INDEX

LIST OF FIGURES

BLOCKCHAIN-BASED FEDERATED LEARNING WITH SMPC MODEL VERIFICATION AGAINST POISONING ATTACK FOR HEALTHCARE SYSTEMS

ABSTRACT

Due to the rising awareness of privacy and security in machine learning applications, federated learning (FL) has received widespread attention and applied to several areas, e.g., intelligence healthcare systems, IoT-based industries, and smart cities. FL enables clients to train a global model collaboratively without accessing their local training data. However, the current FL schemes are vulnerable to adversarial attacks. Its architecture makes detecting and defending against malicious model updates difficult. In addition, most recent studies to detect FL from malicious updates while maintaining the model's privacy have not been sufficiently explored. This paper proposed blockchain based federated learning with SMPC model verification against poisoning attacks for healthcare systems. First, we check the machine learning model from the FL participants through an encrypted inference process and remove the compromised model. Once the participants' local models have been verified, the models are sent to the blockchain node to be securely aggregated. We conducted several experiments with different medical datasets to evaluate our proposed framework.

CHAPTER-1

INTRODUCTION

The Internet of Things (IOT) has been applied in various services, including the healthcare domain. The integration of IOT in the healthcare system is also known as the Internet of Medical Things (IOMT). With the development of IOMT, many healthcare devices are interconnected, allowing devices to exchange information among medical experts and Artificial Intelligence (AI) based services. This interconnectivity helps healthcare industries like hospitals to improve the efficiency and quality of their services. In the medical diagnosis field, medical imaging devices facilitate the process of early diagnosis and treatment for medical staff.

Due to this interconnectivity, medical image retrieval is made easy, resulting in extensive data with wide variations. Consequently, medical image analysis has become a challenging task for medical experts and is prone to human error. In recent years, the success of Deep Learning (DL) in computer vision tasks has provided a significant breakthrough in medical image classification tasks. Several studies of DL in medical imaging fields have shown promising results by providing accurate and efficient diagnoses [1].

As shown in Figure 1, cloud computing is one paradigm that emerged to solve the availability of computing and storage resources. Therefore, the cloud is usually used to deploy the DL model for training and data inference. However, sending the raw data from the IOMT cluster to the cloud will be very expensive. This is where edge computing, like edge servers, will be advantageous to process the data before sending it to the cloud.

It is known that a high-performing Deep Learning (DL) model requires a large and diverse dataset for its training. This large-scale dataset is often obtained from multi-institutional or multi-national data accumulation and voluntary data sharing in the healthcare industry. While massive data collection is essential for the deep learning process, sharing patients' data raises privacy concerns and relative regulations such as the General Data Protection Regulation (GDPR) and Health Insurance Portability and Accountability Act (HIPAA). Due to the rising concerns, healthcare institutions may be prevented from sharing their medical datasets. In some cases where sharing is

possible, some restrictions are applied, resulting in inadequate data sharing.

In recent studies, [2] proposed a federated learning model that allows parties to collaboratively train a model by sharing local model updates with a parameter server. Intuitively, this method is safer than centralized training because machine learning models learn from healthcare IOMT data without relying on a third-party cloud to hold their data [3]. However, federated learning also presents some challenges that may limit its applications in real-world case scenarios. For example, federated learning remains vulnerable to various attacks that may result in leakage of private data [4] or poisoned learning model [5]. Also, the participants in the current FL setup cannot verify the authenticity of the machine learning model. To protect FL participants' privacy, the existing defense method mainly focuses on ensuring the confidentiality of the machine learning gradients. Differential Privacy (DP) [6], [7] is one of the commonly used methods to preserve the privacy of the learning model. Adding DP to a federated learning scenario can improve the privacy of the participants models. However, adding noise into machine learning gradients will reduce the learning model accuracy [7]. DP is also ineffective in mitigating poisoning attacks while maintaining model performance resulting in a faulty global model. To tackle the poisoning attack, existing research on anomaly detection [8],[9] has been explored. However, the existing methods cannot eliminate all the poisoned models and cause the accuracy of the global model to be reduced. Also, they perform the anomaly detection method in a plaintext model. This will lead to another issue where the attacker can perform a parameter stealing attack [10] and a membership inference attack [11]. Thus, a verifiable and secure anomaly detection method for federated learning scenarios is needed.

This paper proposes a privacy-preserving verification method to eliminate poisoned local models in a federated learning scenario. The proposed method eliminates the compromised local model while guaranteeing the privacy of the local model's parameters using an SMPC-based encrypted inference process. Once the local model is verified, the verified share of the local model is sent to the blockchain for the aggregation process. SMPC-based aggregation is used to perform the secure aggregation between the blockchain and the hospital. After the aggregation process, the global model is stored in tampered-proof storage. Later, each hospital receives the global model from the blockchain and verifies the authenticity of the global model. The contributions of our work are summarized as follows:

_ Propose a new block chain-based federated learning architecture for healthcare systems to ensure the security of the global model used for classifying disease.

_ Design a privacy-preserving method for local model anomaly detection in a Federated learning scenario with SMPC as the underlying technology. Our encrypted model verification method eliminates the poisoned model while protecting the local model privacy from membership inference attacks and parameter stealing.

_ Propose an SMPC-based secure aggregation in the block chain as a platform to decentralize the aggregation process.

_ We present a verifiable machine learning model for federated learning participants using block chain in the IOMT scenario.

The rest of this paper is organized as follows. Section II defines the problem and design goals. Section III discusses the related work. Then, we present the system architecture and introduce the proposed frameworks in Section IV. Next, we describe the experimental setup and evaluation results of the proposed work in Section V. Finally, a conclusion is drawn in Section VII.

II. PROBLEM SCENARIO AND DESIGN GOALS

A. Problem Scenario

To discuss and highlight the current issues with current federated learning, we use an IOMT-enabled hospital scenario (see Fig. 2). Assume that several smart hospitals are placed in different regions with varying patient demographics and diseases. Each smart hospital is equipped with a cluster of IOMT devices. The IOMT devices will be used to scan the patient to detect a severe disease. In The current IOMT scenario, IOMT devices will act as data sources since the IOMT devices are resource-constrained and cannot perform any machine learning algorithm. Hence, each hospital has an edge server with computing resources to execute the machine learning tasks using the local datasets. Nevertheless, due to dataset limitations, the machine learning model accuracy generated from the local datasets is relatively low. Therefore the edge server from each hospital participates in the federated learning platform. In the federated learning platform, locally trained models from the hospital's edge server are collected and aggregated to produce a highly accurate machine learning model without sending private datasets to the cloud provider. Later, the aggregated or global model is sent back to the edge server for another round of federated learning processes. Once the global model reaches the desired accuracy, it will be used to recognize the disease more

accurately.

Although the aforementioned federated learning scenario improves the overall machine learning accuracy, it suffers from the following security risks:

_ Risks of local model security: In the current setup of federated learning, every party that sends their local model is sent to the cloud for the aggregation process without checking the model's validity. This traditional FL method introduces the risk of a local model being poisoned. For example, an attacker can perform a poisoning attack and train the model using poisoned data, leading to a faulty local model. Since healthcare data are critical, sending plaintext local models to the cloud can pose privacy risks. Therefore, validating and securing the local model is required to prevent it from various security aspects.

_ Risks of generating a biased aggregated model: The model aggregation process of the local model is performed on the cloud services that can be tampered with and produce a biased global model. For example, an attacker can include a poisoned local model during the aggregation process that may lead the global model to have a false classification. Hence, a secure aggregation method is required to encounter the current security problem.

_ Risk of receiving faulty global model: In the existing federated learning method, the global model generated from the cloud will be sent back to each edge server in the hospitals. However, the hospital cannot verify the global model they received. The attacker can intercept and alter the global model. As a result, the hospital received a faulty global model. From this problem, a global model verification method is required to ensure the integrity of the global model.

B. Design Goals With the risks and threats mentioned above, our goals for preserving privacy in Federated Learning can be decomposed into three aspects as follows:

_ Robustness: The proposed work should have the ability to prevent the adversary from poisoning federated learning. This allows the federated learning participant to learn from a benign global model to improve their model accuracy. Also, a robust aggregation method needs to be developed to secure the aggregation process from an attacker.

_ Privacy: The prior work [12] has shown that an attacker can perform a poisoning attack to decrease the global model accuracy by miss-classifying the machine learning model.To protect the federated learning participants, checking the participant's local learning model while maintaining the local model privacy itself is essential.

_ Verifiability: The designed method should have the ability to verify the machine learning model, specifically the global model. Since the adversary may alter or poison the global model. In the current federated learning scenario, the participant received the global model from the cloud without knowing the model's authenticity.

CHAPTER-2
LITERATURE SURVEY

TITLE: Secure Federated Learning with Blockchain: A Comprehensive Survey
AUTHORS: Xinyi Zhou, Qinghua Lu, Yulei Sui, Liming Zhu, and Xuyun Zhang
ABSTRACT: Federated learning (FL) has gained traction as a solution for privacy-preserving machine learning, allowing multiple participants to collaboratively train models without sharing their private data. However, FL is inherently vulnerable to various security threats, including data and model poisoning attacks, which can significantly degrade model performance and reliability. Blockchain technology, with its decentralized and immutable nature, offers a promising approach to enhance the security of FL systems. This survey provides a comprehensive overview of the intersection of FL and blockchain technology. It begins by discussing the fundamental principles of FL and blockchain, followed by an exploration of security and privacy challenges in FL. The survey reviews existing blockchain-based FL solutions, highlighting how blockchain can be leveraged to secure model updates, ensure data integrity, and provide transparency and auditability. Additionally, it examines the trade-offs and performance impacts of integrating blockchain with FL. The paper concludes with a discussion of open research issues and future directions, emphasizing the need for scalable, efficient, and robust blockchain-based FL frameworks to advance secure collaborative learning in various applications, particularly in sensitive domains like healthcare.

TITLE: Blockchain and Federated Learning for Privacy-Preserving Distributed Machine Learning in Healthcare
AUTHORS: Xiao Liu, Kan Yang, Liqun Chen, and Xiapu Luo
ABSTRACT: The healthcare industry is increasingly adopting federated learning (FL) to enable collaborative model training while preserving patient data privacy. However, the FL paradigm faces significant security challenges, particularly from model poisoning attacks that can compromise the integrity and accuracy of the trained models. This paper proposes a novel framework that integrates blockchain technology with FL to enhance security in healthcare applications. The proposed framework employs secure multiparty computation (SMPC) to verify the integrity of participants' local models before they are aggregated. Blockchain

technology ensures a decentralized, tamper-proof ledger for recording model updates, enhancing transparency and trust among participants. The framework's security mechanisms are designed to detect and mitigate poisoning attacks, ensuring that only verified and uncompromised models contribute to the global model. Extensive experiments conducted using diverse medical datasets demonstrate the framework's effectiveness in maintaining model accuracy and robustness in the face of adversarial attacks. The results show that the integration of blockchain and SMPC not only preserves data privacy but also significantly enhances the security and reliability of FL systems in healthcare.

TITLE: Federated Learning with Blockchain for Intelligent Healthcare Systems: A Survey
AUTHORS: Wen Zhang, Jianping Wang, Xin Zhang, and Chen Wang
ABSTRACT: Federated learning (FL) has emerged as a powerful tool for enabling collaborative machine learning across multiple healthcare institutions without compromising patient data privacy. Despite its advantages, FL is prone to several security vulnerabilities, including data poisoning and integrity attacks. Blockchain technology, known for its decentralized, transparent, and secure characteristics, provides a potential solution to these challenges. This survey comprehensively reviews the integration of blockchain with FL in the context of intelligent healthcare systems. It starts by detailing the principles of FL and blockchain, followed by an analysis of the security threats faced by FL systems. The paper then explores various blockchain-based approaches designed to enhance the security and privacy of FL, including the use of smart contracts, consensus mechanisms, and cryptographic techniques. The survey highlights recent case studies and implementations where blockchain has been successfully integrated with FL to secure healthcare applications. It also discusses the challenges and limitations of current solutions and proposes future research directions to address scalability, efficiency, and interoperability issues. The survey aims to provide a thorough understanding of how blockchain can be leveraged to create robust and secure FL frameworks, fostering trust and collaboration in the healthcare industry.
TITLE: Blockchain-Based Federated Learning with Differential Privacy for Smart Healthcare
AUTHORS: Haoran Wang, Hongwei Li, and Xiaohui Liang
ABSTRACT: The intersection of federated learning (FL) and blockchain technology presents a promising approach for developing secure and privacy-preserving machine learning systems, especially in the context of smart healthcare. This paper proposes a novel framework that

combines FL with blockchain and differential privacy to protect against poisoning attacks and ensure data confidentiality. The framework utilizes secure multiparty computation (SMPC) to validate local model updates from FL participants before they are aggregated. Blockchain technology provides a decentralized, immutable ledger for recording these updates, ensuring transparency and trust. Differential privacy techniques are employed to add noise to the model parameters, further enhancing privacy protection. The framework aims to ensure that only verified and sanitized models contribute to the global model, thus preventing malicious actors from introducing poisoned updates. The paper presents a detailed analysis of the framework's security features and performance, supported by extensive experiments on various medical datasets. The results demonstrate that the proposed framework effectively defends against poisoning attacks while maintaining high model accuracy and protecting sensitive data. The study highlights the potential of integrating blockchain, FL, and differential privacy to create robust, secure, and privacy-preserving machine learning systems for smart healthcare applications.

TITLE: A Blockchain-Based Approach for Secure Federated Learning in Healthcare Systems

AUTHORS: Yiming Zhang, Wei Wang, and Zhen Li

ABSTRACT: Federated learning (FL) has revolutionized collaborative machine learning by enabling multiple institutions to train models on decentralized data while preserving privacy. However, FL's distributed nature makes it vulnerable to adversarial attacks, such as model poisoning, where malicious participants can degrade model performance. This paper presents a blockchain-based FL framework designed to enhance security and trustworthiness in healthcare systems. The framework integrates secure multiparty computation (SMPC) to verify the integrity of local models before aggregation. Blockchain technology is used to create a secure, transparent, and immutable record of all model updates, ensuring that only verified updates are incorporated into the global model. The proposed framework addresses the challenge of detecting and defending against poisoning attacks, thereby enhancing the reliability and robustness of the FL process. Extensive experimental evaluations using various medical datasets demonstrate that the framework maintains high model accuracy and resilience against adversarial behavior. The study underscores the potential of combining blockchain and FL to develop secure, efficient, and privacy-preserving machine learning solutions in healthcare, fostering greater collaboration and data sharing among institutions while safeguarding patient privacy and data integrity.

CHAPTER-3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

In FL, data privacy is achieved by sending the model to the client and performing local training. Later, the locally trained model will be collected by the central server and aggregated into a global model. With this method, the participants only shared the local model and did not send any datasets. However, FL itself is not sufficient to provide a privacy guarantee. Some research has been performed to secure the FL architecture. The author in [6] and [7] enhance the data privacy in FL with differential privacy (DP) by adding noise in the local datasets. In [7], also anonymize the end-user by adding a proxy server. However, the experiment result show there is a significant accuracy reduction. This privacy-preserving method is unsuitable for FL in healthcare systems since accuracy is essential for the inference process.

Zhang et al. [13] use fully homomorphic encryption (FHE) to perform aggregation and training processes by performing a batch encryption method. However, all the homomorphic encryption methods are unusable for healthcare scenarios since the training process takes significant time. Authors in [14], [15], and [16] have successfully performed an adversarial attack on FL architecture. The authors have demonstrated a poisoning attack on the local client's datasets. The poisoned model will be generated and impact the global model. Based on the existing attack, DP and FHE method is insufficient against the poisoning attack. In [17], the author proposed a privacy-enhanced FL against poisoning adversaries. To secure the machine learning model, they encrypt the model using linear homomorphic encryption. Since they encrypt the model from the first round of FL, the training process will take longer than regular machine learning. After the participants finish the encrypted training process, The local model will send to the server for encrypted aggregation. Based on the results of their experiments, their aggregation method reduces the accuracy of the machine learning model.

Blockchain is known for its immutability and is used for tampered-proof storage. The use of blockchain can track the local or global model for audibility purposes. Combining blockchain with

FL can ensure the machine learning model's integrity. Author in [18] proposed verifiable aggregation for FL. Their method follows the concept of blockchain, where they use the hash to compute the digest for verification. Nonetheless, the aggregation and hashing process is performed on a single server. The correct utilization of blockchain technology can overcome the problem. In tackling the issue, [19] proposed decentralized privacy using blockchain-enabled FL. They use blockchain to store and verify the model using cross-validation, but the participant is connected to the same blockchain. In their framework, the participant can use other's local models, which leads to privacy issues.

The work on [20] uses a smart contract to verify the global model. The use of smart contracts can audit the authenticity of the global model. However, they did not perform any checks on the local or global model. Also, the local model is not sent to the blockchain, and not possible to perform any audit process. From the proposed work, they can not handle any poisoning attack.

Disadvantages:

1. The system didn't implement a verifiable Federated Learning (FL) scenario that leverages SMPC to perform an encrypted local model verification process and secure aggregation on the blockchain node.
2. The federated learning scenario not allows each participant to collaboratively train the machine learning model locally with their local datasets. Later the machine learning model will send to the cloud for the model aggregation process.

3.2 PROPOSED SYSTEM:

The system proposes a privacy-preserving verification method to eliminate poisoned local models in a federated learning scenario. The proposed method eliminates the compromised local model while guaranteeing the privacy of the local model's parameters using an SMPC-based encrypted inference process. Once the local model is verified, the verified share of the local model is sent to the blockchain for the aggregation process. SMPC-based aggregation is used to perform the secure aggregation between the blockchain and the hospital. After the aggregation process, the global model is stored in tampered-proof storage. Later, each hospital receives the global model from the blockchain and verifies the authenticity of the global model.

Advantages:

- Propose a new blockchain-based federated learning architecture for healthcare systems to ensure the security of the global model used for classifying disease.
- Design a privacy-preserving method for local model anomaly detection in a Federated learning scenario with SMPC as the underlying technology. Our encrypted model verification method eliminates the poisoned model while protecting the local model privacy from membership inference attacks and parameter stealing.
- Propose an SMPC-based secure aggregation in the blockchain as a platform to decentralize the aggregation process.
- We present a verifiable machine learning model for federated learning participants using blockchain in the IoMT scenario.

CHAPTER-4

SYSTEM REQUIREMENTS

4.1 FUNCTIONAL REQUIREMENTS

Functional requirements will vary for different types of software. For example, functional requirements for a website or mobile application should define user flows and various interaction scenarios.

The major modules of the project are

1. Healthcare Data Manager
2. Healthcare Data Users

4.2 NON-FUNCTIONAL REQUIREMENTS

Nonfunctional requirements are not related to the system's functionality but rather define how the system should perform. They are crucial for ensuring the system's usability, reliability, and efficiency, often influencing the overall user experience. We'll describe the main categories of nonfunctional requirements in detail further on

HARDWARE REQUIREMENTS:

- System : i3
- Hard Disk : 40 GB.
- Floppy Drive : 1.44 Mb.
- Monitor : 15 VGA Colour.
- Mouse : Logitech.
- Ram : 512 Mb.

SOFTWARE REQUIREMENTS:

- Operating system : Windows 7 Ultimate.
- Coding Language : Python.
- Front-End : Python.

- Back-End : Django-ORM
- Designing : Html, css, javascript.
- Data Base : MySQL (WAMP Server).

CHAPTER-5

SYSTEM STUDY

FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

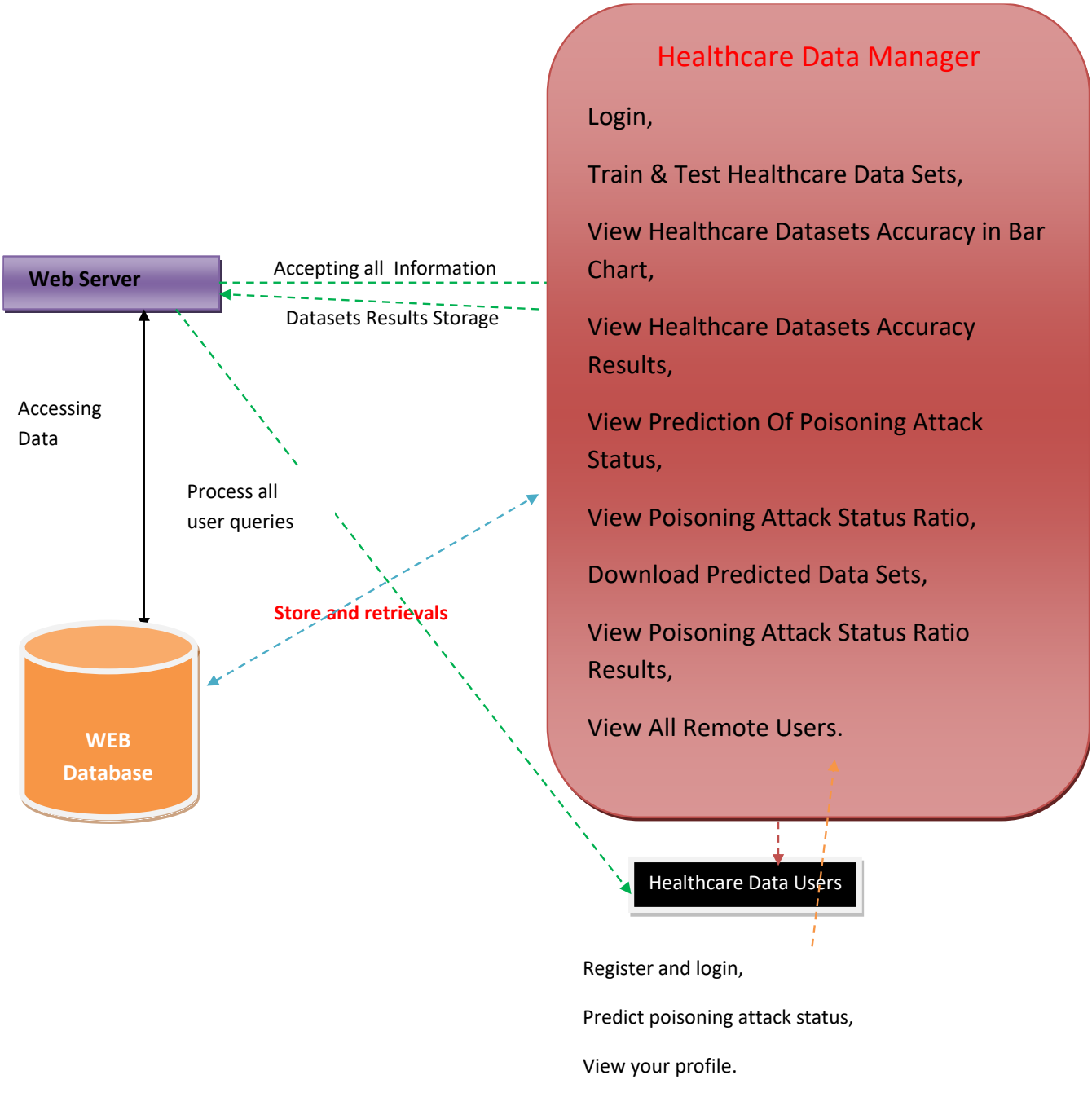
SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

CHAPTER-6

SYSTEM DESIGN

SYSTEM ARCHITECTURE:



4.2 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

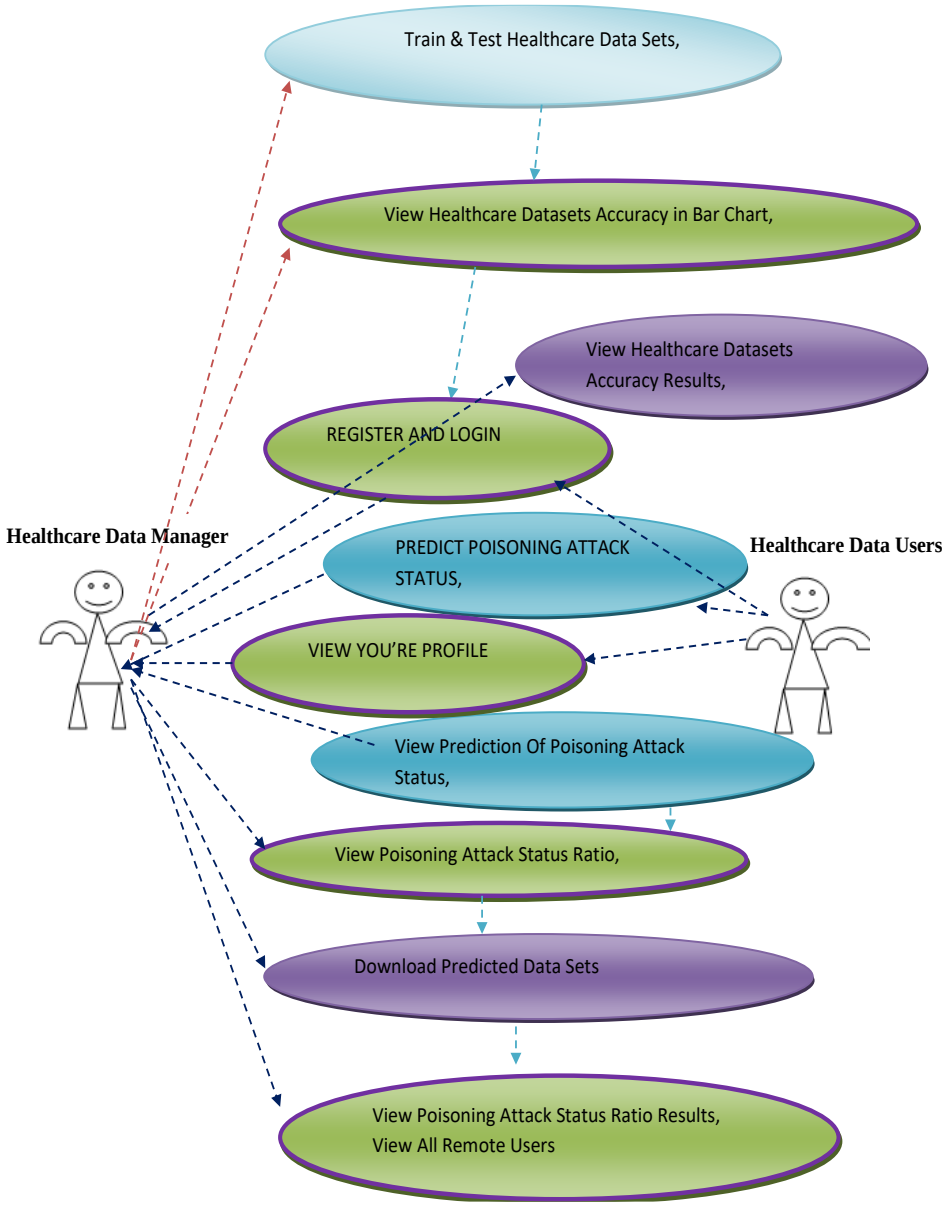
GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

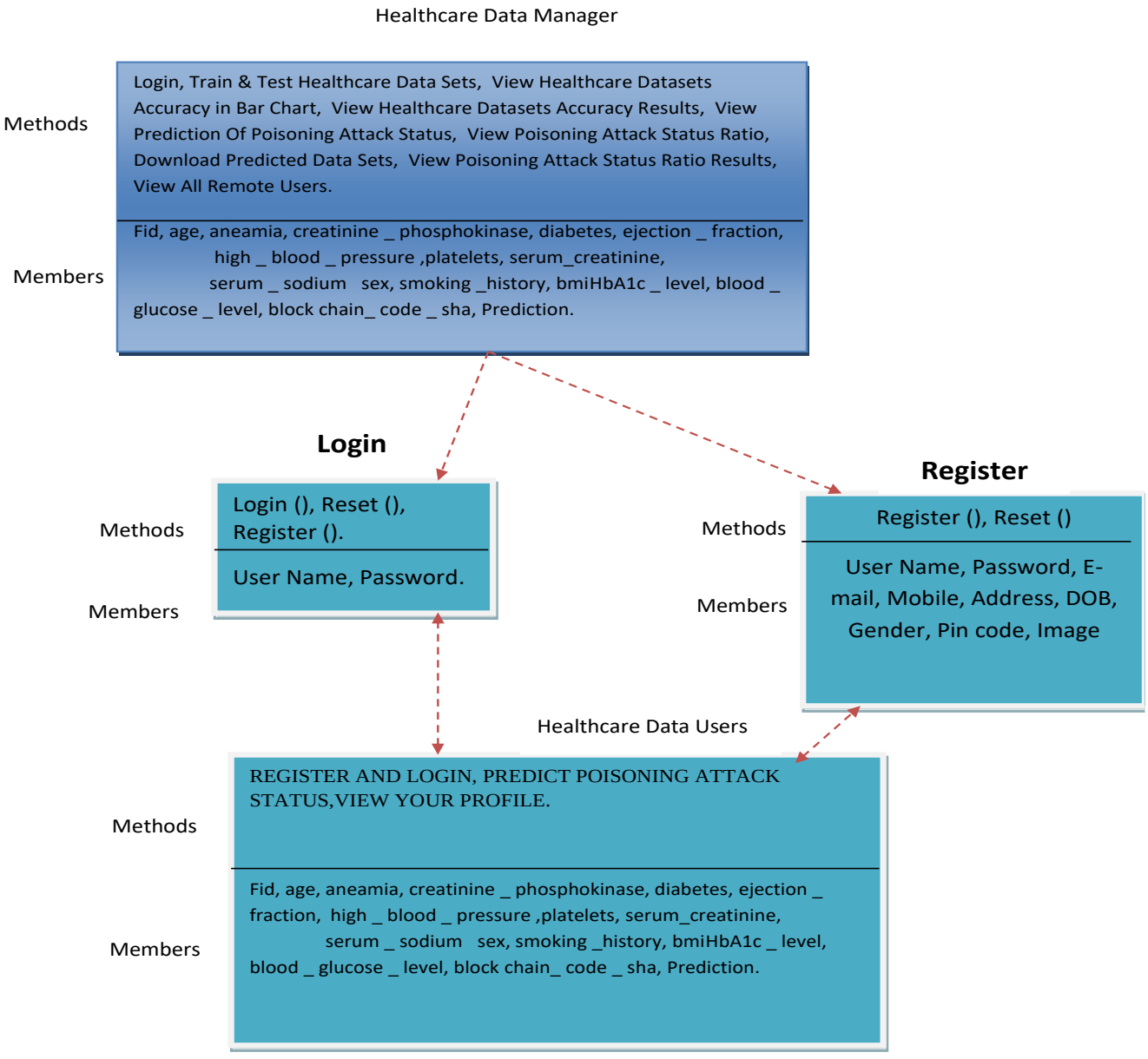
6.2.1 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



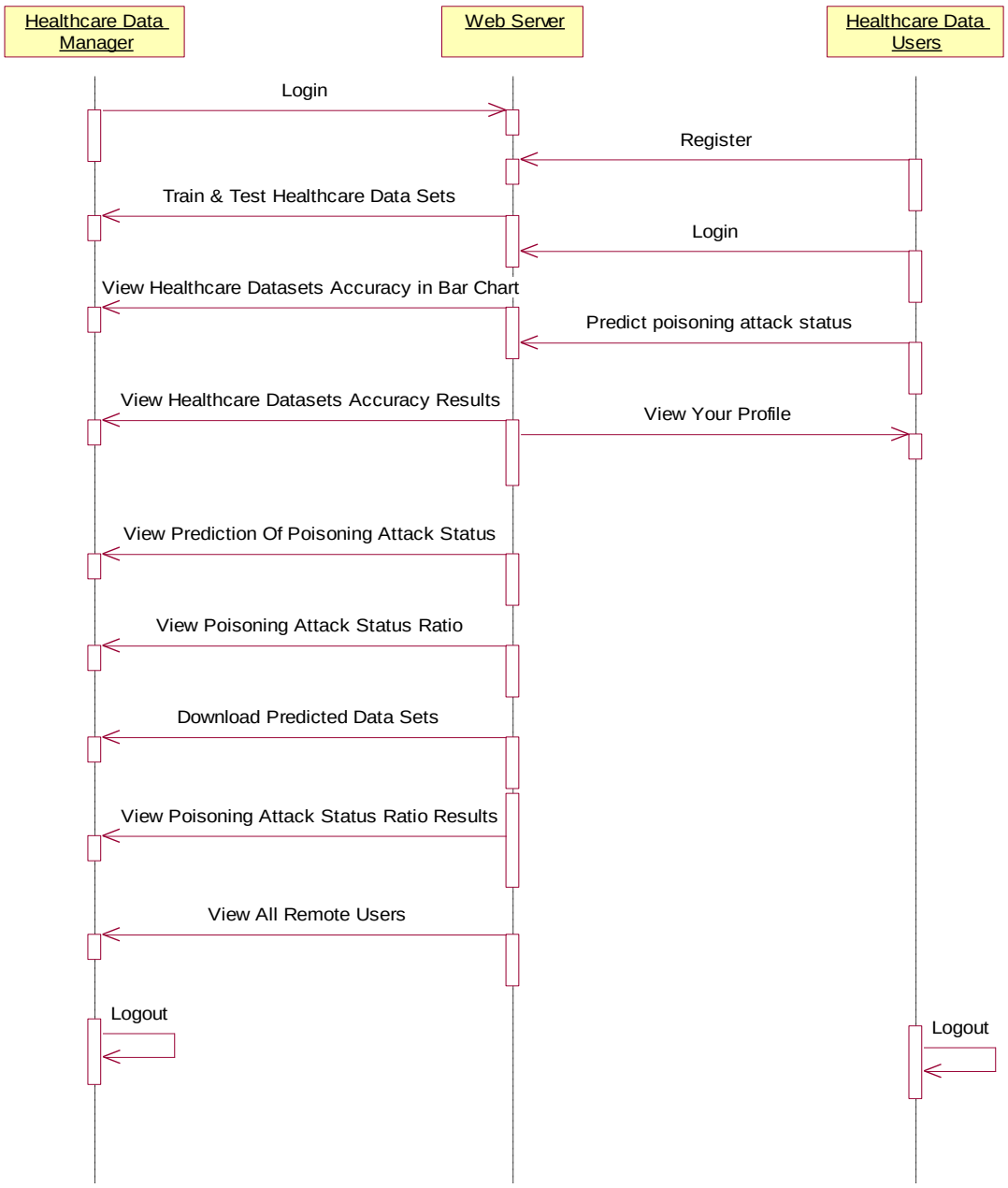
6.2.2 CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which Class contains information.



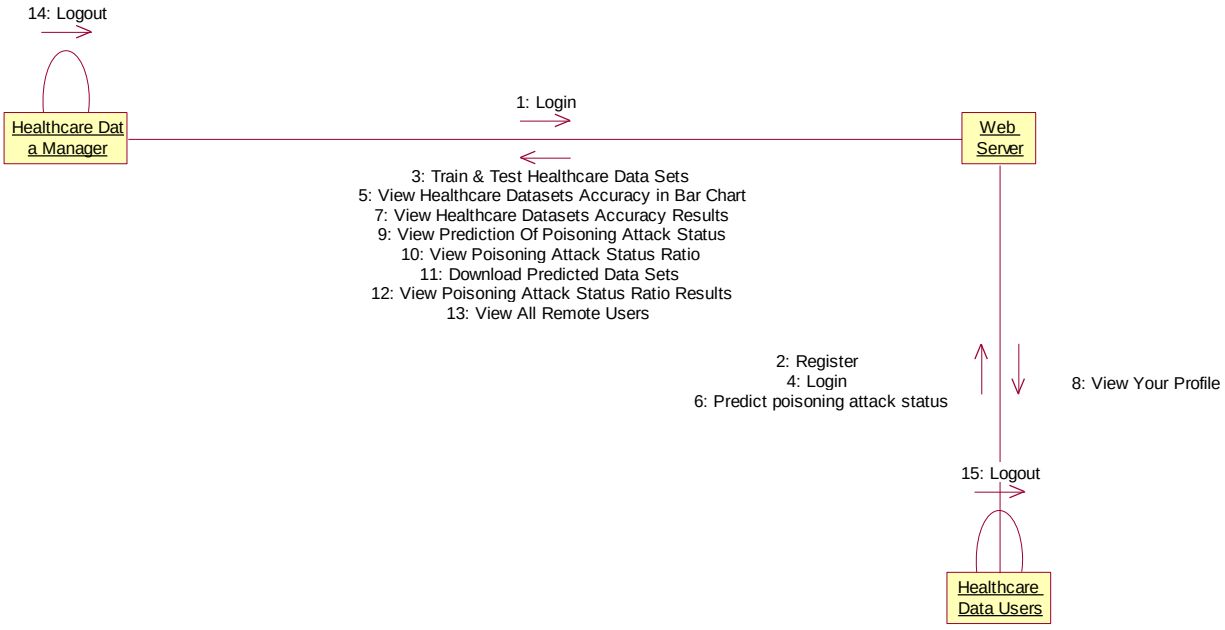
6.2.3 SEQUENCE DIAGRAM:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



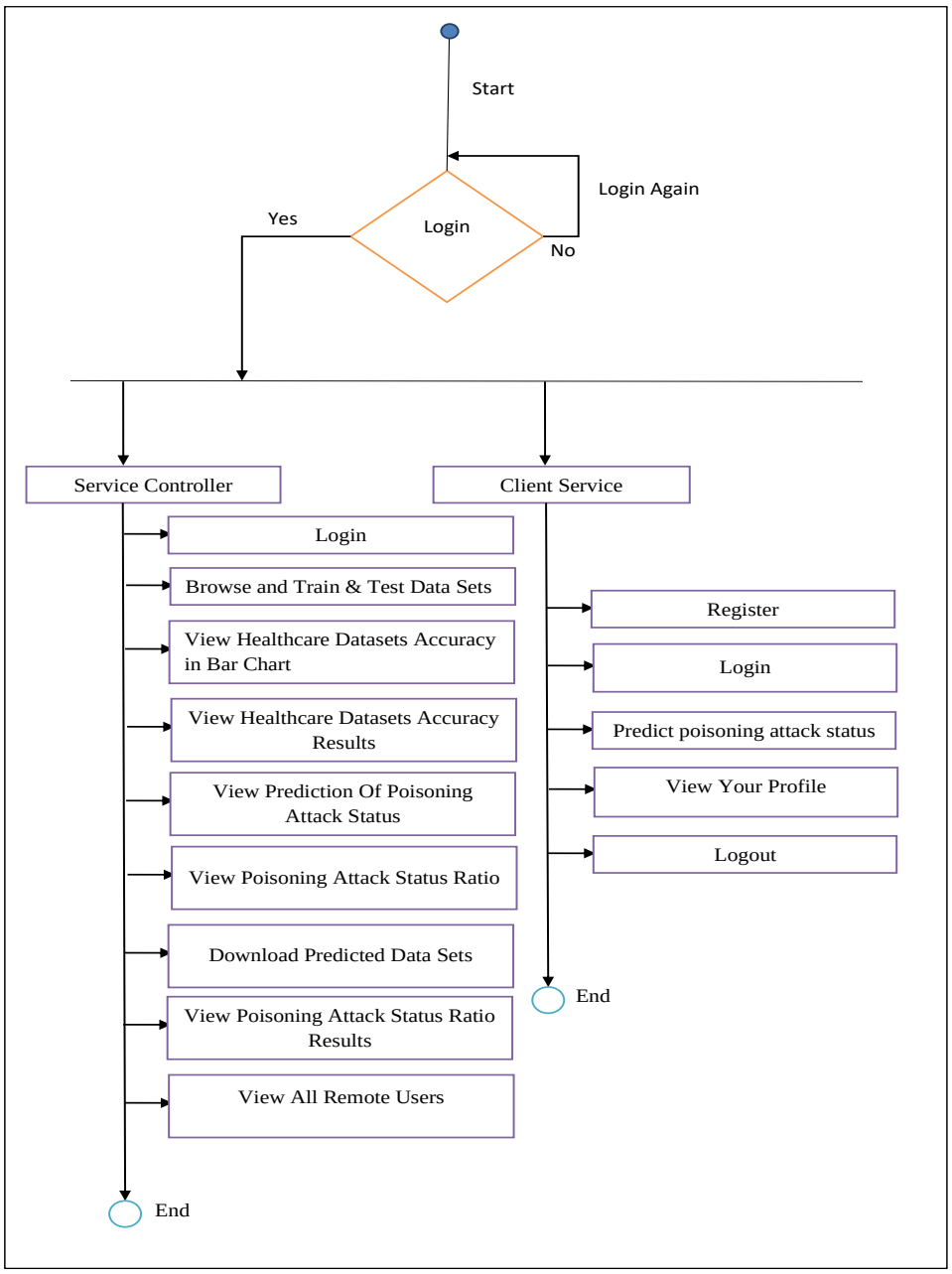
6.2.4 COLLABRATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). Developers can use these diagrams to portray the dynamic behavior of a particular use case and define the role of each object.



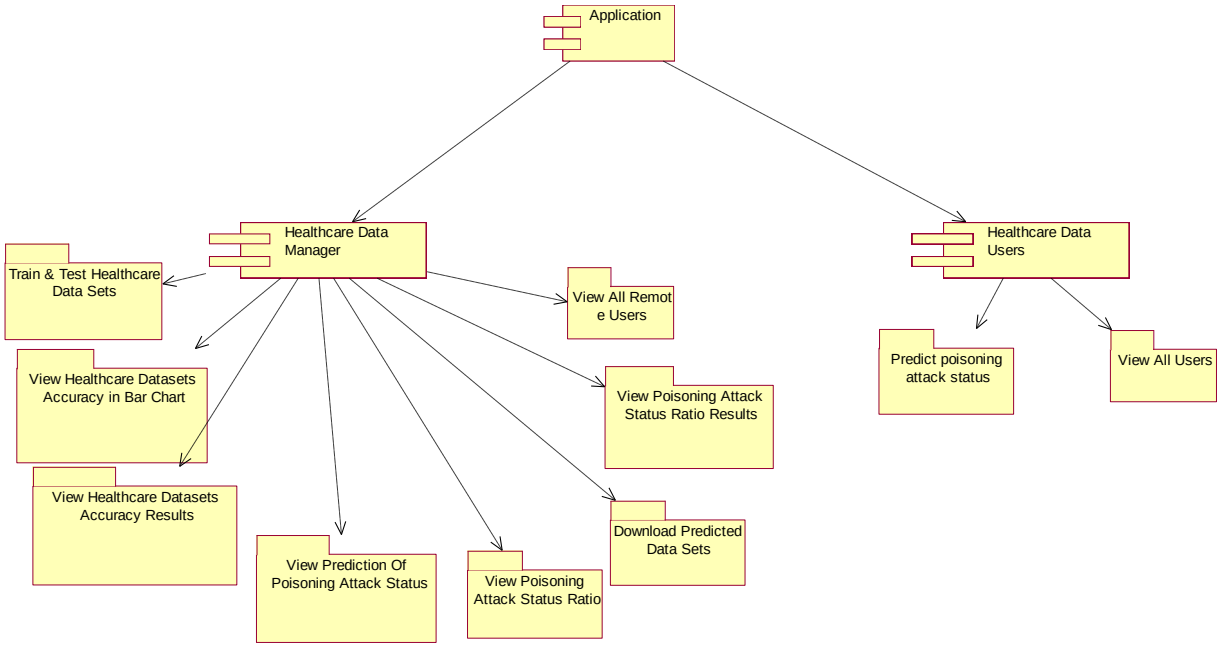
6.2.5 ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



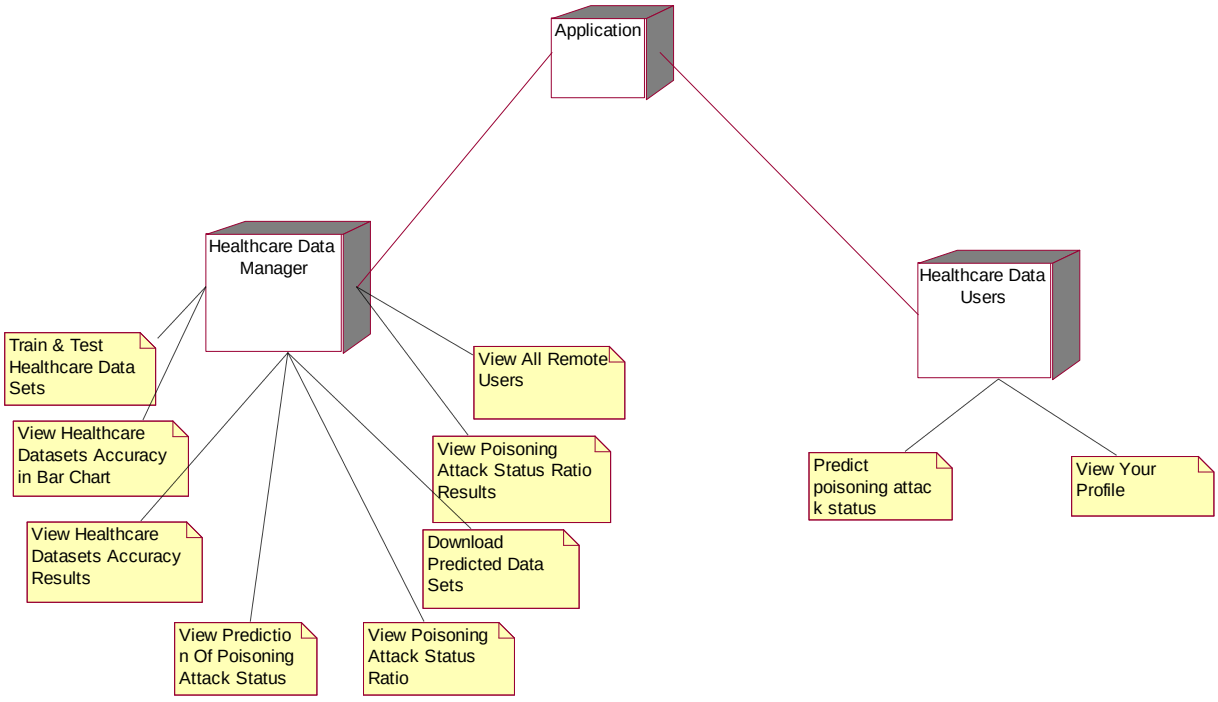
6.2.6 COMPONENT DIAGRAM:

Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.



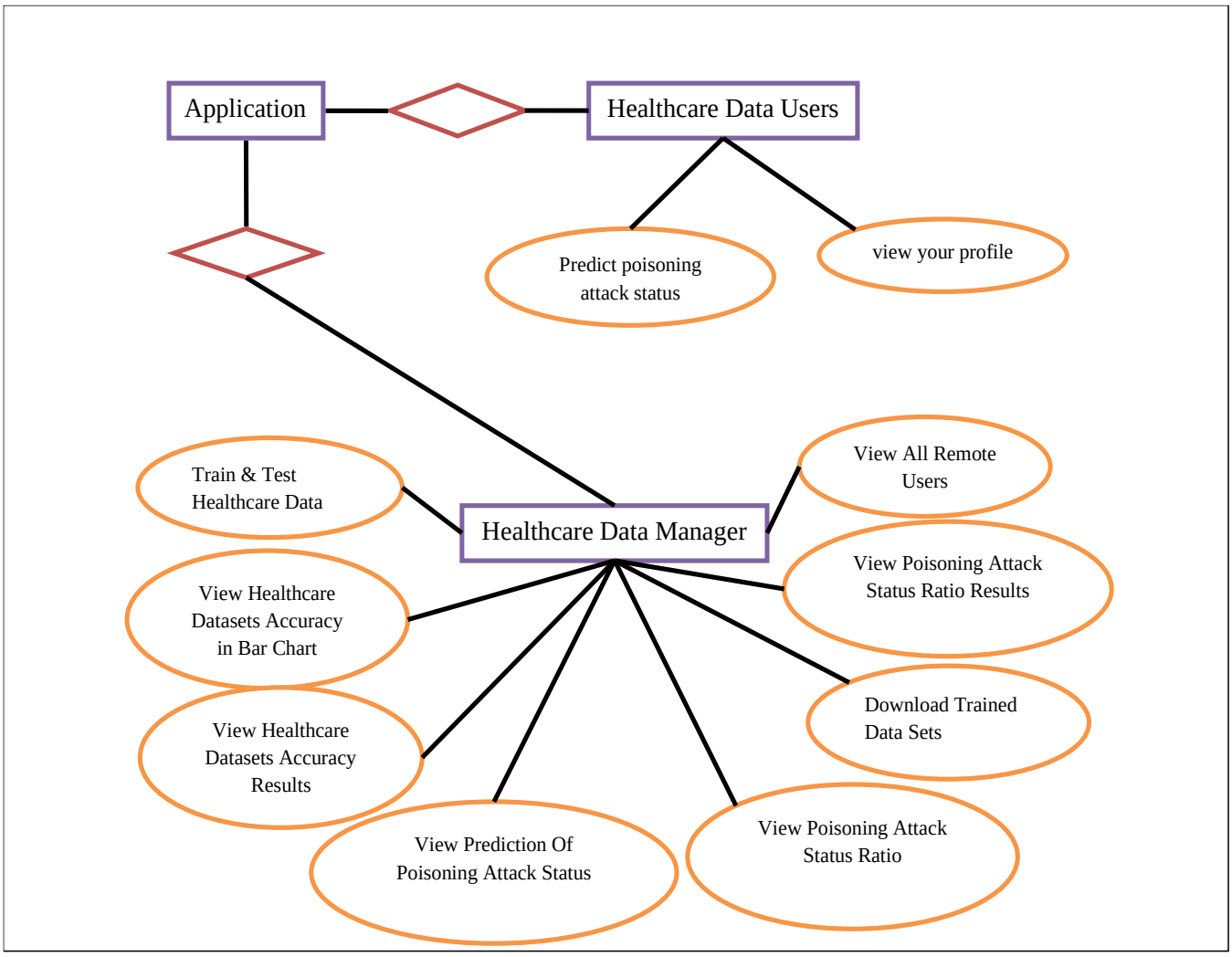
6.2.7 DEPLOYMENT DIAGRAM:

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed. Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.



6.2.8 E-R DIAGRAM

An Entity Relationship Diagram is a diagram that represents relationships among entities in a database. It is commonly known as an ER Diagram. An ER Diagram in DBMS plays a crucial role in designing the database. Today's business world previews all the requirements demanded by the users in the form of an ER Diagram.



6.2.9 DATA DICTIONARY

Database: blockchain_based_federated_learning

Table name: auth_group

Column	Data Type	Constraints	Description
id	Int(11)	Primary key	Unique Identifier
name	Varchar(1000)	Not null	name

Table Name: Auth_group_Permission

Column	Data Type	Constraints	Description
id	Int(11)	Primary key	Unique Identifier
Group id	Int(11)	Primary Key	Unique Identifier
Permission_id	Int(11)	Primary Key	Unique Identifier

Table Name: auth_permission

Column	Data Type	Constraints	Description
id	Int(11)	Primary key	Unique Identifier
Name	Int(255)	Not Null	name
Content_type_id	Int(11)	Primary Key	Unique Identifier
Code name	Int(100)	Not Null	name

Table Name: auth_User

Column	Data Type	Constraints	Description
id	Int(11)	Primary key	Unique Identifier
Password	varchar(128)	Not Null	password
Last_Login	Datetime(6)	Not Null	Last login
Is_superuser	tinyint(1)	Not Null	Name
username	Varchar(150)	Not Null	Username
lastname	Varchar(30)	Not Null	Lastname
email	Varchar(150)	Not Null	Email id
Is_staff	Tinyint(1)	Not Null	Staff
Is_active	Tinyint(1)	Not Null	Active
Date_joined	Datetime(6)	Not Null	Date and time

Table Name: auth_user_groups

Column	Data Type	Constraints	Description
id	Int(11)	Primary key	Unique Identifier
User_id	Int(11)	Primary Key	Unique Identifier
Group_id	Int(11)	Primary Key	Unique Identifier

CHAPTER-7
INPUT/OUTPUT DESIGN

Input design: considering the requirements, procedures to collect the necessary input data in most efficiently designed. The input design has been done keeping in view that, the interaction of the user with the system being the most effective and simplified way.

Also the measures are taken for the following

- Controlling the amount of input
- Avoid unauthorized access to the classroom.
- Eliminating extra steps
- Keeping the process simple
- At this stage the input forms and screens are designed.

Output design: All the screens of the system are designed with a view to provide the user with easy operations in simpler and efficient way, minimum key strokes possible. Instructions and important information is emphasized on the screen. Almost every screen is provided with no error and important messages and option selection facilitates. Emphasis is given for speedy processing and speedy transaction between the screens. Each screen assigned to make it as muchuser friendly as possible by using interactive procedures. So to say user can operate the system without much help from the operating manual.

CHAPTER-8

IMPLEMENTATION

MODULE

The major modules of the project are

1. Healthcare Data Manager
2. Healthcare Data Users

MODULE DESCRIPTION

Healthcare Data Manager

In this module, the Service Provider has to login by using valid user name and password. After login successful he can do some operations such as Train & Test Healthcare Data Sets, View Healthcare Datasets Accuracy in Bar Chart, View Healthcare Datasets Accuracy Results, View Prediction Of Poisoning Attack Status, View Poisoning Attack Status Ratio, Download Predicted Data Sets, View Poisoning Attack Status Ratio Results, View All Remote Users**.**

Healthcare Data Users

In this module, there are n numbers of users are present. User should register before doing any operations. Once user registers, their details will be stored to the database. After registration successful, he has to login by using authorized user name and password. Once Login is successful user will do some operations like Register And Login, Predict Poisoning Attack Status, View Your Profile.

CHAPTER-9

SOFTWARE ENVIRONMENT

1.1 PYTHON

Python is a **high-level, interpreted**, **interactive** and **object-oriented scripting language**. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

1.2 History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

1.3 Python Features

Python's features include:

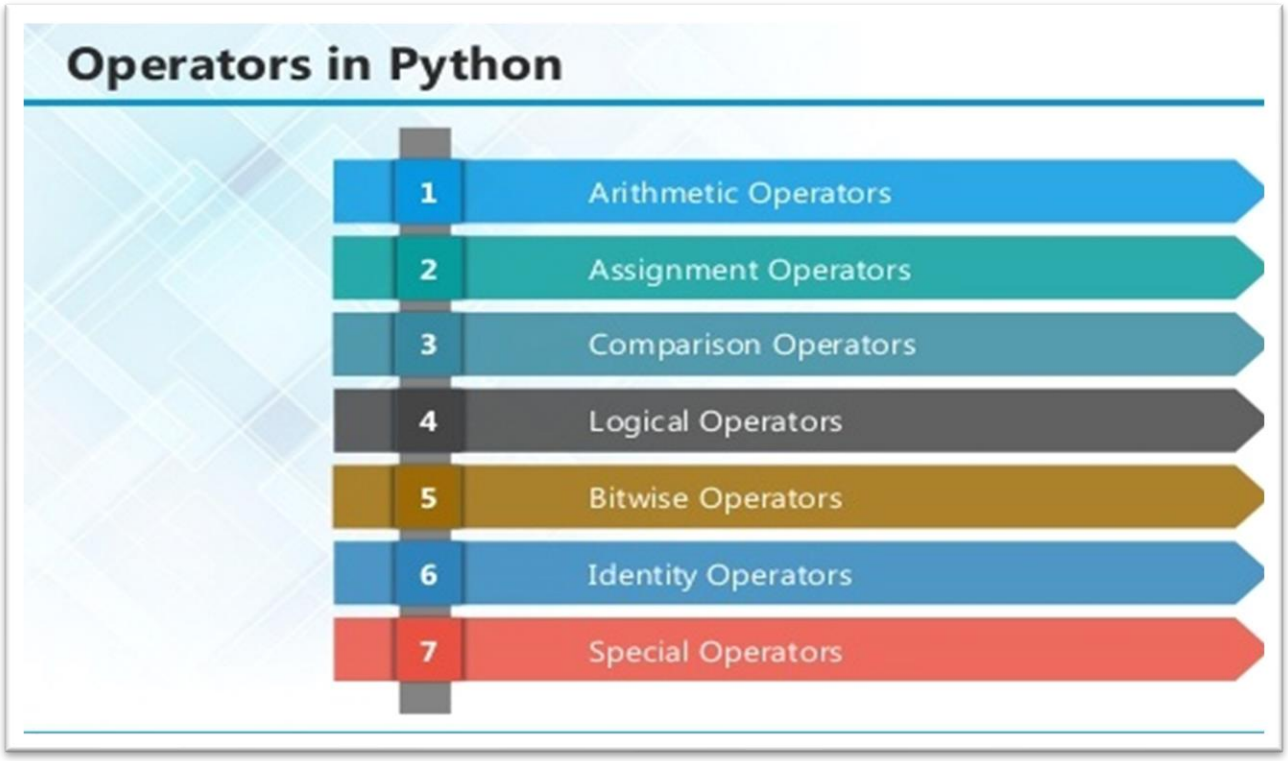
- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Python has a big list of good features:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.

- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.



2.1 ARITHMETIC OPERATORS

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30

- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

2.2ASSIGNMENT OPERATOR

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c

+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a

%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor	It performs floor division on operators	c //= a is

Division	and assign value to the left operand	equivalent to c = c // a

2.3 IDENTITY OPERATOR

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y

2.4 COMPARISON OPERATOR

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)

| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

2.5 LOGICAL OPERATOR

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.

or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

2.6 Membership Operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Operators Precedence

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)

* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

3.1 LIST

The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

Basic List Operations

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Built-in List Functions & Methods:

Python includes the following list functions −

SN	Function with Description
1	cmp(list1, list2) Compares elements of both lists.

2	len(list) Gives the total length of the list.
3	max(list) Returns item from the list with max value.
4	min(list) Returns item from the list with min value.
5	list(seq) Converts a tuple into list.

Python includes following list methods

SN	Methods with Description
1	list.append(obj) Appends object obj to list
2	list.count(obj) Returns count of how many times obj occurs in list
3	list. extend(seq) Appends the contents of seq to list
4	list.index(obj) Returns the lowest index in list that obj appears

5	list.insert(index, obj) Inserts object obj into list at offset index
6	list.pop(obj=list[-1]) Removes and returns last object or obj from list
7	list.remove(obj) Removes object obj from list
8	list.reverse() Reverses objects of list in place
9	list.sort([func]) Sorts objects of list, use compare function if given

3.2 TUPLES

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally we can put these comma-separated values between parentheses also. For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5 );

tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing −

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value −

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

- **Accessing Values in Tuples:**

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

```
tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0]

print "tup2[1:5]: ", tup2[1:5]
```

When the code is executed, it produces the following result −

```
tup1[0]:  physics
tup2[1:5]:  [2, 3, 4, 5]
```

Updating Tuples:

Tuples are immutable which means you cannot update or change the values of tuple elements. We are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
tup1 = (12, 34.56);

tup2 = ('abc', 'xyz');

tup3 = tup1 + tup2;

print tup3
```

When the above code is executed, it produces the following result −

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example:

```
tup = ('physics', 'chemistry', 1997, 2000);
print tup
del tup;
print "After deleting tup : "
print tup
```

Basic Tuples Operations:

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Built-in Tuple Functions

SN	Function with Description

1	**cmp(tuple1, tuple2)**:Compares elements of both tuples.
2	**len(tuple)**:Gives the total length of the tuple.
3	**max(tuple)**:Returns item from the tuple with max value.
4	**min(tuple)**:Returns item from the tuple with min value.
5	**tuple(seq)**:Converts a list into tuple.

3.2 DICTIONARY

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values in Dictionary:

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

Result –

```
dict['Name']:  Zara
dict['Age']:  7
```

Updating Dictionary

We can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Result −

```
dict['Age']:  8
dict['School']:  DPS School
```

Delete Dictionary Elements

We can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement. Following is a simple example –

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key 'Name'
dict.clear();     # remove all entries in dict
del dict ;        # delete entire dictionary

print "dict['Age']: ", dict['Age']
```

```
print "dict['School']: ", dict['School']
```

Built-in Dictionary Functions & Methods –

Python includes the following dictionary functions −

SN	Function with Description
1	cmp(dict1, dict2) Compares elements of both dict.
2	len(dict) Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	str(dict) Produces a printable string representation of a dictionary
4	type(variable) Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Python includes following dictionary methods −

SN	Methods with Description
1	**dict.clear()**:Removes all elements of dictionary *dict*
2	**dict. Copy()**:Returns a shallow copy of dictionary *dict*
3	**dict.fromkeys()**:Create a new dictionary with keys from seq and values *set* to *value*.
4	**dict.get(key, default=None)**:For *key* key, returns value or default if key not in dictionary
5	**dict.has_key(key)**:Returns *true* if key in dictionary *dict*, *false* otherwise
6	**dict.items()**:Returns a list of *dict*'s (key, value) tuple pairs
7	**dict.keys()**:Returns list of dictionary dict's keys
8	**dict.setdefault(key, default=None)**:Similar to get(), but will set dict[key]=default if *key* is not already in dict
9	**dict.update(dict2)**:Adds dictionary *dict2*'s key-values pairs to *dict*
10	**dict.values()**:Returns list of dictionary *dict*'s values

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*

Defining a Function

Simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function −

```
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
```

```
   print str
   return;
# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

When the above code is executed, it produces the following result −

```
I'm first call to user defined function!
Again second call to the same function
```

Function Arguments

You can call a function by using the following types of formal arguments:

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Scope of Variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python −

Global variables Local variables

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope. Following is a simple example −

```
total = 0; # This is global variable.
# Function definition is here
def sum ( arg1, arg2 ):
  # Add both the parameters and return them."
  total = arg1 + arg2; # Here total is local variable.
  print "Inside the function local total : ", total
  return total;
sum ( 10, 20 );
print "Outside the function global total : ", total
```

Result −

```
Inside the function local total : 30
Outside the function global total : 0
```

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Example:

The Python code for a module named *aname* normally resides in a file named *aname.py*. Here's an example of a simple module, support.py

```
def print_func ( par ):
  print "Hello : ", par
```

```
   return
```

The *import* Statement

The *import* has the following syntax:

```
import module1[, module2[,... moduleN]
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module support.py, you need to put the following command at the top of the script −

A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur.

Packages in Python

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-sub packages.

Consider a file *Pots.py* available in *Phone* directory. This file has following line of source code −

```
def Pots():
   print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above −

- *Phone/Isdn.py* file having function Isdn()
- *Phone/G3.py* file having function G3()

Now, create one more file __init__.py in *Phone* directory −

- Phone/__init__.py

To make all of your functions available when you've imported Phone,to put explicit import statements in __init__.py as follows −

```
from Pots import Pots
from Isdn import Isdn
from G3 import G3
```

After you add these lines to __init__.py, you have all of these classes available when you import the Phone package.

```
# Now import your Phone Package.
import Phone
Phone.Pots()
Phone.Isdn()
Phone.G3()
```

RESULT:

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

In the above example, we have taken example of a single functions in each file, but you can keep multiple functions in your files. You can also define different Python classes in those files and then you can create your packages out of those classes.

This chapter covers all the basic I/O functions available in Python.

Printing to the Screen

The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows −

```
print "Python is really a great language,", "isn't it?"
```

Result:

```
Python is really a great language, isn't it?
```

Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are −

- raw_input
- input

The *raw_input* Function

The *raw_input([prompt])* function reads one line from standard input and returns it as a string (removing the trailing newline).

```
str = raw_input("Enter your input: ");
print "Received input is : ", str
```

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this −

```
Enter your input: Hello Python
Received input is : Hello Python
```

The *input* Function

The *input([prompt])* function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

```
str = input("Enter your input: ");
print "Received input is : ", str
```

This would produce the following result against the entered input −

```
Enter your input: [x*5 for x in range(2,10,2)]
Recieved input is : [10, 20, 30, 40]
```

Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.

The *open* Function

Before you can read or write a file, you have to open it using Python's built-in *open()* function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```

Here are parameter details:

- **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

- **access_mode:** The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- **buffering:** If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file −

The *file* Object Attributes

Once a file is opened and you have one *file* object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

Example

```
# Open a file
```

```
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print "Softspace flag : ", fo.softspace
```

This produces the following result −

```
Name of the file:  foo.txt
Closed or not :  False
Opening mode :  wb
Softspace flag :  0
```

The *close()* Method

The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax

```
fileObject.close();
```

Example

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
# Close opend file
fo.close()
```

Result −

```
Name of the file:  foo.txt
```

Reading and Writing Files

The *file* object provides a set of access methods to make our lives easier. We would see how to use *read()* and *write()* methods to read and write files.

The *write()* Method

The *write()* method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.The write() method does not add a newline character ('\n') to the end of the string **Syntax**

```
fileObject.write(string);
```

Here, passed parameter is the content to be written into the opened file. **Example**

```
# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n");

# Close opend file
fo.close()
```

The above method would create *foo.txt* file and would write given content in that file and finally it would close that file. If you would open this file, it would have following content.

```
Python is a great language.
Yeah its great!!
```

The *read()* Method

The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax

```
fileObject.read([count]);
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if *count* is missing, then it tries to read as much as possible, maybe until the end of file.

Example

Let's take a file *foo.txt*, which we created above.

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Close opend file
fo.close()
```

This produces the following result −

```
Read String is :  Python is
```

File Positions

The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The *seek(offset[, from])* method changes the current file position. The *offset* argument indicates the number of bytes to be moved. The *from* argument specifies the reference position from where the bytes are to be moved.

If *from* is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Example

Let us take a file *foo.txt*, which we created above.

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str

# Check current position
position = fo.tell();
print "Current file position : ", position

# Reposition pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str
# Close opend file
fo.close()
```

This produces the following result −

```
Read String is :  Python is
Current file position :  10
Again read String is :  Python is
```

Renaming and Deleting Files

Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

The rename() Method

The *rename()* method takes two arguments, the current filename and the new filename.

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

Following is the example to rename an existing file *test1.txt*:

```
import os

# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

The *remove()* Method

You can use the *remove()* method to delete files by supplying the name of the file to be deleted as the argument.

Syntax

```
os.remove(file_name)
```

Example

Following is the example to delete an existing file *test2.txt* −

```
#!/usr/bin/python
import os

# Delete file test2.txt
os.remove("text2.txt")
```

Directories in Python

All files are contained within various directories, and Python has no problem handling these too. The **os** module has several methods that help you create, remove, and change directories.

The *mkdir()* Method

You can use the *mkdir()* method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

Syntax

```
os.mkdir("newdir")
```

Example

Following is the example to create a directory *test* in the current directory −

```
#!/usr/bin/python
import os

# Create a directory "test"
os.mkdir("test")
```

The *chdir()* Method

You can use the *chdir()* method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax

```
os.chdir("newdir")
```

Example

Following is the example to go into "/home/newdir" directory −

```
#!/usr/bin/python
import os

# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")
```

The *getcwd()* Method

The *getcwd()* method displays the current working directory.

Syntax

```
os.getcwd()
```

Example

Following is the example to give current directory −

```
import os

# This would give location of the current directory
os.getcwd()
```

The *rmdir()* Method

The *rmdir()* method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax:

```
os.rmdir('dirname')
```

Example

Following is the example to remove "/tmp/test" directory. It is required to give fully qualified name of the directory, otherwise it would search for that directory in the current directory.

```
import os
# This would  remove "/tmp/test"  directory.
os.rmdir( "/tmp/test" )
```

EXCEPTION NAME	DESCRIPTION
Exception	Base class for all exceptions
StopIteration	Raised when the next() method of an iterator does not point to any object.
SystemExit	Raised by the sys.exit() function.
StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
AssertionError	Raised in case of failure of the Assert statement.
AttributeError	Raised in case of failure of attribute reference or assignment.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.

ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
LookupError	Base class for all lookup errors.
IndexError KeyError	Raised when an index is not found in a sequence. Raised when the specified key is not found in the dictionary.
NameError	Raised when an identifier is not found in the local or global namespace.
UnboundLocalError EnvironmentError	Raised when trying to access a local variable in a function or method but no value has been assigned to it. Base class for all exceptions that occur outside the Python environment.
IOError IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist. Raised for operating system-related errors.
SyntaxError IndentationError	Raised when there is an error in Python syntax. Raised when indentation is not specified properly.
SystemError	Raised when the interpreter finds an internal problem, but

	when this error is encountered the Python interpreter does not exit.
SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError	Raised when a generated error does not fall into any category.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

File & Directory Related Methods

There are three important sources, which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows −

- File Object Methods: The *file* object provides functions to manipulate files.
- OS Object Methods: This provides methods to process files as well as directories.

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them −

- **Exception Handling:** This would be covered in this tutorial. Here is a list standard Exceptions available in Python: Standard Exceptions.
- **Assertions:** This would be covered in Assertions in Python

List of Standard Exceptions −

What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an exception

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

You can choose the right database for your application. Python Database API supports a wide range of database servers such as −

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix

- Interbase
- Oracle
- Sybase

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following:

- Importing the API module.
- Acquiring a connection with the database.
- Issuing SQL statements and stored procedures.
- Closing the connection

SOURCE CODE:

Settings.py

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'm+1edl5m-5@u9u!b8-=4-4mq&o1%agco2xpl8c!7sn7!eowjk#'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []
```

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'Remote_User',
    'Service_Provider',
]


MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]


ROOT_URLCONF = 'blockchain_based_federated_learning.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [(os.path.join(BASE_DIR,'Template/htmls'))],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'blockchain_based_federated_learning.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
```

```
        'NAME': 'blockchain_based_federated_learning',
        'USER':'root',
        'PASSWORD': '',
        'HOST' :'127.0.0.1',
        'PORT' :'3306',
    }
}



# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR,'Template/images')]
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'Template/media')
```

```
STATIC_ROOT = '/static/'

STATIC_URL = '/static/'
```

Views.py

```
from django.db.models import  Count, Avg
from django.shortcuts import render, redirect
from django.db.models import Count
from django.db.models import Q
import datetime
import xlwt
from django.http import HttpResponse

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.feature_extraction.text import CountVectorizer

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, plot_confusion_matrix, classification_report

# Create your views here.
from Remote_User.models import
ClientRegister_Model,detect_poisoning_attack,detection_ratio,detection_accuracy



def serviceproviderlogin(request):
    if request.method  == "POST":
        admin = request.POST.get('username')
        password = request.POST.get('password')
        if admin == "Admin" and password =="Admin":
            detection_accuracy.objects.all().delete()
            return redirect('View_Remote_Users')

    return render(request,'SProvider/serviceproviderlogin.html')
```

```
def View_Prediction_Of_Poisoning_Attack_Type_Ratio(request):
    detection_ratio.objects.all().delete()
    rratio = ""
    kword = 'No Poisoning Attack Found'
    print(kword)
    obj = detect_poisoning_attack.objects.all().filter(Q(Prediction=kword))
    obj1 = detect_poisoning_attack.objects.all()
    count = obj.count();
    count1 = obj1.count();
    ratio = (count / count1) * 100
    if ratio != 0:
        detection_ratio.objects.create(names=kword, ratio=ratio)

    ratio1 = ""
    kword1 = 'Poisoning Attack Found'
    print(kword1)
    obj1 = detect_poisoning_attack.objects.all().filter(Q(Prediction=kword1))
    obj11 = detect_poisoning_attack.objects.all()
    count1 = obj1.count();
    count11 = obj11.count();
    ratio1 = (count1 / count11) * 100
    if ratio1 != 0:
        detection_ratio.objects.create(names=kword1, ratio=ratio1)



    obj = detection_ratio.objects.all()
    return render(request, 'SProvider/View_Prediction_Of_Poisoning_Attack_Type_Ratio.html', {'objs': obj})

def View_Remote_Users(request):
    obj=ClientRegister_Model.objects.all()
    return render(request,'SProvider/View_Remote_Users.html',{'objects':obj})

def ViewTrendings(request):
    topic = detect_poisoning_attack.objects.values('topics').annotate(dcount=Count('topics')).order_by('-dcount')
```

```
    return  render(request,'SProvider/ViewTrendings.html',{'objects':topic})

def charts(request,chart_type):
    chart1 = detection_ratio.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/charts.html", {'form':chart1, 'chart_type':chart_type})

def charts1(request,chart_type):
    chart1 = detection_accuracy.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/charts1.html", {'form':chart1, 'chart_type':chart_type})

def View_Prediction_Of_Poisoning_Attack_Type(request):
    obj =detect_poisoning_attack.objects.all()
    return render(request, 'SProvider/View_Prediction_Of_Poisoning_Attack_Type.html', {'list_objects': obj})

def likeschart(request,like_chart):
    charts =detection_accuracy.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/likeschart.html", {'form':charts, 'like_chart':like_chart})



def Download_Trained_DataSets(request):

    response = HttpResponse(content_type='application/ms-excel')
    # decide file name
    response['Content-Disposition'] = 'attachment; filename="Predicted_Data.xls"'
    # creating workbook
    wb = xlwt.Workbook(encoding='utf-8')
    # adding sheet
    ws = wb.add_sheet("sheet1")
    # Sheet header, first row
    row_num = 0
    font_style = xlwt.XFStyle()
    # headers are bold
    font_style.font.bold = True
    # writer = csv.writer(response)
    obj = detect_poisoning_attack.objects.all()
```

```
    data = obj  # dummy method to fetch data.
    for my_row in data:
        row_num = row_num + 1

        ws.write(row_num, 0, my_row.Fid, font_style)
        ws.write(row_num, 1, my_row.age, font_style)
        ws.write(row_num, 2, my_row.anaemia, font_style)
        ws.write(row_num, 3, my_row.creatinine_phosphokinase, font_style)
        ws.write(row_num, 4, my_row.diabetes, font_style)
        ws.write(row_num, 5, my_row.ejection_fraction, font_style)
        ws.write(row_num, 6, my_row.high_blood_pressure, font_style)
        ws.write(row_num, 7, my_row.platelets, font_style)
        ws.write(row_num, 8, my_row.serum_creatinine, font_style)
        ws.write(row_num, 9, my_row.serum_sodium, font_style)
        ws.write(row_num, 10, my_row.sex, font_style)
        ws.write(row_num, 11, my_row.smoking_history, font_style)
        ws.write(row_num, 12, my_row.bmi, font_style)
        ws.write(row_num, 13, my_row.HbA1c_level, font_style)
        ws.write(row_num, 14, my_row.blood_glucose_level, font_style)
        ws.write(row_num, 15, my_row.blockchain_code_sha, font_style)
        ws.write(row_num, 16, my_row.Prediction, font_style)


    wb.save(response)
    return response

def train_model(request):
    detection_accuracy.objects.all().delete()
    dataset = pd.read_csv('Healthcare_Datasets.csv', encoding='latin-1')


    def apply_results(label):
        if (label == 0):
            return 0  # No Poisoning Attack Found
        elif (label == 1):
```

```
            return 1  # Poisoning Attack Found

    dataset['results'] = dataset['Label'].apply(apply_results)

    cv = CountVectorizer()

    x = dataset["Fid"]
    y = dataset["results"]

    x = cv.fit_transform(x)

    models = []
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20)
    X_train.shape, X_test.shape, y_train.shape

    print("Convolutional Neural Network (CNN)")
    from sklearn.neural_network import MLPClassifier
    mlpc = MLPClassifier().fit(X_train, y_train)
    y_pred = mlpc.predict(X_test)
    testscore_mlpc = accuracy_score(y_test, y_pred)
    accuracy_score(y_test, y_pred)
    print(accuracy_score(y_test, y_pred))
    print(accuracy_score(y_test, y_pred) * 100)
    print("CLASSIFICATION REPORT")
    print(classification_report(y_test, y_pred))
    print("CONFUSION MATRIX")
    print(confusion_matrix(y_test, y_pred))
    models.append(('MLPClassifier', mlpc))
    detection_accuracy.objects.create(names="Convolutional Neural Network (CNN)",
ratio=accuracy_score(y_test, y_pred) * 100)

    # SVM Model
    print("SVM")
    from sklearn import svm
```

```
lin_clf = svm.LinearSVC()
lin_clf.fit(X_train, y_train)
predict_svm = lin_clf.predict(X_test)
svm_acc = accuracy_score(y_test, predict_svm) * 100
print("ACCURACY")
print(svm_acc)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, predict_svm))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, predict_svm))
detection_accuracy.objects.create(names="SVM", ratio=svm_acc)

print("Logistic Regression")

from sklearn.linear_model import LogisticRegression

reg = LogisticRegression(random_state=0, solver='lbfgs').fit(X_train, y_train)
y_pred = reg.predict(X_test)
print("ACCURACY")
print(accuracy_score(y_test, y_pred) * 100)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, y_pred))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, y_pred))
detection_accuracy.objects.create(names="Logistic Regression", ratio=accuracy_score(y_test, y_pred) * 100)

print("Decision Tree Classifier")
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
dtcpredict = dtc.predict(X_test)
print("ACCURACY")
print(accuracy_score(y_test, dtcpredict) * 100)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, dtcpredict))
```

```
    print("CONFUSION MATRIX")
    print(confusion_matrix(y_test, dtcpredict))
    detection_accuracy.objects.create(names="Decision Tree Classifier", ratio=accuracy_score(y_test, dtcpredict)
* 100)



    labeled = 'Predicted_data.csv'
    dataset.to_csv(labeled, index=False)
    dataset.to_markdown

    obj = detection_accuracy.objects.all()
    return render(request,'SProvider/train_model.html', {'objs': obj})
```

Remote User

predict_poisoning_attack.html

```
{% extends 'RUser/design.html' %}
{% block userblock %}


<link rel="icon" href="images/icon.png" type="image/x-icon" />

        <link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet">
                <link href="https://fonts.googleapis.com/css?family=Righteous" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Fredoka+One" rel="stylesheet">



                <style>
                        body {background-color:#000000;}
                        .container-fluid {padding:50px;}
                        .container{background-color:white;padding:50px;   }
                        #title{font-family: 'Fredoka One', cursive;
}
                        .text-uppercase{
                        font-family: 'Righteous', cursive;

                        }
                        .tweettext{

   border: 2px solid yellowgreen;
   width: 904px;
   height: 202px;
   overflow: scroll;
   background-color:;
}
                .style1 {
        color: #FF0000;
        font-weight: bold;
```

```
}
.style6 {
	font-size: 24px;
	color: #FFFF00;
	font-weight: bold;
}
.style8 {color: #FFFF00; font-weight: bold; }
	</style>

	<body>
	<div class="container-fluid">
		<div class="container">

			<div class="row">
				<div class="col-md-5">

		<form role="form" method="POST" >
							{% csrf_token %}
							<fieldset>
								<p class="text-uppercase pull-center
style1">PREDICTION OF POISONING ATTACK STATUS !!! </p>
			<hr>

			{% csrf_token %}
			<table width="840" align="center">
			  <tr>
			    <td height="44" colspan="4" bgcolor="#FF0000"><div align="center"
class="style8">FEED HERE ALL HEALTH DETAILS  !!! </div></td>
			  </tr>
			  <tr>
			    <td width="192" height="44" bgcolor="#FF0000"><div align="center"
class="style8">Enter Fid</div></td>
			    <td width="161"><input type="text" name="Fid"></td>
			    <td width="229" bgcolor="#FF0000"><span class="style8">Enter Age</span></td>
			    <td width="238"><input type="text" name="age"></td>
```

```
                    </tr>
                    <tr>
                      <td height="44" bgcolor="#FF0000"><div align="center" class="style8">Enter
anaemia</div></td>
                      <td><input type="text" name="anaemia"></td>
                      <td bgcolor="#FF0000"><span class="style8">Enter
creatinine_phosphokinase</span></td>
                      <td><input type="text" name="creatinine_phosphokinase"></td>
                    </tr>
                    <tr>
                      <td height="44" bgcolor="#FF0000"><div align="center" class="style8">Enter
diabetes</div></td>
                      <td><input type="text" name="diabetes"></td>
                      <td bgcolor="#FF0000"><span class="style8">Enter ejection_fraction</span></td>
                      <td><input type="text" name="ejection_fraction"></td>
                    </tr>
                    <tr>
                      <td height="44" bgcolor="#FF0000"><div align="center" class="style8">Enter
high_blood_pressure</div></td>
                      <td><input type="text" name="high_blood_pressure"></td>
                      <td bgcolor="#FF0000"><span class="style8">Enter platelets</span></td>
                      <td><input type="text" name="platelets"></td>
                    </tr>
                    <tr>
                      <td height="44" bgcolor="#FF0000"><div align="center" class="style8">Enter
serum_creatinine</div></td>
                      <td><input type="text" name="serum_creatinine"></td>
                      <td bgcolor="#FF0000"><span class="style8">Enter serum_sodium</span></td>
                      <td><input type="text" name="serum_sodium"></td>
                    </tr>
                    <tr>
                      <td height="44" bgcolor="#FF0000"><div align="center"><span
class="style8">Enter</span> <span class="style8">sex</span></div></td>
                      <td><input type="text" name="sex"></td>
                      <td bgcolor="#FF0000"><span class="style8">Enter</span> <span
```

```
class="style8">smoking_history</span></td>
                    <td><input type="text" name="smoking_history"></td>
                  </tr>
                  <tr>
                    <td height="44" bgcolor="#FF0000"><div align="center"><span
class="style8">Enter</span> <span class="style8">bmi</span></div></td>
                    <td><input type="text" name="bmi"></td>
                    <td bgcolor="#FF0000"><span class="style8">Enter</span> <span
class="style8">HbA1c_level</span></td>
                    <td><input type="text" name="HbA1c_level"></td>
                  </tr>
                  <tr>
                    <td height="44" bgcolor="#FF0000"><div align="center"><span
class="style8">Enter</span> <span class="style8">blood_glucose_level</span></div></td>
                    <td><input type="text" name="blood_glucose_level"></td>
                    <td bgcolor="#FF0000"><span class="style8">Enter</span> <span
class="style8">blockchain_code_sha</span></td>
                    <td><input type="text" name="blockchain_code_sha"></td>
                  </tr>
                  <tr>
                    <td height="44" bgcolor="#FFFFFF"> </td>
                    <td> </td>
                    <td bgcolor="#FFFFFF"><input name="submit" type="submit" class="style1"
value="Predict"></td>
                    <td> </td>
                  </tr>
                </table>
                                    </fieldset>
                                </form>


                                <form role="form" method="POST" >
                                    {% csrf_token %}
                                    <fieldset>
```

<hr>

<table width="870" height="52" border="0" align="center" >
<tr><td width="563" bgcolor="#FF0000"> PREDICTED POISONING ATTACK STATUS :: </td>

<td width="209" bgcolor="#FFFFFF" style="color:red; font-size:20px; font-family:fantasy" > {{objs}} </td></tr>
</table>

</fieldset>
</form>

<!-------null------>

{% endblock %}
<tr>

CHAPTER-10

SYSTEM TESTING

What do you mean by software testing?

Testing involves operation of a system or application under controlled conditions and evaluating the results. The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should. It is oriented to 'detection'.

Unit Testing:

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing is often automated but it can also be done manually. This testing mode is a component of Extreme Programming (XP), a pragmatic method of software development that takes a meticulous approach to building a product by means of continual testing and revision.

Unit tests are written from a programmer's perspective. They ensure that a particular method of a class successfully performs a set of specific tasks. Each test confirms that a method produces the expected output when given a known input.

Performance Testing:

Performance testing is the process of determining the speed or effectiveness of a computer, network, software program or device. This process can involve quantitative tests done in a lab, such as measuring the response time or the number of MIPS (millions of instructions per second) at which a system functions. Qualitative attributes such as

Reliability, scalability and interoperability may also be evaluated. Performance testing is often done in conjunction with stress testing.

Performance testing can verify that a system meets the specifications claimed by its manufacturer or vendor. The process can compare two or more devices or programs in terms of parameters such as speed, data transfer rate, bandwidth, throughput, efficiency or reliability.

Performance testing can also be used as a diagnostic aid in locating communications bottlenecks. Often a system will work much better if a problem is resolved at a single point or in a single component. For example, even the fastest computer will function poorly on today's Web if the connection occurs at only 40 to 50 Kbps (kilobits per second).

Integration Testing:

Integration testing, also known as integration and testing (I&T), is a software development process which program units are combined and tested as groups in multiple ways. In this context, a unit is defined as the smallest testable part of an application. Integration testing can expose problems with the interfaces among program components before trouble occurs in real-world program execution. Integration testing is a component of Extreme Programming (XP), a pragmatic method of software development that takes a meticulous approach to building a product by means of continual testing and revision.

Test cases:

Test case for Login form:

FUNCTION:	**LOGIN**
EXPECTED RESULTS:	Should Validate the user and check his existence in database
ACTUAL RESULTS:	Validate the user and checking the user against the database
LOW PRIORITY	**No**
HIGH PRIORITY	**Yes**

Test case2:

Test case for Remote Access User Registration form:

FUNCTION:	**USER REGISTRATION**
EXPECTED RESULTS:	Should check if all the fields are filled by the user and saving the user to database.
ACTUAL RESULTS:	Checking whether all the fields are field by user or not through validations and saving user.
LOW PRIORITY	**No**
HIGH PRIORITY	**Yes**

Test case3:

Test case for Change Password:

When the old password does not match with the new password ,then this results in displayingan error message as " OLD PASSWORD DOES NOT MATCH WITH THE NEW PASSWORD".

FUNCTION:	**Change Password**

EXPECTED RESULTS:	Should check if old password and new password fields are filled by the user and saving the user to database.
ACTUAL RESULTS:	Checking whether all the fields are field by user or not through validations and saving user.
LOW PRIORITY	**No**
HIGH PRIORITY	**Yes**

Test case 4:

Test case for Forget Password:

When a user forgets his password he is asked to enter Login name, ZIP code, Mobile number. If these are matched with the already stored ones then user will get his Original password.

Module	Functionality	Test Case	Expected Results	Actual Results	Result	Priority
er	Login Usecase	Navigate To Www.Sample.Com Click On Submit Button Without Entering Username and Password	A Validation Should Be As Below “Please Enter Valid Username & Password”	A Validation Has Been Populated As Expected	Pass	High

		aNavigate To Www.Sample.Com Click On Submit Button With Out Filling Password And With Valid Username Test UsernameField	A Validation Should Be As Below “Please Enter Valid Password Or Password Field Can Not Be Empty “	A Validation Is Shown As Expected	Pass	High
		NNavigate To Www.Sample.Com Enter Both Username And	A Validation Shown As Below “The Username Entered Is Wrong”	A Validation Is Shown As Expected	Pass	High

		Password WrongAnd Hit Enter				
		Navigate To Www.Sample.Com Enter Validate Username And Password And Click On Submit	Validate Username And Password In DataBase And Once If They Correct Then Show The Main Page	Main Page/ Home Page Has Been Displayed	Pass	High

SCREENSHOTS

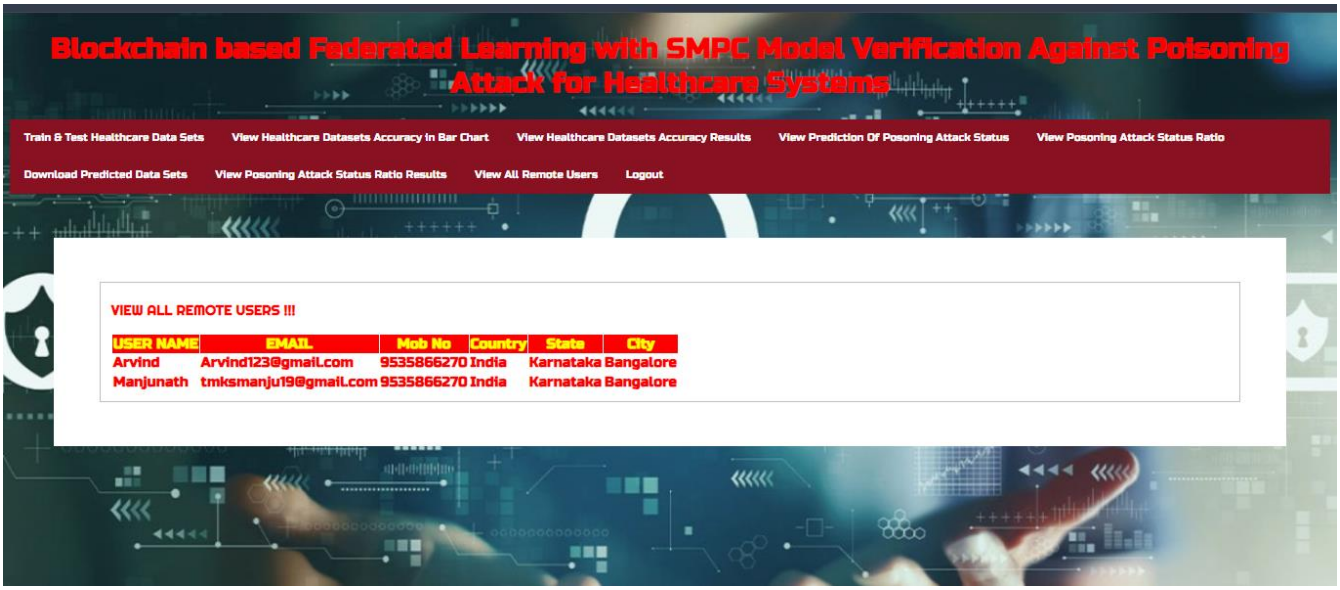


Fig 1: Healthcare Data Manager Home Page

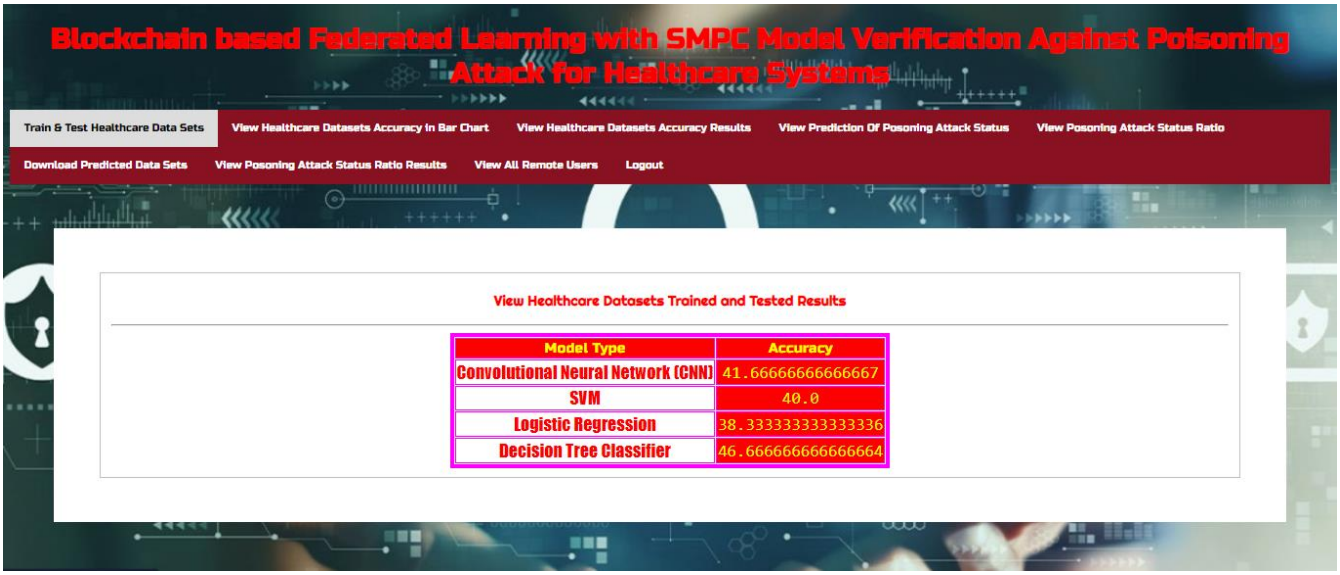


Fig 2: Healthcare Data Manager Operation Train and Test dataset

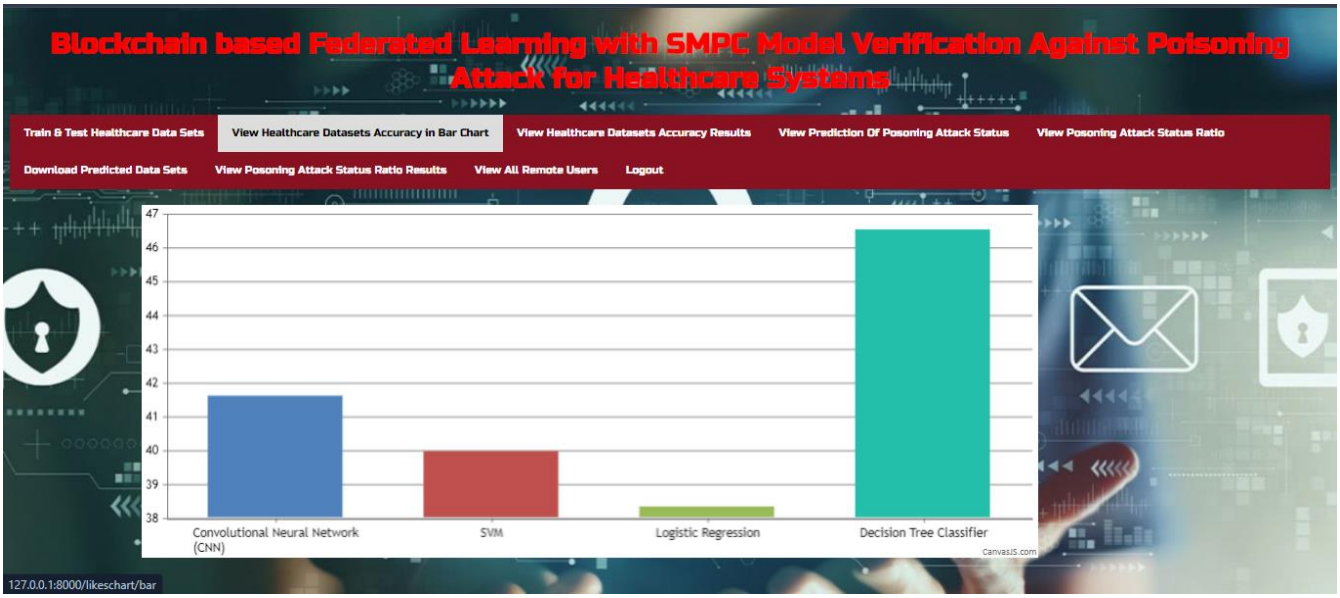


Fig 3: Healthcare Data Manager Operation View Healthcare Datasets Accuracy in bar Chart

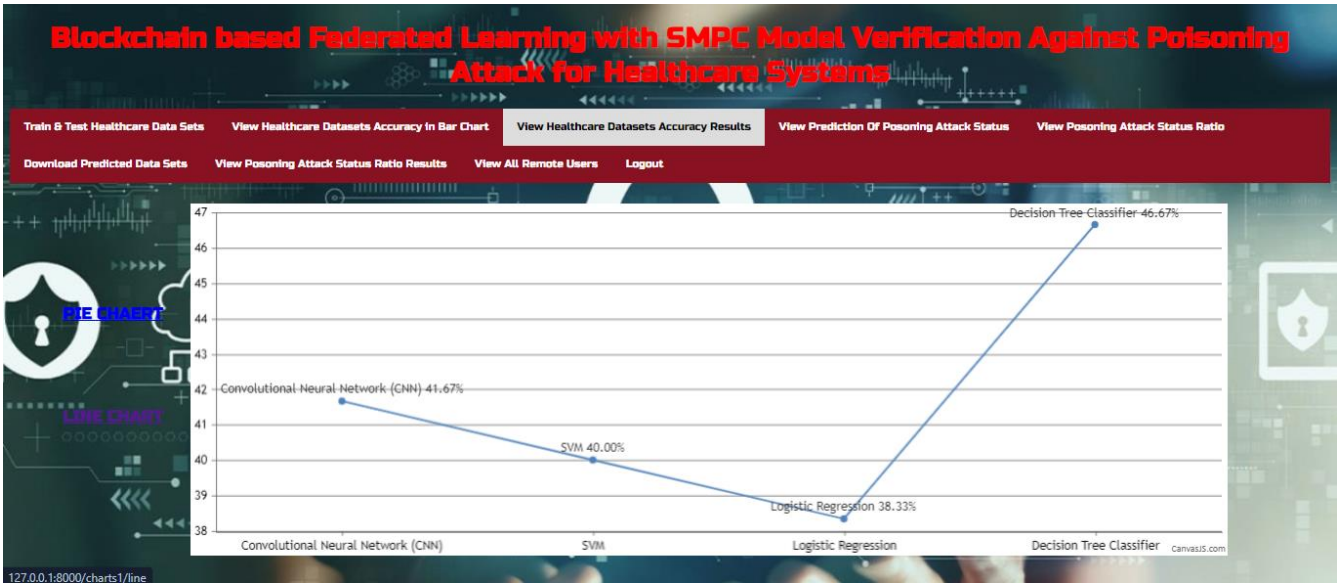


Fig 4: Healthcare Data Manager Operation View Healthcare Datasets Accuracy Results

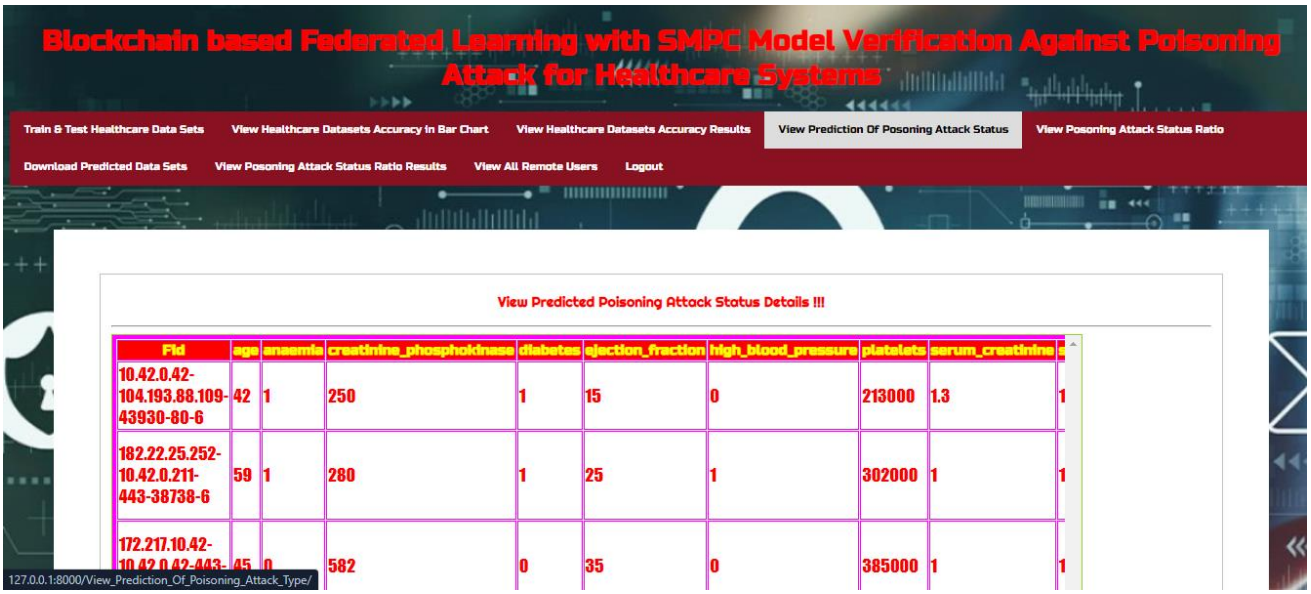


Fig 5: Healthcare Data Manager Operation View Prediction Attack Status

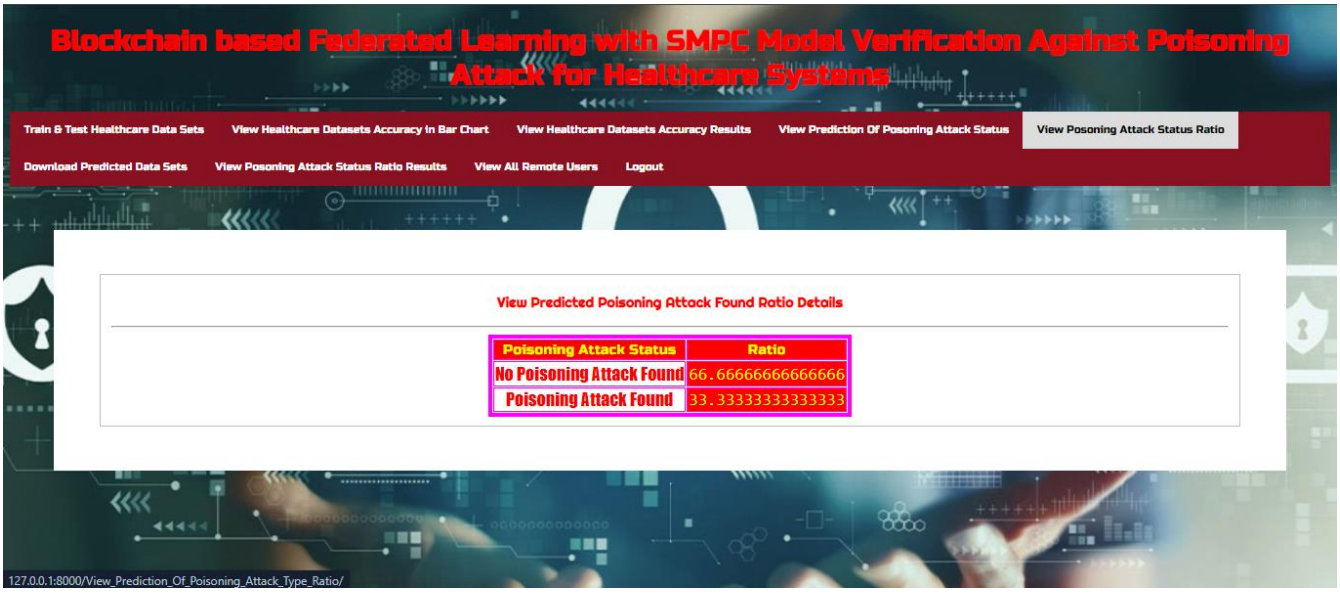


Fig 6: Healthcare Data Manager Operation View Poisoning Attack Status ratio

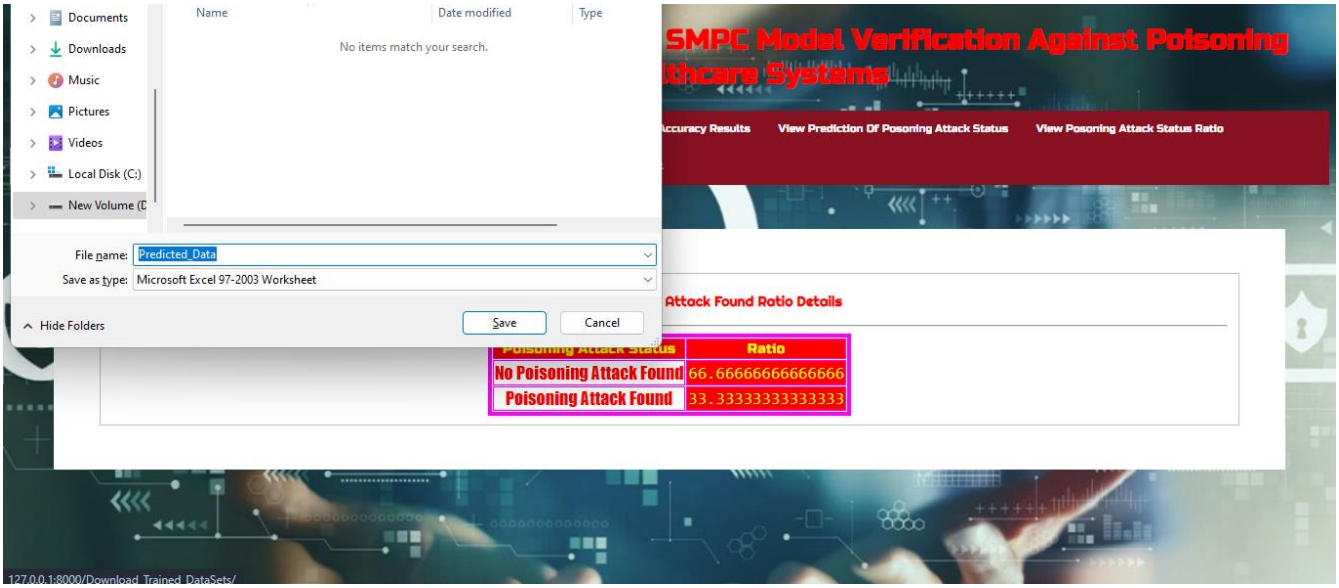


Fig 7: Healthcare Data Manager Operation Download Predicted Dataset

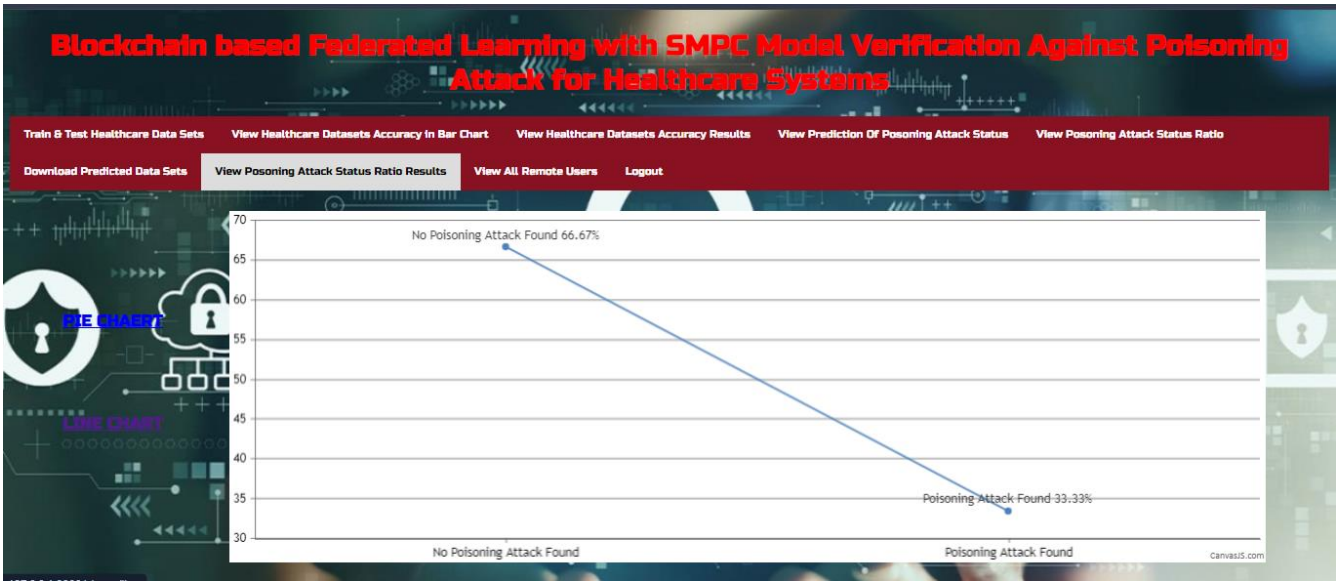


Fig 8: Healthcare Data Manager Operation View Poisoning Attack Status Ratio Results

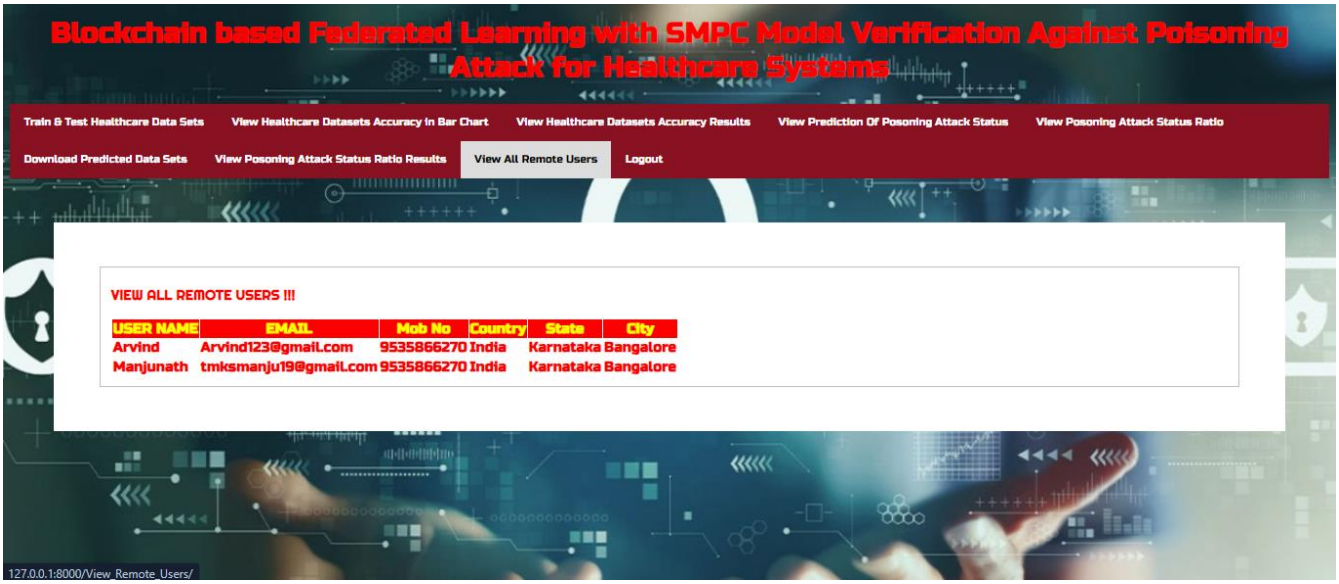


Fig 9: Healthcare Data Manager Operation View All Remote Users

CHAPTER-11

CONCLUSION

This paper proposes block chain-based federated learning with a secure model verification for securing healthcare systems. The main objective is to ensure the local model is poisoned-free while maintaining privacy and providing verifiability for the federated learning participants.

In this framework, we perform a privacy-preserving verification process on the local model before the aggregation process. To preserve privacy on the local model, the verification is performed through an encrypted inference supported by SMPC protocol. This method allows the verifier to check the model with encrypted models and images. Once the local model is verified, the verified share of the local model is sent to the block chain node. Block chain and the hospital will perform SMPC-based secure aggregation. Once the majority of nodes have the same result, the global model is stored in the block chain. Later, the tamper-proof storage will distribute the updated global model to every hospital that joins the federated learning round.

In the experiment, we use Convolutional Neural Network (CNN) based algorithms with several medical datasets to generate local models and aggregate them under FL settings. Our experiment results show that the model encrypted verification process can eliminate all the participants' poisoned models while maintaining the privacy of the local model. In addition, we can recover up to 25% for the global model accuracy. It is essential to mention that our secure inference processing time is almost similar to the original inference process.

In the future, we plan to develop an efficient consensus mechanism for block chain-based aggregation. In this paper, we assume that all hospitals use the homogeneous model and use the same setup to generate their respective local models. However, we plan to broaden our work in the future to support a heterogeneous model in block chain-based federated learning.

.

FUTURE SCOPE

The future scope of this research includes several key areas for further development. Firstly, an efficient consensus mechanism for blockchain-based aggregation needs to be developed to enhance the speed and security of agreement among blockchain nodes. Additionally, the framework should be extended to support heterogeneous models, allowing different hospitals to use varied network architectures. Improving scalability and performance optimization is essential to handle an increasing number of participants without compromising model accuracy or system efficiency. Strengthening defenses against advanced threats and ensuring compliance with healthcare regulations such as HIPAA and GDPR are also crucial. Further enhancement of privacy-preserving techniques through advanced cryptographic methods will help maintain privacy without sacrificing functionality. Real-world deployment and comprehensive case studies will validate the framework in diverse healthcare settings, while ensuring interoperability with existing hospital information systems and electronic health records will facilitate seamless integration. Lastly, developing user-friendly interfaces and tools will make the system accessible and manageable for healthcare professionals, ensuring practical usability and adoption in real-world scenarios.

CHAPTER-12

REFERENCES

[1] L. Sun, X. Jiang, H. Ren, and Y. Guo, “Edge-cloud computing and artificial intelligence in internet of medical things: Architecture, technology and application,” IEEE Access, vol. 8, pp. 101 079–101 092, 2020. [2] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, “On the convergence of fedavg on non-iid data,” arXiv preprint arXiv:1907.02189, 2019.

[3] Z. Yu, S. U. Amin, M. Alhussein, and Z. Lv, “Research on disease prediction based on improved deepfm and iomt,” IEEE Access, vol. 9, pp. 39 043–39 054, 2021.

[4] W. Wei, L. Liu, M. Loper, K.-H. Chow, M. E. Gursoy, S. Truex, and Y. Wu, “A framework for evaluating client privacy leakages in federated learning,” in European Symposium on Research in Computer Security. Springer, 2020, pp. 545–566.

[5] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, “A survey on security and privacy of federated learning,” Future Generation Computer Systems, vol. 115, pp. 619–640, 2021.

[6] Y. Zhao, J. Zhao, M. Yang, T. Wang, N. Wang, L. Lyu, D. Niyato, and K.-Y. Lam, “Local differential privacy-based federated learning for internet of things,” IEEE Internet of Things Journal, vol. 8, no. 11, pp. 8836–8853, 2021.

[7] B. Zhao, K. Fan, K. Yang, Z. Wang, H. Li, and Y. Yang, “Anonymous and privacy-preserving federated learning with industrial big data,” IEEE Transactions on Industrial Informatics, vol. 17, no. 9, pp. 6314–6323, 2021.

[8] X. Wang, S. Garg, H. Lin, J. Hu, G. Kaddoum, M. Jalil Piran, and M. S. Hossain, “Toward accurate anomaly detection in industrial internet of things using hierarchical federated learning,” IEEE Internet of Things Journal, vol. 9, no. 10, pp. 7110–7119, 2022.

[9] V. Mothukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, “Federated-learning-based anomaly detection for iot security attacks,” IEEE Internet of Things Journal, vol. 9, no. 4, pp. 2545–2554, 2022.

[10] B. Wang and N. Z. Gong, “Stealing Hyperparameters in Machine Learning,” in 2018 IEEE Symposium on Security and Privacy (SP), 2018, pp. 36–52.

[11] M. Nasr, R. Shokri, and A. Houmansadr, “Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning,” in 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 739–753.

[12] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, “Data poisoning attacks against federated learning systems,” in European Symposium on Research in Computer Security. Springer, 2020, pp. 480–501.
[13] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, “Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning,” in 2020 fUSENIXg Annual Technical Conference (fUSENIXgfATCg 20), 2020, pp. 493–506.
[14] J. Zhang, B. Chen, X. Cheng, H. T. T. Binh, and S. Yu, “Poisongan: Generative poisoning attacks against federated learning in edge computing systems,” IEEE Internet of Things Journal, vol. 8, no. 5, pp. 3310–3322, 2021.
[15] M. Fang, X. Cao, J. Jia, and N. Gong, “Local model poisoning attacks to byzantine-robust federated learning,” in 29th fUSENIXg Security Symposium (fUSENIXg Security 20), 2020, pp. 1605–1622.
[16] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” in International Conference on Artificial Intelligence and Statistics. PMLR, 2020, pp. 2938–2948.
[17] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, “Privacy-enhanced federated learning against poisoning adversaries,” IEEE Transactions on Information Forensics and Security, vol. 16, pp. 4574–4588, 2021.
[18] X. Guo, Z. Liu, J. Li, J. Gao, B. Hou, C. Dong, and T. Baker, “Verifl: Communication-efficient and fast verifiable aggregation for federated learning,” IEEE Transactions on Information Forensics and Security, vol. 16, pp. 1736–1751, 2021.
[19] Y. Qu, L. Gao, T. H. Luan, Y. Xiang, S. Yu, B. Li, and G. Zheng, “Decentralized privacy using blockchain-enabled federated learning in fog computing,” IEEE Internet of Things Journal, vol. 7, no. 6, pp. 5171–5183, 2020.
[20] Z. Peng, J. Xu, X. Chu, S. Gao, Y. Yao, R. Gu, and Y. Tang, “Vfchain: Enabling verifiable and auditable federated learning via blockchain systems,” IEEE Transactions on Network Science and Engineering, vol. 9, no. 1, pp. 173–186, 2022.’ IEEE Access, vol. 8,