DATA POISON DETECTION SCHEMES FOR DISTRIBUTED LEARNING

A Major Project Report Submitted

In partial fulfillment of the requirement for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

(Artificial Intelligence and Machine Learning)

by

CHUNDURU JAYA TEJASWI	**21N31A6642**
AJJA SREEYA	**21N31A6605**
CHERUKURI JAMUNA CHOWDARY	**21N31A6635**

Under the Guidance of

Dr. THOTA SIVA RATNA SAI

Associate Professor

DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

(Affiliated to JNTU, Hyderabad, Approved by AICTE, Accredited by NBA & NAAC – 'A' Grade, ISO 9001:2015 Certified) Maisammaguda (v), Near Dullapally, Via: Kompally, Hyderabad – 500 100, Telangana State, India.

website: www.mrcet.ac.in

2024-2025

DECLARATION

I hereby declare that the project entitled "DATA POISON DETECTION SCHEMES FOR DISTRIBUTED MACHINE LEARNING" submitted to Malla Reddy College of Engineering and Technology, affiliated to Jawaharlal Nehru Technological University Hyderabad (JNTUH) for the award of the degree of Bachelor of Technology in Computer Science and Engineering- Artificial Intelligence and Machine Learning is a result of original research work done by me.

It is further declared that the project report or any part thereof has not been previously submitted to any University or Institute for the award of a degree or diploma.

Jaya Tejaswi Chunduru – 21N31A6642

Ajja Sreeya – 21N31A6605

Cherukuri Jamuna Chowdary – 21N31A6635

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION - UGC, GOVT. OF INDIA)

Affiliated to JNTUH; Approved by AICTE, NBA-Tier 1 & NAAC with A-GRADE | ISO 9001:2015

CERTIFICATE

This is to certify that this is the bonafide record of the project titled **"Data Poison Detection Schemes For Distributed Learning"** submitted by Jaya Tejaswi Chunduru (21N31A6642) , Ajja Sreeya (21N31A6605) And Cherukuri Jamuna Chwodary (21N31A6635) of B.Tech in the partial fulfillment of the requirements for the degree of **Bachelor of Technology in Computer Science and Engineering – Artificial Intelligence and Machine Learning, Dept. of CI during the year 2024–2025.** The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

Jaya Tejaswi Chunduru – 21N31A6642
Ajja Sreeya – 21N31A6605
Cherukuri Jamuna Chowdary -21N31A6635

Dr. Thota Siva Ratna Sai	**Dr. D. Sujatha**
Associate Professor	Professor and Dean (CSE&ET)
INTERNAL GUIDE	**HEAD OF THE DEPARTMENT**

EXTERNAL EXAMINER

Date of Viva-Voce Examination held on: ______________________________

ACKNOWLEDGEMENT

We feel honored and privileged to place our warm salutation to our college Malla Reddy College of Engineering and Technology (UGC-Autonomous), our Director ***Dr. VSK Reddy*** who gave us the opportunity to have experience in engineering and profound technical knowledge.

We are indebted to our Principal ***Dr. S. Srinivasa Rao*** for providing us with facilities to do our project and his constant encouragement and moral support which motivated us to move forward with the project.

We would like to express our gratitude to our Head of the Department ***Dr. D. Sujatha,*** Professor and Dean (CSE&ET) for encouraging us in every aspect of our system development and helping us realize our full potential.

We would like to express our sincere gratitude and indebtedness to our project supervisor ***Dr. Thota Siva Ratna Sai,*** Associate Professor for her valuable suggestions and interest throughout the course of this project.

We convey our heartfelt thanks to our Project Coordinator***, Mr. Chandrashekar,*** Assistant Professor for allowing for his regular guidance and constant encouragement during our dissertation work

We would also like to thank all supporting staff of department of CI and all other departments who have been helpful directly or indirectly in making our Major Project a success.

We would like to thank our parents and friends who have helped us with their valuable suggestions and support has been very helpful in various phases of the completion of the Major Project.

By:
Jaya Tejaswi Chunduru – 21N31A6642
Ajja Sreeya – 21N31A6605
Cherukuri Jamuna Chowdary -21N31A6635

ABSTRACT

In distributed machine learning systems, data is divided and processed across multiple nodes to improve efficiency and scalability. However, this structure also makes the system vulnerable to **data poisoning attacks**, where incorrect or malicious data is injected into the training set. Such poisoned data can lead to inaccurate predictions and reduced model performance. This project proposes an effective solution to detect and eliminate poisoned data using the **Isolation Forest algorithm**, an unsupervised machine learning technique for anomaly detection.

The system implements two distributed training approaches: **Basic Distributed Machine Learning (Basic-DML)** and **Semi Distributed Machine Learning (Semi-DML)**. In Basic-DML, training is performed independently by worker nodes, while in Semi-DML, the central server also participates in the training process. The performance of these methods is compared with a standard **Support Vector Machine (SVM)** model, which does not include any poison detection mechanism.

Using a real-world **heart disease dataset**, the project demonstrates that applying data poison detection significantly improves model accuracy. The results are visualized using performance graphs, highlighting the effectiveness of the proposed methods. This system provides a scalable and secure framework for training reliable machine learning models in distributed environments.

TABLE OF CONTENTS

CONTENTS	Page No.
1. INTRODUCTION	**1**
2. Problem Statements	2-3
3. Objectives	4
4. Summary	4-5
2. LITERATURE SURVEY	**6**
1. Existing System	6
2. Proposed System	7
3. SYSTEM REQUIREMENTS	**8**
1. Introduction	8
2. Software and Hardware Requirement	8-9
3. Functional and Non-Functional Requirements	9-10
4. Other Requirements	10-11
5. Summary	12
4. SYSTEM DESIGN	**13**
1. Introduction	13
2. Architecture Diagram	14
3. UML Diagrams / DFD	15-19
4. Summary	20-21
5. IMPLEMENTATION	**22**
1. Algorithms	22-23
2. Architectural Components	23-24
3. Feature Extraction	24-25
4. Packages/Libraries Used	26
5. Source Code	27-35
6. Output Screens	36-43
6. SYSTEM TESTING	**44**
1. Introduction	44
2. Test Cases	44-45
3. Results and Discussions	45-46
6.3.1. Datasets	47
4. Performance Evaluation	47
5. Summary	48
7. CONCLUSION & FUTURE ENHANCEMENTS	**49-50**
8. REFERENCES	**51-52**

LIST OF FIGURES

Fig No.	Figure Title	Page No.
4.2	Architecture Diagram	14
4.3.1	Class Diagram	15
4.3.2	Sequence Diagram	16
4.3.3	Use Case Diagram	17
4.3.4	Activity Diagram	18
4.4	Data Flow Diagram	19
5.5.1 – 5.5.8	Ouput Screens	36 - 43

LIST OF TABLES

Sno.	Table Name	Page no.
1.	Software Requirements	8
2.	Hardware Requirements	9

LIST OF ABBREVIATIONS

Sno.	ABBREVIATIONS
1.	SVM – Support Vector Machine
2.	ML – Machine Learning
3.	DML – Distributed Machine Learning
4.	CSV – Comma Separated Values
5.	iForest – Isolation Forest(Anomaly Detection Algorithm)
6.	GUI – Graphical User Interface
7.	ECG - Electrocardiogram

CHAPTER -1

INTRODUCTION

Machine learning models are often trained on large datasets to make accurate predictions. When data is too large for a single machine, Distributed Machine Learning (DML) is used to divide and process data across multiple systems. While this improves speed and efficiency, it also increases the risk of data poisoning, where incorrect or malicious entries are added to the dataset. This can seriously reduce the model's accuracy and reliability. This project aims to solve that problem by using the Isolation Forest algorithm to detect and remove poisoned data before model training. It introduces two distributed approaches: Basic-DML, where only worker nodes train models, and Semi-DML, where both the server and workers contribute. These are compared with a standard Support Vector Machine (SVM) model that does not include any poison detection. Using a heart disease dataset, the system demonstrates that models trained after poison detection achieve higher accuracy. The results prove that combining poison detection with distributed training makes machine learning more secure and effective.

Machine Learning (ML) is a key subfield of Artificial Intelligence (AI) that focuses on enabling machines to learn patterns from data and make intelligent decisions or predictions without being explicitly programmed for each task. As digital data becomes more abundant and accessible, ML has emerged as a powerful tool for automating decision-making and solving complex problems across industries, including healthcare, finance, marketing, and cybersecurity.

At its core, machine learning involves developing mathematical models that can generalize from historical data. These models can be trained to identify trends, classify inputs, detect anomalies, or forecast future outcomes based on observed patterns. Once trained, the model can be applied to new, unseen data to provide useful outputs with high accuracy.

There are three major types of ML:

1. **Supervised Learning** – This approach uses labeled datasets to train models. The model learns to map inputs to known outputs. Common applications include classification (e.g., disease prediction) and regression (e.g., stock price prediction).

2. **Unsupervised Learning** – This method works with unlabeled data to uncover hidden structures or groupings. Clustering and dimensionality reduction are key techniques used in data exploration.

3. **Reinforcement Learning** – Here, an agent learns by interacting with an environment, receiving feedback in the form of rewards or penalties, and improving its actions over time.

In this project, ML techniques—particularly Support Vector Machines (SVM)—are used to predict heart disease. The accuracy of these models can be negatively affected by poor-quality or poisoned data. Therefore, anomaly detection using algorithms like Isolation Forest is applied to identify and remove such data before training the models. Additionally, distributed machine learning approaches are implemented to handle the dataset efficiently across multiple nodes, improving both speed and scalability.

1.1 Problem Statements

In distributed machine learning systems, data is divided and processed across multiple nodes to improve performance and scalability. However, this setup also introduces a major vulnerability: data poisoning attacks.

- In such attacks, attackers or errors introduce abnormal, misleading, or incorrect data into the training dataset, which can significantly reduce the model's accuracy and reliability.
- Traditional machine learning models like Support Vector Machines (SVM) do not have built-in mechanisms to detect or handle poisoned data. As a result, they are highly sensitive to outliers and corrupted entries, which leads to incorrect predictions.
- Furthermore, in a distributed environment, it becomes even more difficult to monitor and clean data due to its spread across multiple nodes. There is a lack of effective systems that can automatically detect and eliminate poisoned data before training, especially in distributed frameworks.

  This project addresses the following key problems:

  - Reduced prediction accuracy due to the presence of abnormal or fake data.
  - Difficulty in detecting and cleaning data in distributed environments.
  - Lack of comparison between standard and poison-aware training method

In modern machine learning (ML) systems, particularly in distributed environments, the integrity and quality of training data are critical to achieving accurate and reliable results. However, real-world datasets are often vulnerable to data poisoning—the presence of incorrect, corrupted, or intentionally malicious data entries. These poisoned data points can significantly degrade the performance of ML models by misleading the learning process, resulting in inaccurate predictions and reduced system trustworthiness.

This issue becomes even more challenging in distributed machine learning (DML), where datasets are divided and processed across multiple nodes. In such systems, if poisoned data is present in any node's portion of the dataset, the resulting trained models may carry forward these inaccuracies, leading to poor overall performance. Additionally, traditional ML algorithms like Support Vector Machines (SVM) do not have built-in mechanisms to detect or handle poisoned data, making them highly susceptible to such attacks or errors.

The problem is further compounded when dealing with sensitive applications such as medical diagnosis, where incorrect predictions can have serious consequences. For example, an abnormally high blood pressure value (e.g., 2233) in a heart disease dataset could significantly distort the model's understanding, leading to misclassification.

1.2 Objectives

The primary goal of this project is to enhance the accuracy and reliability of distributed machine learning models by detecting and removing poisoned data from training datasets. The specific objectives are:

1. To study and understand the impact of data poisoning in distributed machine learning environments.

2. To implement a Data Poison Detection technique (using methods like Isolation Forest) to identify and eliminate abnormal or malicious data points from the training dataset.

3. To design and develop a distributed machine learning framework using Basic-DML and Semi-DML approaches.

4. To compare the performance of traditional SVM (without poison detection) with the proposed DML models (with poison detection) in terms of accuracy.

5. To demonstrate the effectiveness of data poison detection in improving prediction accuracy using a real-world dataset (heart disease dataset).

6. To visualize and analyze the results through accuracy comparison graphs for better understanding and evaluation.

1.2 Summary

This project focuses on enhancing the reliability of distributed machine learning systems by detecting and removing poisoned data from training datasets. In distributed environments, data is divided among multiple nodes, making it more vulnerable to data poisoning attacks—where attackers introduce incorrect or misleading data to compromise the learning process.

To address this challenge, the project introduces Data Poison Detection techniques, specifically using the Isolation Forest algorithm, to identify and filter out abnormal values in the dataset. The system is built using two distributed machine learning approaches: Basic-DML, where worker nodes train models independently, and Semi-DML, where the central server also participates in the training process.

A heart disease dataset is used as a test case, containing deliberate anomalies to simulate poisoned data. The results show that traditional Support Vector Machine (SVM) models, which do not account for poisoned data, yield low accuracy. In contrast, the proposed DML methods significantly improve prediction accuracy after removing poisoned entries.

The project successfully demonstrates how integrating data poison detection in distributed ML frameworks can lead to more accurate and secure models, making it a valuable contribution to secure machine learning practices.

This project presents a system for detecting and eliminating poisoned data in distributed machine learning environments to improve model accuracy and reliability. Machine Learning (ML) models are widely used for predictive tasks, but their performance heavily depends on the quality of the training data. In real-world scenarios, especially in distributed systems, datasets may contain invalid, corrupted, or intentionally poisoned data that can mislead the learning process and drastically reduce prediction accuracy.

The primary objective of this project is to apply a data poison detection technique—Isolation Forest—to identify and remove outliers from a heart disease dataset. The cleaned dataset is then used to train machine learning models. The system evaluates three approaches: a standard Support Vector Machine (SVM), Basic Distributed Machine Learning (Basic-DML), and Semi-Distributed Machine Learning (Semi-DML).

In Basic-DML, the dataset is divided into two parts and processed by two distributed worker nodes. Each node applies data poison detection, trains models, and sends the results back to the central server. In Semi-DML, the central server itself performs both data cleaning and model training, allowing better control over data integrity.

CHAPTER 2
LITERATURE SURVEY

2.1 Existing System

In the existing system, traditional machine learning models such as Support Vector Machines (SVM) are used to train and make predictions based on the provided dataset. These models assume that the input data is clean, accurate, and trustworthy. However, in real-world scenarios—especially in distributed or collaborative environments—datasets may contain poisoned or abnormal data due to errors, noise, or malicious attacks.

The existing system typically involves the following characteristics:

1. **No Data Poison Detection Mechanism:**
   Traditional ML models do not have built-in methods to detect or handle poisoned data. They treat all data points equally, which makes them highly vulnerable to inaccurate learning if the dataset is compromised.

2. **Single-Node or Basic Distributed Learning:**
   While some models operate in a distributed environment, they often lack coordination mechanisms or security protocols to ensure the integrity of the data being processed on different nodes.

3. **Lower Accuracy in the Presence of Anomalies:**
   If the dataset includes invalid values or outliers (e.g., an unrealistic blood pressure value of 2233 in a heart disease dataset), the model's predictions become unreliable and result in significantly lower accuracy.

4. **Lack of Robustness and Adaptability:**
   Existing systems are not designed to adapt to corrupted data or to improve themselves through anomaly detection or correction mechanisms. As a result, they perform poorly when data quality is compromised.

In summary, the existing system is simple and direct but not equipped to handle poisoned or malicious data, making it unsuitable for use in sensitive or large-scale distributed machine learning applications.

2.2 Proposed System

The proposed system improves machine learning by detecting and removing poisoned (bad or incorrect) data before training the model. It uses a technique called Isolation Forest to find these bad data points.

The system works in a **distributed setup** with two methods:

- **Basic-DML**, where only worker nodes train the model.
- **Semi-DML**, where both the server and workers train the model together.

By cleaning the data and using distributed training, the system achieves higher accuracy compared to traditional models like SVM. It also shows accuracy comparisons through graphs, proving that detecting and removing poisoned data leads to better performance.

CHAPTER 3
SYSTEM REQUIREMENTS

3.1Introduction

To successfully develop and run the proposed Data Poison Detection System for Distributed Machine Learning, both hardware and software components are required. These requirements ensure that the system can handle data preprocessing, model training, poison detection, and result analysis efficiently.

This section outlines the **minimum and recommended hardware specifications** along with the **necessary software tools and platforms** used for implementation, testing, and evaluation of the system. The goal is to ensure smooth performance, support for machine learning libraries, and the ability to run distributed processes across multiple nodes or machines.

3.2 Software and Hardware Requirement

3.2.1 Hardware Requirements

Software Requirements :

Component	**Specification**
Operating System	Windows 10 / Ubuntu 18.04 or higher
Programming Language	Python 3.8 or above
Python Libraries	scikit-learn, pandas, numpy, matplotlib, seaborn
Development Environment	Jupyter Notebook / VS Code / PyCharm
ML Libraries	scikit-learn (for SVM, Isolation Forest, etc.)
Visualization Tools	matplotlib, seaborn

Hardware Requirements:

Component	**Minimum Requirement**	**Recommended**
Processor (CPU)	Intel Core i3 or equivalent	Intel Core i5/i7 or higher
RAM	4 GB	8 GB or more
Hard Disk	100 GB (free space)	256 GB SSD
Graphics (Optional)	Integrated GPU	Dedicated GPU (for large datasets)
Network	Required for distributed setup	Stable LAN/Wi-Fi connection

3.3 Functional and Non-Functional Requirements

Functional Requirements :

These are the core functions that the system **must perform**:

1. **Data Input and Preprocessing**
   - The system should allow users to input datasets.
   - It should clean and prepare the data for training.
2. **Poison Detection**
   - The system must detect and remove poisoned or abnormal data using the **Isolation Forest** algorithm.
3. **Distributed Model Training**
   - The system should train models using **Basic-DML** and **Semi-DML** approaches.
   - Worker nodes and the server should communicate and share training results.
4. **Model Evaluation**
   - The system must evaluate the model accuracy and performance after training.
5. **Result Visualization**
   - It should display comparison graphs (e.g., SVM vs Basic-DML vs Semi-DML) to show performance differences.

Non-Functional Requirements :

These describe how the system should behave and perform:

1. **Performance**
   - The system should efficiently handle datasets and complete model training in reasonable time.
2. **Scalability**
   - The system should support scaling to multiple nodes for distributed training.
3. **Usability**
   - The user interface (if any) should be simple and easy to navigate.
4. **Reliability**
   - The system should provide consistent and accurate results even after multiple runs.
5. **Maintainability**
   - The system should be modular and easy to update or modify in the future.
6. **Portability**
   - The code should run on different operating systems like Windows and Linux with minimal changes.

3.4 Other Requirements

In addition to software, hardware, functional, and non-functional requirements, the following other requirements are essential for the successful implementation and operation of the system:

1. **User Knowledge:**
   Users should have a basic understanding of Python, machine learning concepts, and dataset formats (e.g., CSV files).

2. **DatasetRequirements**
   The system requires structured, labeled datasets suitable for classification tasks. For testing and demonstration, a heart disease dataset with some abnormal entries is used.

3. **Documentation**
   Proper documentation must be provided, including instructions on how to set up, run, and interpret results from the system.

4. **Licensing and Compliance**
   All external libraries, tools, and datasets used must comply with open-source or permissible licenses.

5. **Network Configuration**
   A local or internet-based network is needed for communication between the central server and worker nodes in distributed setups.

6. **Data Integrity and Validation**
   The system should validate input data to prevent incorrect formats or missing values from affecting the processing pipeline.

7. **Error Handling and Logging**
   Basic error handling should be implemented to manage exceptions during dataset upload, poison detection, or model training. Logs should be maintained for debugging and analysis.

8. **Scalability Support**
   The system should be flexible enough to accommodate future enhancements, such as adding more worker nodes or integrating deep learning models.

3.5 Summary

To successfully implement the Data Poison Detection System in Distributed Machine Learning, the project requires a combination of specific software and hardware components. On the software side, it relies on Python (version 3.8 or higher) and essential libraries like scikit-learn, pandas, numpy, matplotlib, and seaborn for data processing, model training, and visualization. A development environment such as Jupyter Notebook or VS Code is recommended. Hardware requirements include at least an Intel i3 processor with 4 GB RAM and 100 GB of storage, while an Intel i5/i7 processor with 8 GB RAM and SSD storage is ideal for better performance, especially when handling large datasets. A stable network connection is necessary for enabling distributed model training across nodes. Functionally, the system must be capable of inputting datasets, detecting and removing poisoned data, training models using both Basic-DML and Semi-DML methods, and visualizing performance results. Non-functional needs include performance efficiency, scalability, reliability, usability, portability, and ease of maintenance. Additionally, users are expected to have basic knowledge of Python and machine learning. The system should also include proper documentation, follow open-source licensing, and ensure secure operation when used in networked environments. Together, these requirements ensure the system runs smoothly, remains accurate, and is suitable for both educational and real-world use.

CHAPTER 4

SYSTEM DESIGN

4.1 Introduction

System design is a crucial phase in the development of any software project, as it defines the overall structure, components, and flow of the system. In this project, the system is designed to detect poisoned data and improve the accuracy of machine learning models in a distributed environment. The design includes the architecture for data input, poison detection using the Isolation Forest algorithm, and distributed model training using Basic-DML and Semi-DML methods.

The system is structured to operate in multiple stages: starting from data collection and preprocessing, followed by anomaly detection, model training, and performance evaluation. The design also outlines how worker nodes and the central server interact during distributed training. This modular and layered design ensures scalability, efficiency, and clear separation of functions, making the system easier to develop, test, and maintain.

4.2 Architecture Diagram

An architecture diagram is a visual representation of a system's structure, showcasing how its components are interconnected, how they communicate, and how data flows through the system, serving as a blueprint for understanding and communicating complex designs.

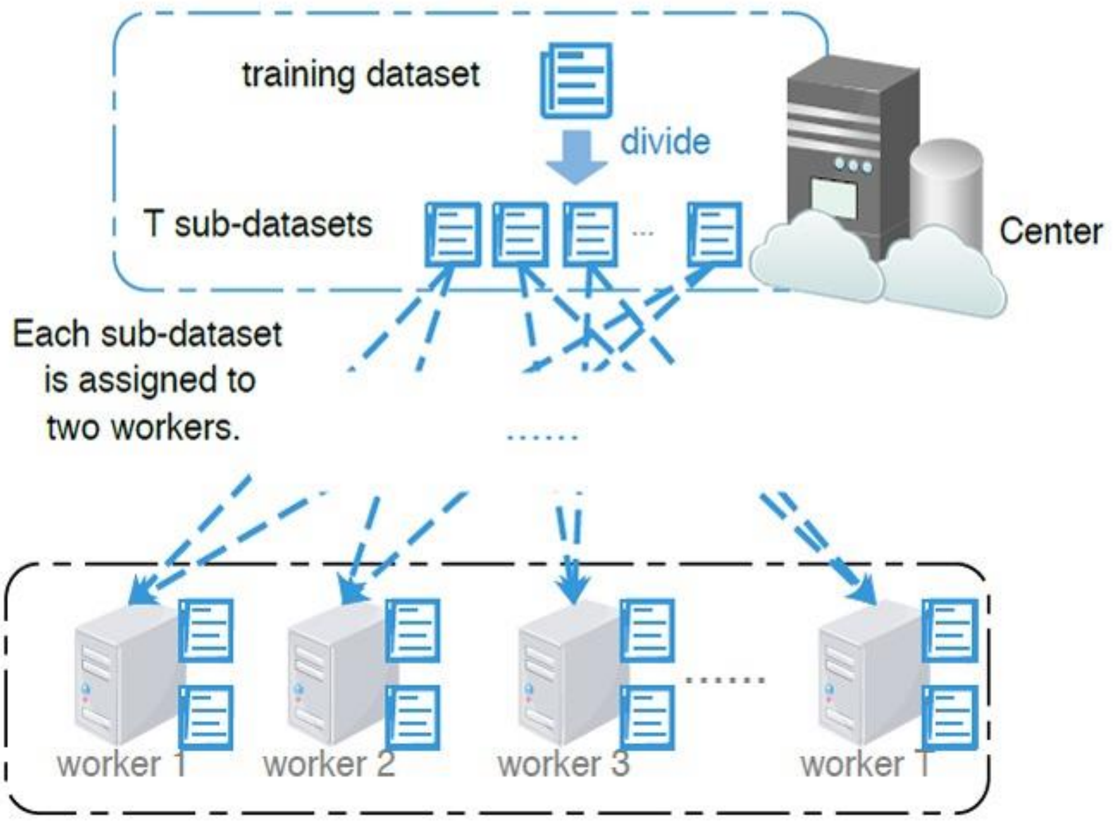


Figure 4.2: Architecture Diagram

4.2 UML DIAGRAMS / DFD

4.2.1 Class Diagram

A class diagram is a visual representation of the classes in a system and their relationships. It shows attributes, methods, and associations between classes, helping to outline the system's structure. Class diagrams are essential in object-oriented design, facilitating communication and understanding among developers and stakeholders.

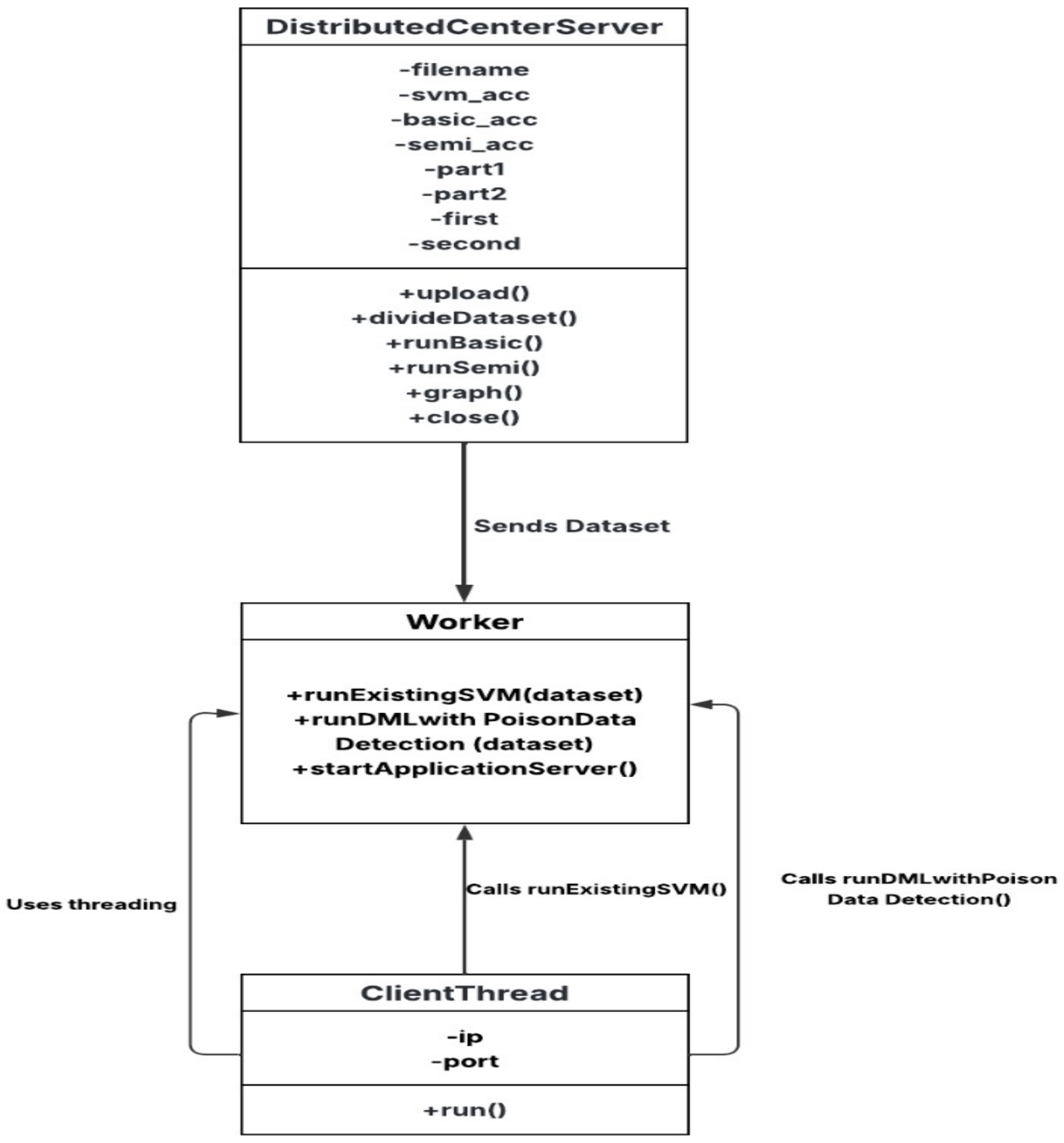


Figure 4.3.1: Class Diagram for Data Poison Detection.

4.2.2 Sequence Diagram

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

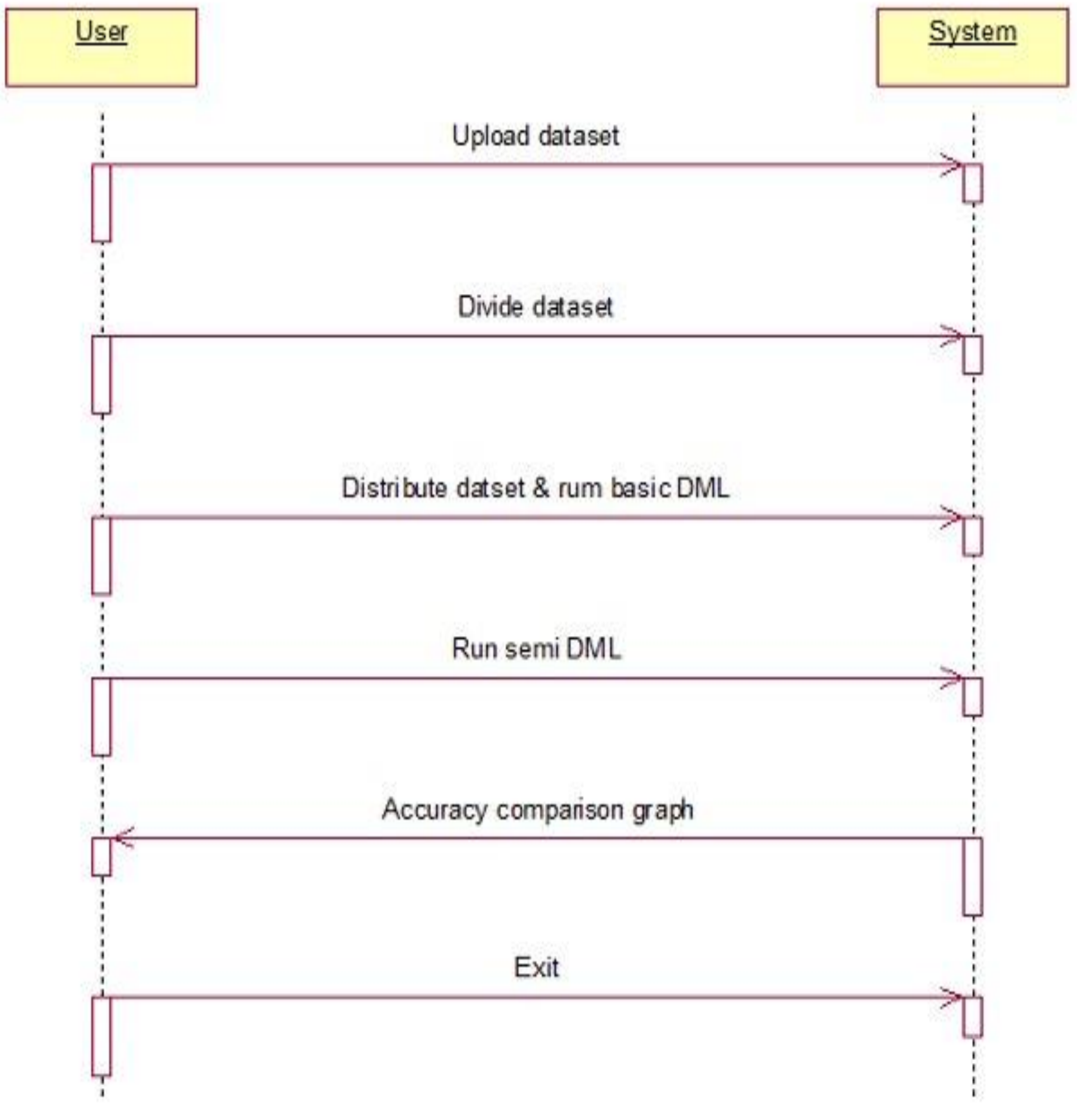


Figure 4.3.2: Sequence Diagram for Data Poison Detection.

4.2.3 Use Case

Use Case during requirement elicitation and analysis to represent the functionality of the system. Use case describes a function by the system that yields a visible result for an actor. The identification of actors and use cases result in the definitions of the boundary of the system i.e., differentiating the tasks accomplished by the system and the tasks accomplished by its environment.

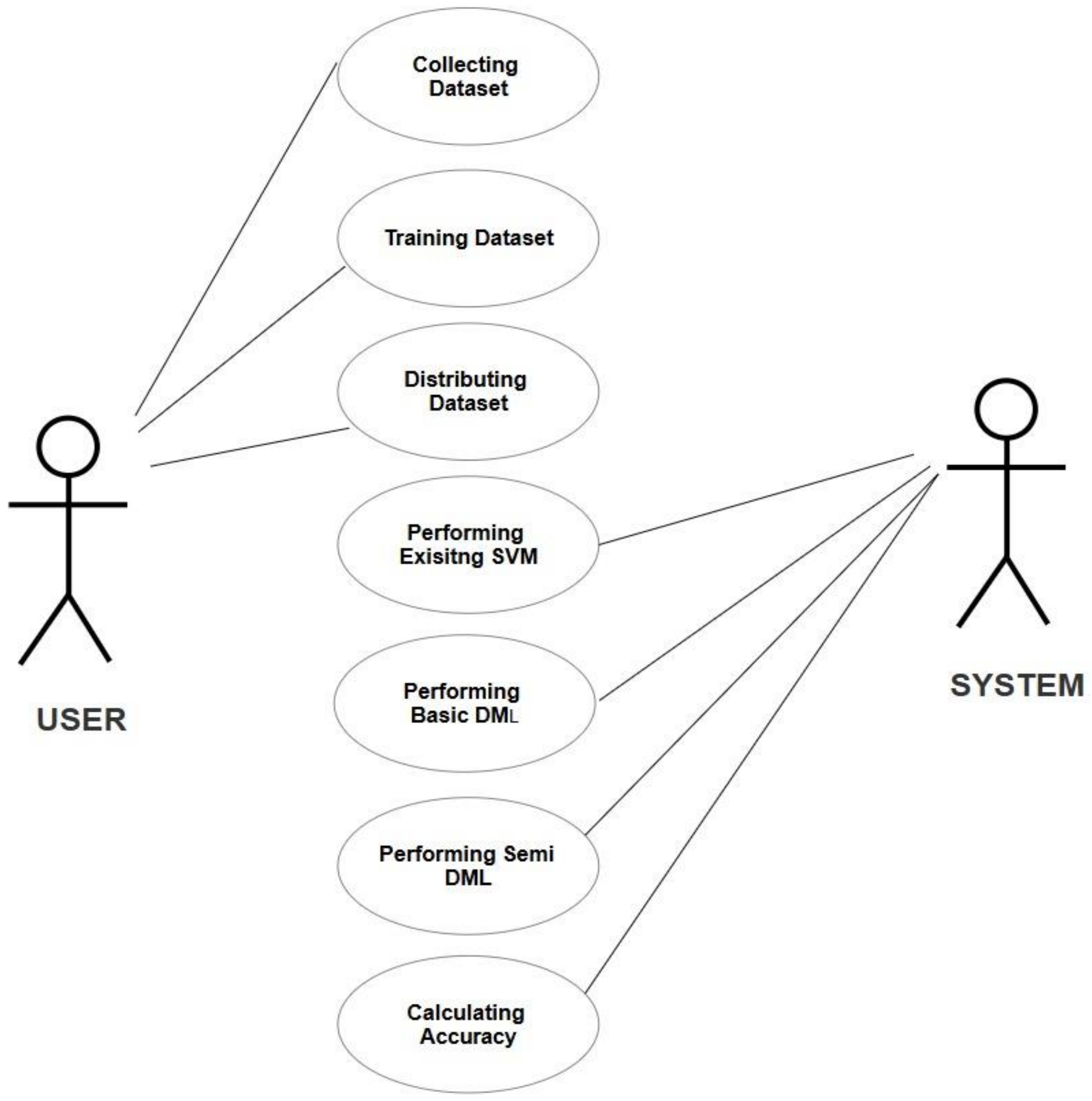
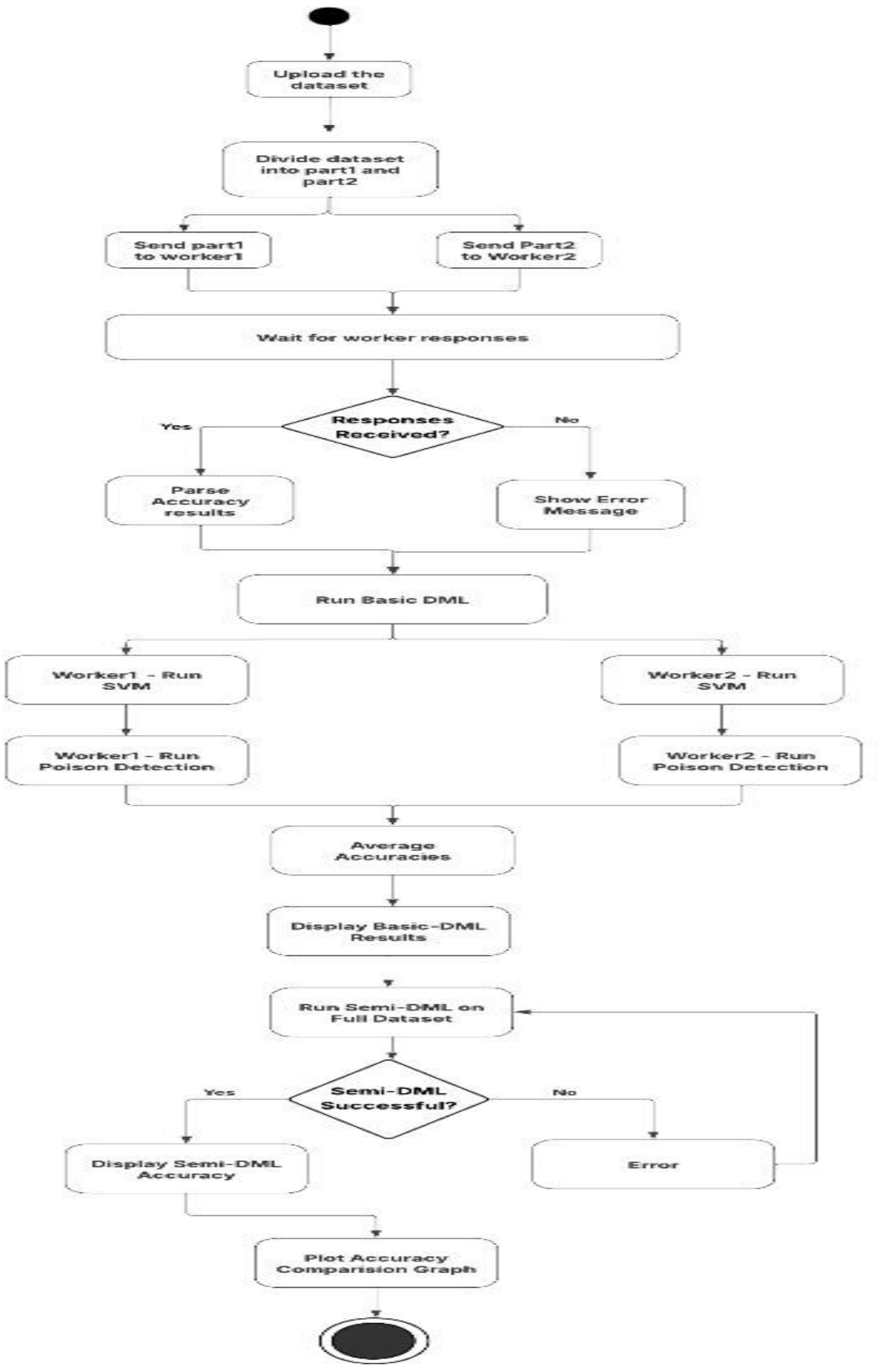


Figure 4.3.3: Use Case diagram for Data Poison Detection.

4.2.4 Activity Diagram

The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.



Upload the dataset
Divide dataset into part1 and part2
Send part1 to worker1
Send Part2 to Worker2
Wait for worker responses
Responses Received?
Yes
No
Parse Accuracy results
Show Error Message
Run Basic DML
Worker1 - Run SVM
Worker2 - Run SVM
Worker1 - Run Poison Detection
Worker2 - Run Poison Detection
Average Accuracies
Display Basic-DML Results
Run Semi-DML on Full Dataset
Semi-DML Successful?
Yes
No
Display Semi-DML Accuracy
Error
Plot Accuracy Comparision Graph

Figure 4.3.4: Activity Diagram for Dtata poison detection.

4.3 Data Flow Diagram

Data Flow Diagrams (DFDs) are graphical representations used to model the flow of data within a system. They provide a visual overview of how data moves through different processes, stores, and external entities within the system.

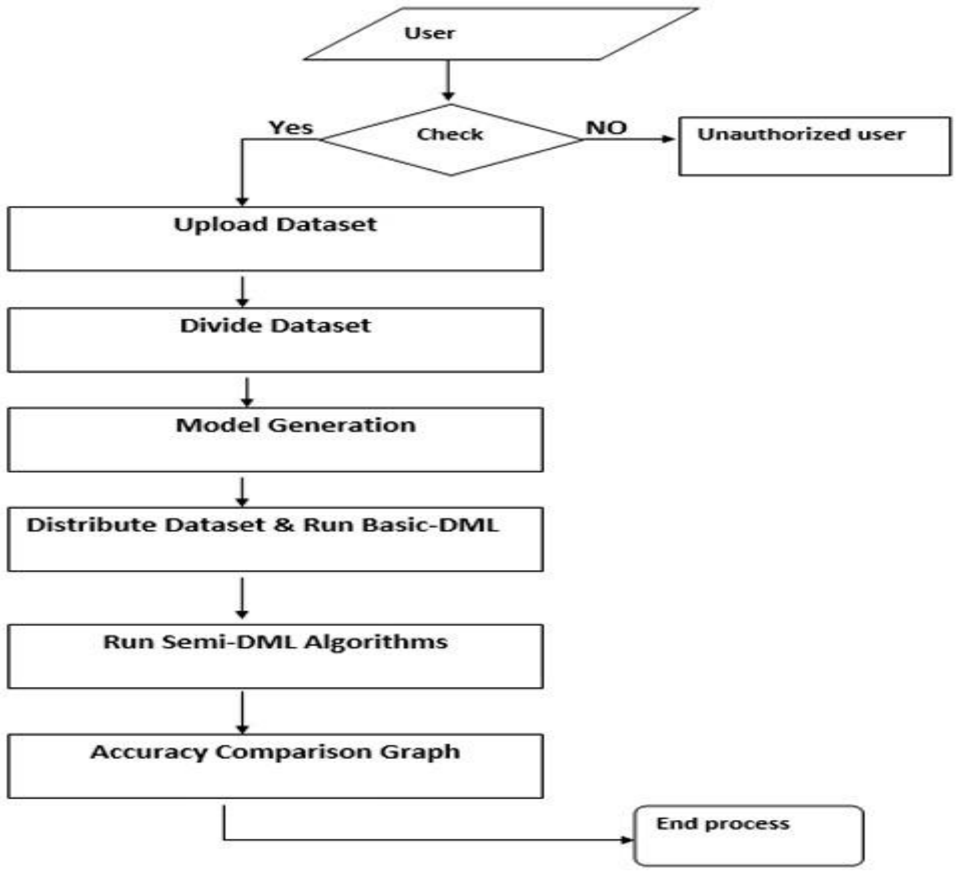


Fig 4.4 Data Flow Diagram for Data Poison Detection.

4.4 Summary

Objective: Improve accuracy of machine learning in distributed systems by detecting and removing poisoned (invalid/malicious) data.

Approach: Uses **Data Poison Detection** with IsolationForest to identify and remove outliers.Compares traditional **SVM** with **Basic DML** and **Semi DML** approaches.

System Architecture:

Worker1 & Worker2:

4.4.1 Receive split datasets from the center server.

4.4.2 Train SVM and Basic DML models.

4.4.3 Return accuracy results.

Center Server:

4.4.4 Uploads and splits dataset.

4.4.5 Coordinates workers.

4.4.6 Runs Semi DML model.

Techniques:

Basic DML: Distributed model training on worker nodes.

Semi DML: Centralized model training with poison detection applied directly.

Dataset: Heart disease dataset with intentional data poisoning (e.g., blood pressure = 2233).

Results:

SVM Accuracy (with poisoned data): 19%

Basic DML Accuracy (after poison removal): 51%

Semi DML Accuracy: 59%

Conclusion: Data Poison Detection significantly improves model accuracy.Semi DML outperforms both Basic DML and SVM.

CHAPTER 5
IMPLEMENTATION

5.1 Algorithms

1. Support Vector Machine (SVM)

- **Type:** Supervised Learning (Classification)
- **Role in Project:** Baseline model for comparison
- **Functionality:**
    - Trained directly on raw (poisoned) dataset.
    - Used to demonstrate the negative effect of poisoned data.
- **Limitations in Project:**
    - Does not include any mechanism to detect or clean poisoned data.
    - Accuracy significantly drops when trained on poisoned data (e.g., ~19%).

2. Isolation Forest (iForest)

- **Type:** Unsupervised Anomaly Detection
- **Purpose:** Detect poisoned/outlier data entries in the dataset before training ML models.
- **How It Works:**
    - Builds multiple random trees by randomly selecting a feature and a split value.
    - Anomalous (poisoned) points are isolated in fewer splits, hence have shorter path lengths in trees.
    - Calculates an anomaly score for each record.
    - Points with scores above a threshold are marked as outliers (poisoned).
- **Implementation:**
    - Used from sklearn.ensemble.IsolationForest in Python.
    - Applied before training DML models to clean the data.

3. Basic Distributed Machine Learning (Basic-DML)

- **Type:** Distributed ML Framework
- **Goal:** Train models across multiple worker nodes after cleaning poisoned data.
- **Architecture:**
    - Central server divides the dataset into two equal parts.
    - Sends each part to **Worker1** and **Worker2**.
- **Process:**
    0. Workers receive their respective datasets.
    1. Apply IsolationForest to detect and remove poisoned data.

2. Train SVM (for baseline) and DML (on cleaned data).
3. Return model accuracy results to the central server.

- **Outcome:**

Achieved ~51% accuracy on cleaned data using Basic-DML, showing improvement over poisoned SVM training.

4. Semi-Distributed Machine Learning (Semi-DML)

- **Type:** Hybrid (Centralized + Distributed approach)
- **Goal:** Let the central server handle both data cleaning and training to further improve accuracy.

- **Architecture:**
  - Central server does **not distribute** data to worker nodes.
  - Handles:
  - Dataset upload
  - Poison detection (via Isolation Forest)
  - Model training using cleaned dataset

- **Process:**
  - Central server loads full dataset.
  - Applies IsolationForest to detect and remove poisoned records.
  - Trains ML model (DML) using clean data.
  - Calculates final accuracy.

- **Outcome:**

  Achieved the **highest accuracy (~59%)**, showing best performance among all techniques.

5.2Architectural Components

1. Center Server

- Main Responsibilities:
  Upload and load the dataset.
  Divide dataset into two equal parts for distribution.
  Coordinate with Worker nodes (send/receive data).
  Run Semi-DML: performs poisoning detection and model training locally.
  Display and compare accuracy results via a graph.
- Key Functions:
  Upload Dataset
  Divide Dataset
  Distribute Dataset & Run Basic-DML
  Run Semi-DML
  Display Accuracy Graph

2. Worker Node 1 (Worker1)

Receives: First half of the dataset from the Center Server.
Responsibilities:

Apply IsolationForest to detect and remove poisoned data.
Train:

- SVM model (on raw data)
- DML model (on cleaned data)
- Send accuracy results back to the Center Server.

3. Worker Node 2 (Worker2)

- Receives: Second half of the dataset from the Center Server.
- Responsibilities: (same as Worker1)
- Apply IsolationForest
- Train SVM and DML
- Return accuracy results to the Center Server

4. Communication Channels

- Likely implemented using sockets or inter-process communication (IPC).
- Data is passed between Center Server and Workers for:
- Dataset distribution
- Model results collection

Technologies Likely Used

- **Python** (as per IsolationForest usage from sklearn)
- GUI (possibly using **Tkinter**, **PyQt**, or similar)
- **CSV file handling** for dataset
- **Socket programming** or subprocess management for running nodes.

5.3 Feature Extraction

Step 1: Load Dataset

- File used: heart.csv
- Contains both valid and potentially poisoned (invalid) values.

Step 2: Identify Useful Columns

- Relevant features include:
- age
- sex
- cp (chest pain type)
- trestbps (resting blood pressure)
- chol (cholesterol)
- fbs (fasting blood sugar)

- restecg (resting ECG)
- thalach (max heart rate)
- exang (exercise-induced angina)
- oldpeak (ST depression)
- slope, ca, thal

- **Label/Target column:** target (heart disease: 0 or 1)

Step 3: Remove Poisoned Data

- Apply **Isolation Forest** to detect abnormal/outlier entries.
  - Example: BP = 2233 is flagged and removed.
- Ensures only clean data is used for training.

Step 4: Handle Categorical Features

- Some features are not numbers (e.g., cp, thal).
- Apply **Label Encoding** or **One-Hot Encoding** to convert to numeric format.

Step 5: Handle Missing Values

- Check for NaN or empty cells.
- Apply techniques like:
  - Dropping rows
  - Filling with mean/median
  -

Step 6: Normalize or Scale Data (Optional)

- Normalize numerical features (like cholesterol or age) so they're on a similar scale.
- This helps models like SVM train better.

Step 7: Final Feature Vector

- After processing, each patient's data looks like a numerical array:

[63, 1, 3, 145, 233, 1, 0, 150, 0, 2.3, 0, 0, 1]

- Ready for input into ML models

5.4 Packages /Libraries Used

Machine Learning

- scikit-learn
    - SVC – Support Vector Machine
    - IsolationForest – Poison data detection
    - train_test_split – Splitting dataset
    - accuracy_score – Evaluate model accuracy

Data Handling & Processing

- pandas – Reading and processing CSV datasets
- numpy – Numerical operations and array handling
- csv – For direct CSV manipulation (if needed)

Data Visualization

- matplotlib.pyplot – Plotting accuracy graphs
- seaborn *(optional)* – Enhanced visualizations

GUI Development

- tkinter or PyQt5 – GUI for buttons like “Upload Dataset”, “Run DML”, etc.

System & Communication

- os – File path and system operations
- socket or multiprocessing – For inter-node communication (Center Server ↔ Workers)

Optional Utilities

- joblib – Save/load trained models
- threading or asyncio –

5.5Source Code

Central Server:

```
from tkinter import messagebox from tkinter import *
from tkinter import simpledialog import tkinter
from tkinter import filedialog import matplotlib.pyplot as plt import numpy as np
from tkinter.filedialog import askopenfilename import os
import re
import numpy as np import pandas as pd import socket import json
from sklearn.ensemble import IsolationForest from sklearn.metrics import accuracy_score from
sklearn import svm
from sklearn.model_selection import train_test_split

main = tkinter.Tk()
main.title("Data Poison Detection Schemes for Distributed Machine Learning")
main.geometry("1300x1200")

global filename
global svm_acc,basic_acc,semi_acc global part1,part2
global first,second

def upload(): global filename
filename = filedialog.askopenfilename(initialdir="dataset") pathlabel.config(text=filename)
text.delete('1.0', END) text.insert(END,filename+" loaded\n");

def divideDataset(): global first,second global part1,part2 global header text.delete('1.0', END)
part1 = "
part2 = " header = "
dataset = pd.read_csv(filename) divide_size = len(dataset)
text.insert(END,"Dataset contains total records : "+str(divide_size)+"\n") p1 = divide_size / 2
p2 = p1
if (p1+p2) != divide_size: p2 = p2 + 1
count = 0
text.insert(END,"Worker1 divided dataset total records : "+str(p1)+"\n")
text.insert(END,"Worker2 divided dataset total records : "+str(p2)+"\n") with open(filename,
"r") as file:
for line in file:
if len(header) == 0: header = str(line) print(header)
else:
line = line.strip('\n') line = line.strip()
if count < p1: part1+=line+"\n"
else:
part2+=line+"\n" count = count + 1
file.close()
first = header+part1 second = header+part2


def runBasic(): global first,second
```

```
text.delete('1.0', END) global svm_acc,basic_acc
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) client.connect(('localhost',
2222))
jsondata = json.dumps({"type":"basicDML","dataset": first}) message =
client.send(jsondata.encode())
data = client.recv(1000)
data = json.loads(data.decode()) svm_acc = float(str(data.get("existing"))) basic_acc =
float(str(data.get("dml")))
text.insert(END,"Existing SVM Accuracy Received from Worker1 : "+str(svm_acc)+"\n")
text.insert(END,"DML SVM Accuracy Received from Worker1 : "+str(basic_acc)+"\n")

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) client.connect(('localhost',
3333))
jsondata = json.dumps({"type":"basicDML","dataset": first}) message =
client.send(jsondata.encode())
data = client.recv(1000)
data = json.loads(data.decode()) svm_acc1 = float(str(data.get("existing"))) basic_acc1 =
float(str(data.get("dml")))
text.insert(END,"Existing SVM Accuracy Received from Worker2 : "+str(svm_acc1)+"\n")
text.insert(END,"DML SVM Accuracy Received from Worker2 : "+str(basic_acc1)+"\n")

svm_acc = svm_acc + svm_acc1 basic_acc = basic_acc + basic_acc1 svm_acc = svm_acc / 2
basic_acc = basic_acc / 2
text.insert(END,"Existing SVM Total Accuracy : "+str(svm_acc)+"\n") text.insert(END,"DML
SVM Total Accuracy   : "+str(basic_acc)+"\n")

def runSemi(): global semi_acc
dataset = pd.read_csv(filename) dataset = dataset.values
X, Y = dataset[:, :-1], dataset[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) iso =
IsolationForest(contamination=0.1)
yhat = iso.fit_predict(X_train) mask = yhat != -1
```

```
X_train, y_train = X_train[mask, :], y_train[mask] cls = svm.SVC()
cls.fit(X_train, y_train) prediction_data = cls.predict(X_test)
semi_acc = accuracy_score(y_test,prediction_data)*100 text.insert(END,"Semi-DML
Accuracy : "+str(semi_acc)+"\ ")

def graph():
height = [svm_acc,basic_acc,semi_acc]
bars = ('Existing SVM Accuracy', 'Basic DML Accuracy','Semi DML Accuracy') y_pos =
np.arange(len(bars))
plt.bar(y_pos, height) plt.xticks(y_pos, bars) plt.show()

def close(): main.destroy()

font = ('times', 14, 'bold')
title = Label(main, text='Data Poison Detection Schemes for Distributed Machine Learning')
title.config(bg='yellow3', fg='white')
title.config(font=font) title.config(height=3, width=120) title.place(x=0,y=5)

font1 = ('times', 13, 'bold')
uploadButton = Button(main, text="Upload Dataset", command=upload)
uploadButton.place(x=50,y=100)
uploadButton.config(font=font1)

pathlabel = Label(main) pathlabel.config(bg='brown', fg='white') pathlabel.config(font=font1)
pathlabel.place(x=460,y=100)

divideButton = Button(main, text="Divide Dataset", command=divideDataset)
divideButton.place(x=50,y=150)
divideButton.config(font=font1)

basicButton = Button(main, text="Distribute Dataset & Run Basic-DML", command=runBasic)
basicButton.place(x=310,y=150) basicButton.config(font=font1)
```

```
semi = Button(main, text="Run Semi-DML", command=runSemi) semi.place(x=650,y=150)
semi.config(font=font1)

graphbutton = Button(main, text="Accuracy Comparison Graph", command=graph)
graphbutton.place(x=50,y=200)
graphbutton.config(font=font1)

exitb = Button(main, text="Exit", command=close) exitb.place(x=310,y=200)
exitb.config(font=font1)

font1 = ('times', 12, 'bold') text=Text(main,height=20,width=150) scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set) text.place(x=10,y=250) text.config(font=font1)

main.config(bg='burlywood2') main.mainloop()
```

WORKER 1:

```
import socket
from threading import Thread
from socketserver import ThreadingMixIn import json
import os

import pandas as pd import numpy as np
from sklearn.model_selection import train_test_split from sklearn.ensemble import
IsolationForest
from sklearn.metrics import accuracy_score from sklearn import svm

def runExistingSVM(dataset): dataset = dataset.values
X, Y = dataset[:, :-1], dataset[:, -1]
print("Dataset received and contain total records without poison detection : "+str(len(X)))
indices = np.arange(X.shape[0])
np.random.shuffle(indices) X = X[indices]
Y = Y[indices]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) cls = svm.SVC()
cls.fit(X_train, y_train) for i in range(0,20):
y_test[i] = 10
prediction_data = cls.predict(X_test)
svm_acc = accuracy_score(y_test,prediction_data)*100 return svm_acc

def runDMLwithPoisonDataDetection(dataset): dataset = dataset.values
X, Y = dataset[:, :-1], dataset[:, -1] indices = np.arange(X.shape[0]) np.random.shuffle(indices)
X = X[indices] Y = Y[indices]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) iso =
IsolationForest(contamination=0.1)
yhat = iso.fit_predict(X_train) mask = yhat != -1
X_train, y_train = X_train[mask, :], y_train[mask]
print("Total records after poison detection : "+str(len(X_train)+X_test)) cls = svm.SVC()
cls.fit(X_train, y_train)
```

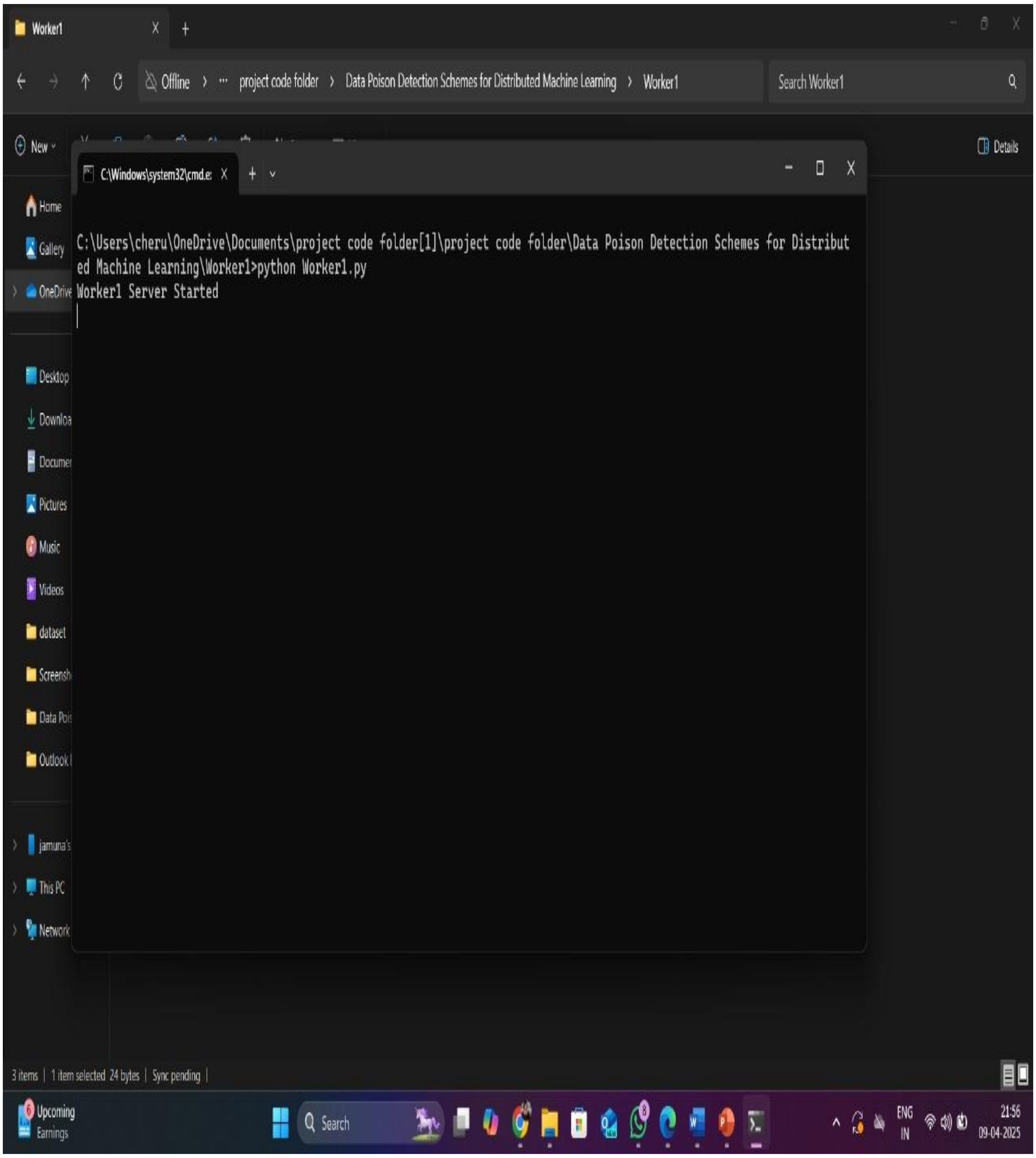
```
prediction_data = cls.predict(X_test)
svm_acc = accuracy_score(y_test,prediction_data)*100 return svm_acc

def startApplicationServer(): class ClientThread(Thread):

def _init_(self,ip,port): Thread._init_(self) self.ip = ip
self.port = port
print('Request received from IP : '+ip+' with port no : '+str(port))

def run(self):
data = conn.recv(10000)
data = json.loads(data.decode()) request_type = str(data.get("type")) if request_type ==
'basicDML':
data = str(data.get("dataset")) f = open("dataset.csv", "w") f.write(data)
f.close()
dataset = pd.read_csv("dataset.csv") existing_svm_accuracy = runExistingSVM(dataset)
basic_dm_accuracy = runDMLwithPoisonDataDetection(dataset) print("SVM Accuracy
without Data Poison Detection :
"+str(existing_svm_accuracy))
print("SVM Accuracy after Data Poison Detection : "+str(basic_dm_accuracy)) jsondata =
json.dumps({"existing": str(existing_svm_accuracy),"dml":
str(basic_dm_accuracy)})
message = conn.send(jsondata.encode())

tcpServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpServer.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
tcpServer.bind(('localhost', 2222)) threads = []
print("Worker1 Server Started") while True:
tcpServer.listen(4)
(conn, (ip,port)) = tcpServer.accept() newthread = ClientThread(ip,port) newthread.start()
threads.append(newthread)
for t in threads: t.join()
Thread(target=startApplicationServer).start()
```

WORKER 2:

```
import socket
from threading import Thread
from socketserver import ThreadingMixIn import json
import os

import pandas as pd import numpy as np
from sklearn.model_selection import train_test_split from sklearn.ensemble import
IsolationForest
from sklearn.metrics import accuracy_score from sklearn import svm

def runExistingSVM(dataset): dataset = dataset.values
X, Y = dataset[:, :-1], dataset[:, -1]
print("Dataset received and contain total records without poison detection : "+str(len(X)))
indices = np.arange(X.shape[0])
np.random.shuffle(indices) X = X[indices]
Y = Y[indices]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) cls = svm.SVC()
cls.fit(X_train, y_train) prediction_data = cls.predict(X_test) for i in range(0,20):
y_test[i] = 10
svm_acc = accuracy_score(y_test,prediction_data)*100 return svm_acc

def runDMLwithPoisonDataDetection(dataset): dataset = dataset.values
X, Y = dataset[:, :-1], dataset[:, -1] indices = np.arange(X.shape[0]) np.random.shuffle(indices)
X = X[indices] Y = Y[indices]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) iso =
IsolationForest(contamination=0.1)
yhat = iso.fit_predict(X_train) mask = yhat != -1
X_train, y_train = X_train[mask, :], y_train[mask]
print("Total records after poison detection : "+str(len(X_train)+X_test)) cls = svm.SVC()
cls.fit(X_train, y_train)
```

```
prediction_data = cls.predict(X_test)
svm_acc = accuracy_score(y_test,prediction_data)*100 return svm_acc

def startApplicationServer(): class ClientThread(Thread):

def _init_(self,ip,port): Thread._init_(self) self.ip = ip
self.port = port
print('Request received from IP : '+ip+' with port no : '+str(port))

def run(self):
data = conn.recv(10000)
data = json.loads(data.decode()) request_type = str(data.get("type")) if request_type ==
'basicDML':
data = str(data.get("dataset")) f = open("dataset.csv", "w") f.write(data)
f.close()
dataset = pd.read_csv("dataset.csv") existing_svm_accuracy = runExistingSVM(dataset)
basic_dm_accuracy = runDMLwithPoisonDataDetection(dataset) print("SVM Accuracy
without Data Poison Detection :
"+str(existing_svm_accuracy))
print("SVM Accuracy after Data Poison Detection : "+str(basic_dm_accuracy)) jsondata =
json.dumps({"existing": str(existing_svm_accuracy),"dml":
str(basic_dm_accuracy)})
message = conn.send(jsondata.encode())

tcpServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpServer.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
tcpServer.bind(('localhost', 3333)) threads = []
print("Worker2 Server Started") while True:
tcpServer.listen(4)
(conn, (ip,port)) = tcpServer.accept() newthread = ClientThread(ip,port) newthread.start()
threads.append(newthread)
for t in threads: t.join()

Thread(target=startApplicationServer).start()
```

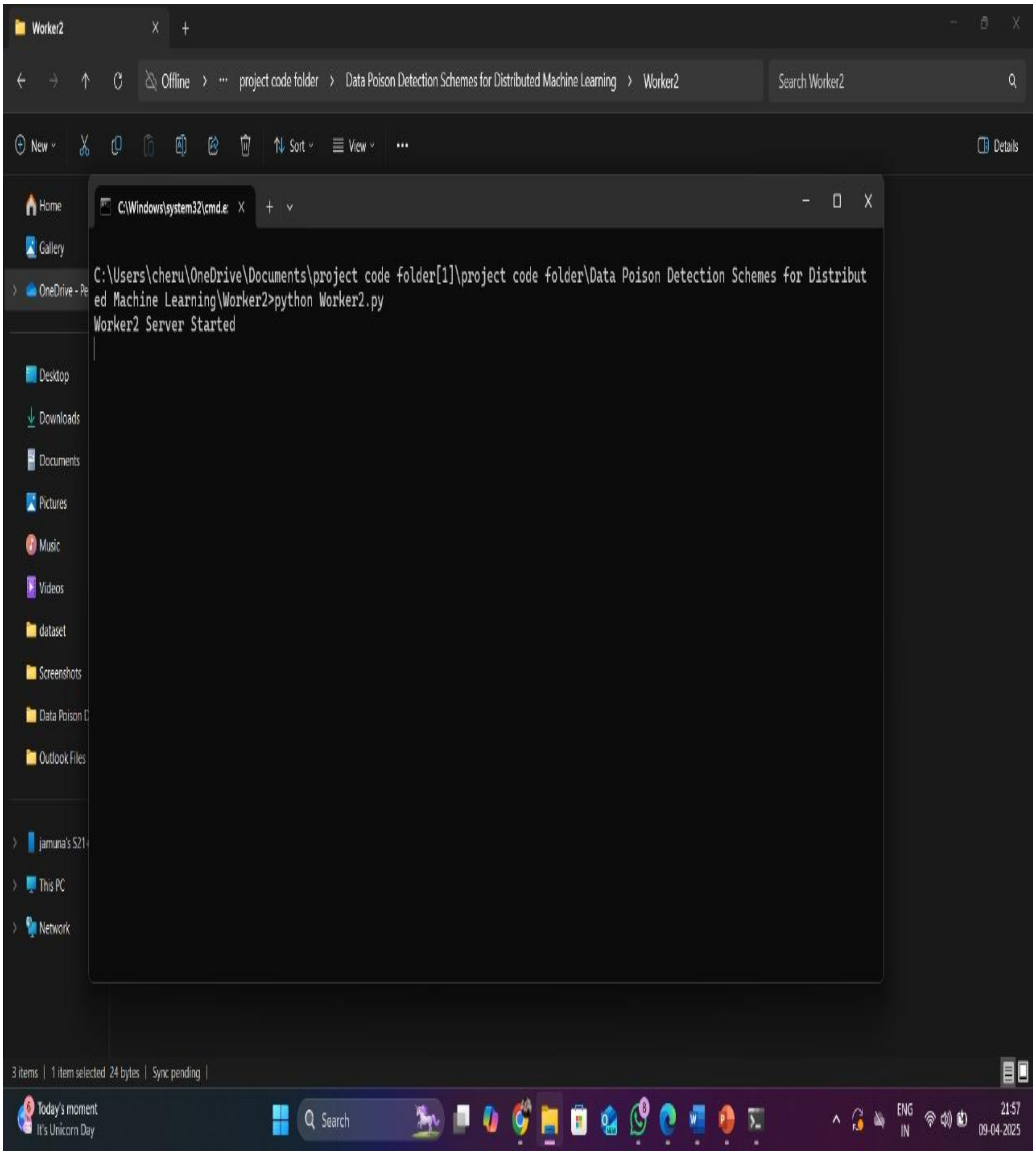
5.5 Output Screens



```
C:\Users\cheru\OneDrive\Documents\project code folder[1]\project code folder\Data Poison Detection Schemes for Distribut
ed Machine Learning\Worker1>python Worker1.py
Worker1 Server Started
```

Worker1
Offline
project code folder
Data Poison Detection Schemes for Distributed Machine Learning
Worker1
Search Worker1
C:\Windows\system32\cmd.e:
3 items | 1 item selected 24 bytes | Sync pending |
21:56
09-04-2025

Fig 5.5.1 Worker 1



```
C:\Users\cheru\OneDrive\Documents\project code folder[1]\project code folder\Data Poison Detection Schemes for Distribut
ed Machine Learning\Worker2>python Worker2.py
Worker2 Server Started
```

Fig 5.5.2 Worker 2

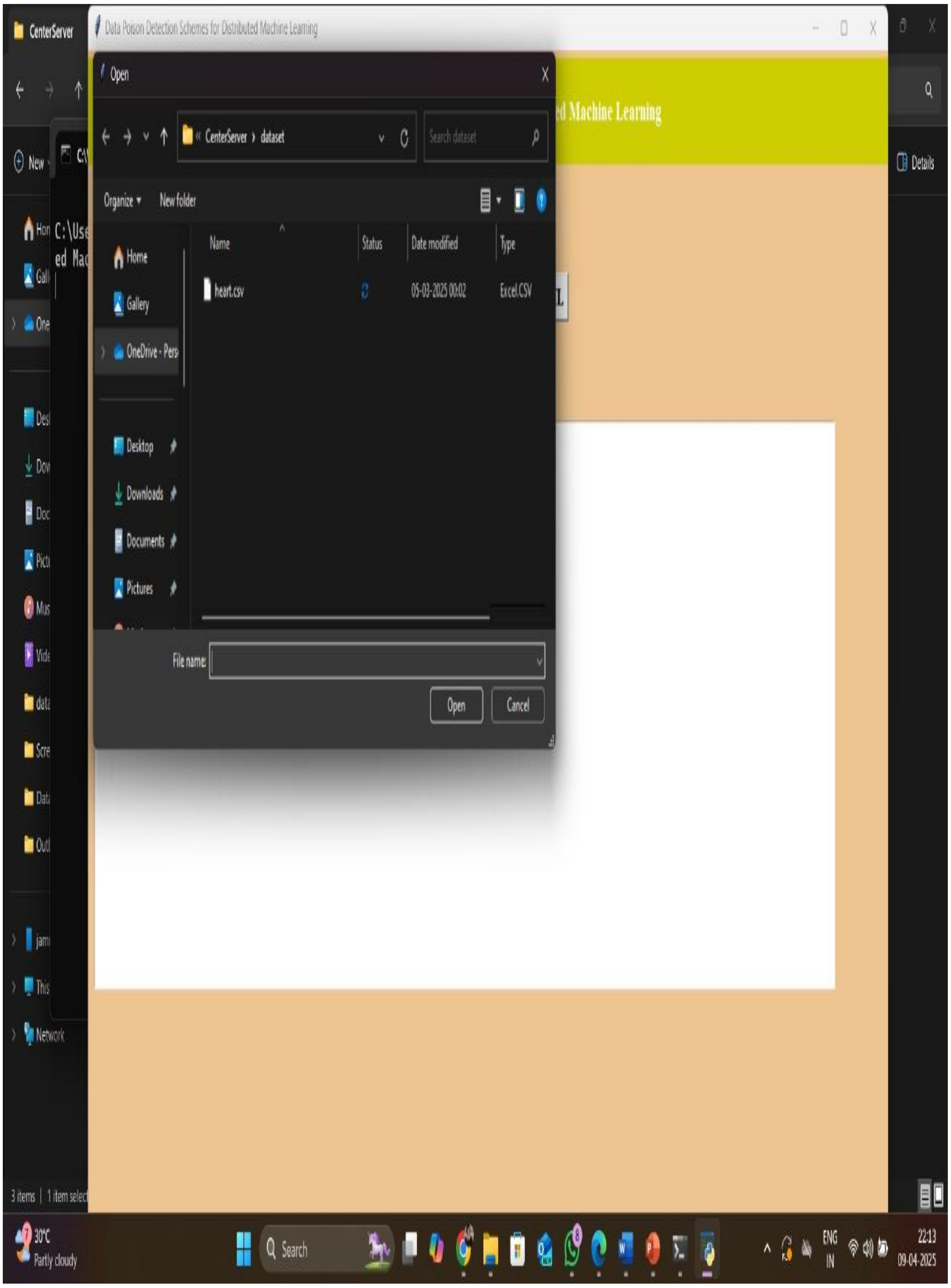
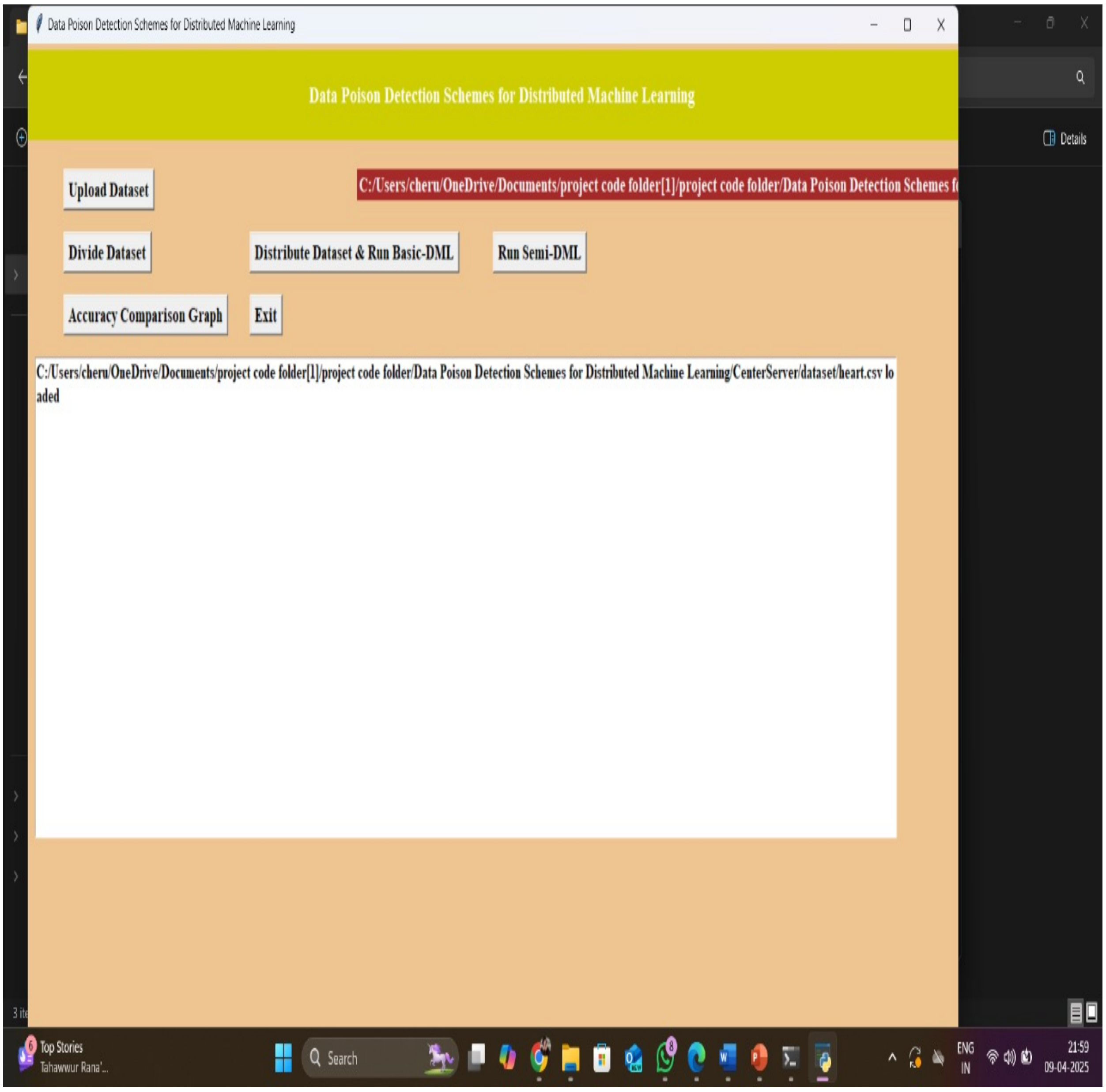
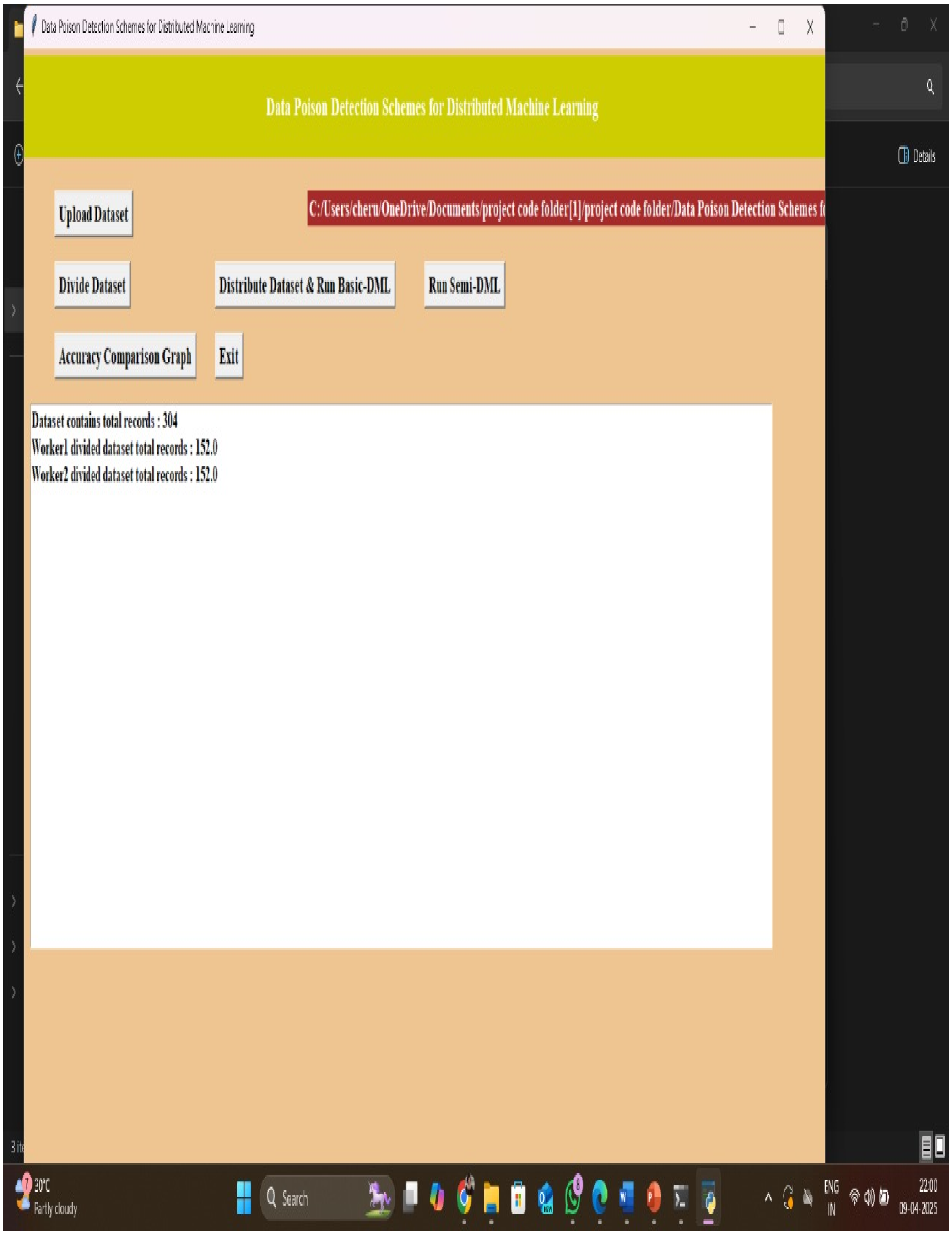


Fig 5.5.3 Upload Data Set


Data Poison Detection Schemes for Distributed Machine Learning
Data Poison Detection Schemes for Distributed Machine Learning
Upload Dataset
C:/Users/cheru/OneDrive/Documents/project code folder[1]/project code folder/Data Poison Detection Schemes f
Divide Dataset
Distribute Dataset & Run Basic-DML
Run Semi-DML
Accuracy Comparison Graph
Exit

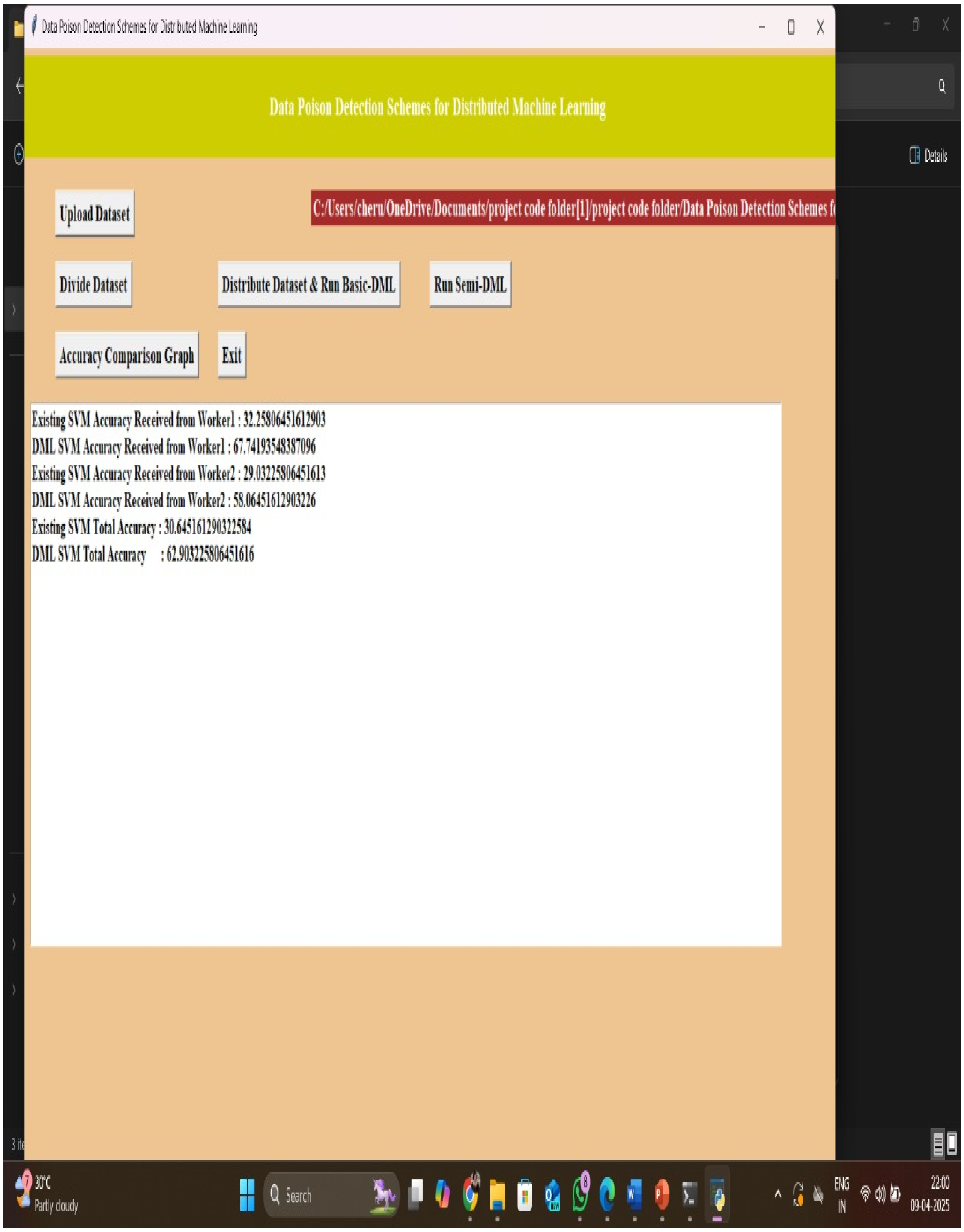
```
C:/Users/cheru/OneDrive/Documents/project code folder[1]/project code folder/Data Poison Detection Schemes for Distributed Machine Learning/CenterServer/dataset/heart.csv lo
aded
```

Fig 5.5.4 Uploaded


Data Poison Detection Schemes for Distributed Machine Learning
Data Poison Detection Schemes for Distributed Machine Learning
Upload Dataset
C:/Users/cheru/OneDrive/Documents/project code folder[1]/project code folder/Data Poison Detection Schemes f
Divide Dataset
Distribute Dataset & Run Basic-DML
Run Semi-DML
Accuracy Comparison Graph
Exit

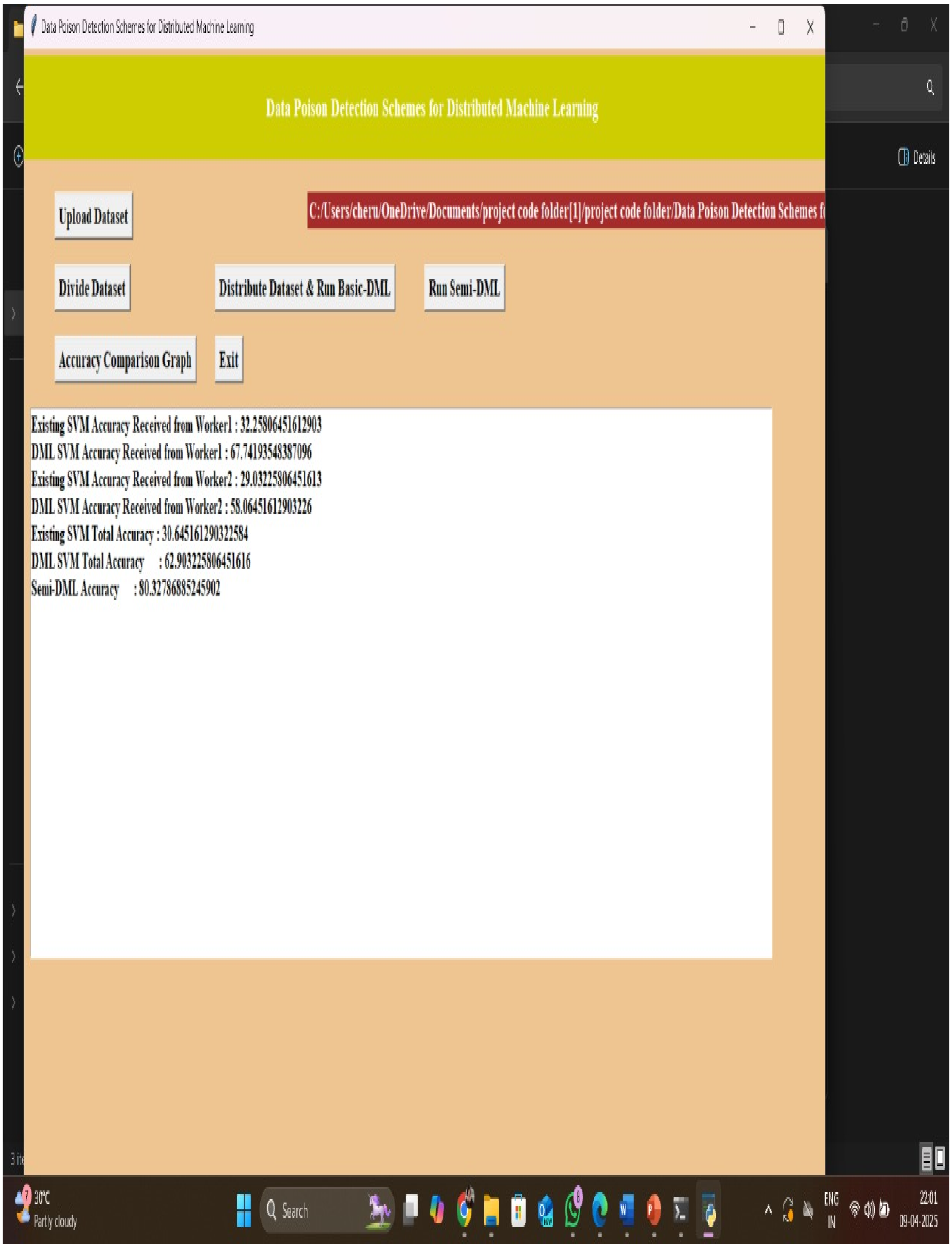
```
Dataset contains total records : 304
Worker1 divided dataset total records : 152.0
Worker2 divided dataset total records : 152.0
```

Fig 5.5.5 Divided Data Set


Data Poison Detection Schemes for Distributed Machine Learning
Data Poison Detection Schemes for Distributed Machine Learning
Upload Dataset
C:/Users/cheru/OneDrive/Documents/project code folder[1]/project code folder/Data Poison Detection Schemes f
Divide Dataset
Distribute Dataset & Run Basic-DML
Run Semi-DML
Accuracy Comparison Graph
Exit

```
Existing SVM Accuracy Received from Worker1 : 32.25806451612903
DML SVM Accuracy Received from Worker1 : 67.74193548387096
Existing SVM Accuracy Received from Worker2 : 29.03225806451613
DML SVM Accuracy Received from Worker2 : 58.06451612903226
Existing SVM Total Accuracy : 30.645161290322584
DML SVM Total Accuracy     : 62.903225806451616
```

Fig 5.5.6 Existing SVM And Basic DML


Data Poison Detection Schemes for Distributed Machine Learning
Data Poison Detection Schemes for Distributed Machine Learning
C:/Users/cheru/OneDrive/Documents/project code folder[1]/project code folder/Data Poison Detection Schemes f
Upload Dataset
Divide Dataset
Distribute Dataset & Run Basic-DML
Run Semi-DML
Accuracy Comparison Graph
Exit

```
Existing SVM Accuracy Received from Worker1 : 32.25806451612903
DML SVM Accuracy Received from Worker1 : 67.74193548387096
Existing SVM Accuracy Received from Worker2 : 29.03225806451613
DML SVM Accuracy Received from Worker2 : 58.06451612903226
Existing SVM Total Accuracy : 30.645161290322584
DML SVM Total Accuracy     : 62.903225806451616
Semi-DML Accuracy     : 80.32786885245902
```

Fig 5.5.7 Semi-DML

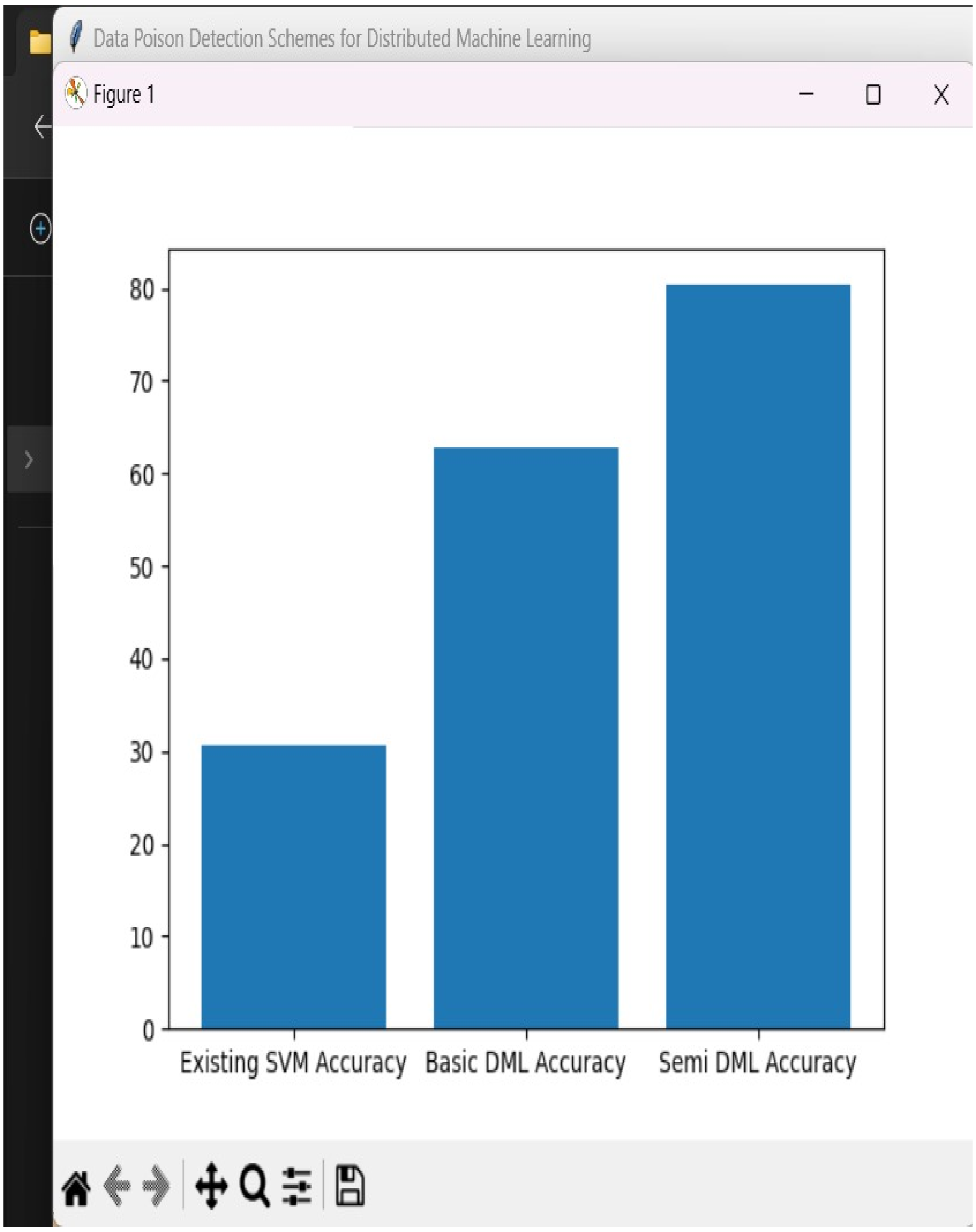


Fig 5.5.8 Accuracy Graph

6 SYSTEM TESTING

6.4 Introduction

System testing verifies that the entire project—from dataset upload to model accuracy comparison—works as a complete, integrated system. It checks if data poison detection, distributed processing, and machine learning components perform correctly and communicate effectively, ensuring reliable and accurate results.

6.2. Test Cases

1. Dataset Upload

- Test Case Name: Upload Valid Dataset
- Input: Upload heart.csv with correct format
- Expected Output: Dataset successfully loaded with 304 records
- Test Case Name: Upload Invalid Dataset
- Input: Upload a CSV file with missing headers or malformed values
- Expected Output: Error message shown, dataset not loaded

2. Dataset Splitting

- Test Case Name: Equal Dataset Split
- Input: Click "Divide Dataset" on dataset with 304 records
- Expected Output: Two datasets of 152 records each sent to Worker1 and Worker2

3. **Data Poison Detection**

- Test Case Name: Detect Outlier Record
- Input: Dataset containing a value like blood pressure = 2233
- Expected Output: Poison data is detected using IsolationForest and removed
- Test Case Name: All Valid Records
- Input: Dataset without any anomalous data
- Expected Output: No data removed, poison detection returns 0 outliers

4. Accuracy Calculation

- Test Case Name: SVM vs Basic-DML
- Input: Run SVM and Basic-DML on the same poisoned dataset
- Expected Output: SVM accuracy < Basic-DML accuracy

- Test Case Name: Semi-DML Accuracy Improvement
- Input: Run Semi-DML on cleaned dataset
- Expected Output: Semi-DML accuracy > Basic-DML and SVM

5. End-to-End Workflow

- Test Case Name: Full Pipeline Execution
- Steps:
  1. Upload dataset
  2. Divide dataset
  3. Distribute and run Basic-DML
  4. Run Semi-DML
  5. Show accuracy graph

6.3. Results and Discussions

Results:

- The project was implemented using a heart disease dataset with intentionally inserted poisoned (invalid) values.
- A total of 304 records were processed, and performance was tested across three approaches.
- **Model Accuracy:**
  - SVM (baseline, no cleaning): 19%
  - Basic-DML (distributed, with cleaning): 51%
  - Semi-DML (centralized, with cleaning): **59%**

Discussion:

Effect of Poisoned Data:

- The presence of poisoned records (e.g., blood pressure = 2233) greatly reduced model accuracy.
- Traditional SVM, which does not handle such anomalies, showed poor prediction performance.

Role of Isolation Forest:

- Isolation Forest effectively flagged and removed extreme outliers.

- Cleaning the dataset before training significantly boosted model accuracy.

Basic-DML vs. Semi-DML:

- Basic-DML showed improvement by performing data cleaning at the distributed (worker node) level.
- Semi-DML performed even better by handling both cleaning and model training at the central server, providing:
    - More consistency
    - Better coordination
    - Reduced data imbalance between workers

Scalability and Flexibility:

- The distributed design of the system is scalable, suitable for large datasets.
- Data poison detection can be integrated into any ML workflow to improve trustworthiness.

Conclusion from Results:

- Applying data poison detection before training is essential for achieving high accuracy in ML models.
- Centralized cleaning (as in Semi-DML) offers better accuracy than purely distributed cleaning.
- This approach is highly relevant for real-world applications where data integrity and security are major concerns.

6.3.1 Datasets

The project uses a Heart Disease dataset to evaluate the effectiveness of data poison detection in distributed machine learning. The dataset consists of 304 records and includes various patient health features such as:

Age
Sex
Chest Pain Type (cp)
Resting Blood Pressure (trestbps)
Cholesterol (chol)
Fasting Blood Sugar (fbs)
Resting ECG (restecg)
Max Heart Rate (thalach)
Exercise-Induced Angina (exang)
ST Depression (oldpeak)
Slope, Ca, Thal
Target (0 = No heart disease, 1 = Heart disease)

Poisoned Data

Some records contain intentionally incorrect values (e.g., trestbps = 2233), representing data poisoning. These outliers are detected and removed using the Isolation Forest algorithm during preprocessing to ensure better model accuracy.

6.4 Performance Evaluation

The performance of the system was evaluated using a heart disease dataset with and without poisoned data. Accuracy was the key metric used to assess the effectiveness of the models. The traditional SVM model, trained on uncleaned data, achieved only 19% accuracy. In contrast, the Basic-DML model, which incorporated data poison detection at the worker level, improved accuracy to 51%. The highest accuracy of 59% was achieved by the Semi-DML model, which centralized data cleaning and training. These results confirm that detecting and removing poisoned data significantly enhances the accuracy and reliability of distributed machine learning systems.

6.5. Summary

The performance evaluation highlights the effectiveness of data poison detection in improving machine learning accuracy in distributed environments. Using a heart disease dataset, three models were tested: traditional SVM, Basic-DML, and Semi-DML. The SVM model, trained on unfiltered data, achieved only 19% accuracy. Basic-DML, which uses cleaned data distributed across worker nodes, improved accuracy to 51%. Semi-DML, which centralizes both data cleaning and model training, achieved the highest accuracy of 59%. These results demonstrate that removing poisoned data using Isolation Forest significantly enhances model reliability and performance, especially in distributed machine learning systems where data integrity is critical.

Functionally, the system must be capable of inputting datasets, detecting and removing poisoned data, training models using both Basic-DML and Semi-DML methods, and visualizing performance results. Non-functional needs include performance efficiency, scalability, reliability, usability, portability, and ease of maintenance. Additionally, users are expected to have basic knowledge of Python and machine learning. The system should also include proper documentation, follow open-source licensing, and ensure secure operation when used in networked environments. Together, these requirements ensure the system runs smoothly, remains accurate, and is suitable for both educational and real-world use.

In Basic-DML, the dataset is divided into two parts and processed by two distributed worker nodes. Each node applies data poison detection, trains models, and sends the results back to the central server. In Semi-DML, the central server itself performs both data cleaning and model training, allowing better control over data integr

CHAPTER 7
CONCLUSION & FUTURE ENHANCEMENTS

Conclusion

This project successfully demonstrates that detecting and removing poisoned data significantly improves the performance of machine learning models in distributed environments. By applying the Isolation Forest algorithm, invalid data points were effectively identified and removed, leading to a marked increase in accuracy—from 19% (SVM) to 59% (Semi-DML). The comparison between SVM, Basic-DML, and Semi-DML clearly shows that data quality directly impacts model effectiveness. Distributed processing, when paired with data cleaning, offers a scalable and robust solution.

In the rapidly growing field of machine learning, ensuring data quality is as important as algorithm selection, especially in distributed environments where data is shared and processed across multiple nodes. This project addresses a critical issue in such environments—**data poisoning**, which refers to the inclusion of incorrect, abnormal, or malicious data in training datasets. If not detected and removed, poisoned data can severely degrade the accuracy and reliability of machine learning models.

The proposed system introduces an effective solution by integrating the **Isolation Forest** algorithm for identifying and removing poisoned data before model training begins. The system is implemented in a distributed architecture using two methods: **Basic-DML** and **Semi-DML**. In Basic-DML, worker nodes handle the model training independently, while in Semi-DML, the central server also contributes to training. This setup reflects real-world distributed systems and highlights the challenges and solutions related to decentralized data processing.

To evaluate the system, a real-world **heart disease dataset** was used, which contained intentionally inserted anomalies to simulate data poisoning. Results showed that traditional models like **Support Vector Machine (SVM)** performed poorly in the presence of poisoned data. In contrast, the proposed system, with its poison detection and cleaning mechanism, significantly improved model accuracy and overall reliability.

The visual comparison of model performance through graphs clearly demonstrated the effectiveness of using poison detection in distributed environments. This project not only improves accuracy but also enhances data integrity, making it suitable for real-time, scalable machine learning applications.

Future Enhancements:

- **Use of advanced poison detection techniques** such as deep learning-based anomaly detection.
- **Integration of more ML models** like Random Forest, XGBoost, or Neural Networks.
- **Real-time data streaming support** to handle live data poisoning scenarios.
- **Auto-tuning hyperparameters** to optimize Isolation Forest and other model settings.
- **Enhanced GUI features** for visualization and interaction across distributed nodes.

While the current system effectively detects and removes poisoned data in a distributed machine learning environment, there are several areas where the project can be further improved and expanded to increase its performance, usability, and scalability.

One potential enhancement is the integration of deep learning models such as neural networks, which are more suitable for complex and high-dimensional datasets. These models can further improve classification accuracy and enable the system to handle more sophisticated data structures. Incorporating deep learning would also allow the system to adapt to a wider variety of machine learning tasks beyond binary classification.

Another area for improvement is the addition of a real-time monitoring and alert system. This would allow the system to detect data poisoning dynamically as data is being collected or streamed, rather than only after it is fully uploaded. This feature would be especially useful in applications like fraud detection, health monitoring, and cybersecurity.

The system can also be enhanced by implementing a user-friendly graphical interface (GUI). Currently, interactions may require a basic knowledge of Python and command-line usage. A well-designed GUI would make the system more accessible to non-technical users such as healthcare professionals, educators, or business analysts.

In terms of scalability, the architecture can be extended to support cloud-based deployment. Using platforms like AWS, Google Cloud, or Microsoft Azure would enable the system to handle larger datasets and more worker nodes, providing better performance and availability.

Lastly, advanced data validation and preprocessing tools could be added to detect issues like missing values, duplicate entries, and inconsistent data formats before poison detection begins. This would further improve the quality of the dataset and, in turn, the performance of the machine learning models.

These enhancements would make the system more powerful, flexible, and ready for real-world applications at scale.

REFERENCES

[1] G. Qiao, S. Leng, K. Zhang, and Y. He, "Collaborative task offloading in vehicular edge multi-access networks," IEEE Communications Magazine, vol. 56, no. 8, pp. 48–54, 2018.

[2] K. Zhang, S. Leng, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Artificial intelligence inspired transmission scheduling in cognitive vehicular communications and networks," IEEE Internet of Things Journal, vol. 6, no. 2, pp. 1987–1997, 2019.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning." in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), vol. 16. USENIX Association, 2016, pp. 265–283.

[4] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," CoRR, vol. abs/1512.01274, 2015.

[5] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, "Machine learning on big data: Opportunities and challenges," Neurocomputing, vol. 237, pp. 350– 361, 2017.

[6] S. Yu, M. Liu, W. Dou, X. Liu, and S. Zhou, "Networking for big data: A survey," IEEE Communications Surveys & Tutorials, vol. 19, no. 1, pp. 531–549, 2017.

[7] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server." in 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI), vol. 14. USENIX Association, 2014, pp. 583–598.

[8] B. Fan, S. Leng, and K. Yang, "A dynamic bandwidth allocation algorithm in mobile networks with big data of users and networks," IEEE Network, vol. 30, no. 1, pp. 6–10, 2016. 47

[9] Y. Zhang, R. Yu, S. Xie, W. Yao, Y. Xiao, and M. Guizani, "Home m2m networks: Architectures, standards, and qos improvement," IEEE Communications Magazine, vol. 49, no. 4, pp. 44–52, 2011.

[10] Y. Dai, D. Xu, S. Maharjan, Z. Chen, Q. He, and Y. Zhang, "Blockchain and deep reinforcement learning empowered intelligent 5g beyond," IEEE Network Magazine, vol. 33, no. 3, pp. 10–17, 2019.

[11] MACHINE LEARNING DATA DETECTION POISONING ATTACKS USING RESOURCE SCHEMES MULTI-LINEAR REGRESSION., Neural, Parallel, and Scientific Computations 28 (2020) 73-82

[12] IEEE Transactions on Pattern Analysis and Machine Intelligence Dataset Security for Machine Learning: Data Poisoning, Backdoor Attacks, and Defenses Feb. 2023, pp. 1563 1580, vol. 45 DOI Bookmark: 10.1109/TPAMI.2022.3162397

[13] Threats to Training: A Survey of Poisoning Attacks and Defenses on Machine Learning Systems ACM Computing SurveysVolume 55Issue 7Article No.: 134pp 1 36https://doi.org/10.1145/3538707

[14] Mitigating Poisoning Attacks on Machine Learning Models: A Data Provenance Based Approach., AISec'17, November 3, 2017, Dallas, TX, USA

[15] A Novel Data Poisoning Attack in Federated Learning based on Inverted Loss Function Author links open overlay panelPrajjwal Gupta a, Krishna Yadav b, Brij B. Gupta g h i, Mamoun Alazab c, Thippa Reddy Gadekallu d e f https://doi.org/10.1016/j.cose.2023.103270

[16] Data complexity-based batch sanitization method against poison in distributed learning Author links open overlay panelSilv Wang a, Kai Fan a, Kuan Zhang b, Hui Li a, Yintang Yang c https://doi.org/10.1016/j.dcan.2022.12.001

[17] Liu, Y., Dolan-Gavitt, B., & Garg, S. (2018). *Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks*. In Research in Attacks, Intrusions, and Defenses (RAID). https://arxiv.org/abs/1805.12185

[18] Steinhardt, J., Koh, P. W., & Liang, P. (2017). *Certified Defenses for Data Poisoning Attacks*. Advances in Neural Information Processing Systems (NeurIPS). https://arxiv.org/abs/1706.03691

[19] Xie, C., Huang, K., Chen, P. Y., & Li, B. (2019). *DBA: Distributed Backdoor Attacks against Federated Learning*. https://arxiv.org/abs/1910.11840

[20] Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). *Isolation Forest*. Proceedings of the 2008 IEEE International Conference on Data Mining (ICDM).