Security-aware Data Offloading and Optimization in Healthcare System

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Computer Science and Engineering

Submitted By

Kashish Kaushal

(802332039)

Under the supervision of:

Dr. Rajkumar Tekchandani

Associate Professor

Dr. Anil Singh

Assistant Professor

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

PATIALA – 147004

July 2025

CERTIFICATE

I hereby certify that the work which is being presented in the seminar entitled, **“Security-aware Data Offloading and Optimization in Healthcare System”**, in partial fulfillment of the requirements for the award of the degree of **Master of Engineering in Computer Science and Engineering**, submitted in the **Computer Science and Engineering Department** of **Thapar Institute of Engineering and Technology, Patiala**, is an authentic record of my own work carried out under the supervision of **Dr. Rajkumar Tekchandani and Dr. Anil Singh**.

[illegible]

(Kashish Kaushal)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

[illegible] [illegible]

Dr. Rajkumar Tekchandani — *Associate Professor*

Dr. Anil Singh — *Assistant Professor*

Acknowledgement

This project would not have been possible without the help of many people, and it is my pleasure to convey my gratitude to them for their various forms of support, encouragement, and assistance during the project's duration.

I would like to express my gratitude to my supervisors, **Dr. Anil Singh** and **Dr. Rajkumar Tekchandani**, for allowing me to work under their capable guidance and for their unwavering support. This endeavor would not have been possible without their assistance and advice. They were the ones who first taught me topics like federated learning, foundational technologies like Cloud–Fog–Edge computing and containerization using Docker, as well as providing me with extremely useful insights through regular discussions during this project. Continuous talks have aided me in acquiring a good understanding of this thesis.

Abstract

Healthcare data is growing at a rapid rate, which presents both opportunities and obstacles in enhancing healthcare services and data privacy. Due to the humongous amount of data generated by healthcare systems, it becomes necessary to transport some of the data to cloud servers. The federated environment is better suitable for healthcare systems due to several key factors that address the challenges in handling sensitive medical data and improving healthcare outcomes. Due to the dynamic nature of the federated environment and uncertain operating environment with a large volume of data generated by healthcare systems and considering the criticality of service, it is very challenging and, at the same time, important to make sure that the federated environment is reliable in terms of Quality of Service(QoS) and security. To ensure the QoS and securely transporting/storing healthcare data between local servers and the cloud, this work presents a secure federated framework. First, the classification of the data into several categories, such as operational data, research data, essential patient information, and archived data, is conducted. Second, a trade-off is proposed that decides where a particular data chunk will reside (i.e., local node, fog node, or cloud). Third, a secure method for ensuring data security is introduced within the federated system. The experiments conducted based on the multiple datasets demonstrate that the performance of the proposed framework is improved significantly as compared to state of the art algorithms.

Contents

List of Figures

List of Tables

List of Symbols

Table 1: List of Symbols and Their Descriptions

Symbol	**Description**
T_i	Task i
S_i	Size of task i (in bytes)
Z_i	Task size parameter for scheduling
Δ_i	Delay tolerance of task i
ST_i	Start time of task i
ET_i	Execution time of task i
D_i	Deadline for task i
$CD_{i,j}$	Communication delay of task i on node j
T_{init}	Initialization time to establish communication
$SC_{i,j}$	Signaling cost between task i and node j
BW	Bandwidth of the network link (in Mbps)
C_k	Capacity of node k (in GHz)
L_k	Current load of node k (in GHz)
S_k	Storage capacity of node k (in GB)
UT_j	Utilization of node j
UT_{system}	Overall system utilization
$cost_i$	Deployment cost of task i
N_i	Node value associated with task i (based on job size)
J_i	Tier value for task i (Edge=1, Fog=2, Cloud=8) i
N_a	Average of all N_i values in a batch
J_a	Average reference tier value (typically = 3.67)
Y	Total number of scheduled tasks
Y_0	Number of tasks completed within deadline
SR	Task success ratio
e	Number of tasks allocated to Edge nodes
f	Number of tasks allocated to Fog nodes
c	Number of tasks allocated to Cloud nodes
D_{total}	Overall system delay
NL	Node load (used in node classification)
LT	Latency (used in Algorithm 2)
N'	Under-loaded node

Chapter 1
Introduction

In the current era of digitization, data has become a critical asset. The digital transformation of healthcare is generating a huge amount of data, which includes: patient records, medical imaging, and other administrative information. This data has huge potential to improve medical research, hospital operations, and patient care. However, handling and using healthcare data securely is extremely difficult due to its sensitive nature and strict privacy laws. This is especially important when it comes to data storage and movement between local servers and cloud environments. One way of addressing these issues is federated learning (FL), which allows machine learning models to be developed across decentralized datasets while maintaining localized data.
Federated Learning (FL) [1] is a method to train machine learning models without sharing sensitive data. Instead of sending the actual data to a central server, each device or node trains the model locally and only shares updates to the model. In healthcare, federated learning allows the creation of machine learning models between hospitals and other medical facilities without requiring the exchange of private patient information [2]. FL improves diagnosis, treatment plans, and research because here models are trained locally and only communicate updates while maintaining patient privacy and security [3]. Figure-1.1 is the depiction of smart healthcare's wide-ranging benefits and applications.

The benefits of federated learning in healthcare are significant. Since sensitive patient data is retained within medical facilities, this lowers the possibility of security breaches and guarantees obedience to privacy regulations [4]. With the use of a variety of data from different sources, hospitals, and other institutions can collaborate to improve AI models, which would improve patient care. As these models learn from varied patient cases and conditions across multiple sites, they become more accurate and dependable.

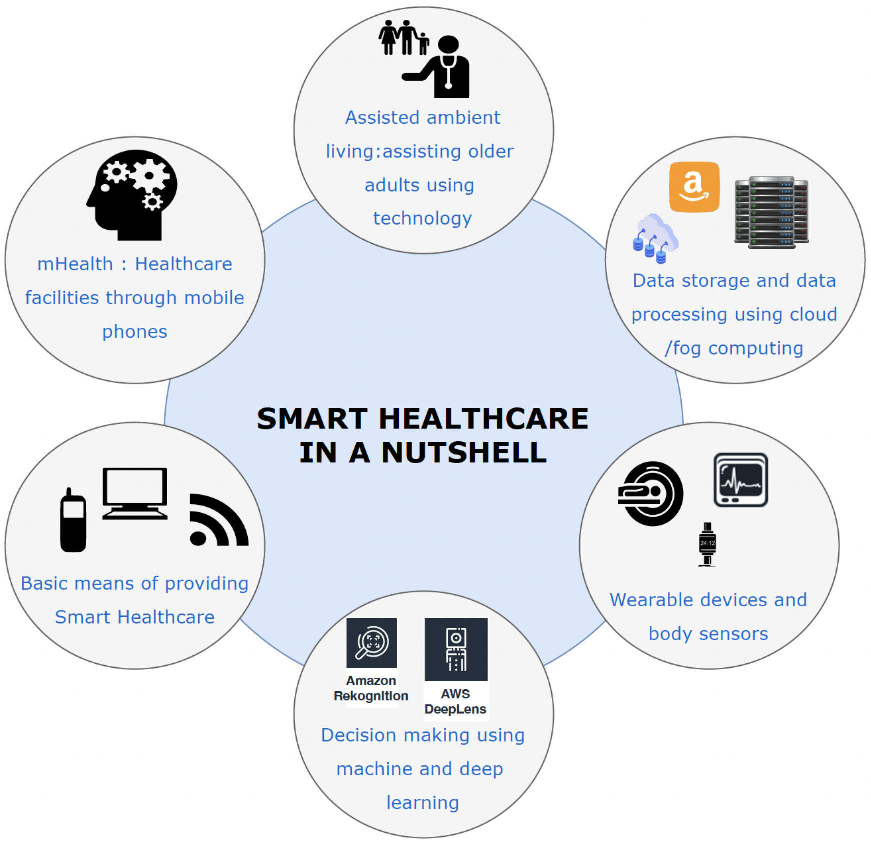
Assisted ambient living:assisting older adults using technology
mHealth : Healthcare facilities through mobile phones
Data storage and data processing using cloud /fog computing
SMART HEALTHCARE IN A NUTSHELL
Basic means of providing Smart Healthcare
Wearable devices and body sensors
Amazon Rekognition
AWS DeepLens
Decision making using machine and deep learning

Figure 1.1: Transformation of Healthcare through Smart Technology

When it comes to the security of healthcare, one of the key strengths is the protection of sensitive data. Since this actual sensitive data never leaves local servers, the risk of data breaches is significantly reduced. However, security challenges remain: even with raw data safe, model updates can still leak sensitive information via inference attacks [5, 6]. To address this, FL systems employ advanced encryption and secure aggregation techniques—such as homomorphic encryption and differential privacy—to protect model updates from tampering or leakage [7–10]

Despite the substantial benefits of Federated Learning in healthcare, several real-world challenges hinder its full-scale deployment. One major obstacle is the heterogeneity of data and computing infrastructure across medical facilities [2, 3]. Different hospitals may use varied formats, devices, or standards to record and store patient data. This diversity leads to issues in model convergence and synchronization, as inconsistencies across local nodes can negatively affect overall training performance. In addition, resource constraints at edge devices, such as hospital servers, IoT-enabled patient monitors, or wearable health trackers, can limit the speed and efficiency of training [11–13].

These nodes may not have equal storage capacity, computational power, or network bandwidth. As a result, naive or uniform scheduling strategies often cause performance bottlenecks, idle time, or even dropped updates, impacting the model's overall quality and timeliness.

Moreover, the volume and frequency of updates shared between participants can introduce considerable communication overhead [14, 15]. In high-resolution or time-sensitive applications such as real-time patient monitoring or emergency diagnostics, this latency can be critical [16]. Efficient scheduling, compression, or prioritization techniques must be deployed to manage these bottlenecks and minimize energy and bandwidth consumption. From a security standpoint, the vulnerability of gradient updates to inference attacks remains a concern [6, 17]. While FL inherently avoids raw data transmission, attackers may still exploit patterns in shared parameters to reconstruct private information [18]. Ensuring robust security and privacy thus requires integrating privacy-preserving technologies like Differential Privacy (DP), Homomorphic Encryption (HE), or Secure Multiparty Computation (SMC).

Given these challenges, this thesis aims to explore an optimized and secure data offloading strategy for federated healthcare environments. The proposed approach focuses on intelligent scheduling within a Cloud–Edge–Fog continuum to minimize operational costs, ensure data availability, and enhance security—making FL more scalable and effective in real-world healthcare systems.

Chapter 2

Background

Implementing a safe and effective federated learning environment for healthcare applications requires a basic understanding of many different technologies as shown in fig 2.1. For example, cloud computing allows for centralized, flexible storage and processing, while edge computing offers low-latency processing and privacy through geographically proximate processing. Then, there's federated learning, which is a way to train ML models. In this multiple parties can train a model while avoiding the transmission of sensitive patient data. Such an evaluation acknowledges the relative advantages and disadvantages between various solutions and highlights what's similar yet different when appreciating real-time healthcare versus big data management.

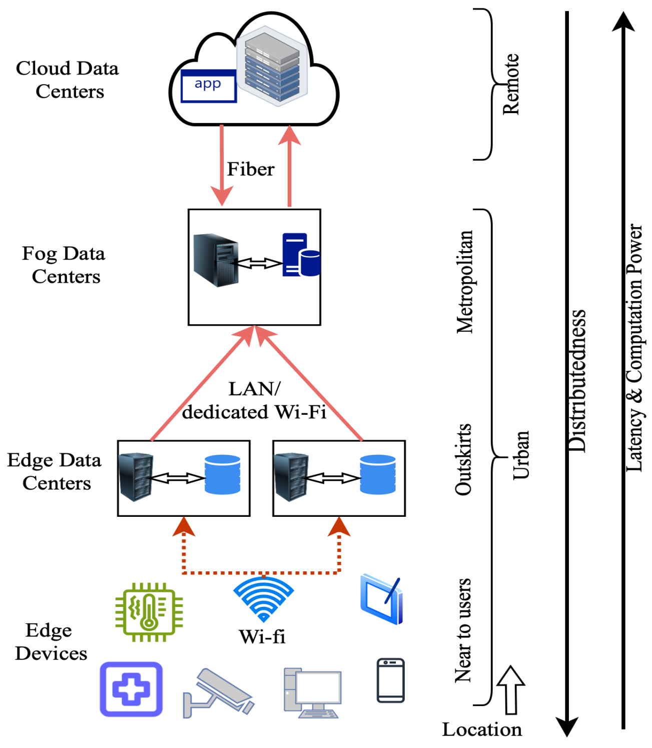

Figure 2.1: Federated environment

2.1 Cloud Computing

Cloud computing involves the distribution of computing services such as storage, processing, software via the Internet on a pay-as-you-go basis without active, direct user control [11,19]. The collaborative nature of this cloud standard involves processing and storing extensive amounts of data in centralized data warehouses which offer a cloud that is more customizable, more flexible, and based on a pay-per-use structure [20,21]. The services fall into three distinct categories: *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)*, and *Software as a Service (SaaS)*.

For instance, cloud computing provides healthcare facilities access to processing vast amounts of *electronic health records (EHR)*, undertaking extensive data analytics, or accessing AI decision-support software. However, the centralized approach to such computing can create latency issues and cyber-security flaws—especially with sensitive and time-sensitive information [16,22].

2.2 Edge Computing

Edge computing relocates computation and data storage to the location, where they are needed, which is usually at or near the data source itself [11,23]. In healthcare, edge computing is used on local gateways or edge nodes, like hospital servers or diagnostic center systems, to make it easier to process data in real-time and keeping delay to minimal.

This computing paradigm cuts down on the amount of data that needs to be sent to remote cloud servers [13, 16]. This makes data more private and speeds up response times. It is especially useful when you need to do emergency diagnostics, get real-time alerts, and keep a watch on a patient all the time [16,23].

2.3 Federated Learning

Federated Learning (FL) is a type of machine learning that is decentralized and lets models be trained on many different devices or servers that have local data samples without having to share them with each other [24,25]. In FL, only model updates (like weights or gradients) are sent to a central server. This maintains the privacy by keeping the raw data local.

FL is an effective solution to problems with privacy and data governance in healthcare. It lets medical institutions work together to learn from each other without putting sensitive patient information at risk [25,26]. This makes it easier to create stronger and more general models that can be used with a wide range of patients.

2.4 Difference Between Cloud and Edge Computing

Cloud and edge computing both have big benefits for healthcare systems, but serve different purposes and have different pros and cons. The table below summarizes key differences:

Table 2.1: Comparison of Cloud and Edge Computing

Parameter	**Cloud Computing**	**Edge Computing**
Latency	High (data travels to a distant server)	Low (data processed locally)
Data Privacy	Moderate (data must be transmitted over the internet)	High (data stays near source)
Computation Power	High (centralized infrastructure)	Limited (dependent on local devices)
Scalability	Highly scalable	Less scalable, hardware-constrained
Use Case	Long-term storage, global analytics	Real-time alerts, sensitive data handling
Network Dependency	Requires reliable high-bandwidth connectivity	Operates even with intermittent connectivity
Security	Vulnerable to centralized breaches	Better privacy control but with limited protections
Maintenance	Centralized and handled by providers	Decentralized, requires local technical support

Chapter 3

Literature Review

The integration of Federated Learning (FL) into healthcare systems has become a focal point in recent research. FL enables collaborative model building across multiple universities while keeping sensitive data local, which addresses privacy issues. Several research have proved the feasibility and benefits of using FL in healthcare.

3.1 Federated Learning for Healthcare: Security and Privacy Perspectives

In [13], a software reference architecture for security data orchestration, analysis and reporting is proposed. This work introduces a Software Reference Architecture (SRA) called *SecDOAR*, which aims to standardize and improve how security data platforms work together. *SecDOAR* helps organize, analyze, and report security data across different tools and systems. *SecDOAR* is designed to offer a method for managing the security of data and making it easier for different security tools to work together. It also helps organizations improve how they monitor, analyse and make report on cybersecurity. SecDOAR is made up of various components including; Security tools and infrastructure layer that manages the capabilities and actions enabled by various security tools. Security integration layer manages the integration of security data from diverse sources and solutions. Semantic layer allows for semantic integration of security data, ensuring consistency and meaningful interpretation across many tools. The security data processing layer processes raw security data and extracts additional information for analysis. The threat intelligence and management layer analyzes security data to identify risks and manage threat intelligence. Orchestration layer integrates security data from many tools and platforms to provide consistent data management and analysis. Analysis and reporting layers: Conduct actual security data analysis and provide reports based on the results.

In [27], the blockchain-based federated learning technique for privacy preservation and security of smart electronic health records is presented. This work improves the security and privacy of smart device health records by integrating blockchain technology with federated learning. Concerns related to the scalability of healthcare data records in a cloud environment are raised, especially within the Internet Of Medical Things(IOMT) framework. Using lightweight encryption and blockchain decentralized property, this system ensures that data is securely stored and transmitted.

In [28], the authors proposed preserving privacy and security in the federated learning framework. Concerns related to both privacy and security in FL are raised especially against poisoning attacks. The use of protocols such as *Zero Knowledge Proof(ZKP)*, which allows users to prove that their model updates are not poisoned, without revealing the actual updates themselves. Along with that, homomorphic encryption is also used that allows us to perform certain mathematical operations on encrypted data without any need to decrypt it.

In [16] a framework called *BEdgeHealth* is presented. *BEdgeHealth* is a decentralized architecture for edge-based IoMT networks that uses blockchain. It provides a decentralized architecture that combines *Mobile-edge Computing(MEC)* and blockchain to tackle the challenges in healthcare. This includes *low-quality servers(QOS)*,data privacy and security vulnerabilities. Use of *MEC* to offload healthcare data from mobile nodes to nearby servers for better processing with the integration of blockchain. Other than this, interplanetary file system *(IPES)* which is a peer-to-peer decentralized file storage and sharing protocol is used to make the web faster, more secure, and less centralized. Smart-control authentication mechanism which operates on the network edge ensures access verification enhancing security and traceability. The data retrieval process is optimized by removing traditional global *distributed hash table(DHT)* from *IPES*, this speeds up data lookups and reduces the time for data retrieval. This method is particularly good for low-latency healthcare applications.

3.2 Performance Optimization Techniques in Federated Learning for Healthcare

In [29], the author presented the allocation and scheduling of linear workflows across fog and cloud infrastructures. To provide security for *IOT* devices, this study categorized the workloads with varying security requirements by differentiating between ”*cloud jobs*” with fewer security requirements and ”*fog jobs*” with higher security requirements. The study describes two variants of scheduling strategy that prioritize *fog jobs* by using approximate computations for *cloud jobs*, resulting in faster *fog job* execution. Fog computing enhances the performance of *IOT* applications by extending computational resources near the data source. Thus reducing latency in data. Fog computing sports better bandwidth management because it filters and processes data before sending only relevant information to the cloud.

In [12], the author provides a mechanism for secure and lightweight communication in heterogeneous IoT environments. This paper discusses strategies for improving communication security and efficiency in *IoT* networks. This study focuses on the *Constrained Application Protocol (CoAP)*, which is a lightweight protocol developed for restricted devices, and how it works with *Datagram Transport Layer Security (DTLS)* to ensure safe communication. Issues such as limited memory, processing power, and the energy requirements of cryptographic operations make building secure communication protocols difficult.

In [30], a federated learning optimization that comprises a computational blockchain process with offloading analysis to enhance security is presented. The main focus is on the optimization of computational processes through offloading tasks while enduring data security in IOT systems by training data in local nodes rather than sending it to a central server. The technique evaluates which tasks can be offloaded from primary IOT devices to other nodes, thus reducing the burden on individual nodes. Furthermore, Date Blocks are used to improve data transmission.

In [31], a framework for lightweight privacy and security computing for blockchain federated learning in IoT is proposed. This framework addresses the issues of computational efficiency, security, and privacy in *Internet of Things (IoT)* contexts. To guarantee that *Internet of Things (IoT)* devices can train machine learning models securely and effectively without disclosing sensitive data, the study emphasizes the integration of blockchain technology with federated learning. Layered BFL Architecture; the *LPBFL* architecture takes a layered approach, combining blockchain and federated learning. This architecture ensures that local model data is kept private even during aggregation procedures by utilizing the Paillier encryption algorithm for safe, efficient computing. *Lightweight Digital Signature;* it provides a lightweight digital signature technique with batch verification, which lowers the computational load and guarantees the integrity and authenticity of the data being processed. This mechanism helps fend against typical attacks like model forging or denial. *Reputation-Based Worker Selection;* using a reputation-based selection method, the paper tackles the problem of unreliable workers in federated learning. By evaluating and choosing only reliable workers for model training, this method raises the global model's accuracy and efficiency. Comprehensive Security and Performance Analysis: Confidentiality, non-repudiation, and unforgeability are just a few of the security attributes of *LPBFL* that are carefully examined in this work. Additionally, a comparison of *LPBFL's* computational overhead with other approaches demonstrates notable advances in security and efficiency.

Table 3.1: Summary of Literature on Federated Learning in Healthcare

Authors	**Methodology**	**Advantages**	**Limitations**
Zhang et al. [1]	Comprehensive survey on FL	Covers a wide range of FL techniques	Not specific to healthcare
Arikumar et al. [4]	FL-PMI system for wearable device data	Smart healthcare integration	Limited to specific devices
Gupta et al. [11]	iFogSim toolkit for edge/fog simulation	Useful for performance evaluation	Simulation-based; lacks real-time analysis
Antunes et al. [3]	Systematic review on FL in healthcare	Architecture proposal	Implementation details lacking
Nguyen et al. [2]	Survey on FL for smart healthcare	Detailed comparison	No new framework
Chauhan et al. [13]	SecDOAR architecture for security orchestration	Modular and semantic design	Not healthcare-specific
Guduri et al. [27]	Blockchain-enhanced FL for EHRs	Privacy-preserving architecture	Ethereum overhead
Shitharth et al. [30]	Blockchain-based offloading strategy	Enhanced security	High resource usage
Nguyen and Thai [28]	Survey on preserving FL privacy	Broad coverage	No comparative metrics
Nguyen et al. [16]	BEdgeHealth using blockchain + MEC	Low-latency design	Architecture complexity
Siddiqui et al. [12]	Lightweight secure comms for IoT	Energy-efficient	Not FL-specific
Karatza [29]	Workflow scheduling with security constraints	Fog-cloud integration	No privacy in FL considered
Fan et al. [31]	Lightweight privacy framework for FL	Blockchain integration	Scalability concerns

Table 3.3: Comparative Feature Support Among Approaches

Approach	**Security**	**Privacy**	**Lightweight**	**Decentralized**	**Deadline-Aware**	**Offloading Opt.**	**Success Ratio**
Fan et al. [12]	✓	✓	✓	✓	×	×	×
Nguyen et al. [10]	✓	✓	✓	×	×	×	×
Karatza et al. [9]	✓	×	×	×	✓	✓	×
Nguyen et al. [7]	✓	✓	×	×	×	×	×
Chauhan et al. [5]	✓	×	×	✓	×	×	×
Proposed approach	✓	✓	✓	✓	✓	✓	✓

Chapter 4

Research Gaps or Problem Statement

4.1 Research Gaps

While federated learning has showed considerable promise in the healthcare area, numerous difficulties must be addressed before it can completely achieve its potential:

Dynamic task offloading: There is currently a lack of comprehensive solutions for dynamically offloading tasks among edge, fog, and cloud nodes based on data sensitivity, size, and delay tolerance. Ongoing approaches are generally focusing on static classification, which may not be good for real-time healthcare systems. [11, 13]

Security and Privacy Efficiency: While other security techniques such as homomorphic encryption and Zero-Knowledge proofs have been used in FL, their computational overhead is still considerable [12, 31]. We need lightweight and robust technologies that balance security measures with system performance to avoid latency and resource wastage, especially in resource-constrained IoT environments.

Integration of Decentralized Architectures: Decentralized technologies like blockchain show potential for improving security and scalability, but their integration with edge and fog computing frameworks is yet underexplored [27, 30]. Research regarding the seamless integration of blockchain with FL for healthcare applications is necessary.

Efficient Scheduling for Heterogeneous Networks: While scheduling strategies for fog and cloud systems exist, improving these strategies for heterogeneous networks with variable security and latency constraints is crucial for implementing them in healthcare like environment [13, 29].

Low-Latency Solutions for Healthcare Applications: Studies such as BEdgeHealth show latency reduction using MEC [16], but there has been little study into merging low-latency solutions with scalable storage systems such as IPFS for healthcare applications. Creating such solutions can considerably improve performance for real-time health monitoring.

4.2 Problem Statement and Objectives

The increasing digitalization of healthcare systems has resulted in an unprecedented volume of distributed data, such as electronic health records, diagnostic images, and administrative log. This data has a lot of potential to improve diagnostics, treatment outcomes, and operational efficiency, but it isn't being used as much as it could be because of privacy concerns, regulatory restrictions, and the fact that healthcare systems are not very well integrated. It can be hard to make strong AI models across institutions when centralized data collection isn't possible or is limited by law. This is a big problem: how do you use decentralized healthcare data for smart applications without putting privacy at risk?
Federated Learning (FL) is a promising path because it lets different groups to work together to train models without sharing raw data. However, the practical deployment of FL in healthcare comes with its own set of problems, such as different types of data, problems with communication, limited resources at edge nodes, and security threats like model inversion attacks. These issues must be carefully addressed to ensure the scalability and effectiveness of FL systems in real-world healthcare scenarios.
By avoiding direct data sharing, we can protect patient privacy, therefore enabling facilitates to collaborate among organizations without worrying about privacy concern. Healthcare firms use distributed data and advanced analytics to improve patient outcomes [25, 26]. However, the delicate nature of healthcare data requires diligent privacy and security measures [6, 9]. Traditional centralized data processing systems are ineffective due to high latency, security threats, and compliance issues with rules such as *HIPAA* [16, 19]. Furthermore, managing activities with shifting sensitivity and size in dynamic IoT contexts, such as Edge–Fog–Cloud frameworks, is difficult. There is an urgent need for a scalable and efficient system that protects privacy, decreases latency, optimizes work distribution, and increases security.

The system must emphasize on data offloading and make sure this data is protected by ensuring that sensitive patient data never leaves local servers. This goal is important for meeting privacy requirements such as *HIPAA* and enabling healthcare facilities

to collaborate on enhancing predictive models. The data that isn't stored locally must be stored at relevant storage facilities, finding the right balance between cost optimization and security. Also, cutting down on communication overhead is another important goal. The system needs to make sure that updates are transferred as efficiently as possible between central server and local servers. Achieving both practicality and efficacy in real-time situations for the system requires minimizing delays.
Another purpose is to ensure data security throughout the entire procedure. Although Federated Learning reduces the amount of data that is exposed, but still attacks on the shared model updates still need to be prevented [28, 31]. Advanced security features like lightweight encryption should be included in the system to handle this. The system architecture should prioritize cost-efficient strategies for data management. Since healthcare data is very different in terms of sensitivity and frequency of use, so another important objective is to create a strategy or system that determines which data is stored locally and which is moved to the cloud or other storage devices. This method will help us to reduce storage costs while maintaining fast access to the critical or most frequently used data. A Framework for Node Load Balancing in Federated Learning should be developed to handle node overload and under-utilization by exploiting Docker containers for job transfer.

The Key-objectives are:

- To develop strategies for classifying and storing data based on sensitivity and frequency of use.
- To develop a cost efficient Federated Learning framework integrated with edge, Fog, and Cloud nodes for secure and efficient healthcare data processing.
- To emphasize on data offloading while ensuring data security by keeping the most sensitive patient data on local servers.
- To design a framework for node load balancing in federated learning in order to handle node overload and under-utilization.
- To include advanced security features like lightweight encryption in order to protect shared model updates from attacks.

Chapter 5

Methodology

5.1 Federated Environment and Distribution of Nodes

In the proposed healthcare federated learning (FL) system, the distribution of data is across various nodes, each node possesses unique features and roles. These nodes are categorized into edge nodes, fog nodes, and cloud nodes, and each layer is essential for the proper functioning of the system. Here is the description of the node types and how data is managed within this environment. One scenario of a federated environment in healthcare is depicted in 5.1.

The node distribution of the proposed Federated Learning (FL) system for healthcare is designed to maximize both security and performance. The system consists of three-layer node architecture - edge nodes, fog nodes and cloud nodes, each has its own specific roles and characteristics.

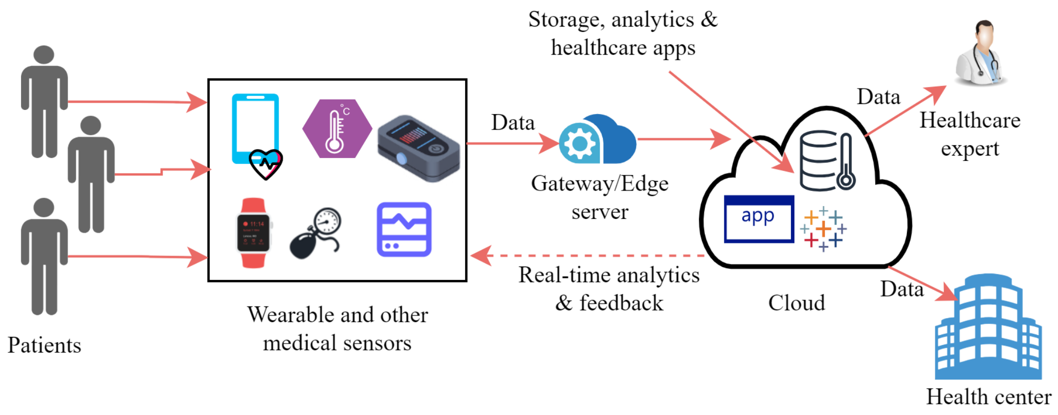

Figure 5.1: A smart healthcare scenario

Edge Nodes (E)

Edge nodes or edge data centers, are located in close proximity to the data source — usually at hospitals, clinics, diagnostic centers, or research institutions. These nodes act as the first point of data generation and processing, where sensitive patient data is collected and stored locally.

Edge nodes are responsible for:

- Training machine learning models on local datasets without transmitting raw data.
- Maintaining high data privacy by avoiding data transmission.
- Handling low-latency operations due to their physical closeness to patients and medical equipment.
- Supporting frequent operational tasks such as appointment scheduling or localized diagnostics.

However, they have limited computational capacity, storage, and bandwidth in comparison to fog and cloud nodes. Despite this, they remain essential for privacy-preserving local computations in federated healthcare.

Characteristics of Edge Nodes:

- Located close to data sources and end users.
- Store and process sensitive patient data.
- Perform initial model training using local data.
- Offer low latency but limited processing/storage capabilities.
- Located near to users, usually situated on the outskirts of urban areas.

Fog Nodes (F)

Fog nodes are intermediate computational entities that are deployed between edge and cloud layers. They are tasked with aggregating model updates from multiple edge nodes within a specific region and performing partial computations or pre-processing before forwarding data to the cloud.

They are critical in:

- Performing regional model aggregation for collaborative learning.

- Managing medium-sensitivity data from multiple institutions.
- Providing faster response times than the cloud due to their geographical proximity to edge nodes.
- Offloading tasks such as operational coordination, resource allocation, or staff scheduling.

Fog nodes serve as a scalable middle layer that decreases the workload on the cloud and reduces network communication overhead by locally handling intermediate data flows.

Characteristics of Fog Nodes:

- Positioned between edge and cloud layers.
- Serve multiple edge nodes within the same region.
- Execute mid-level computations and aggregations.
- Offer medium latency and resource capability.

Cloud Node (C)

The cloud node or central cloud server, act as the federated learning network's central aggregation sites. It is the core aggregation point for global model training. It receives regional updates from fog nodes and performs global model aggregation, which is then redistributed back to fog and edge layers.

The cloud infrastructure is designed for:

- Handling high-volume, long-term data for global research and benchmarking.
- Performing complex computations such as global orchestration, trend analysis, and full-model retraining.
- Storing large datasets from multiple healthcare institutions across regions.

However, due to its remoteness from the data source, the cloud introduces higher latency and is less suited for time-sensitive healthcare operations.

Characteristics of the Cloud Node:

- Centralized infrastructure with high computational and storage capacity.
- Manages orchestration of the global federated model.

- Suitable for archival storage, AI benchmarking, and global analytics.
- Exhibits higher latency than fog and edge layers.

Mathematical Representation

Let C represent the central cloud node responsible for global aggregation and orchestration.

Let $E = \{E_1, E_2, \ldots, E_n\}$ represent the set of edge nodes, $F = \{F_1, F_2, \ldots, F_m\}$ the fog nodes, and C the cloud node. Each node type has distinct characteristics:

- **Edge Nodes** (E_i): Located close to data sources, these nodes have low latency but limited computational capacity. They store sensitive patient data.
- **Fog Nodes** (F_j): Intermediate nodes aggregating model updates and handling medium sensitivity tasks.
- **Cloud Node** (C): Centralized high-capacity node responsible for global model aggregation and archival storage.

Each node N_k, where k is a node in Edge, Fog, or Cloud layers, has the following attributes:

- Capacity: C_k (GHz)
- Current Load: L_k (GHz)
- Bandwidth: B_k (Mbps)
- Storage: S_k (GB)

This hierarchical model supports privacy-preserving, latency-sensitive federated learning within a distributed healthcare environment, enabling intelligent data flow from local to global layers.

5.2 Data Dividing and Categories

The healthcare data distributed across these nodes is separated into several categories based on sensitivity, frequency of access and security requirements. Each type of data is handled accordingly to ensure both cost-efficiency and security. Here are the Categories:

- **Critical Patient-Centric Data (CPD):** This category contains very sensitive and private information such patient medical records, diagnoses, and treatment plans. Because of its sensitive nature, this data is kept only and only within the edge nodes (E), assuring maximum privacy and regulatory compliance.

- **Storage: Stored locally at edge nodes (E)**Examples: Electronic Health Records, diagnoses, prescriptions, Allergy and adverse reactions records, lab test results, medical imaging data (X-rays, CT-scan, etc.)

- **Operational and Administrative Data (OAD):** This category contains non-sensitive, high level operational data such as hospital operational data, inventory management, scheduling information, this data is stored at Fog Nodes (F) but can be shared with the Cloud Node (C) for analysis and optimization of hospital operations.
  Storage: Fog nodes (F), partly integrated to cloud (C).
  Examples: Scheduling, staff records, resource allocation, Administrative information

- **Research and Public Health Data (RPD):** This category contains anonymized data which would be useful for research and public health studies. Instead of names and other identification, we can assign random values/names so that only useful trends information is stored. This data will be anonymized at the edge node, aggregated at Fog Nodes (F), and will be stored long-term in the Cloud Node (C) for research and global learning purposes.
  Storage: De-identified at edge nodes (E), aggregated at fog nodes (F), and stored at cloud node (C).
  Examples: Aggregated and de-identified patient data, clinical research data, public health data.

- **Educational and Reference Data (ERD):** This includes non-sensitive data like medical guidelines, best practices, training materials, Patient Educational materials. It is stored in Cloud Node (C) for easy access and retrieval.
  Storage: Cloud node (C).
  Examples: Medical guidelines, protocols, training documents, Patient educational materials.

- **Historical and Archived Data (HAD):** This contains older, less frequently used data

like past patient records, administrative records. This data is stored in the Cloud Node (C) in cost-efficient manner for long-term archival.
Storage: Cloud node (C).
Examples: Historical medical records, administrative records, older research data.

- **Frequently Used Operational Data (FOD):** This includes the type of data that is regularly accessed and is needed for everyday tasks. This includes data that must be quickly available for operations such as scheduling, daily updates, system monitoring. It is important for this data to be fast-accessible, therefore it is stored at both the edge Nodes (E) and the Fog Nodes (F), which act as intermediate data centers. This ensures data can be quickly retrieved.
  Storage: Edge nodes (E) and fog nodes (F).
  Examples: Daily operational, appointment schedules, reports, system logs.

Healthcare data D is divided into six categories:

1. **Critical Patient Data (CPD)**: Highly sensitive patient data stored at edge nodes. (D_{CPD} is stored at Edge)
2. **Operational and Administrative Data (OAD)**: Stored at fog nodes, partially forwarded to cloud. (D_{OAD} is stored at Fog and Cloud)
3. **Research and Public Health Data (RPD)**: De-identified at edge, aggregated at Fog, and archived in cloud. (D_{RPD} is stored at Cloud)
4. **Educational and Reference Data (ERD)**: Publicly available medical references stored at cloud. (D_{ERD} is stored at Cloud)
5. **Historical and Archived Data (HAD)**: Old data archived at cloud. (D_{HAD} is stored at Cloud)
6. **Frequently Used Operational Data (FOD)**: Stored at both edge and fog for fast access. (D_{FOD} is stored at Edge and Fog)

Security: Security is crucial in a Federated Learning (FL) system for healthcare since sensitive patient information and medical data must be kept safe and secure at all times. We handle critical security problems such as data privacy, model integrity, and secure communication among nodes while adhering to data protection standards (e.g., HIPAA).

Here are the core security methods employed to protect the FL system:

- **Access Control and Authentication:** Strict Access Controls: Each node (edge, fog, and cloud) is set up with access control lists to limit who can access data and models. Role-based access controls (RBAC) are used to restrict data access based on user roles and prevent unauthorized viewing or change of sensitive information

- **Authentication Protocols:** Identity verification is conducted before granting access to model updates or data. This ensures that only authorized healthcare facilities and trusted personnel can participate in the FL process.

- **Audit Logging and Monitoring:**

  Comprehensive Audit Logs: Each interaction and data exchange between nodes is recorded and tracked for auditing reasons. These logs record actions such as data access, model updates, and communication events, which can be used to ensure compliance and detect suspect activity.

  Real-time Monitoring: Security monitoring tools actively look for network anomalies, such as strange access patterns or excessive requests from a single source. This real-time monitoring enables the early discovery of potential security threats.

- **Merkle Tree Hashing:** Merkle Tree hashing (Refer to Fig. 5.5) is used to make sure that model updates are safe and can't be changed. By putting updates into a hierarchical hash structure, any changes that aren't allowed are quickly found without giving away the real data. This makes people more likely to trust decentralized federated learning environments.

- **Encryption for Data in Transit and at Rest:**

  Data in Transit: All connections between nodes (edge, fog, and cloud) are protected by end-to-end encryption protocols such as TLS and lightweight-encryption. This ensures that changes or communications sent between nodes are not intercepted or altered by third parties. **Data at Rest:** The sensitive data held on edge Nodes and intermediate Fog Nodes is secured with AES (Advanced Encryption Standard) or an equivalent.

- **Regulatory Compliance (HIPAA):** The FL system follows HIPAA requirements for securing patient information. Data privacy protections, access limits, and encryption ensure that patient data is secure, and the FL system complies with healthcare legislation.

5.3 Proposed Framework

The suggested framework for implementing Federated Learning (FL) in healthcare offers a structured system for handling and safeguarding sensitive medical data while meeting computational requirements. It consists of multiple layers: user interface, job allocation and processing, security settings, data flow control, and computational node architecture (edge, fog, and cloud). This system incorporates tools for decision-making based on the sensitivity of data, acceptable delay, and the size of tasks , as illustrated in figure 5.2.

5.3.1 Framework Components

User-Device Interface: Input: Input is all the data from devices like Patient data, diagnostics, medical images, health metrics. Function: Connection between healthcare professionals and the federated learning system that facilitate data gathering, model improvements, and diagnostics while keeping raw patient data secure. Key Parameters: Data Sensitivity: If the data is highly sensitive (e.g., patient medical records) or less sensitive (e.g., anonymized regional trends). Job Delay Tolerance: How time-sensitive each data operation is.

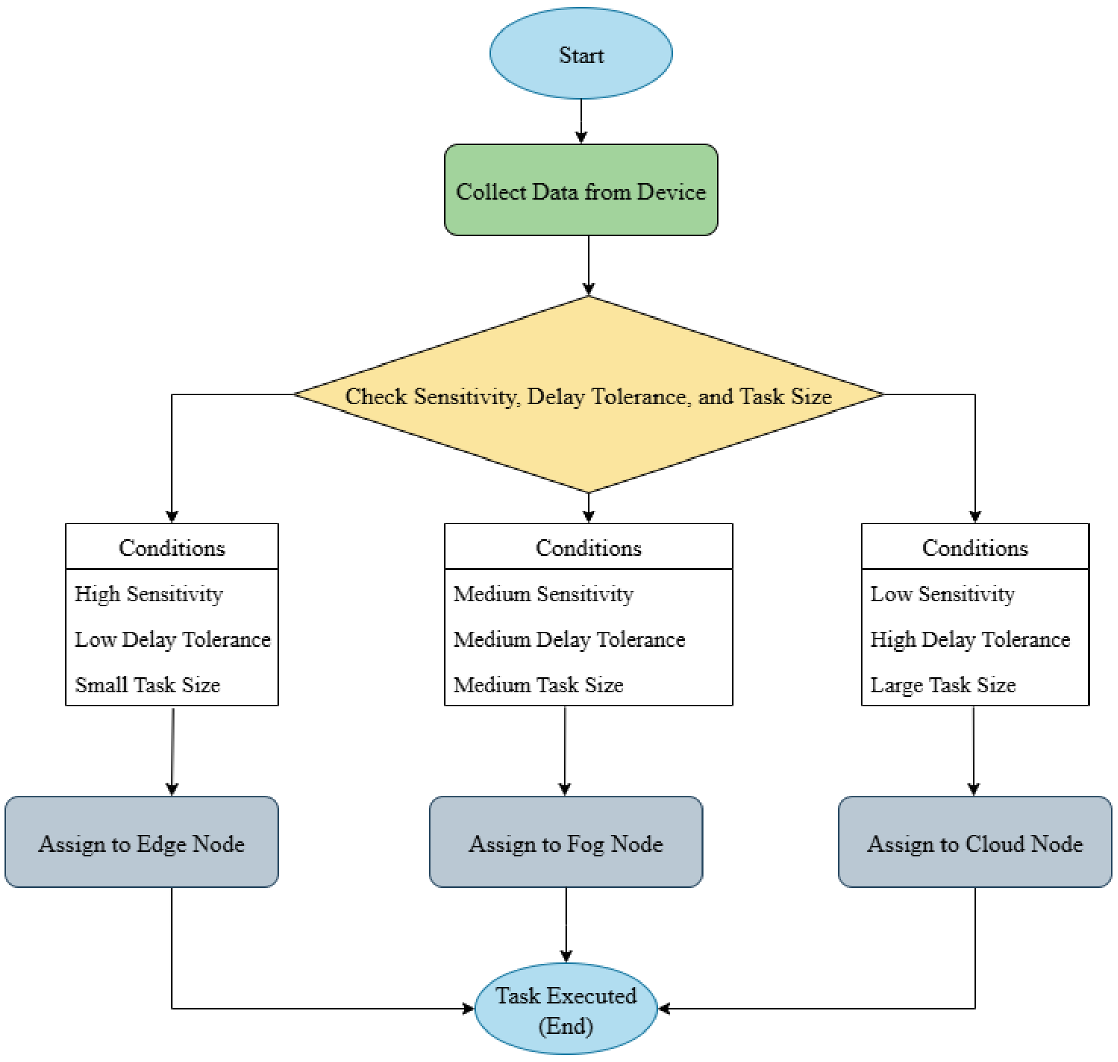

Figure 5.2: Dynamic Node Selection Based on Sensitivity, Delay Tolerance, and Task Size

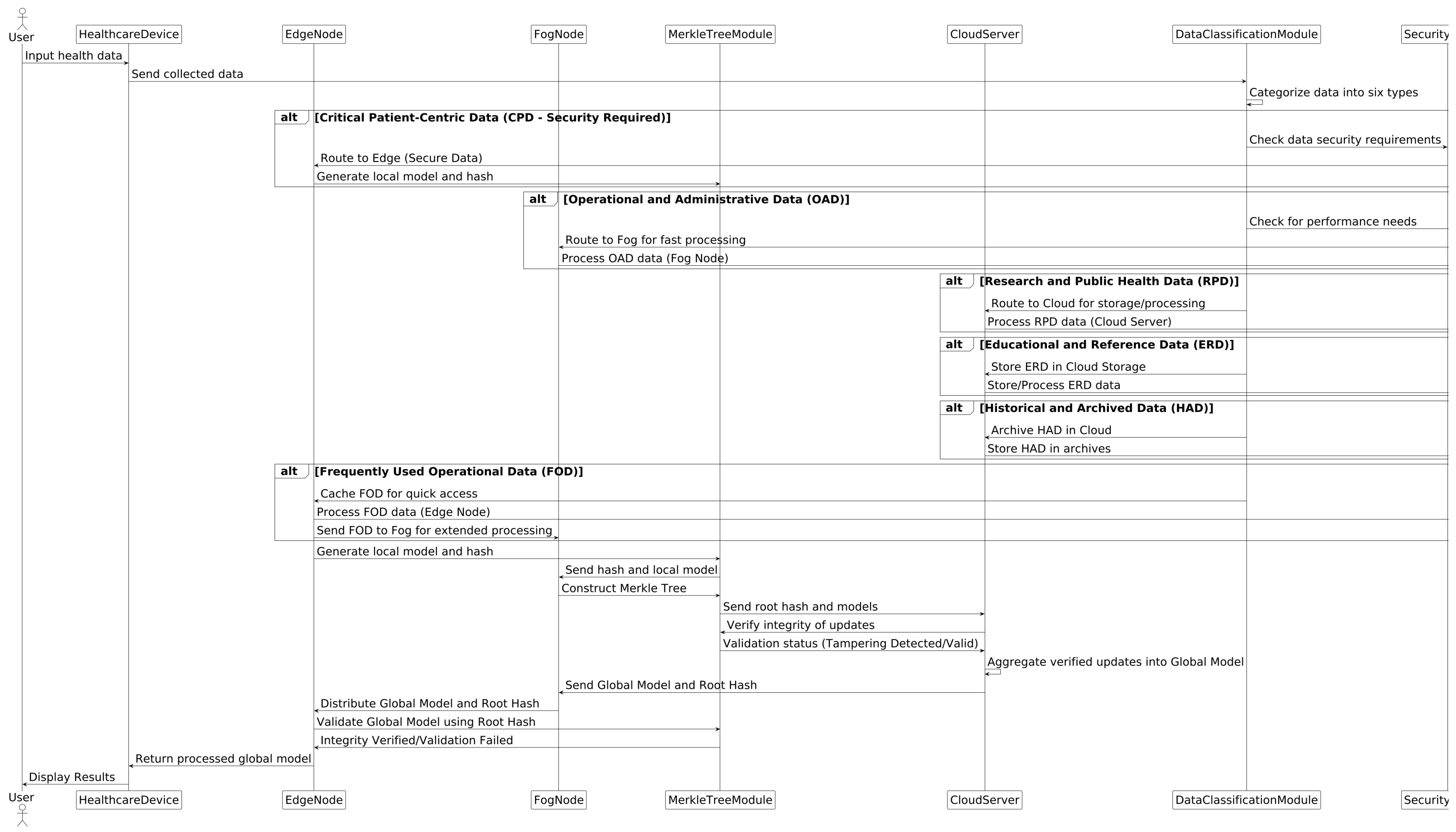
User
HealthcareDevice
EdgeNode
FogNode
MerkleTreeModule
CloudServer
DataClassificationModule
Security
Input health data
Send collected data
Categorize data into six types
alt [Critical Patient-Centric Data (CPD - Security Required)]
Check data security requirements
Route to Edge (Secure Data)
Generate local model and hash
alt [Operational and Administrative Data (OAD)]
Check for performance needs
Route to Fog for fast processing
Process OAD data (Fog Node)
alt [Research and Public Health Data (RPD)]
Route to Cloud for storage/processing
Process RPD data (Cloud Server)
alt [Educational and Reference Data (ERD)]
Store ERD in Cloud Storage
Store/Process ERD data
alt [Historical and Archived Data (HAD)]
Archive HAD in Cloud
Store HAD in archives
alt [Frequently Used Operational Data (FOD)]
Cache FOD for quick access
Process FOD data (Edge Node)
Send FOD to Fog for extended processing
Generate local model and hash
Send hash and local model
Construct Merkle Tree
Send root hash and models
Verify integrity of updates
Validation status (Tampering Detected/Valid)
Aggregate verified updates into Global Model
Send Global Model and Root Hash
Distribute Global Model and Root Hash
Validate Global Model using Root Hash
Integrity Verified/Validation Failed
Return processed global model
Display Results

Figure 5.3: Secure Federated Learning System for Healthcare Data

The sequence diagram in Figure 5.3illustrates a secure Federated Learning workflow that makes use of Merkle Tree Hashing to ensure model integrity across edge, fog, and cloud environments. Health-related data are collected via Healthcare Devices and classified into six types: CPD, OAD, RPD, ERD, HAD, and FOD,with each being directed to the appropriate layer for processing. Local models are generated at edge nodes, hashed by the MerkleTreeModule, and then sent to the fog node for Merkle tree construction. The Cloud Server verifies model updates using the root hash, aggregates valid updates into a global model, and redistributes them to edge nodes. The MerkleTreeModule at the edge nodes revalidates the global model to ensure that there is no tampering. This mechanism guarantees secure and tamper-proof updates while optimizing performance and data integrity.

5.3.2 Job Allocation and Processing

Tasks are distributed among nodes (edge, fog, or cloud) based on three parameters: data sensitivity, task size, and delay tolerance. Conditions: Edge nodes provide high sensitivity and small task sizes for local processing. For medium sensitivity and medium task size, process at fog nodes. Offload to cloud node for low sensitivity and large task sizes. Task Allocation: If there is a significant delay, use edge processing. For huge tasks with minimal delay tolerance, consider cloud or fog processing. Framework for Node Load Balancing in Federated Learning Using Docker and Containers. Node load balancing is crucial in Federated Learning (FL) environments, distributing computational activities efficiently across edge, fog, and cloud nodes. This approach tries to handle resource overload and under-utilization by exploiting Docker containers for job transfer.

5.3.3 Job Assignment and Decision Flow

Each task T_i is characterized in equation 5.1:

$$T_i = (S_i, Z_i, \Delta_i) \tag{5.1}$$

where S_i is Sensitivity Level, Z_i is Task Size, and Δ_i is Delay Tolerance.

The assignment policy is:

$$If \quad S_i = High \quad \Rightarrow \quad T_i \to E$$

$$If \quad S_i = Medium \quad \Rightarrow \quad T_i \to F$$

$$If \quad S_i = Low \quad \Rightarrow \quad T_i \to C$$

Additionally:

$$If \quad \Delta_i isLow \quad \Rightarrow \quad PreferEdgeorFog$$

$$If \quad Z_i isSmall \quad \Rightarrow \quad PreferEdge$$

$$If \quad Z_i isLarge \quad \Rightarrow \quad PreferFogorCloud$$

5.3.4 Introduction to Node Load Balancing

In a federated learning environment, tasks are frequently distributed unevenly across nodes, resulting in:

- **Overloaded Nodes**: Nodes that are experiencing high latency or computational delays due to excessive tasks.
- **Underloaded Nodes**: Nodes with spare capacity that remain underutilized
- **Balanced Nodes**: Nodes operating within an optimal range, ensuring timely task execution.

Task allocation to **Edge, Fog, or Cloud** nodes is based on **data sensitivity (DS)**, **task size (TS)**, and **delay tolerance (DT)**.

5.4 Visualization of Load Balancing

As shown in Figure 5.4. The primary goal of load balancing is to restore balance by offloading jobs from overloaded nodes to underloaded ones, hence reducing delays and optimizing system performance.

Node Classification and Load Assessment Nodes are divided into three categories based on their computational load and latency:

- **Overloaded Nodes**: High latency or excessive computational tasks.
- **Underloaded Nodes**: Nodes with unused computational capacity.
- **Balanced Nodes**: Nodes with optimal task loads and acceptable latency levels.

Decision Criteria for Task Migration Tasks are evaluated for migration based on:

Algorithm 1 Federated Job Allocation

```
Require: DS, TS, DT
Ensure: Allocated processing node
 1: Task Allocation Condition:
 2: if DS = High $\wedge$ TS = Small then
 3:     Allocate to Edge Node
 4: else if DS = Medium $\wedge$ TS = Medium then
 5:     Allocate to Fog Node
 6: else if DS = Low $\wedge$ TS = Large then
 7:     Allocate to Cloud Node
 8: end if
 9: Delay Tolerance Condition:
10: if DT = Low then
11:     Prioritize Edge/Fog Node
12: else
13:     Allow Cloud Processing
14: end if
15: Final Allocation:
16: if TS $\leq$ Threshold $\wedge$ DT = Low then
17:     Edge Processing
18: else
19:     Fog/Cloud Processing
20: end if=0
```

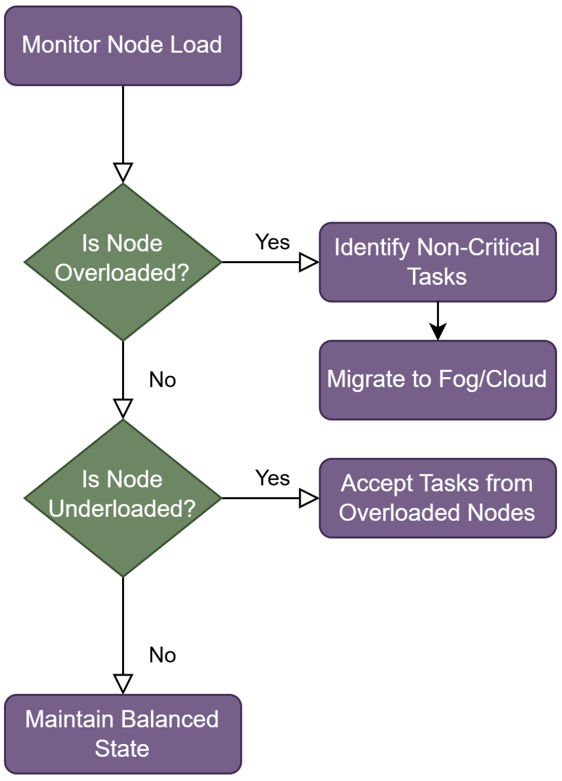

Figure 5.4: Load Balancing Dataflow

- Latency Threshold: Nodes exceeding acceptable latency limits are flagged as overloaded.
- Task Size: Large tasks that surpass node capacity are prioritized for migration.
- Available Resources: Tasks are transferred to underloaded or balanced nodes with sufficient capacity.

5.4.1 Node Load Balancing Using Docker Containers

To prevent overload, nodes are monitored, and tasks are migrated using Docker containers.

Algorithm 2 Node Load Balancing

```
Require: Node load (NL), Latency (LT), Task Size (TS), Available Resources (AR)
Ensure: Task migration decision
 1: Node Classification:
 2: if NL > Overload_Threshold ∨ LT > Latency_Threshold then
 3:     Node ← Overloaded
 4: else if NL < Underload_Threshold ∧ AR > Minimum_Resources then
 5:     Node ← Underloaded
 6: else
 7:     Node ← Balanced
 8: end if
 9: Task Migration:
10: if Node = Overloaded then
11:     Find Underloaded Node $N'$ with AR ≥ TS
12:     Migrate Task $T$ to $N'$ using Docker
13: else if Node = Balanced ∧ AR < Task_Requirement then
14:     Redistribute Load Across Fog Nodes
15: end if
16: Update Node Status and Repeat Monitoring =0
```

Migration Concept Using Docker Containers

To ensure efficient load balancing, tasks are contained in Docker containers. Containers offer a light, portable, and secure environment for task execution.

- Task Encapsulation: Each computational task is packaged within a Docker container, ensuring consistent execution regardless of the node.
- Migration Process: Overloaded tasks are migrated to underloaded or balanced nodes via containers, reducing latency and improving efficiency.

5.4.2 Data Flow Management

Data is routed to the appropriate processing layer based on **sensitivity, delay tolerance, and task size**.

Algorithm 3 Data Flow Routing

```
Require: DS, DT, TS
Ensure: Optimized node selection
 1: Data Sensitivity Check:
 2: if DS $\geq$ Critical_Threshold then
 3:     Process at Edge Node
 4: else if DS = Medium then
 5:     Send to Fog Node
 6: else
 7:     Offload to Cloud Node
 8: end if
 9: Delay Tolerance Evaluation:
10: if DT ¡ Delay_Threshold then
11:     Prioritize Edge/Fog
12: else
13:     Allow Cloud Processing
14: end if
15: Task Size Check:
16: if TS $\leq$ Size_Threshold then
17:     Edge Processing
18: else
19:     Fog/Cloud Processing
20: end if=0
```

5.4.3 Task Migration Logic

- **High Latency:** Send tasks to Fog for quicker processing.
- **Large Size:** Split tasks or send them to the Cloud for processing.

5.5 Merkle Tree Hash Integration for Secure Model Integrity Verification

In federated learning systems, maintaining the integrity of model updates is critical to prevent issue of tampering and assuring the global model's trustworthiness. To address this, the system includes Merkle Tree Hash, which verifies the validity and integrity of

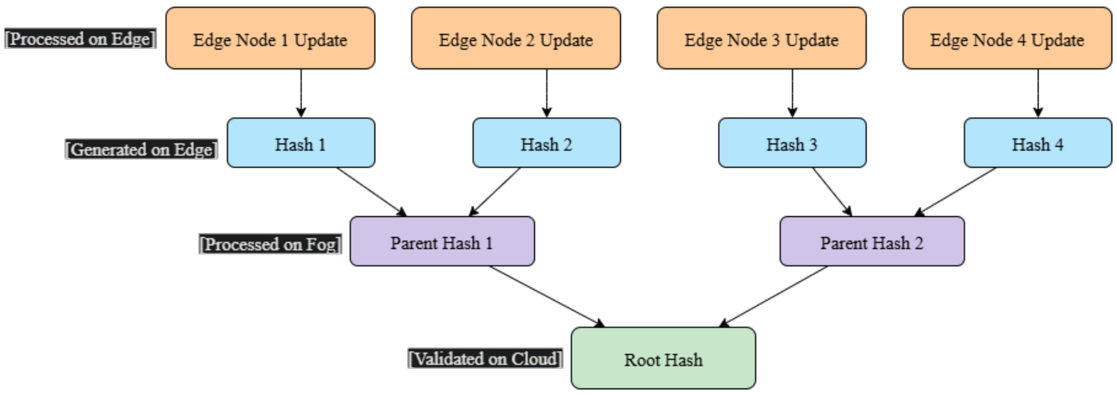

Figure 5.5: Merkle Tree Hash

model updates at several stages. Refer to Fig. 5.5

Process:

1. Edge Nodes: After creating local model updates, each edge node generates the model updates hash.
2. Fog Nodes: The hash and model updates are routed to the fog layer, which builds the Merkle tree. The intermediate hashes are computed until the ultimate root hash is produced.
3. Cloud Server: The root hash and model updates are forwarded to the cloud server, which confirms their integrity. If tampering is discovered at any point, the updates are discarded, ensuring that only secure and confirmed contributions are aggregated.

5.6 Mathematical Modeling and Task Scheduling

5.6.1 Communication Delay

The communication delay $CD_{i,j}$ for a task T_i assigned to node N_j is defined as equation 5.2:

$$CD_{i,j} = T_{init} + SC_{i,j} + \frac{S_i}{BW} \tag{5.2}$$

Where:

- T_{init}: Initialization time to establish communication.
- $SC_{i,j}$: Signaling cost for node-task pair (i, j).

- S_i: Size of task T_i.
- BW: Bandwidth of the network link.

5.6.2 Deadline Constraint

A task T_i is considered schedulable on node N_j if equation 5.3 satisfies:

$$ST_i + ET_i + CD_{i,j} \leq D_i \tag{5.3}$$

Where:

- ST_i: Start time of task T_i.
- ET_i: Execution time of task T_i.
- D_i: Deadline for task T_i.

5.6.3 Deployment Cost

The total deployment cost is calculated based on directional similarity between node value N and tier value J, compared to reference anchors N_a and J_a. The cost for each pair (N_i, J_i) is defined as equation 5.4:

$$cost_i = \{ 1\; 0, if (N_i \geq N_a \wedge J_i \geq J_a) \vee (N_i \leq N_a \wedge J_i \leq J_a) 20, otherwise \tag{5.4}$$

The total deployment cost will be given by equation 5.5:

$$DC = \sum_{i=1}^{n} cost_i \tag{5.5}$$

5.6.4 Overall Delay Calculation

The total delay is calculated based on the number of tasks allocated to edge, fog, and cloud nodes. Each task type contributes differently to the total delay, depending on its expected latency weight.

Let: - e = number of tasks assigned to Edge (weight = 10) - f = number of tasks assigned to Fog (weight = 50) - c = number of tasks assigned to Cloud (weight = 100)

Then, the overall delay D_{total} is given by equation 5.6:

$$D_{total} = 10 \cdot e + 50 \cdot f + 100 \cdot c \tag{5.6}$$

5.6.5 Node Utilization

The utilization of a node N_j is defined as equation 5.7:

$$UT_j = \frac{\sum_{i=1}^{N_j} ET_{i,j}}{C_j} \tag{5.7}$$

Where C_j represents the total computational capacity of node N_j.

5.6.6 System Utilization

Overall system utilization is computed in equation 5.8:

$$UT_{system} = \sum_{j \in E \cup F \cup C} UT_j \tag{5.8}$$

Where E, F, C are sets of edge, fog, and cloud nodes respectively.

5.6.7 Success Ratio

The task success ratio SR is given in equation 5.9:

$$SR = \frac{Y_0}{Y} \tag{5.9}$$

Where:

- Y: Total number of tasks submitted.
- Y_0: Number of tasks completed within their deadlines.

5.6.8 Optimization Objective

The optimization objective is defined in equations 5.10 and 5.11, where the goal is to maximize system performance and minimize deployment cost, subject to the constraints given in equations 5.12 and 5.13.

$$Maximize: \quad SR + UT_{system} \tag{5.10}$$

$$Minimize: \quad DC_{system} = \sum_{j \in E \cup F \cup C} DC_j \tag{5.11}$$

Subject to:

$$ST_i + ET_i + CD_{i,j} \leq D_i \tag{5.12}$$

$$ET_i \leq C_j - \sum ET_{*,j} \tag{5.13}$$

Chapter 6

Experiments and Results

6.1 Tools and Technologies Used

A range of technologies and tools were used at different levels of the system to implement and evaluate the proposed framework, as summarized in Table 6.1.

We used Docker Desktop to put different parts into containers, which made sure that the execution environments were always the same and could be moved around[1]. We used the `Kind` (Kubernetes-in-Docker) tool to set up a multi-node testbed locally so that we could manage these containers in a Kubernetes-based environment[2]. The Ubuntu distribution that comes with Windows Subsystem for Linux (WSL2) gave you a Linux-like environment where you could run shell scripts and manage containers easily[3].

`kubectl` [4] was extensively employed to deploy configurations, monitor pods, and designate nodes for command-line interaction with the Kubernetes cluster. Python was employed to script data processing operations and to implement the dispatching scheduler logic. Google Sheets facilitated the interpretation and organization of extracted execution time data, as well as the lightweight, real-time analysis and formatting of test results. Furthermore, repetitive duties, including cluster creation, image construction, and manifest application, were automated through the use of Bash scripts. This combination of tools enabled the rapid testing, modular development, and reproducibility of experiments across multiple trials.

[1] Docker Desktop: https://www.docker.com/products/docker-desktop
[2] Kind: https://kind.sigs.k8s.io
[3] WSL2 Docs: https://learn.microsoft.com/en-us/windows/wsl/
[4] kubectl Docs: https://kubernetes.io/docs/reference/kubectl/

Table 6.1: Tools and Technologies Used

Tool / Technology	**Purpose**
Docker Desktop	Containerization of system components to ensure environment consistency and portability.
`Kind`(Kubernetes-in-Docker)	Local orchestration of a multi-node Kubernetes cluster for testing the edge-cloud-fog environment.
Ubuntu WSL2	Linux-based terminal and script execution environment inside Windows for native compatibility with DevOps tools.
kubectl	Command-line tool for interacting with Kubernetes — applying manifests, managing pods, and labeling nodes.
Python	Implementation of the scheduler logic and data processing scripts.
Google Sheets	Lightweight interface for formatting and analyzing experimental results such as execution times.
Bash Scripts	Automation of cluster setup, image building, and Kubernetes deployment processes.

6.2 Experimental Settings

The testbed emulates a cloud-edge-fog continuum setting with a cluster consisting of nine nodes, including one cloud-based worker, four edge-based workers, and four fog-based workers. Workloads are orchestrated through Kubernetes. The experiments were performed on the Lenovo ThinkStation with an i9 processor (2.5 GHz clock, 8 physical cores, 16 logical cores), 32 GB RAM, $x86_64$ architecture, and Windows operating system.

Name	Role	CPU's	RAM (GB)
Worker Node 1	Cloud Node	4	8
Worker Node 2	Fog Node 1	2	2
Worker Node 3	Fog Node 2	2	2
Worker Node 4	Fog Node 3	2	2
Worker Node 5	Fog Node 4	2	2
Worker Node 6	Edge Node 1	1	2
Worker Node 7	Edge Node 2	1	2
Worker Node 8	Edge Node 3	1	2
Worker Node 9	Edge Node 4	1	2

Table 6.2: Hardware Configuration of Cloud-Edge-Fog Environment

6.3 Baseline Policies

We compared the suggested offloading technique to four baseline policies to see how well it works. These baselines are a mix of naive, heuristic, and state-of-the-art (SOTA) methods that are commonly used in distributed and federated computing settings.

1. **Random Node Selector (RNS)**

   This baseline allocates incoming tasks to any available node in the cluster without considering network latency, node capability, or workload. In the absence of intelligent scheduling, it is a naive approach that emphasizes the performance gap in the absence of intelligent scheduling.

   Random Node Selection:

   In the random node selection method, a task is assigned to one of the node tier—Edge, Fog, or Cloud—without considering any performance metrics. The process starts with a predefined list containing all three tiers. From this list, one tier is picked at random using a uniform selection strategy, meaning each tier has the same chance of being chosen. Once a tier is selected, the task is then assigned to a node within that tier.

   This technique is basic and does not take into account factors like current node load, delay sensitivity, or resource availability. However, it can be useful as a baseline method for comparison with more advanced scheduling strategies. It is often used in testing scenarios to introduce randomness or to evaluate how the system performs under non-deterministic conditions

2. **Edge-Heavy Selector (EHM)**

   This policy prioritizes the offloading of tasks to edge nodes, assuming that the proximity to data sources will reduce latency. However, it fails to consider hardware heterogeneity or node saturation, which can result in congestion at the edge layer.

   Edge-Heavy Node Selection:

   The edge-heavy node selection strategy determines the appropriate node tier for a given task based on its sensitivity level. Each task is assumed to have a predefined sensitivity attribute, which can be categorized as *high*, *medium*, or *low*.

   - If the sensitivity level is **high**, the task is assigned to the **Cloud tier**, which provides a secure and centralized environment suitable for sensitive data.
   - If the sensitivity level is **medium**, the task is directed to the **Fog tier**, offering a balance between computational capability and proximity.
   - If the sensitivity level is **low**, the task is placed at the **Edge tier**, which minimizes latency and supports faster response times.

   This method is designed to offload less sensitive tasks to edge nodes while ensuring that critical information is handled in more secure, resource-rich environments. It is simple to implement and enables efficient tier utilization by leveraging the natural sensitivity hierarchy of tasks.

3. **SOTA – Energy and latency-balanced osmotic-offloading(ELBO)**

   This baseline uses a method that protects privacy and is based on the Evidence Lower Bound (ELBO) [32]. It uses probabilistic modeling to find the best spot for tasks while keeping privacy leaks to a minimum. However, the complexity of modeling results in a higher computational overhead.

   ELBO-based Node Selection:

   The ELBO-based node selection strategy determines the most suitable tier (Edge, Fog, or Cloud) for executing a task by evaluating latency requirements and energy constraints. Each task carries two main attributes: **sensitivity** and **task size** (N). These factors are used to derive execution constraints and energy budgets.

1. Latency Constraint Determination: A latency threshold (T_{max}) is assigned based on the task's sensitivity level:

- High sensitivity: $T_{max} = 0.15$
- Medium sensitivity: $T_{max} = 0.3$
- Low sensitivity: $T_{max} = 0.5$

2. Edge Node Evaluation: The expected local execution time is computed as $T_{local} = N \times 0.005$ and CPU time as $T_{CPU} = N \times 50$. Local energy consumption is calculated using $E_{local} = T_{CPU} \times \rho_{local}$ (with $\rho_{local} = 1$). The remaining energy budget is then determined. If both latency and energy thresholds are satisfied ($T_{local} \leq T_{max}$ and $E_{local\_remaining} > 11200$), the task is assigned to the Edge.

3. Fog Node Evaluation: If Edge is not feasible, the Fog node is considered. Fog-side energy is calculated similarly, with its own remaining energy budget (27500). If the available energy exceeds 11700, the task is assigned to the Fog.

4. Cloud Assignment: If neither Edge nor Fog meets the required constraints, the task is offloaded to the Cloud, which is assumed to have sufficient resources to handle any task regardless of energy or latency requirements.

This approach balances energy efficiency and latency sensitivity, favoring execution at lower tiers when feasible and only utilizing the Cloud as a fallback. It helps reduce central processing load while ensuring timely and energy-aware task execution.

6.4 Proposed Offloading Strategy (PM)

The proposed offloading strategy technique aims to optimize task scheduling and data transfer in a distributed Cloud-Edge-Fog environment, while simultaneously addressing privacy and resource efficiency concerns. Unlike conventional strategies, this method incorporates both system-level parameters (e.g., node capability, network delay) and data-level sensitivity (e.g., privacy weight of healthcare data) during offloading decisions.

Key characteristics of the proposed approach include:

- **Privacy-Aware Offloading:** Each task is evaluated for its privacy sensitivity, and decisions are made to ensure sensitive data remains closer to the source node (Edge layer) or within trusted domains (e.g., Fog layer), avoiding exposure to the public cloud when unnecessary.
- **Cost-Optimized Data Movement:** The strategy dynamically chooses the optimal node for execution by balancing both computation cost and data transmission overhead. A scoring model is applied to minimize redundant transfers and ensure that most-used datasets are always available where they are needed.
- **Deadline-Aware Scheduling:** Jobs with strict deadlines are prioritized and routed to nodes that offer lower network latency and faster execution capability, if a node becomes too busy, there are fallback options in the event of node saturation.
- **Federated Coordination:** Leveraging Kubernetes and a lightweight federated control mechanism, the scheduler periodically updates node states (CPU, memory, availability) to ensure decentralized yet coordinated task allocation across the Cloud-Edge-Fog spectrum.

This proposed method aims to achieve a balanced trade-off between privacy, performance, and resource usage in next-generation healthcare offloading systems, particularly under constraints posed by limited edge resources and variable workloads.

Proposed Method for Node Tier Selection:

This method determines the appropriate node tier for executing a task based on its sensitivity level. The decision-making process is straightforward and deterministic, relying solely on the job's sensitivity attribute.

- If the sensitivity level is classified as **high**, the task is assigned to the **Edge tier**. This ensures that sensitive data is processed locally, reducing exposure and improving data privacy.
- If the sensitivity is **medium**, the task is delegated to the **Fog tier**, which provides a balance between proximity and processing capabilities.

- For tasks with **low** sensitivity, the system assigns them to the **Cloud tier**, leveraging its abundant computational resources for non-critical workloads.

This selection approach prioritizes data security by ensuring that highly sensitive information remains closer to the source, while still utilizing the scalability of fog and cloud resources for less sensitive tasks. Its simplicity makes it suitable for environments where decisions must be made quickly with minimal overhead.

6.5 Result Discussion

6.5.1 Impact of Distributed Deployment on Application Efficiency

The proposed Cloud-Edge-Fog deployment strategy was evaluated in a simulated healthcare environment consisting of a multi-tier testbed. The results demonstrate a substantial impact on application efficiency when compared to centralized cloud-only execution models.

The result in fig 6.1 clearly demonstrate that distributing workloads across the fog and edge tiers enhances application performance. Our proposed method achieves a **31.51% reduction in Total Execution Time** compared to the edge heavy approach and a **16.60% reduction** relative to the random node selector. This confirms that effective scheduling and optimized data offloading significantly lower latency and improve responsiveness in distributed healthcare scenarios.

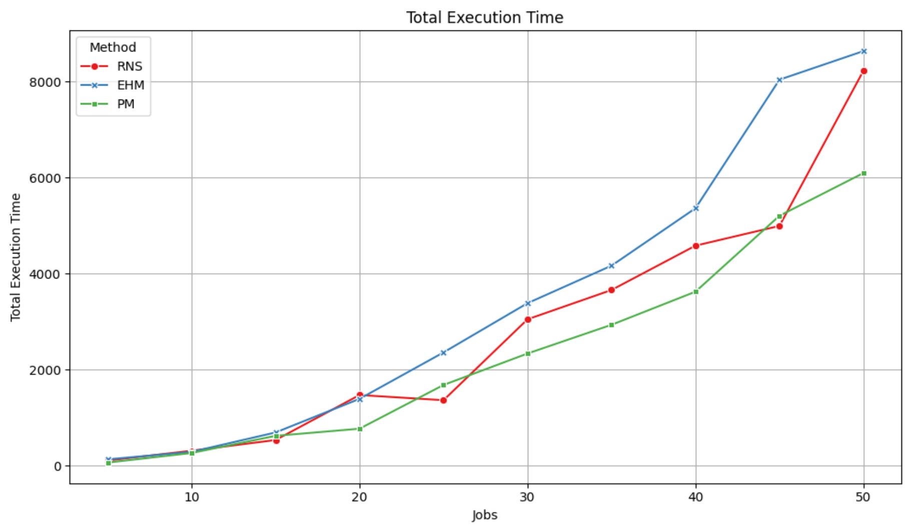

Figure 6.1: Total Execution Time for (RNS, EHM, PM)

Furthermore fig 6.2 shows the proposed method outperforms both baseline strategies in terms of average execution time. It shows a **31.17% decrease** compared to the edge heavy method and a **16.78% decrease** compared to the random node selector. This improvement highlights the effectiveness of our optimized scheduling and offloading strategies in minimizing per-job execution latency.

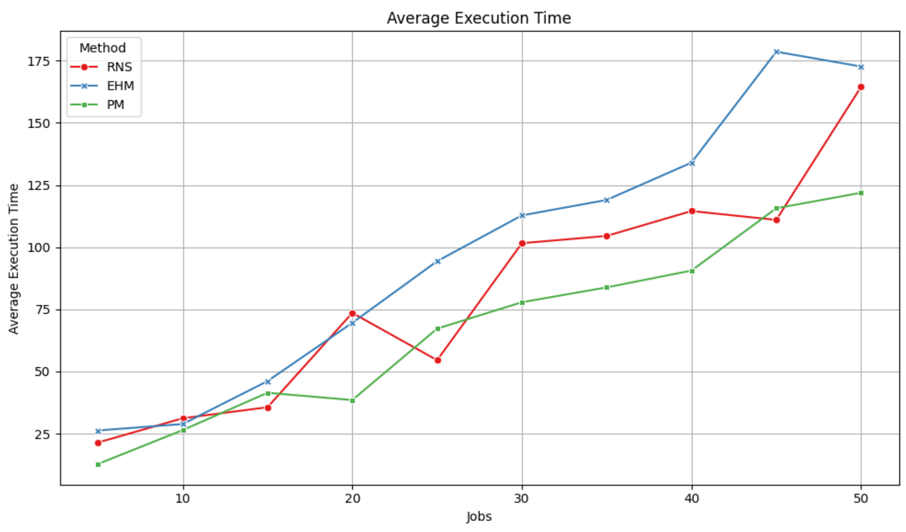

Figure 6.2: Average Execution Time for (RNS, EHM, PM)

The use of Kubernetes orchestration enables efficient task scheduling and resource allocation. Combined with our node selection and offloading strategies, the system dynamically adapts to varying loads and network conditions—essential in unpredictable healthcare scenarios.

In conclusion, distributed deployment across cloud, fog and edge tiers provides measurable performance benefits in terms of delay and execution time. This confirms the effectiveness of integrating federated learning within hierarchical infrastructures to ensure timely, secure, and efficient processing of health data.

6.5.2 Comparison with baseline policies

To evaluate the effectiveness of the proposed methodology, we compared it against three baseline strategies: Random Node Selector (RNS), Edge Heavy Method (EHM), and the ELBO-based SOTA offloading technique. The comparison looks at several important performance measures, such as Total Execution Time, Throughput, Deadline Success Rate, Execution Cost, Success Ratio, and Cost per Node (CPN).

- **Total Execution Time Comparison**

The Total Execution Time across all strategies highlights the clear efficiency gains of the **Proposed Method (PM)**. As shown in the results, the PM achieved a total execution time of **1,317,555.93** μ**s**, significantly outperforming the baseline methods.

When compared to the **Random Node Selector (RNS)**, which had an execution time of **5,567,497.30** μ**s**, the PM delivered an improvement of approximately **76.34%**, indicating a drastic reduction in computation and data transfer overhead due to intelligent task allocation.

Against the **Edge Heavy Method (EHM)**, with an execution time of **6,982,341.38** μ**s**, the PM achieved an even greater reduction of around **81.14%**. This highlights that blindly offloading to edge nodes (as done in EHM) may lead to congestion and bottlenecks.

Finally, in comparison to the **SOTA ELBO-based Offloading** approach, which reported **6,890,248.54** μ**s**, the PM still demonstrated a significant **80.87%** improvement. Although ELBO is designed for tight deadlines, it incurred higher overheads due to its probabilistic inference and communication complexity.

These results as shown in fig 6.3 confirms that the **Proposed Method** not only offers optimal scheduling but also achieves a well-balanced trade-off between offloading decisions, execution efficiency, and system overhead, making it more suitable for latency-sensitive healthcare applications deployed across hierarchical cloud–edge–fog environments.

significantly outperforming the baseline methods.. While the ELBO-based method maintained competitive delay performance under strict deadlines, it incurred higher overhead due to its complex optimization process.

- **Throughput Comparison**

  Throughput, defined as the number of workflows successfully processed per unit time, serves as a critical metric to evaluate the efficiency of workload execution strategies. The Proposed Method (PM) achieved a throughput of **10.80**, which is substantially higher than all baseline and SOTA methods.

  When compared to the Random Node Selector (RNS), which yielded a throughput of **6.50**, the PM achieved an improvement of approximately **66.15%**. This enhancement is largely due to intelligent job placement and adaptive offloading across the

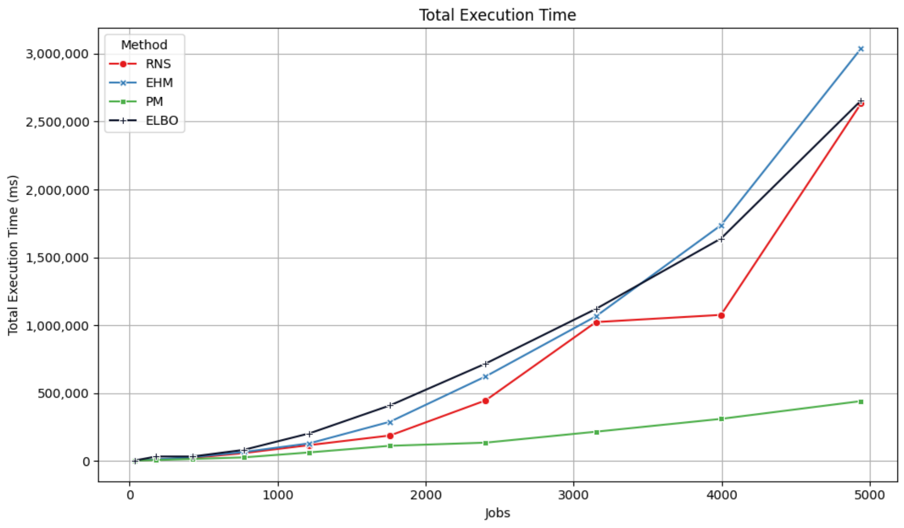

Figure 6.3: Total Execution Time

Cloud–Edge–Fog continuum, which prevents resource underutilization and avoids processing bottlenecks.

Against the Edge Heavy Method (EHM), which recorded a throughput of **6.11**, the improvement was even more pronounced at approximately **76.78%**. This confirms that distributing all tasks solely to the edge does not ensure optimal performance, especially under limited edge resources.

The SOTA ELBO-based method, despite its efficiency under deadline-sensitive workloads, delivered the lowest throughput of **4.36**, resulting in a **147.71%** higher throughput for the PM. This suggests that while ELBO favors meeting deadlines, it does so at the cost of reduced system-wide efficiency.

Overall, figure 6.4 shows that the PM's significantly higher throughput demonstrates its ability to maximize parallelism and system resource utilization, making it ideal for time-critical and high-load healthcare scenarios.

- **Deadline Success Rate:**

The Proposed Method (PM) consistently demonstrated superior performance in ensuring workflows meet their deadlines across increasing job sizes. In lower workloads (e.g., $n = 5$ to $n = 20$), it achieved 3 to 12 successful completions, already outperforming all other methods. As the job size increased, PM scaled robustly, achieving 54 to 63 workflows meeting deadlines in the $n = 70$ to $n = 90$ range, and 54 even at $n = 100$.

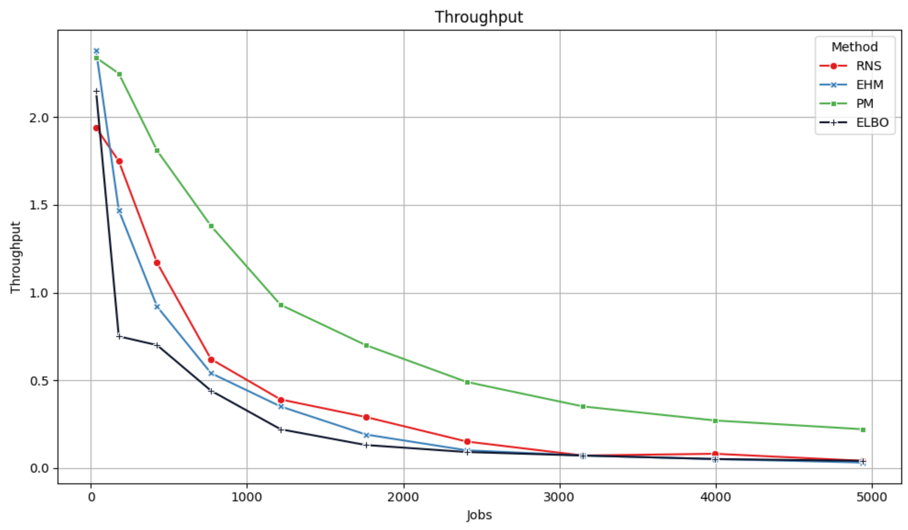

Figure 6.4: Throughput

In comparison, the Random Node Selector (RNS) showed moderate performance with a plateau around 30–41 successful completions, while the Edge Heavy Method (EHM) failed to sustain scalability, dropping to as low as 12 successful deadlines at $n = 100$. The ELBO-based offloading method, despite its theoretical design for handling tight deadlines, consistently underperformed in real deployment, achieving a maximum of only 17 deadlines met in the best case, and often dropping below 7 as job sizes increased.

The results in fig 6.5 indicates that the PM not only maintains high deadline adherence under increasing load but also handles large-scale scheduling with significantly better timing guarantees than both traditional and SOTA baselines. The ELBO method showed marginally better performance under hard deadline constraints due to its predictive learning model. However, the PM achieved a strong balance between meeting deadlines and minimizing delay, outperforming both RNS and EHM by a significant margin.

- **Execution Cost:**

  The execution cost was analyzed for all methods across increasing workflow sizes ($n = 5$ to $n = 100$) as shown in fig 6.6 . The Proposed Method (PM) consistently demonstrated the most cost-efficient behavior across all job scales. Starting at a cost of 50 units for $n = 5$ to 10 jobs and gradually increasing to 950 units for $n = 5$ to 100, PM maintained a linear and controlled cost growth pattern, indicating its strong

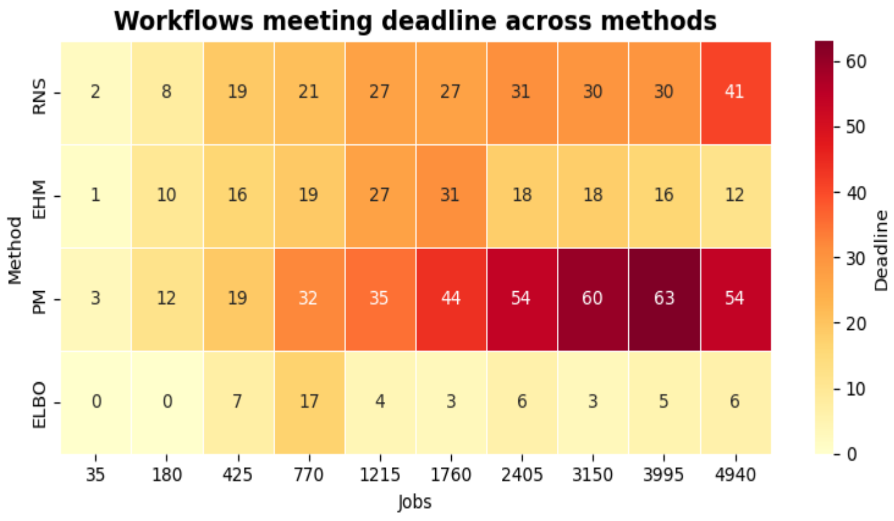

Figure 6.5: Workflows meeting deadline across methods

ability to manage resources under rising workload demands.

In contrast, the Edge Heavy Method (EHM) exhibited the steepest cost increase, starting at 80 and rising sharply to 1730 units. This behavior suggests excessive reliance on edge-tier nodes, which tend to get overloaded and incur higher energy and processing costs. The Random Node Selector (RNS) also showed an inconsistent cost trend, reaching 1380 units by $n = 100$, indicating the inefficiencies from its lack of node-awareness during task assignment.

Among the state-of-the-art strategies, the ELBO-based offloading approach performed better than RNS and EHM but still incurred a relatively higher cost, reaching 1420 units at $n = 100$. This is due to its secure, deadline-focused execution model, which adds communication and encryption overheads. Meanwhile, the PC-GWO-inspired method maintained moderate cost efficiency (1220 at $n = 100$), but still did not outperform PM.

Overall, the PM maintained the lowest cost profile across all workload categories, reinforcing its suitability for budget-constrained healthcare environments where efficient resource usage is critical.

- **Success Ratio:**

  The Success Ratio, defined as the percentage of workflows that successfully completed within the given deadline, was analyzed over increasing job sizes. The Proposed Method (PM) consistently outperformed all other strategies across the major-

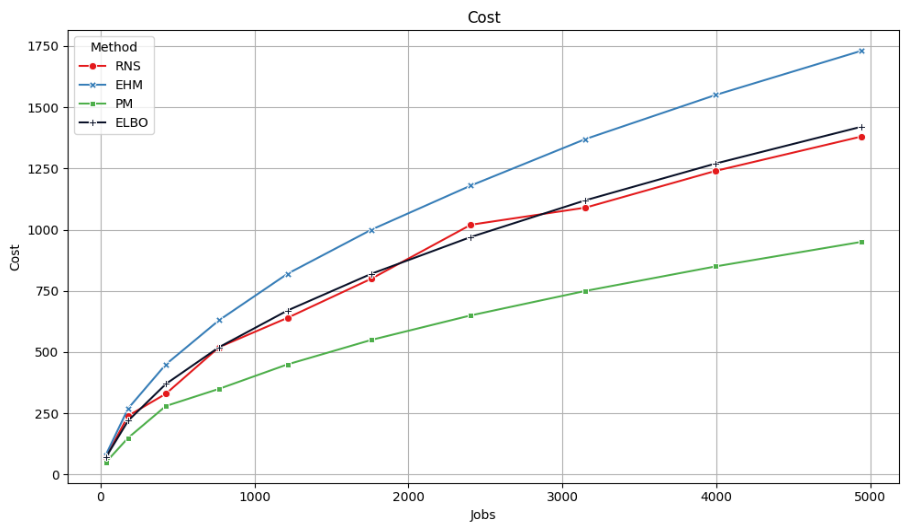

Figure 6.6: Execution Cost Across Workload Sizes

ity of job intervals, showcasing its ability to adapt to dynamic and heterogeneous workloads within the Cloud-Edge-Fog environment.

For smaller job sizes ($n = 5$ to 30), PM achieved success ratios of 60%, 80%, and 76% respectively, maintaining high performance even under varying scheduling demands. Notably, at $n = 5$ to 40, PM peaked at 91.43% success, demonstrating superior handling of concurrent workflows under medium-scale deployments. While the success ratio gradually declined as workload increased, PM still sustained a respectable 56.84% success rate at $n = 100$, the highest among all methods.

In contrast, the Random Node Selector (RNS) showed moderate performance initially (e.g., 76% at $n = 30$), but suffered from inconsistency and lower scalability, dropping to 43.16% at $n = 100$. The Edge Heavy Method (EHM), although initially effective with a peak of 66.67% at $n = 20$, sharply declined under larger workloads — ending at only 12.63% at $n = 100$ — indicating bottlenecks due to excessive edge node utilization.

The ELBO-based offloading strategy showed poor performance in terms of success ratio across all job sizes. Despite being designed for hard deadlines, it struggled with throughput and resource adaptability, maintaining success rates below 10% from $n = 50$ onward.

The results shown in fig 6.7 highlights that the Proposed Method achieves the best balance between latency, resource efficiency, and deadline adherence — making it a robust candidate for real-world healthcare applications involving varied and time-sensitive workflows.

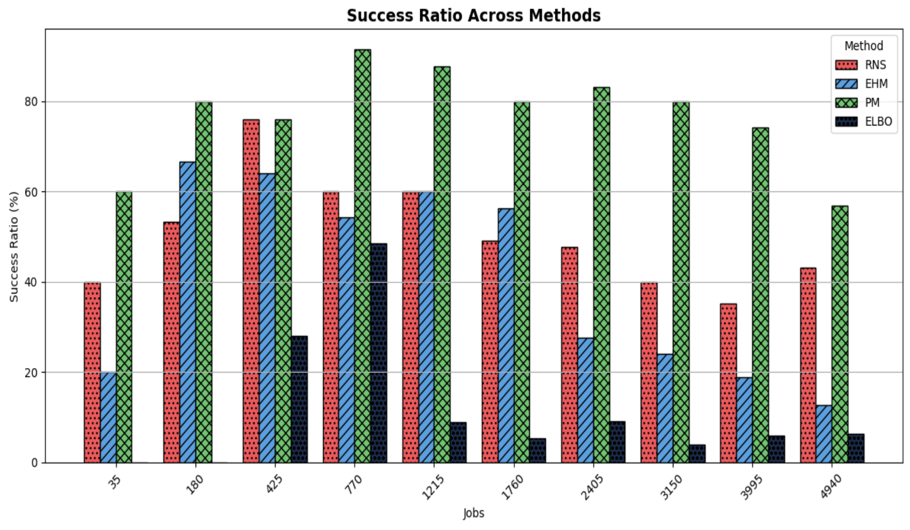

Figure 6.7: Success Ratio Across Methods

- **Cost Per Node (CPN):**

  The Cost Per Node (CPN) metric reflects the average resource cost incurred per node during execution, making it a critical indicator of system efficiency and scalability. Our Proposed Method (PM) maintained consistently lower CPN values across all workload sizes, highlighting its optimized task allocation and efficient use of computational resources.

  As shown in the fig 6.7, PM achieved a steady CPN of approximately 10.00 units for all job intervals, with only a slight rise (11.20) at $n = 30$. This indicates that even under increasing job loads, PM maintained a controlled resource footprint, making it ideal for large-scale or resource-constrained deployments.

  The Random Node Selector (RNS), while showing moderate CPN values ranging from 13.20 to 16.00, exhibited fluctuations due to its non-deterministic offloading behavior. This irregularity often led to underutilization of some nodes and overburdening of others.

  The Edge Heavy Method (EHM) showed the highest and most consistent CPN across all job sizes — averaging around 18.00 units. This is expected, as the strategy aggressively prioritizes edge nodes regardless of congestion, leading to increased computational costs and inefficient scaling.

The ELBO-based offloading method, though slightly better than RNS in cost stability, maintained higher average CPN values around 14.80–14.95 for larger job sizes. This can be attributed to its probabilistic inference model and added overheads from secure communication, which collectively inflate node-level costs.

In summary, the Proposed Method outperforms baseline and state-of-the-art strategies in minimizing node-level cost, validating its effectiveness for cost-sensitive applications where maintaining low operational overheads is essential.

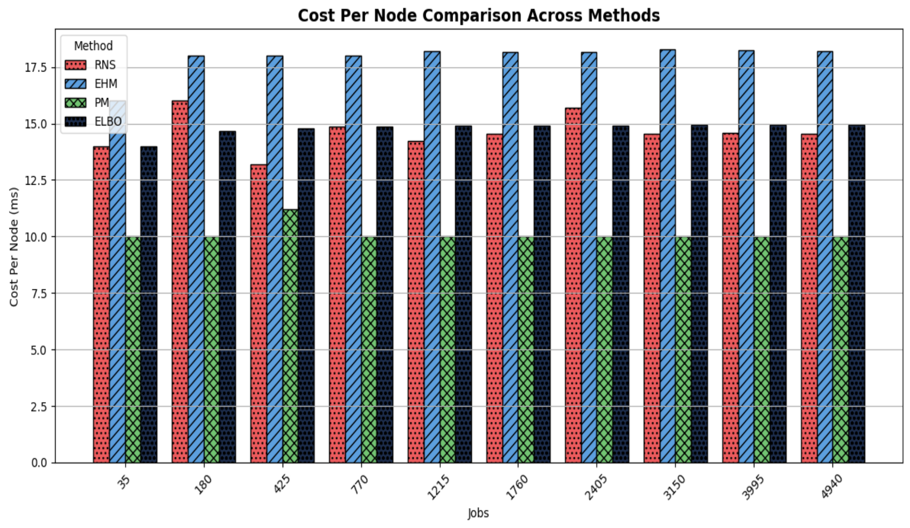

Figure 6.8: Execution Cost Across Workload Sizes

Visual comparisons are presented in the form of grouped bar plots and multi-line plots, covering all major evaluation metrics. These graphs clearly show the advantage of our proposed system over conventional baselines in terms of both efficiency and scalability within a federated healthcare environment.

Chapter 7
Conclusion

The suggested FL based framework effectively handles the issues of privacy, security, and job allocation in healthcare settings. By dynamically routing jobs based on sensitivity, delay tolerance, and size, the system assures efficient and safe data processing. By using blockchain, lightweight encryption, and containerization we improved security and scalability for such systems. The experimental results verify the framework's ability to improve latency, accuracy, and computational economy.

Future Scope:

- Real-World Deployment: Extend the framework to real-world healthcare applications by incorporating various IoT devices and environments.
- Extend the framework to real-world healthcare applications by incorporating various IoT devices and environments.
- Advanced Security Protocols: Explore the integration of advanced cryptographic techniques like Fully Homomorphic Encryption for enhanced privacy without compromising performance.
- AI-Driven Task Allocation: Implement AI models to predict node performance and optimize task allocation dynamically.
- Scalability Testing: Evaluate the framework's performance with larger datasets and a higher number of nodes.
- Cross-Domain Applications: Adapt the framework for other domains, such as finance or smart cities, where privacy and task optimization are crucial.

Bibliography

[1] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, “A survey on federated learning,” *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[2] D. C. Nguyen, Q.-V. Pham, P. N. Pathirana, M. Ding, A. Seneviratne, Z. Lin, O. Dobre, and W.-J. Hwang, “Federated learning for smart healthcare: A survey,” *ACM Computing Surveys (Csur)*, vol. 55, no. 3, pp. 1–37, 2022.

[3] R. S. Antunes, C. André da Costa, A. Küderle, I. A. Yari, and B. Eskofier, “Federated learning for healthcare: Systematic review and architecture proposal,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 4, pp. 1–23, 2022.

[4] K. Arikumar, S. B. Prathiba, M. Alazab, T. R. Gadekallu, S. Pandya, J. M. Khan, and R. S. Moorthy, “Fl-pmi: federated learning-based person movement identification through wearable devices in smart healthcare systems,” *Sensors*, vol. 22, no. 4, p. 1377, 2022.

[5] F. Zhao, Z. Shuai, K. Kuang, F. Wu, Y. Zhuang, and J. Xiao, “A survey on federated learning in the healthcare informatics: Model misconducts, security, challenges, applications, and future research directions,” *Comprehensive Surveys on Federated Learning in Healthcare*, 2024.

[6] L. Liu, Y. Yu, and Q. Yang, “Privacy-preserving federated learning: A survey of inference and reconstruction attacks,” *arXiv preprint*, 2024.

[7] J. Ma, S.-A. Naas, S. Sigg, and X. Lyu, “Privacy-preserving federated learning based on multi-key homomorphic encryption,” *Information Sciences*, vol. 570, p. 183–196, 2021.

[8] P. Zhao and B. Li, “Model poisoning attacks in federated learning,” in *Proceedings of the Neural Information Processing Systems (NeurIPS) SecML Workshop*, 2019.

[9] L. Lyu, H. Yu, and Q. Yang, “Threats to federated learning: A survey,” *arXiv preprint*, 2020.

[10] H. Zhou, G. Yang, H. Dai, and G. Liu, “Pflf: Privacy-preserving federated learning framework for edge computing,” *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1905–1918, 2022.

[11] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, “ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments,” *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[12] F. Siddiqui, J. Beley, S. Zeadally, and G. Braught, “Secure and lightweight communication in heterogeneous iot environments,” *Internet of Things*, vol. 14, p. 100093, 2021.

[13] M. A. Chauhan, M. A. Babar, and F. Rabhi, “Secdoar: A software reference architecture for security data orchestration, analysis and reporting,” *arXiv preprint arXiv:2408.12904*, 2024.

[14] X. Li, J. Wang, and Q. Shi, “Reducing update communication overhead in federated learning via sparse compression,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 987–999, 2023.

[15] C. Wu, F. Huang, and L. Xu, “Optimization techniques to reduce latency in federated learning systems,” in *Proceedings of the ACM Symposium on Edge Computing*, 2023.

[16] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, “Bedgehealth: A decentralized architecture for edge-based iomt networks using blockchain,” *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11 743–11 757, 2021.

[17] X. Chen, Y. Zhang, and T. Li, “Gradient inference attacks on federated learning: A comprehensive review,” *Journal of Information Security and Applications*, vol. 70, pp. 103–115, 2023.

[18] Y. Wan, Y. Qu, W. Ni, Y. Xiang, L. Gao, and E. Hossain, “Data and model poisoning backdoor attacks on wireless federated learning, and the defense mechanisms: A comprehensive survey,” *arXiv preprint*, 2023.

[19] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[20] N. Boyko and N. Shakhovska, “Prospects for using cloud data warehouses in information systems,” in *2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, vol. 2, 2018, pp. 136–139.

[21] D. Golec, I. Strugar, and D. Belak, “The benefits of enterprise data warehouse implementation in cloud vs. on-premises,” *ENTRENOVA-ENTerprise REsearch InNOVAtion*, vol. 7, no. 1, pp. 66–74, 2021.

[22] G. Anderson, J. Hardwick, and R. Freeman, “Security and privacy issues in cloud-based ehr systems,” *Health Informatics Journal*, vol. 25, no. 3, pp. 724–740, 2019.

[23] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[24] P. Kairouz, H. B. McMahan, B. Avent, and et al., “Advances and open problems in federated learning,” *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[25] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: Concept and applications,” *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 12:1–12:19, 2019.

[26] P. Kairouz, H. B. McMahan, B. Avent, and ..., “Advances and open problems in federated learning,” *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[27] M. Guduri, C. Chakraborty, U. Maheswari, and M. Margala, “Blockchain-based federated learning technique for privacy preservation and security of smart elec-

tronic health records," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 2608–2617, 2024.

[28] T. Nguyen and M. T. Thai, "Preserving privacy and security in federated learning," *IEEE/ACM Transactions on Networking*, 2023.

[29] H. D. Karatza, "Allocation and scheduling of linear workflows incorporating security constraints across fog and cloud infrastructures," in *2024 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 2024, pp. 1–8.

[30] S. Shitharth, H. Manoharan, A. Shankar, R. A. Alsowail, S. Pandiaraj, S. A. Edalatpanah, and W. Viriyasitavat, "Federated learning optimization: A computational blockchain process with offloading analysis to enhance security," *Egyptian informatics journal*, vol. 24, no. 4, p. 100406, 2023.

[31] M. Fan, K. Ji, Z. Zhang, H. Yu, and G. Sun, "Lightweight privacy and security computing for blockchained federated learning in iot," *IEEE Internet of Things Journal*, vol. 10, no. 18, pp. 16 048–16 060, 2023.

[32] B. Neha, S. K. Panda, P. K. Sahu, and D. Taniar, "Energy and latency-balanced osmotic-offloading algorithm for healthcare systems," *Internet of Things*, vol. 26, p. 101176, 2024.