Learning and Beyond**. *IEEE Internet of Things Journal*, **8**(7):5476–5497, 2021.

[20] Safa Ben Atitallah, Maha Driss, and Henda Ben Ghezala. **FedMicro-IDA: A federated learning and microservices-based framework for IoT data analytics**. *Elsevier Internet of Things*, **23**:100845, 2023.

[21] S. Wang J. Li A. Imteaj, U. Thakker and M. H. Amini. **Federated Learning for Resource-Constrained IoT Devices: Panoramas and State-of-the-art, Book chapter Federated and Transfer Learning, Springer**. **27**, 2023.

[22] P. N. Pathirana A. Seneviratne J. Li D. C. Nguyen, M. Ding and H. Vincent Poor. **Federated Learning for Internet of Things: A Comprehensive Survey**. *IEEE Communications Surveys Tutorials*, **23**(3):1622–1658, 2021.

[23] A. Costa M. M. Gomes, R. Righi and D. Griebler. **Simplifying IoT data stream enrichment and analytics in the edge**. *Computers and Electrical Engineering, Elsevier*, **92**:107110, 2021.

[24] D. Ramage S. Hampson H. B. McMahan, E. Moore and B. A. Arcas. **Communication-Efficient Learning of Deep Networks from Decentralized Data**. Proceedings of Machine Learning Research, 2023.

[25] Y. Lan Z. Cui J. Cai J. Wen, Z. Zhang and W. Zhang. **A survey on federated learning: challenges and applications**. *International Journal of Machine Learning and Cybernetics*, **14**:513–535, 2023.

[26] E. Simmon. **The NIST Definition of Cloud Computing**. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf. Online; accessed 16 Dec 2018.

[27] **OpenFog Reference Architecture for Fog Computing**. https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf. Online; accessed 26 Dec 2018.

[28] A. Kamsin S. Soomro M. R. Belgaum M. H. Miraz Z. Alansari, N. B. Anuar and J. Alshaer. **Challenges of Internet of Things and Big Data Integration**. arXiv:1806.08953, 2018.

[29] S. Chhabra and D. Singh. **Data fusion and Data aggregation/ summarization techniques in WSNs: A review**. *International Journal of Computer Applications*, **121**(19), 2015.

[30] Gaurav Mohindru, Koushik Mondal, and Haider Banka. **Different hybrid machine intelligence techniques for handling IoT-based imbalanced data**. *CAAI Transactions on Intelligence Technology, IET*, **6**(4):405–416.

[31] D. Gopinathan A. Nayyar R. Krishnamurthi, A. Kumar and B. Qureshi. **An Overview of IoT Sensor Data Processing, Fusion, and Analysis Techniques**. *MDPI Sensors*, **20**(21), 2020.

[32] H. E. Khoukhi M. Bakhouya V. D. Florio D. E. Ouadghiri S. Latre Y. N. Malek, A. Kharbouch and C. Blondia. **On the use of IoT and Big Data Technologies for Real-time Monitoring and Data Processing**. *Procedia Computer Science*, **113**:429–434, 2017.

[33] J. L. Chen K. D. Chang, C. Y. Chen and H. C. Chao. **Internet of Things and Cloud Computing for Future Internet**. In T. H. Kim R. S. Chang and S. L. Peng, editors, *Security-Enriched Urban Computing and Smart Grid*, pages 1–10. Springer, 2011.

[34] S R Marganiec B. Alturki and C. Perera. **A Hybrid Approach for Data Analytics for Internet of Things**. *International Conference on the Internet of Things, ACM*, 22-25 Oct 2017.

[35] H. Moungla M. A. Cherif H. Afifi M. Laroui, B. Nour and M. Guizani. **Edge and fog computing for IoT: A survey on current research activities future directions**. *Elsevier Computer Communications*, **180**:210–231, 2021.

[36] V. Hegde and S. Usmani. **Parallel and Distributed Deep Learning**. *ICME, Standford University*, 2016.

[37] M. Madiajagan and S. Sridhar Raj. **Chapter 1 - Parallel Computing, Graphics Processing Unit (GPU) and New Hardware for Deep Learning in Computational Intelligence Research**. In Arun Kumar Sangaiah, editor, *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, pages 1–15. Academic Press, 2019.

[38] M. LangerZ. He, Wenny Rahayu, and Yanbo Xue. **Distributed Training of Deep Learning Models: A Taxonomic Perspective**. *IEEE Trans. Parallel Distrib. Syst.*, **31**(12):2802–2818, dec 2020.

[39] Stephen Bonner, Ibad Kureshi, John Brennan, and Georgios Theodoropoulos. **Chapter 14 - Exploring the Evolution of Big Data Technologies**. In Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, and Bruce Maxim, editors, *Software Architecture for Big Data and the Cloud, Elsevier*, pages 253–283. Morgan Kaufmann, Boston, 2017.

[40] A. Ghodsi J. Gonzalez S. Shenker I. Stoica R. S. Xin P. Wendell T. Das M. Armbrust A. Dave X. Meng J. Rosen M. Zaharia, M. J. Franklin and S. Venkataraman. **Apache Spark: A Unified Engine for Big Data Processing**. *Commun. ACM*, **59**(11):56–65, Oct 2016.

[41] J. Sun J. Fu and K. Wang. In *International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII)*, 2016.

[42] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. **MLlib: Machine Learning in Apache Spark**. *J. Mach. Learn. Res.*, **17**(1):1235–1241, jan 2016.

[43] **Distributed Machine Learning Toolkit**. http://www.dmtk.io/. Online; accessed 18 Oct 2019.

[44] **Cognitive Toolkit**. https://docs.microsoft.com/en-us/cognitive-toolkit/. Online; accessed 18 Oct 2019.

[45] Abid Malik, Micheal Lu, Nathenial Wang, Yeiwei Lin, and Shinjae Yoo. **Detailed Performance Analysis of Distributed Tensorflow on a GPU Cluster using Deep Learning Algorithms**. In *2018 New York Scientific Data Summit (NYSDS)*, pages 1–8, 2018.

[46] **Distributed Training with TensorFlow**. https://www.tensorflow.org/guide/distributed_training. Online; accessed 20 Apr 2023.

[47] Afaf Taïk and Soumaya Cherkaoui. **Federated Edge Learning : Design Issues and Challenges**, arXiv:2009.00081, 2022.

[48] K. S. Prado. **Face Recognition: Understanding LBPH Algorithm**. https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b. Online; accessed 16 Oct 2018.

[49] **OpenCV library**. https://opencv.org. Online; accessed 20 Oct 2022.

[50] **World Alzheimer Report 2018: The state of the art of dementia research: New frontiers**. https://www.alzint.org/u/WorldAlzheimerReport2018.pdf. Online; accessed 20 Jan 2019.

[51] **Samvedna Senior Care**. https://www.samvednacare.com/blog/2018/04/09/5-types-of-multi-sensory-stimulation-for-dementia-patients. Online; accessed 10 Jan 2019.

[52] M. A. Dzulkifli and M. F. Mustafar. **The Influence of Colour on Memory Performance: A Review**. *The Malaysian Journal of Medical Sciences*, **20**:3–9, 2013.

[53] L. T. Sharpe F. A. Wichmann and K. R. Gegenfurtner. **The contributions of color to recognition memory for natural scenes**. *Journal of Experimental Psychology, Learning, Memory, and Cognition*, **28**:509–520, 2002.

[54] **Auditory Stimulation for Alzheimer's Disease and Dementia**. https://best-alzheimers-products.com/auditory-stimulation.html. Online; accessed 18 Nov 2018.

[55] K.S. Gayathri and K.S. Easwarakumar. **Intelligent Decision Support System for Dementia Care Through Smart Home**. *Procedia Computer Science*, **93**:947–955, 2016. Proceedings of the 6th International Conference on Advances in Computing and Communications.

[56] **OpenHAB**. https://openhab.org. Online; accessed 21 Dec 2018.

[57] **PIR Sensor**. https://www.epitran.it/ebayDrive/datasheet/44.pdf. Online; accessed December 24, 2022.

[58] **RPi Camera Documentation**. https://www.raspberrypi.com/documentation/accessories/camera.html/. Online; accessed 24 Jan 2023.

[59] **Philips Hue Lights**. https://www.philips-hue.com/en-in. Online; accessed 19 Nov 2018.

[60] M. Carikci and F. Ozen. **A Face Recognition System Based on Eigenfaces Method**. *Procedia Technology*, **1**:118–123, 2012. First World Conference on Innovation and Computer Sciences (INSODE 2011).

[61] Prof Lata, Chandra Kiran, Bharadwaj Tungathurthi, H Ram, Ram Mohan Rao Hanmanth, Dr Govardhan, and Dr Reddy. **Facial recognition using eigenfaces by PCA**. *SHORT PAPER International Journal of Recent Trends in Engineering*, **1**, 04 2009.

[62] G. S. Behera. **Face Detecton with Haar Cascade**. https://towardsdatascience.com/. Online; accessed 19 Jan 2021.

[63] **OpenCV: Face detection using Haar Cascades**. https://docs.opencv.org/3.4/d2/d99/tutorial_js_face_detection.html. Online; accessed 20 Dec 2020.

[64] M. Slavkovic and D. Jevtic. **Face Recognition Using Eigenface Approach**. *Serbian Journal of Electrical Engineering*, **9**:121–130, 2012.

[65] S. Tsang. **MobileNetV2 - Light Weight Model (Image Classification)**. https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c, Online; accessed 15 Nov 2019.

[66] X-zhangyang. **Face Mask Dataset**. https://github.com/X-zhangyang/Real-World-Masked-Face-Dataset. Online; accessed 21 Jun 2020.

[67] **Transfer Learning**. https://www.tensorflow.org/tutorials/images/transfer_learning. Online; accessed 21 May 2020.

[68] **SLURM Workload Manager**. https://slurm.schedmd.com/quickstart.html. Online; accessed 25 Aug 2019.

[69] **Message Passing Interface**. https://computing.llnl.gov/tutorials/mpi/. Online; accessed 15 Aug 2019.

[70] K. Doucet and J. Zhang. **Learning Cluster Computing by Creating a Raspberry Pi Cluster**. In *Proceedings of the SouthEast Conference*, ACM SE '17, page 191–194, New York, NY, USA, 2017. Association for Computing Machinery.

[71] G. Mills. **Building a Raspberry Pi Cluster**. https://glmdev.medium.com/building-a-raspberry-pi-cluster-784f0df9afbd. Online; accessed 20 Aug 2019.

[72] N. Ha. **Raspberry Pi Cluster for Parallel and Distributed Computing**. https://digitalcommons.mtech.edu/cgi/viewcontent.cgi?article=1198&context=grad_rsch. MS Project Report, Montana Technological University, 2019.

[73] P. J. Basford W.. Hajji C.S. Perkins F. P. Tso D. Pezaros R. D. Mullins E. Yoneki S. J. Cox J. Singer, H. Herry and S. J. Johnston. **Next generation single board clusters**. In *NOMS 2018 -*

2018 IEEE/IFIP Network Operations and Management Symposium, pages 1–3, 2018.

[74] I. I. Ivanov. **Utility Computing: Reality and Beyond**. In J. Filipe and M. S. Obaidat, editors, *E-business and Telecommunications*, pages 16–29, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[75] J. Mishra and S. Ahuja. **P2PCompute: A Peer-to-peer computing system**. In *2007 International Symposium on Collaborative Technologies and Systems*, pages 169–176, 2007.

[76] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. **The Hadoop Distributed File System**. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[77] **Apache Hadoop YARN**. https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html. Online: accessed 26 Jan 2019.

[78] **Apache Hadoop**. https://hadoop.apache.org/. Online: accessed 25 Jan 2019.

[79] **Establishing a Hadoop Cluster**. https://www.linode.com/docs/guides/how-to-install-and-set-up-hadoop-cluster/. Online; accessed 30 Oct 2019.

[80] **Apache Spark**. https://spark.apache.org/downloads.html. Online; accessed 20 Nov 2019.

[81] K. Sarafrazi. **Ecommerce Customer dataset**. https://spark.apache.org/downloads.html. Online; accessed 25 Nov 2019.

[82] I. Yeh. **Credit Card dataset**. https://archive.ics.uci.edu/dataset/350/default+of+credit+card+clients. Online; accessed 25 Nov 2019.

[83] **Flower: A Friendly Federated Learning Framework**. https://flower.dev/. Online; accessed 15 Feb 2023.

[84] **gRPC documentation**. https://grpc.io/docs/. Online; accessed 15 Feb 2023.

[85] **TensforFlow**. https://www.tensorflow.org/. Online; accessed 15 Feb 2023.

[86] **CNN with TensorFlow and Keras**. https://medium.com/geekculture. accessed 22 Oct 2022.

[87] **Implementing a CNN in TensorFlow Keras**. https://learnopencv.com/implementing-cnn-tensorflow-keras/. Online; accessed 1 Feb 2023.

[88] **Scikit-learn Machine Learning**. https://scikit-learn.org/stable/. Online; accessed 22 Jan 2022.

[89] F. Hashmi. **Face Recognition using Deep Learning CNN**. https://thinkingneuron.com/. Online; accessed 24 Nov 2022.

[90] V. Patel. **Face Recognition Dataset**. https://www.kaggle.com/datasets/vasukipatel/face-recognition-dataset. Online; accessed 10 Mar 2023.

[91] J. Young. **Rain in Australia**. https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package/code. Online; accessed 21 Feb 2023.

[92] B. Shetty. **Data Visualization using Matplotlib**. https://towardsdatascience.com/data-visualization-using-matplotlib-16f1aae5ce70, Online; accessed 14 March 2021.

[93] D. Gopinathan A. Nayyar R. Krishnamurthi, A. Kumar and B. Qureshi. **An Overview of IoT Sensor Data Processing, Fusion, and Analysis Techniques**. *MDPI Sensors*, **20**(21), 2020.

[94] **IMX219-77 Camera - Waveshare wiki**. https://www.waveshare.com/wiki/IMX219-77_Camera. Online; accessed March 12, 2023.

[95] **Digital Humidity and Temperature Sensor**. https://www.mouser.com/datasheet. Online; accessed January 25, 2022.

[96] **Apache Kafka**. https://kafka.apache.org/. Online; accessed 25 Jan 2023.

[97] Bhole Rahul Hiraman, Chapte Viresh M., and Karve Abhijeet C. **A Study of Apache Kafka in Big Data Stream Processing**. In *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*, pages 1–3, 2018.

[98] **Apache Zookeeper**. https://zookeeper.apache.org/. Online; accessed 25 Jan 2023.

[99] **Paramiko documentation**. https://www.paramiko.org/. Online; accessed 22 Feb 2023.

[100] R. Sukthanker B. Ru T. Elsken A. Zela D. Dey C. White, M. Safari and F. Hutter. **Neural Architecture Search: Insights from 1000 Papers**, 2023.

[101] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. **MnasNet: Platform-Aware Neural Architecture Search for Mobile**, 2019.

[102] **Image Classification - AutoKeras**. https://autokeras.com/tutorial/image_classification/. Online; accessed 23 Mar 2023.

[103] **Netdata: Monitoring and Troubleshooting**. https://www.netdata.cloud/. Online; accessed 21 Jan 2022.

[104] **Cloud Computing Services - Google Cloud**, 2023.

Heterogeneous Sensor Data Streaming for Federated Learning in Fog-Centric IoT Applications

by Srinivasa Raju R

Submission date: 18-Oct-2023 02:15PM (UTC+0530)
Submission ID: 2199518926
File name: 18MCPC13_Thesis_for_Plagiarism_Check.pdf (11.38M)
Word count: 27841
Character count: 151978

Heterogeneous Sensor Data Streaming for Federated Learning in Fog-Centric IoT Applications

ORIGINALITY REPORT

39%	25%	31%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

Overall Similarity Index: 39 - (8+7+7+4+3+1+1+1) = 07%.

1	Srinivasa Raju Rudraraju, N.K. Suryadevara, Atul Negi. "Heterogeneous Sensor Data Acquisition and Federated Learning for Resource Constrained IoT Devices - A validation", IEEE Sensors Journal, 2023 Publication This Publication is by the Student	8%
2	ebin.pub Internet Source This publication is by the Student	7%
3	annals-csis.org Internet Source This publication is by the Student	7%
4	Srinivasa Raju Rudraraju, Nagender Kumar Suryadevara, Atul Negi. "Face Recognition in the Fog Cluster Computing", 2019 IEEE International Conference on Signal Processing, Information, Communication & Systems (SPICSCON), 2019 Publication This Publication is by the Student	4%
5	vdoc.pub Internet Source This Publication is by the Student	3%

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

6	Srinivasa Raju Rudraraju, Nagender Kumar Suryadevara, Atul Negi. "Edge computing for visitor identification using eigenfaces in an assisted living environment", Elsevier BV, 2020 This Publication is lyte Student Publication	1 %
7	Srinivasa Raju Rudraraju, Nagender Kumar Suryadevara, Atul Negi. "Heterogeneous Sensor Data Acquisition and Federated Learning for Resource Constrained IoT Devices—A Validation", IEEE Sensors Journal, 2023 This publication is lyte Student Publication	1 %
8	Srinivasa Raju Rudraraju, Nagender Kumar Suryadevara, Atul Negi. "Fog computing framework for Big Data processing using cluster management in a resource-constraint environment", Institution of Engineering and Technology (IET), 2021 Publication This publication is lyte Student	1 %
9	dokumen.pub Internet Source	1 %
10	pub.nkumbauniversity.ac.ug Internet Source	1 %
11	Submitted to Higher Education Commission Pakistan Student Paper	1 %

12	www.scilit.net Internet Source	<1 %
13	www.researchgate.net Internet Source	<1 %
14	"Advances in Decision Sciences, Image Processing, Security and Computer Vision", Springer Science and Business Media LLC, 2020 Publication	<1 %
15	Submitted to Canadian University of Dubai Student Paper	<1 %
16	github.com Internet Source	<1 %
17	arxiv.org Internet Source	<1 %
18	www.intellectyx.com Internet Source	<1 %
19	"Security Issues in Fog Computing from 5G to 6G", Springer Science and Business Media LLC, 2022 Publication	<1 %
20	"NEO 2016", Springer Science and Business Media LLC, 2018 Publication	<1 %
21	www.bbsmax.com Internet Source	

		<1%
22	"Second International Conference on Image Processing and Capsule Networks", Springer Science and Business Media LLC, 2022 Publication	<1%
23	Submitted to Iowa State University Student Paper	<1%
24	Yang Cao. "Chapter 3 Near Real-Time Federated Machine Learning Approach Over Chest Computed Tomography for COVID-19 Diagnosis", Springer Science and Business Media LLC, 2022 Publication	<1%
25	Qiang Duan, Jun Huang, Shijing Hu, Ruijun Deng, Zhihui Lu, Shui Yu. "Combining Federated Learning and Edge Computing Toward Ubiquitous Intelligence in 6G Network: Challenges, Recent Advances, and Future Directions", IEEE Communications Surveys & Tutorials, 2023 Publication	<1%
26	researchr.org Internet Source	<1%
27	Sameer Wadkar, Madhu Siddalingaiah. "Pro Apache Hadoop", Springer Science and Business Media LLC, 2014	<1%

	Publication	
28	"Fog and Fogonomics", Wiley, 2020 Publication	<1 %
29	B. Ankayarkanni, Niroj Kumar Pani, M. Anand, V. Malathy, Bhupati. "P2FLF: Privacy-Preserving Federated Learning Framework Based on Mobile Fog Computing", International Journal of Interactive Mobile Technologies (iJIM), 2023 Publication	<1 %
30	Submitted to University of Oklahoma Student Paper	<1 %
31	"Euro-Par 2017: Parallel Processing Workshops", Springer Nature, 2018 Publication	<1 %
32	test.jpier.org Internet Source	<1 %
33	Submitted to Berlin School of Business and Innovation Student Paper	<1 %
34	oops.uni-oldenburg.de Internet Source	<1 %
35	"Cyberphysical Smart Cities Infrastructures", Wiley, 2022 Publication	<1 %
36	Submitted to University of Johannsburg	

	Student Paper	<1%
37	Submitted to University of Hong Kong Student Paper	<1%
38	Youyang Qu, Mohammad Reza Nosouhi, Lei Cui, Shui Yu. "Chapter 4 Personalized Privacy Protection Solutions", Springer Science and Business Media LLC, 2021 Publication	<1%
39	"Federated Learning", Springer Science and Business Media LLC, 2020 Publication	<1%
40	Yuvraj Sahni, Jiannong Cao, Shigeng Zhang, Lei Yang. "Edge Mesh: A New Paradigm to Enable Distributed Intelligence in Internet of Things", IEEE Access, 2017 Publication	<1%
41	"Design, User Experience, and Usability. User Experience Design for Diverse Interaction Platforms and Environments", Springer Nature, 2014 Publication	<1%
42	Submitted to Università degli Studi di Trieste Student Paper	<1%

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		