BFL-IoT: A Blockchain-Enabled Federated Learning Model for Secure and Efficient Data Transmission in IoT Networks

Report submitted to the SASTRA Deemed to be University

In partial fulfillment of the requirements

for the award of the degree of

Bachelor of Technology

Submitted by

PRAKADEESH B S

(Reg. No.: 225003104, B.Tech-CSE)

May 2025

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SRINIVASA RAMANUJAN CENTRE,
KUMBAKONAM, TAMILNADU,
INDIA – 612001

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SRINIVASA RAMANUJAN CENTRE,
KUMBAKONAM, TAMILNADU,
INDIA - 612001

Bonafide Certificate

This is to certify that the project report titled **"BFL-IoT: A Blockchain-Enabled Federated Learning Model for Secure and Efficient Data Transmission in IoT Networks"** submitted in partial fulfillment of the requirements for the award of the degree of **B. Tech. Computer Science and Engineering** to the SASTRA Deemed to be University is a bona-fide record of the work done by **Mr. PRAKADEESH B S (Reg. No. 225003104)** during the final semester of the academic year 2024- 25 in Srinivasa Ramanujan Centre, under my supervision. This report has not formed the basis for the award of any degree, diploma, associateship, fellowship or other similar title to any candidate of any University.

Signature of Project Supervisor : 15/5/25

Name with Affiliation : **Dr. N. Rajesh Kumar**
AP-II / CSE / SRC / SASTRA

Date : 05.05.2025

Project *Viva voce* held on 15/05/2025

Examiner 1 15/5

Examiner 2

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SRINIVASA RAMANUJAN CENTRE,
KUMBAKONAM, TAMILNADU,
INDIA - 612001

Declaration

I declare that the project report titled **"BFL-IoT: A Blockchain-Enabled Federated Learning Model for Secure and Efficient Data Transmission in IoT Networks"** submitted by me is an original work done by me under the guidance of **Dr. N. RAJESH KUMAR / AP-II/CSE/SASTRA, Srinivasa Ramanujan Centre, SASTRA Deemed to be University** during the final semester of the academic year 2024-25, in the Srinivasa Ramanujan Centre. The work is original and wherever I have used materials from other sources, I have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associate-ship, fellowship or other similar title to any candidate of any University.

Signature of the candidate :

Name of the candidate : Prakadeesh B S

Date : 05.05.2025

Acknowledgements

I pay my sincere obeisance to God Almighty for his grace and infinite mercy and for showing me his choicest blessings.

I would like to express my gratitude to **Prof. R. Sethuraman, Chancellor, Dr. S. Vaidhyasubramaniam, Vice Chancellor, Dr. R. Chandramouli, Registrar, and Dr. S. Swaminathan, Dean- Planning and Development,** for providing me with the opportunity to be student of this esteemed institution.

I express my deepest gratitude to **Dr. V. Ramaswamy, Dean, and Dr. A. Alli Rani, Associate Dean, Srinivasa Ramanujan Centre**, for their constant support and suggestions, which were provided without reservation.

I express my gratitude to **Dr. V. Kalaichelvi, Associate Professor, CSE, Srinivasa Ramanujan Centre,** for her constant support and valuable suggestions for the completion of the project.

I exhibit my pleasure in expressing my thanks to **Dr. N. RAJESH KUMAR, Assistant Professor, CSE,** my guide for his ever-encouraging spirit and meticulous guidance for the completion of the project.

I would like to place on record the benevolent approach and painstaking efforts of guidance and correction of **Dr. S. Priyanga, Assistant Professor, CSE,** the project coordinator and all department staff to whom I owe my heartfelt thanks forever.

Without the support of my parents and friends, this project would never have become a reality.

I dedicate this work to my well-wishers, with love and affection.

TABLE OF CONTENTS

List of Figures

ABBREVIATIONS

1D-CNN	1-Dimensional Convolutional Neural Network
DNN	Deep Neural Network
LR	Logistic Regression
IPFS	InterPlanetary File System
IOT	Internet Of Things
POS	Proof Of Stack
ETH	Ethereum

ABSTRACT

The growth of Internet of Things (IoT) networks across industries, it will become increasingly important to ensure that the information transmitted is secure due to the rise in cyber threats and the sensitivity of the information. This project will present a Blockchain-enabled Federated Learning model (BFL-IoT) for secure and efficient data transmission in generic IoT networks. By leveraging the decentralized and tamper-resistant architecture of blockchain along with the privacy-preserving training of federated learning, the proposed model will guarantee secure communication between IoT devices while maintaining data privacy. The BFL-IoT model will decentralize the storage of information, with only encrypted model parameters being shared for collaborative training, thus reducing the risks of unauthorized access. The model will also be capable of detecting and countering different types of cyberattacks in IoT networks with high accuracy, distinguishing between normal and adversarial activities. The model will be tested through experimental evaluations and is expected to outperform traditional approaches in terms of security, scalability, and performance for securing and authenticating IoT data transactions across various applications.

KEY WORDS: - IoT, Blockchain, Federated learning, Security.

SPECIFIC CONTRIBUTION:

- Dataset is pre-processed and trained using CNN, DNN and LR models using Federated Learning. Utilized blockchain technology to store model updates and ran the model on localhost with authentication and encryption.

SPECIFIC LEARNING:

- Gained hands-on experience in leveraging Blockchain and Federated Learning for secure model updates, and trained CNN, DNN, and LR models for attack detection.

CHAPTER 1
INTRODUCTION

With the rapid expansion of smart devices and interconnected systems, the Internet of Things (IoT) has become a crucial part of modern infrastructure, particularly in healthcare and smart environments. However, this connectivity also increases the risk of cyberattacks, making IoT networks highly vulnerable to threats such as intrusions, data theft, and system manipulation. Traditional centralized detection methods pose significant limitations, especially regarding data privacy, real-time detection, and trust among multiple stakeholders. To overcome these challenges, this project introduces a hybrid approach combining Deep Learning, Federated Learning, and Blockchain technologies to securely and efficiently detect and respond to attacks in IoT-based networks.

The goal is to classify traffic as either normal or attack using three different models: Deep Neural Networks (DNN), Convolutional Neural Networks (CNN-1D), and Logistic Regression (LR). These models are trained locally on different clients to preserve data privacy. Instead of sharing raw data, each client shares only its trained model, which is stored securely on a Blockchain using smart contracts. This ensures immutability, trust, and transparency in the learning process.

Furthermore, the system includes a FastAPI-based UI for predicting threats and transmitting only safe (normal) data. It also incorporates encryption and secure downloading, enhancing data confidentiality and user control. This integrated approach ensures early attack detection, protects sensitive information, and offers a scalable solution for securing future IoT networks.

1.1 SUMMARY OF THE BASE PAPER

Title	**:** A blockchain-orchestrated deep learning approach for secure data transmission in IoT-enabled healthcare system
Journal Name	**:** Journal of Parallel and Distributed Computing
Authors	**:** Prabhat Kumar, Randhir Kumar, Govind P. Gupta, et al.
Year	**:** 2023

This paper proposes a secure and intelligent framework for IoT-enabled healthcare systems by integrating blockchain technology with deep learning. The approach addresses critical issues such as data privacy, integrity, and real-time analysis of sensitive medical data. Blockchain ensures decentralized and tamper-proof data transmission, while deep learning enhances diagnostic accuracy through intelligent pattern recognition. Together, they provide a scalable and secure solution for modern healthcare environments.

1.2 DEMERITS OF EXISTING SYSTEM

- The current methods for securing data transmission have several challenges. Centralized systems are prone to failure and can be attacked easily. Encryption helps keep data private but doesn't stop unauthorized access or data changes.
- Machine learning models need a lot of labeled data and usually store it centrally, which can lead to privacy risks.
- Deep learning methods require sharing data for training, which also raises privacy concerns
- Existing security systems improves data integrity, but it is slow, uses a lot of computing power, and struggles to scale. These issues make existing solutions inefficient for secure data transmission.

1.3 PROBLEM DEFINITION

The rapid growth of digital data transmission, confidential information is vulnerable to cyber threats, unauthorized access and data violations. Traditional security approaches face challenges such as centralized risks, scalability problems and high computational costs. Ensuring the privacy and integrity of the data while allowing efficient automatic learning in distributed data is an important challenge. This project aims to develop a safe and decentralized data transmission model using blockchain for the integrity of data and federated learning for privacy prevention. The objective is to improve security, reduce cyber threats and guarantee data transmission without problems and reliable in modern digital environments.

- To implement secure data transmission with blockchain and federated learning over IoT network.
- To detect cyber-attacks and prevent unauthorized access.
- To evaluate the performance of proposed model for ensuring scalability and efficiency.

1.4 DATASET DESCRIPTION

Dataset Name: ToN_IoT (Telemetry & Network IoT Dataset)
Category: Network Traffic Data
Total Files: 23 CSV files
Columns: 44 features, including network packet details, traffic behaviour, and attack label
Target Column: label (indicates normal or attack traffic)
Additional Column: type (specifies attack category)
Issues Identified: Column variations, missing values, class imbalance, and outliers.
Purpose: Used for anomaly detection and cybersecurity research in IoT networks.

1.5 PROPOSED METHODOLOGY

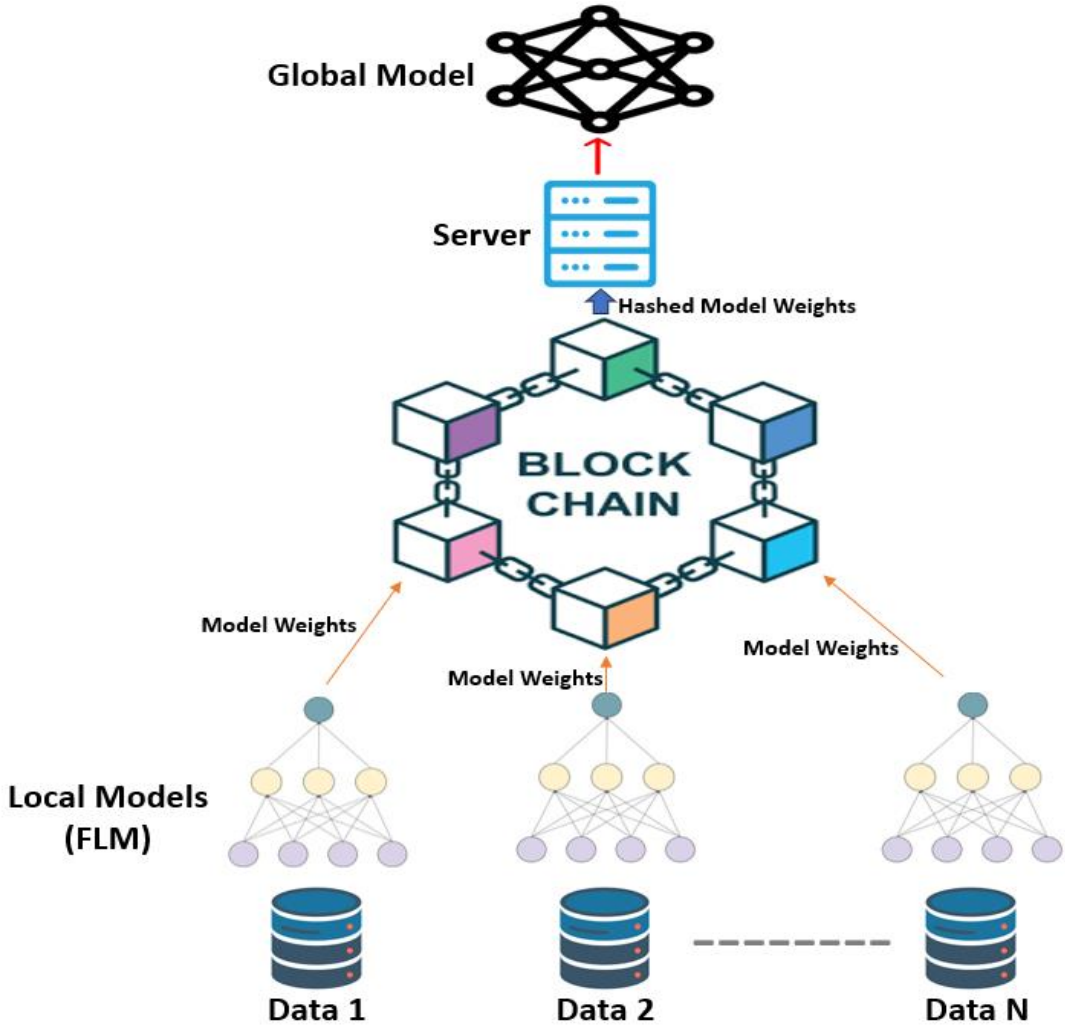


Fig. 1.1 BFL-IoT Architecture

The overall architecture of proposed BFL-IoT is shown in Fig. 1.1. In the federated learning system, the process begins with the creation of Local Models (FLM), each trained independently on its respective private dataset for example, Data 1, Data 2, up to Data N. Each local environment uses its own data to train a neural network model without sharing the actual data externally. This ensures complete data privacy while allowing each local model to learn important features, such as patterns, anomalies, or attack behaviours specific to its dataset. After completing the training, the system does not expose any raw data. Instead, it extracts only the learned model weights the essential parameters that the model has internalized during learning to move forward in the process.

Following the local training phase, the system proceeds to Model Weights Transmission. In this step, every client node sends its trained model's weights, which represent the "knowledge" acquired from its data to a central blockchain network. These weights, which capture the insights learned locally, serve as the building blocks for the global intelligence to be formed later.

Before reaching the central server, however, there is a crucial security step: the Blockchain Layer. Rather than sending the model weights directly, the system first hashes and securely stores them on a blockchain. Blockchain technology ensures several critical properties here. Firstly, immutability guarantees that once model weights are recorded, they cannot be altered or tampered with. Secondly, transparency ensures that all weight submissions and updates are visible, traceable, and auditable by participants in the network. Thirdly, security is fortified, protecting against potential threats such as model poisoning or manipulation during transmission. Each participating blockchain node can validate the legitimacy of the submitted model weights, establishing a foundation of trust even among nodes that may not inherently trust each other.

After successfully recording the model updates in the blockchain, the Server Federated Aggregation phase begins. The central server acts as an aggregator, retrieving the validated hashed weights from the blockchain. These weights undergo verification before being aggregated using an algorithm like Federated Averaging (FedAvg). This method averages the model weights from different clients to generate a unified representation of the collective learning. Through aggregation, the server synthesizes insights from multiple diverse datasets into one single model without ever accessing the actual local datasets.

Finally, the result of this process is the Global Model. This global model is the culmination of knowledge from all participating local models, crafted without compromising the privacy of any dataset. It becomes a powerful predictive model capable of identifying attacks, detecting anomalies, and understanding network behaviours across various environments. Because the system draws from multiple datasets without centralized data storage, it offers a privacy-preserving, decentralized, and secure approach to collaborative machine learning ideal for real-world applications in cybersecurity, healthcare, finance, and more.

1.6 1D CONVOLUTIONAL NEURAL NETWORK (CNN-1D)

A 1D Convolutional Neural Network (1D-CNN) is a type of deep learning model designed to process sequential or tabular data (not images).Instead of working over two dimensions (like images), it slides a filter (kernel) along one dimension — typically across features or time. Utilized 1D-CNN to detect normal vs attack instances based on tabular network data features.

1. Input Layer

Data: Each input sample is a row of numeric features from the CSV file (e.g., packet size, duration, bytes, flags).

Shape: (Number of Features, 1) → features treated as a 1D sequence.

2. First Convolution Layer (Conv1D)

Apply a 1D filter across the feature sequence to learn local patterns between neighboring features. The convolution operation multiplies small groups of adjacent features with the

filter weights and sums them, moving step-by-step (stride) along the sequence. Highlights useful local patterns for example, how src_bytes and dst_bytes together affect the outcome.

Parameters:
Filters (e.g., 64)
Kernel size (e.g., 3)

3. MaxPooling1D Layer
Reduce the size of the convolved feature maps and keep only the most important information. Takes the maximum value in each small window, thus shrinking the feature map size and focusing on strong patterns.Speeds up training, reduces overfitting, and allows the model to focus on dominant features.

4. Dropout Layer
Prevent overfitting during training. Randomly "drops" (sets to zero) a fraction of neurons in the layer during each training pass. Forces the model to not depend too heavily on any one part of the network.

5. Second Convolution Layer (Conv1D)
Learn higher-level patterns from the already extracted local features. A second convolution is applied to the max-pooled and dropout-processed features, building deeper feature representations. Makes the network capable of learning more complex relationships among multiple feature combinations.

6. Second MaxPooling1D Layer
Same as the first MaxPooling, further compresses and strengthens the important features extracted.

7. Second Dropout Layer
Same role — continues to reduce overfitting and improve generalization.

8. Flatten Layer
Transform the 3D output of convolution/pooling layers into a 1D flat vector. All extracted features are lined up into a single long vector. Prepares the data for input to fully connected (Dense) layers.

9. Dense Layer (Fully Connected)
Take the rich feature representation from previous layers and learn global, non-local patterns. Every neuron in the Dense layer is connected to every input neuron. Allows the model to make sophisticated decisions based on combinations of features.

Activation:
ReLU (Rectified Linear Unit) for non-linear learning.

10. Dropout Layer (after Dense)
Final Dropout before output to improve robustness.

11. Output Layer
Dense Layer (1 neuron)

Activation function: Sigmoid
Outputs a probability between 0 and 1.
If probability > 0.5 → classified as Attack (1)
If probability <= 0.5 → classified as Normal (0)

1.7 DEEP NEURAL NETWORK (DNN)

A Deep Neural Network (DNN) is a type of Artificial Neural Network with multiple hidden layers between the input and output layers. It learns complex relationships by combining many linear and non-linear transformations. DNN is used to classify whether the network activity is normal or an attack based on IoT dataset features.

1. **Input Layer:**
   The model takes structured input data with multiple features from the dataset. No reshaping is needed as DNN operates on feature vectors directly.

2. **First Dense Layer:**
   A fully connected (Dense) layer with 128 neurons. Each neuron takes input from all features and applies weighted sums.

   Activation: ReLU function is used to introduce non-linearity and make the network capable of learning complex patterns.

3. **Dropout Layer 1:**
   30% of neurons are randomly dropped during training. Prevents overfitting by forcing the model to not rely on particular neurons heavily.

4. **Second Dense Layer:**
   A Dense layer with 64 neurons. Further extracts deeper patterns from the previously learned features.

   Activation: Again ReLU for introducing non-linearity.

5. **Dropout Layer 2:**
   30% of neurons dropped again to maintain robustness.

6. **Third Dense Layer:**
   A Dense layer with 32 neurons. Reduces the feature dimensions and refines feature learning.

   Activation: ReLU to continue non-linear transformation.

7. **Dropout Layer 3:**
   20% of neurons dropped after the third dense layer to regularize the learning.

8. **Output Layer:**
   A single neuron with Sigmoid activation function. Outputs a probability between 0 and 1 indicating the confidence of the prediction. Thresholded (usually at 0.5) to decide between Normal (0) and Attack (1).

1.8 LOGISTIC REGRESSION (LR)

Logistic Regression (LR) is a traditional machine learning algorithm mainly used for binary classification tasks. It predicts whether an instance belongs to class 0 (Normal) or class 1 (Attack). Despite its name, it is a classification algorithm, not a regression algorithm.

1. **Input Features:**
   The features (after scaling) from the dataset are passed directly into the model.Each feature is multiplied by a weight and added together along with a bias term.

2. **Linear Combination:**
   Logistic Regression first computes a linear combination of input features:
   $z = w1*x1 + w2*x2 + ... + wn*xn + b$
   where w = weights, x = input features, and b = bias.

3. **Sigmoid Activation:**
   The result z is passed through a Sigmoid function:
   $sigmoid(z) = 1 / (1 + exp(-z))$
   This compresses the output to a probability between 0 and 1.
   If probability > 0.5 → Predict class 1 (Attack).
   If probability <= 0.5 → Predict class 0 (Normal).

1.9 TECHNOLOGIES USED

To implement this project some of the technologies are used are described below:

1. **TensorFlow and Keras**
   A deep learning framework used to build, train, and save neural network models like DNN and CNN.

2. Scikit-learn
A machine learning library used for preprocessing (like scaling), training logistic regression models, and evaluating performance.

3. FastAPI
A fast and modern Python web framework used to build APIs for model prediction, file handling, encryption, and download.

4. React.js
A JavaScript library for building the user interface (UI) where users upload CSVs, trigger predictions, and download results.

5. IPFS (InterPlanetary File System)
A decentralized file storage system used to store the trained model files (.h5) securely.

6. Ethereum Blockchain (Sepolia Testnet)
A public blockchain network where smart contracts store and manage references (hashes) to uploaded models.

7. Web3.py
A Python library that allows interaction with Ethereum smart contracts (e.g., submitting and fetching model hashes).

8. Infura
A gateway that connects your Python app to the Ethereum blockchain without running a full node.

1.10 EXPERIMENTAL SETUP

The machine used for this project is Google Colab, Jupyter and laptop with following specifications.

- Processor: Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz
- Random Access Memory (RAM) : 8GB
- Operating System : Windows 11 64-bit

1.11 FUNCTIONS USED IN SOURCE CODE

- **pandas.read_csv() -** To read CSV files into a DataFrame.
- **pandas.DataFrame.drop() -** To remove unwanted columns from a DataFrame.
- **numpy.array() -** To convert data into arrays.

- **numpy.mean() -** To compute the average value across arrays (used in model weight averaging).
- **tensorflow.keras.models.load_model() -** To load pre-trained Keras models (.h5 files).
- **tensorflow.keras.models.save_model()** - To save trained models in .h5 format.
- **tensorflow.keras.Sequential() -** To build sequential deep learning models (DNN, CNN).
- **tensorflow.keras.layers.Dense() -** To add fully connected (dense) layers.
- **tensorflow.keras.layers.Dropout() -** To add dropout regularization layers.
- **tensorflow.keras.layers.Conv1D() -** To add 1D convolutional layers (for CNN-1D).
- **tensorflow.keras.layers.MaxPooling1D() -** To downsample feature maps (for CNN-1D).
- **tensorflow.keras.layers.Flatten() -** To flatten output before passing to dense layers.
- **sklearn.model_selection.train_test_split() -** To split the dataset into training and testing parts.
- **sklearn.preprocessing.StandardScaler() -** To normalize the features by removing the mean and scaling to unit variance.
- **joblib.load() -** To load the saved scaler for later use.
- **web3.Web3() -** To create a connection with Ethereum blockchain using Infura.
- **web3.contract.ContractFunction.call() -** To call smart contract functions and read data.
- **requests.get() -** To make HTTP GET requests, used for downloading models from IPFS.
- **cryptography.fernet.Fernet() -** To handle encryption and decryption of files.
- **fastapi.FastAPI() -** To create API endpoints in Python backend.

CHAPTER 2
MERITS AND DEMERITS OF BASE PAPER

2.1 RELATED WORKS

Archana D. Wankhade et al. [1] proposed a model for attack detection and mitigation in cloud-based IoT networks using centralized learning methods. However, the system's real-time efficiency and scalability were not deeply addressed. Maryam Anwer et al. [2] used traditional machine learning techniques for intrusion detection in IoT, achieving good performance but lacked adaptation to continuously evolving threats. Zhiqiang Feng [3] integrated blockchain and federated learning for secure IoT data sharing, providing a strong base for decentralized intelligence, but the paper did not discuss implementation-level constraints. Hui Chen et al. [4] introduced a synaptic intelligent CNN for intrusion detection in dynamic IoT environments. Their model effectively learned temporal patterns but required high computation, limiting deployment on edge devices. Bhukya Madhu et al. [5] focused on deep learning models like CNN and LSTM for IoT intrusion detection, showing improved accuracy, yet they lacked in combining learning with secure data transmission. Mahmudul Hasan et al. [6] experimented with various machine learning models for anomaly detection in IoT sensor data and reported high accuracy but failed to ensure secure data sharing or resistance to adversarial attacks. Khawla Shalabi et al. [7] systematically reviewed blockchain-based IDS systems in IoT, suggesting enhanced security but missing hands-on implementation strategies. Sowmya T et al. [8] developed a stable feature selection algorithm for machine learning-based IDS, which improved model performance, yet did not cover model interpretability or deployment. C. Malathi et al. [9] proposed ML-based techniques for identifying cyber-attacks in IoT smart networks. Their approach was practical but lacked integration with modern deep learning frameworks or decentralized systems like blockchain.

2.2 MERITS

- Combines blockchain, Zero Knowledge Proof (ZKP), and deep learning to create a multi-layered defense against cyberattacks.
- Blockchain immutability ensures that healthcare data cannot be altered or tampered with once recorded.
- The use of Zero Knowledge Proof authenticates devices without revealing sensitive details, protecting user identities.
- Integrates IPFS (InterPlanetary File System) to handle and store massive volumes of IoT-generated healthcare data efficiently.
- The proposed Intrusion Detection System (IDS) based on Deep Sparse AutoEncoder and BiLSTM achieved about 99% accuracy on standard datasets.

- Designed to defend against impersonation, MITM (Man-in-the-Middle), replay attacks, and insider threats.

- Ethereum smart contracts automate the verification and trust-building process between IoT devices, removing third-party dependence.

- Comprehensive experiments were conducted on two credible datasets (ToN-IoT and CICIDS-2017), strengthening the credibility of results.

- Introduces an optimized lightweight Proof-of-Work (ePoW) algorithm for faster transaction validation compared to traditional PoW.

2.3 DEMERITS

- Deep learning models like DSAE and BiLSTM can demand significant computational power, which may strain resource-limited IoT devices.

- Even with optimizations, blockchain transactions and consensus processes introduce delay compared to centralized systems.

- Integrating blockchain, smart contracts, ZKP, IPFS, and deep learning makes real-world deployment complicated.

- While IPFS helps, blockchain still faces scalability challenges (e.g., network congestion) when handling millions of IoT devices.

- Bugs in smart contracts (common in Ethereum) could introduce new security risks if not rigorously audited.

- The framework was only tested in a simulated environment (using Ganache and Ethereum testnets) — not yet validated in real healthcare settings.

- Experiments are based only on ToN-IoT and CICIDS-2017 datasets; broader testing on diverse, real-world healthcare IoT datasets is missing.

- Even lightweight PoW (ePoW) involves computational work, potentially leading to energy consumption issues for battery-powered devices.

- IPFS storage, while decentralized, is not entirely free or universally stable; maintaining persistence in IPFS may incur hidden costs.

CHAPTER 3
SOURCE CODE

3.1 DATASET PROCESSING

```
import pandas as pd
import os
import gc
folder_path =  "C:/Users/Prakadeesh/Downloads/Processed_Network_dataset/Processed_Network_dataset"
output_file  = "D:/bal3/combined_dataset.csv"
csv_files = [file for file in os.listdir(folder_path) if file.endswith('.csv')]
first_file = pd.read_csv(os.path.join(folder_path, csv_files[0]), nrows=100, low_memory=False, dtype={8: str})
reference_columns = first_file.columns
with open(output_file, 'w') as f:
    first_file.to_csv(f, index=False)
for file in csv_files:
    file_path = os.path.join(folder_path, file)
    try:
        for chunk in pd.read_csv(file_path, chunksize=100000, low_memory=False, dtype={8: str}):
            if list(chunk.columns) == list(reference_columns):
                chunk.to_csv(output_file, mode='a', header=False, index=False)
            else:
                print(f"Skipping {file} due to column mismatch.")
            del chunk
            gc.collect()
    except Exception as e:
        print(f"Error processing {file}: {e}")
print(f"Combined CSV saved at: {output_file}")
```

```
import os
import dask.dataframe as dd


input_folder = "D:/bal1"
dtype_dict = {
    'src_bytes': 'object',
    'label': 'int64'
}


balanced_data = dd.read_csv(os.path.join(input_folder, "*.part"), dtype=dtype_dict)


output_folder = "D:/bal2"


balanced_data = balanced_data.repartition(npartitions=4)


for i in range(4):
    output_file_path = os.path.join(output_folder, f"balanced_split_{i+1}.csv")


    balanced_data.get_partition(i).to_csv(output_file_path, index=False, single_file=True, compute=True)

    print(f"Saved: {output_file_path}")

print("Balancing and splitting completed successfully!")
```

```
import os
import dask.dataframe as dd


input_folder = "D:/bal2"
output_folder = "D:/fi"


dtypes = {'src_bytes': 'object'}


file_paths = [os.path.join(input_folder, f"balanced_split_{i+1}.csv") for i in range(4)]
dfs = [dd.read_csv(file_path, dtype=dtypes, low_memory=False) for file_path in file_paths]


normal_data = dd.concat([dfs[0], dfs[1]])
attack_data = dd.concat([dfs[2], dfs[3]])

normal_count = normal_data.shape[0].compute()
attack_count = attack_data.shape[0].compute()

total_splits = 4
normal_per_split = normal_count / total_splits
attack_per_split = (normal_per_split * 70) / 30

if attack_per_split * total_splits > attack_count:
    raise ValueError("Not enough attack data to create 70:30 splits across all files.")

normal_frac = normal_per_split / normal_count
attack_frac = attack_per_split / attack_count

for i in range(total_splits):
    normal_split = normal_data.sample(frac=normal_frac, random_state=42+i)
    attack_split = attack_data.sample(frac=attack_frac, random_state=42+i)

    combined_split = dd.concat([normal_split, attack_split]).sample(frac=1, random_state=42+i)

    output_file_path = os.path.join(output_folder, f"balanced_final_split_{i+1}.csv")
    combined_split.to_csv(output_file_path, index=False, single_file=True, compute=True)

    print(f"Saved balanced split: {output_file_path}")

print("Balancing and splitting completed successfully!")
```

```python
import pandas as pd
import os

input_folder = "D:/fi"
output_folder = "D:/res"
csv_files = [f"{input_folder}/balanced_final_split_{i+1}.csv" for i in range(4)]

def analyze_and_clean(files):
    columns_to_remove = []

    dtype_issues = {}
    missing_values = {}

    for file in files:
        print(f"Analyzing {file}...")

        df = pd.read_csv(file, low_memory=False)

        empty_columns = df.columns[df.isnull().all()].tolist()
        if empty_columns:
            print(f"Empty columns in {file}: {empty_columns}")
            columns_to_remove.append(empty_columns)

        for col in df.columns:
            if df[col].dtype == 'object':
                try:
                    df[col] = df[col].astype(float)
                except ValueError:
                    if col not in dtype_issues:
                        dtype_issues[col] = []
                    dtype_issues[col].append(file)
                    print(f"Column {col} has dtype issues in {file}")

        missing_cols = df.columns[df.isnull().sum() > 0].tolist()
        if missing_cols:
            missing_values[file] = missing_cols
            print(f"Missing values in {file}: {missing_cols}")

    for file in files:
        df = pd.read_csv(file, low_memory=False)

        for cols in columns_to_remove:
            if all(col in df.columns for col in cols):
                df = df.drop(cols, axis=1)

        for col in df.columns:
            if df[col].isnull().sum() > 0:
                df[col] = df[col].fillna(df[col].mean())

        output_file = os.path.join(output_folder, os.path.basename(file))
        df.to_csv(output_file, index=False)
        print(f"Saved cleaned file: {output_file}")

analyze_and_clean(csv_files)
```

Fig. 3.1 dataset Processing

3.2 Models Implementation

3.2.1. Logistic Regression

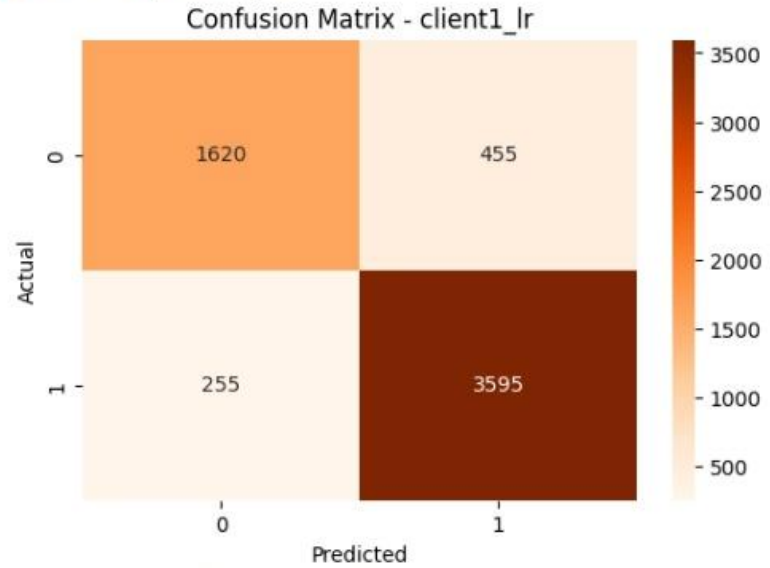
```
import os
import pandas as pd
import joblib
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

csv_files = [
    "/content/drive/MyDrive/abc/processed_balanced_final_split_1_processed.csv",
    "/content/drive/MyDrive/abc/processed_balanced_final_split_2_processed.csv",
    "/content/drive/MyDrive/abc/processed_balanced_final_split_3_processed.csv",
    "/content/drive/MyDrive/abc/processed_balanced_final_split_4_processed.csv"
]

for i, csv_path in enumerate(csv_files, 1):
    print(f"\n📁 Training Logistic Regression model for Client {i}")

    data = pd.read_csv(csv_path)
    X = data.drop(columns=['label', 'type', 'src_ip', 'dst_ip'], errors='ignore')
    X = X.select_dtypes(include='number')
    y = data['label']

    X_train, X_val, y_train, y_val = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)

    model = LogisticRegression(max_iter=1000, solver='lbfgs')
    model.fit(X_train_scaled, y_train)

    y_pred = model.predict(X_val_scaled)

    print("Confusion Matrix:")
    print(confusion_matrix(y_val, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_val, y_pred, digits=4))

    save_dir = f"/content/client{i}_lr"
    os.makedirs(save_dir, exist_ok=True)
    joblib.dump(model, os.path.join(save_dir, 'model_lr.pkl'))
    joblib.dump(scaler, os.path.join(save_dir, 'scaler_lr.pkl'))
    print(f"✅ Model and scaler saved to: {save_dir}")
```

```
📂 Training Logistic Regression model for Client 1
Confusion Matrix:
[[28626 23575]
 [ 6726 71575]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8097    0.5484    0.6539     52201
           1     0.7522    0.9141    0.8253     78301

    accuracy                         0.7678    130502
   macro avg     0.7810    0.7312    0.7396    130502
weighted avg     0.7752    0.7678    0.7567    130502

✅ Model and scaler saved to: /content/client1_lr

📂 Training Logistic Regression model for Client 2
Confusion Matrix:
[[28396 23805]
 [ 6643 71658]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8104    0.5440    0.6510     52201
           1     0.7506    0.9152    0.8248     78301

    accuracy                         0.7667    130502
   macro avg     0.7805    0.7296    0.7379    130502
weighted avg     0.7745    0.7667    0.7553    130502

✅ Model and scaler saved to: /content/client2_lr


📂 Training Logistic Regression model for Client 3
Confusion Matrix:
[[28673 23528]
 [ 6847 71454]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8072    0.5493    0.6537     52201
           1     0.7523    0.9126    0.8247     78301

    accuracy                         0.7672    130502
   macro avg     0.7798    0.7309    0.7392    130502
weighted avg     0.7743    0.7672    0.7563    130502

✅ Model and scaler saved to: /content/client3_lr

📂 Training Logistic Regression model for Client 4
Confusion Matrix:
[[28249 23952]
 [ 6490 71811]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8132    0.5412    0.6499     52201
           1     0.7499    0.9171    0.8251     78301

    accuracy                         0.7667    130502
   macro avg     0.7815    0.7291    0.7375    130502
weighted avg     0.7752    0.7667    0.7550    130502

✅ Model and scaler saved to: /content/client4_lr
```

Fig. 3.2 Training of LR

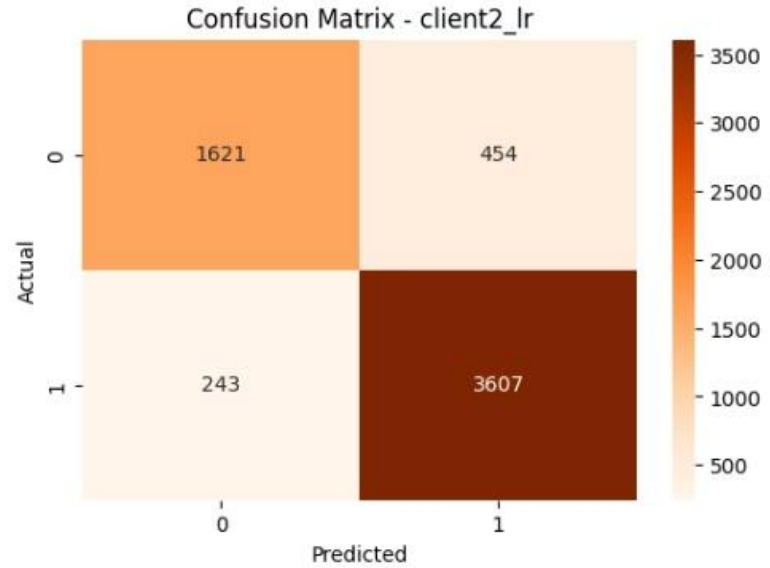
```python
import os
import pandas as pd
import numpy as np
import joblib
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

test_csv = '/content/drive/MyDrive/testdataset.csv'
test_data = pd.read_csv(test_csv)
y_test = test_data['label']
X_test_raw = test_data.drop(columns=['label'], errors='ignore')
X_test_numeric = X_test_raw.select_dtypes(include=[np.number])

base_path = '/content/drive/MyDrive'

for client in sorted(os.listdir(base_path)):
    client_path = os.path.join(base_path, client)

    if not os.path.isdir(client_path) or not client.endswith('_lr'):
        continue
    try:
        print(f"\n📊 Evaluating {client}")

        scaler = joblib.load(os.path.join(client_path, 'scaler_lr.pkl'))
        expected_columns = scaler.feature_names_in_
        X_test = X_test_numeric.reindex(columns=expected_columns, fill_value=0)
        X_test_scaled = scaler.transform(X_test)
        model = joblib.load(os.path.join(client_path, 'model_lr.pkl'))
        y_pred = model.predict(X_test_scaled)

        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(6, 4))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Oranges", xticklabels=[0, 1], yticklabels=[0, 1])
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title(f"Confusion Matrix - {client}")
        plt.show()

        print(classification_report(y_test, y_pred, digits=4))

    except Exception as e:
        print(f"❌ Error evaluating {client}: {e}")
```

Evaluating client1_lr



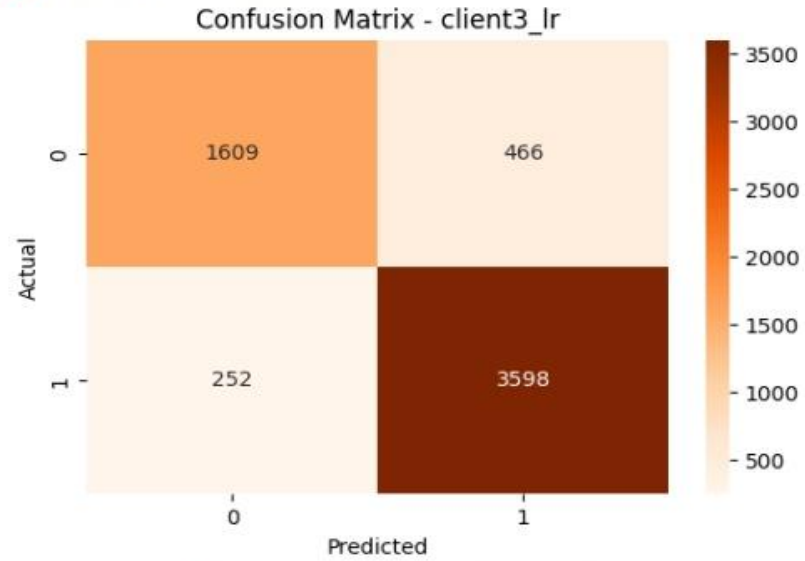
	precision	recall	f1-score	support
0	0.8640	0.7807	0.8203	2075
1	0.8877	0.9338	0.9101	3850
accuracy			0.8802	5925
macro avg	0.8758	0.8572	0.8652	5925
weighted avg	0.8794	0.8802	0.8787	5925

Evaluating client2_lr



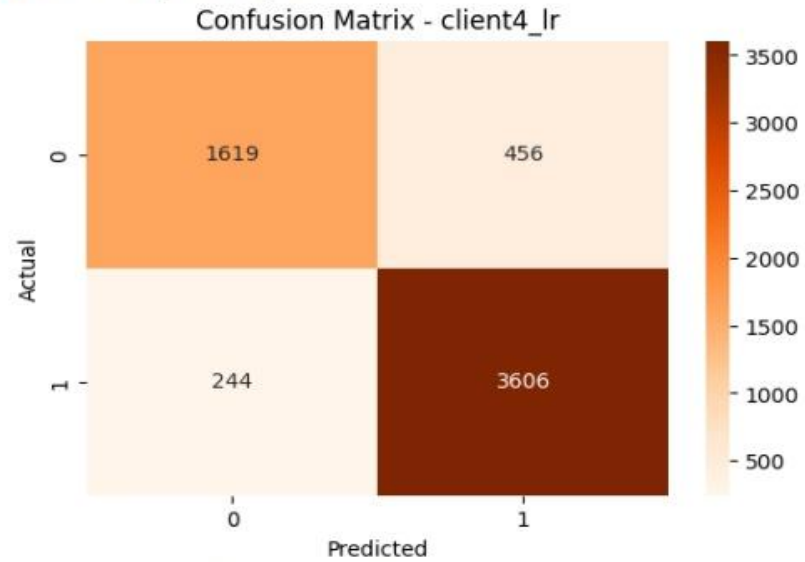
	precision	recall	f1-score	support
0	0.8696	0.7812	0.8231	2075
1	0.8882	0.9369	0.9119	3850
accuracy			0.8824	5925
macro avg	0.8789	0.8590	0.8675	5925
weighted avg	0.8817	0.8824	0.8808	5925

Evaluating client3_lr



	precision	recall	f1-score	support
0	0.8646	0.7754	0.8176	2075
1	0.8853	0.9345	0.9093	3850
accuracy			0.8788	5925
macro avg	0.8750	0.8550	0.8634	5925
weighted avg	0.8781	0.8788	0.8772	5925

Evaluating client4_lr



	precision	recall	f1-score	support
0	0.8690	0.7802	0.8222	2075
1	0.8877	0.9366	0.9115	3850
accuracy			0.8819	5925
macro avg	0.8784	0.8584	0.8669	5925
weighted avg	0.8812	0.8819	0.8803	5925

Fig. 3.3 Testing LR local models

```python
import os
import joblib
import numpy as np
from sklearn.linear_model import LogisticRegression

client_dirs = [
    '/content/drive/MyDrive/client1_lr',
    '/content/drive/MyDrive/client2_lr',
    '/content/drive/MyDrive/client3_lr',
    '/content/drive/MyDrive/client4_lr'
]

coefs = []
intercepts = []
scalers = []

for client_path in client_dirs:
    model = joblib.load(os.path.join(client_path, 'model_lr.pkl'))
    scaler = joblib.load(os.path.join(client_path, 'scaler_lr.pkl'))

    coefs.append(model.coef_)
    intercepts.append(model.intercept_)
    scalers.append(scaler)

    print(f"✅ Loaded model from {client_path}")

avg_coef = np.mean(coefs, axis=0)
avg_intercept = np.mean(intercepts, axis=0)

global_scaler = scalers[0]

global_model = LogisticRegression()
global_model.coef_ = avg_coef
global_model.intercept_ = avg_intercept
global_model.classes_ = np.array([0, 1])

joblib.dump(global_model, '/content/global_lr_model.pkl')
joblib.dump(global_scaler, '/content/global_lr_scaler.pkl')

print("\n✅ Global Logistic Regression model and scaler saved successfully.")
```

```
✅ Loaded model from /content/drive/MyDrive/client1_lr
✅ Loaded model from /content/drive/MyDrive/client2_lr
✅ Loaded model from /content/drive/MyDrive/client3_lr
✅ Loaded model from /content/drive/MyDrive/client4_lr

✅ Global Logistic Regression model and scaler saved successfully.
```

Fig. 3.4 Updating the LR global model

```
import pandas as pd
import numpy as np
import joblib
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

test_csv = '/content/drive/MyDrive/testdataset.csv'
test_data = pd.read_csv(test_csv)

y_test = test_data['label']
X_test_raw = test_data.drop(columns=['label'], errors='ignore')
X_test_numeric = X_test_raw.select_dtypes(include=[np.number])

scaler = joblib.load('/content/drive/MyDrive/global_lr_scaler.pkl')
model = joblib.load('/content/drive/MyDrive/global_lr_model.pkl')

expected_columns = scaler.feature_names_in_
X_test = X_test_numeric.reindex(columns=expected_columns, fill_value=0)
X_test_scaled = scaler.transform(X_test)

y_pred = model.predict(X_test_scaled)

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Oranges", xticklabels=[0, 1], yticklabels=[0, 1])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Global LR Model")
plt.show()

print(classification_report(y_test, y_pred, digits=4))
```

Fig. 3.5 Testing of LR global model

3.2.2. Convolution Neural Network (CNN)

```
import pandas as pd
import numpy as np
import joblib
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout, Input
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import mixed_precision
mixed_precision.set_global_policy('mixed_float16')
csv_files = [
    "/content/drive/MyDrive/abc/processed_balanced_final_split_1_processed.csv",
    "/content/drive/MyDrive/abc/processed_balanced_final_split_2_processed.csv",
    "/content/drive/MyDrive/abc/processed_balanced_final_split_3_processed.csv",
    "/content/drive/MyDrive/abc/processed_balanced_final_split_4_processed.csv"
]
for i, csv_path in enumerate(csv_files, 1):
    print(f"\n--- Training model for client {i} ---")
    data = pd.read_csv(csv_path)
    X = data.drop(columns=['label', 'type', 'src_ip', 'dst_ip'], errors='ignore')
    X = X.select_dtypes(include=[np.number])
    y = data['label']
    X_train, X_val, y_train, y_val = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)
    save_dir = f"/content/client{i}_cnn"
    os.makedirs(save_dir, exist_ok=True)
    joblib.dump(scaler, os.path.join(save_dir, "scaler.pkl"))
    X_train_scaled = X_train_scaled[..., np.newaxis]
    X_val_scaled = X_val_scaled[..., np.newaxis]

    model = Sequential([
        Input(shape=(X_train_scaled.shape[1], 1)),
        Conv1D(64, kernel_size=3, activation='relu'),
        MaxPooling1D(pool_size=2),
        Dropout(0.3),
        Conv1D(128, kernel_size=3, activation='relu'),
        MaxPooling1D(pool_size=2),
        Dropout(0.3),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(1, activation='sigmoid', dtype='float32')  # Ensure final output is float32
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
    model.fit(X_train_scaled, y_train, validation_data=(X_val_scaled, y_val),
              epochs=10, batch_size=128, callbacks=[early_stop], verbose=1)
    model.save(os.path.join(save_dir, "cnn1d_model.h5"))
```

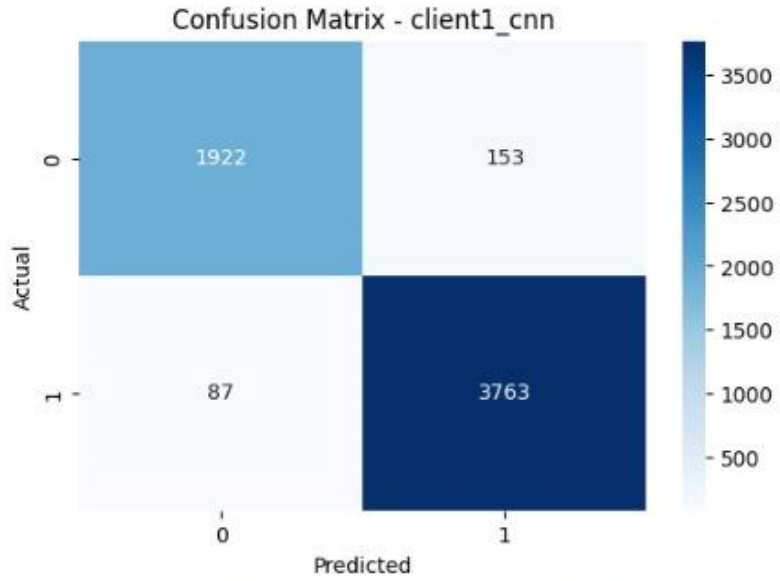
```
--- Training model for client 1 ---
Epoch 1/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 27s 5ms/step - accuracy: 0.8443 - loss: 0.3674 - val_accuracy: 0.9389 - val_loss: 0.1750
Epoch 2/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 31s 3ms/step - accuracy: 0.9316 - loss: 0.2002 - val_accuracy: 0.9562 - val_loss: 0.1382
Epoch 3/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 23s 4ms/step - accuracy: 0.9439 - loss: 0.1699 - val_accuracy: 0.9578 - val_loss: 0.1298
Epoch 4/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.9484 - loss: 0.1571 - val_accuracy: 0.9595 - val_loss: 0.1234
Epoch 5/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.9506 - loss: 0.1503 - val_accuracy: 0.9610 - val_loss: 0.1178
Epoch 6/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.9517 - loss: 0.1450 - val_accuracy: 0.9615 - val_loss: 0.1130
Epoch 7/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 20s 3ms/step - accuracy: 0.9528 - loss: 0.1409 - val_accuracy: 0.9620 - val_loss: 0.1118
Epoch 8/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 21s 3ms/step - accuracy: 0.9542 - loss: 0.1362 - val_accuracy: 0.9622 - val_loss: 0.1076
Epoch 9/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.9559 - loss: 0.1331 - val_accuracy: 0.9618 - val_loss: 0.1044
Epoch 10/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.9551 - loss: 0.1316 - val_accuracy: 0.9629 - val_loss: 0.1012
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format

--- Training model for client 2 ---
Epoch 1/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 20s 4ms/step - accuracy: 0.8471 - loss: 0.3631 - val_accuracy: 0.9370 - val_loss: 0.1582
Epoch 2/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 16s 3ms/step - accuracy: 0.9343 - loss: 0.1854 - val_accuracy: 0.9558 - val_loss: 0.1228
Epoch 3/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 20s 3ms/step - accuracy: 0.9464 - loss: 0.1523 - val_accuracy: 0.9608 - val_loss: 0.1063
Epoch 4/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 13s 3ms/step - accuracy: 0.9511 - loss: 0.1367 - val_accuracy: 0.9653 - val_loss: 0.0936
Epoch 5/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.9563 - loss: 0.1248 - val_accuracy: 0.9709 - val_loss: 0.0836
Epoch 6/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 21s 3ms/step - accuracy: 0.9593 - loss: 0.1170 - val_accuracy: 0.9727 - val_loss: 0.0796
Epoch 7/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.9605 - loss: 0.1133 - val_accuracy: 0.9719 - val_loss: 0.0747
Epoch 8/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 21s 3ms/step - accuracy: 0.9622 - loss: 0.1070 - val_accuracy: 0.9751 - val_loss: 0.0709
Epoch 9/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 20s 3ms/step - accuracy: 0.9641 - loss: 0.1029 - val_accuracy: 0.9737 - val_loss: 0.0693
Epoch 10/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 20s 3ms/step - accuracy: 0.9655 - loss: 0.1010 - val_accuracy: 0.9718 - val_loss: 0.0705
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format
--- Training model for client 3 ---
Epoch 1/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 20s 4ms/step - accuracy: 0.8391 - loss: 0.3749 - val_accuracy: 0.9280 - val_loss: 0.1816
Epoch 2/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 16s 3ms/step - accuracy: 0.9261 - loss: 0.2078 - val_accuracy: 0.9492 - val_loss: 0.1424
Epoch 3/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 20s 3ms/step - accuracy: 0.9409 - loss: 0.1744 - val_accuracy: 0.9592 - val_loss: 0.1198
Epoch 4/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 20s 3ms/step - accuracy: 0.9468 - loss: 0.1539 - val_accuracy: 0.9594 - val_loss: 0.1111
Epoch 5/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 13s 3ms/step - accuracy: 0.9509 - loss: 0.1406 - val_accuracy: 0.9623 - val_loss: 0.0988
Epoch 6/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.9536 - loss: 0.1342 - val_accuracy: 0.9658 - val_loss: 0.0893
Epoch 7/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 13s 3ms/step - accuracy: 0.9557 - loss: 0.1244 - val_accuracy: 0.9712 - val_loss: 0.0863
Epoch 8/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 21s 3ms/step - accuracy: 0.9582 - loss: 0.1164 - val_accuracy: 0.9733 - val_loss: 0.0745
Epoch 9/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 13s 3ms/step - accuracy: 0.9614 - loss: 0.1120 - val_accuracy: 0.9723 - val_loss: 0.0746
Epoch 10/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 13s 3ms/step - accuracy: 0.9644 - loss: 0.1048 - val_accuracy: 0.9760 - val_loss: 0.0747
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format

--- Training model for client 4 ---
Epoch 1/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 21s 4ms/step - accuracy: 0.8245 - loss: 0.3954 - val_accuracy: 0.9296 - val_loss: 0.1896
Epoch 2/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.9269 - loss: 0.2092 - val_accuracy: 0.9544 - val_loss: 0.1388
Epoch 3/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 21s 3ms/step - accuracy: 0.9426 - loss: 0.1721 - val_accuracy: 0.9568 - val_loss: 0.1273
Epoch 4/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.9479 - loss: 0.1572 - val_accuracy: 0.9579 - val_loss: 0.1226
Epoch 5/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 20s 3ms/step - accuracy: 0.9516 - loss: 0.1444 - val_accuracy: 0.9599 - val_loss: 0.1147
Epoch 6/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 13s 3ms/step - accuracy: 0.9533 - loss: 0.1383 - val_accuracy: 0.9629 - val_loss: 0.1040
Epoch 7/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 21s 3ms/step - accuracy: 0.9550 - loss: 0.1314 - val_accuracy: 0.9628 - val_loss: 0.1016
Epoch 8/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 23s 6ms/step - accuracy: 0.9563 - loss: 0.1280 - val_accuracy: 0.9648 - val_loss: 0.0962
Epoch 9/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 32s 3ms/step - accuracy: 0.9577 - loss: 0.1239 - val_accuracy: 0.9678 - val_loss: 0.0915
Epoch 10/10
4079/4079 ━━━━━━━━━━━━━━━━━━━━ 21s 3ms/step - accuracy: 0.9594 - loss: 0.1185 - val_accuracy: 0.9735 - val_loss: 0.0871
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format
```

Fig. 3.6 Training of CNN

```python
import os
import pandas as pd
import numpy as np
import joblib
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

test_csv = '/content/drive/MyDrive/testdataset.csv'
test_data = pd.read_csv(test_csv)
y_test = test_data['label']
X_test_raw = test_data.drop(columns=['label'], errors='ignore')
X_test_numeric = X_test_raw.select_dtypes(include=[np.number])
base_path = '/content/drive/MyDrive'

for client_folder in sorted(os.listdir(base_path)):
    if not client_folder.startswith("client") or not client_folder.endswith("_cnn"):
        continue
    client_path = os.path.join(base_path, client_folder)
    model_path = os.path.join(client_path, 'cnn1d_model.h5')
    scaler_path = os.path.join(client_path, 'scaler.pkl')
    try:
        print(f"\n📊 Evaluating {client_folder}")
        scaler = joblib.load(scaler_path)
        expected_columns = scaler.feature_names_in_
        X_test = X_test_numeric.reindex(columns=expected_columns, fill_value=0)
        X_test_scaled = scaler.transform(X_test)
        X_test_scaled = X_test_scaled[..., np.newaxis]
        model = tf.keras.models.load_model(model_path, compile=False)

        y_pred_probs = model.predict(X_test_scaled)
        y_pred = (y_pred_probs > 0.5).astype(int).flatten()
        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(6, 4))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=[0, 1], yticklabels=[0, 1])
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title(f"Confusion Matrix - {client_folder}")
        plt.show()
        print(classification_report(y_test, y_pred, digits=4))
    except Exception as e:
        print(f"❌ Error evaluating {client_folder}: {e}")
```

Evaluating client1_cnn

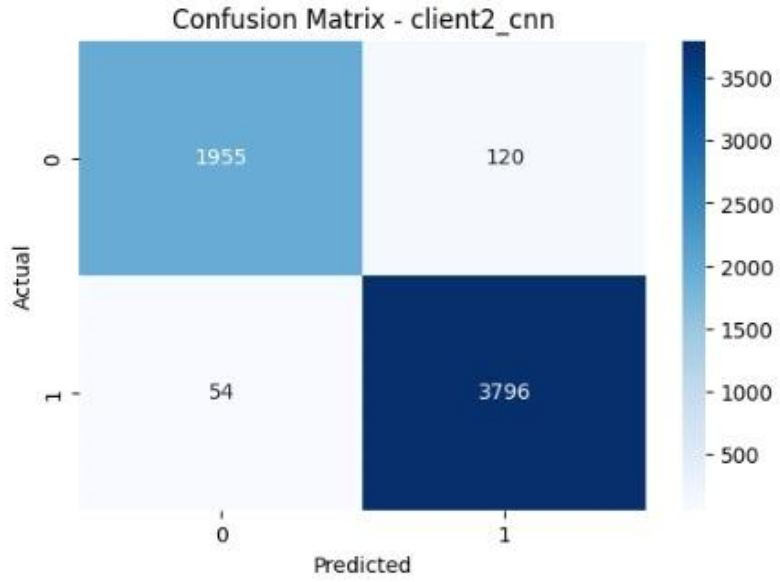
186/186 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step



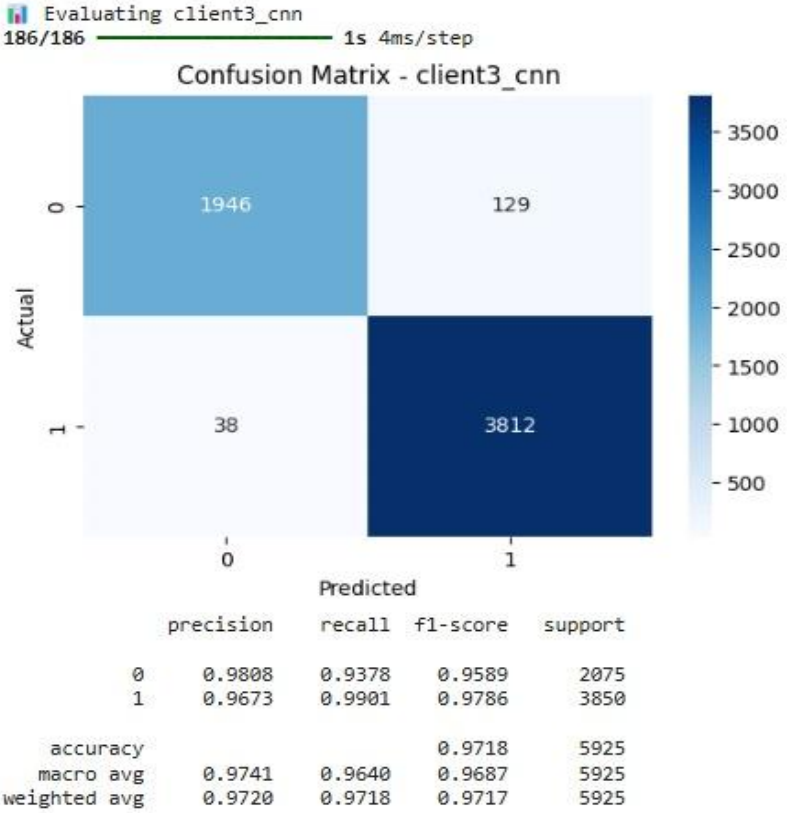
	precision	recall	f1-score	support
0	0.9567	0.9263	0.9412	2075
1	0.9609	0.9774	0.9691	3850
accuracy			0.9595	5925
macro avg	0.9588	0.9518	0.9552	5925
weighted avg	0.9594	0.9595	0.9593	5925

Evaluating client2_cnn

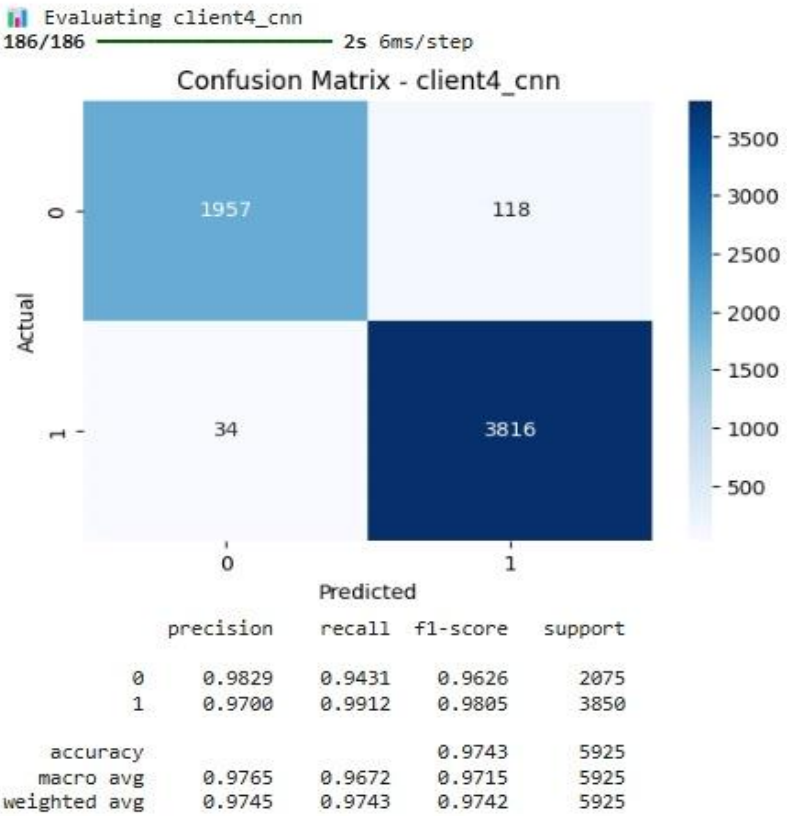
186/186 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step



	precision	recall	f1-score	support
0	0.9731	0.9422	0.9574	2075
1	0.9694	0.9860	0.9776	3850
accuracy			0.9706	5925
macro avg	0.9712	0.9641	0.9675	5925
weighted avg	0.9707	0.9706	0.9705	5925



	precision	recall	f1-score	support
0	0.9808	0.9378	0.9589	2075
1	0.9673	0.9901	0.9786	3850
accuracy			0.9718	5925
macro avg	0.9741	0.9640	0.9687	5925
weighted avg	0.9720	0.9718	0.9717	5925



	precision	recall	f1-score	support
0	0.9829	0.9431	0.9626	2075
1	0.9700	0.9912	0.9805	3850
accuracy			0.9743	5925
macro avg	0.9765	0.9672	0.9715	5925
weighted avg	0.9745	0.9743	0.9742	5925

Fig. 3.7 Testing of CNN local models

```python
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model, clone_model

base_path = '/content'
client_folders = [f for f in os.listdir(base_path) if f.startswith("client") and f.endswith("_cnn")]

models = []
for folder in sorted(client_folders):
    try:
        model_path = os.path.join(base_path, folder, 'cnn1d_model.h5')
        model = load_model(model_path, compile=False)
        models.append(model)
        print(f"✅ Loaded model from: {folder}")
    except Exception as e:
        print(f"❌ Error loading from {folder}: {e}")

if not models:
    raise ValueError("No models loaded successfully.")

global_model = clone_model(models[0])
global_model.build(input_shape=models[0].input_shape)

final_weights = []
for i in range(len(models)):
    weights = models[i].get_weights()
    final_weights.append(weights[-2])
    final_weights.append(weights[-1])

kernel_avg = np.mean([final_weights[i] for i in range(0, len(final_weights), 2)], axis=0)
bias_avg = np.mean([final_weights[i] for i in range(1, len(final_weights), 2)], axis=0)

base_weights = models[0].get_weights()
base_weights[-2] = kernel_avg
base_weights[-1] = bias_avg

global_model.set_weights(base_weights)

global_model_path = "/content/global_cnn_model_partial_avg.h5"
global_model.save(global_model_path)
print(f"\n✅ Global CNN model saved successfully with final layer averaging at: {global_model_path}")
```

```
✅ Loaded model from: client1_cnn
✅ Loaded model from: client2_cnn
✅ Loaded model from: client3_cnn
✅ Loaded model from: client4_cnn
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`.

✅ Global CNN model saved successfully with final layer averaging at: /content/global_cnn_model_partial_avg.h5
```

Fig. 3.8 Updating the CNN global model

```python
import os
import pandas as pd
import numpy as np
import joblib
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

test_csv = '/content/drive/MyDrive/testdataset.csv'
model_path = '/content/drive/MyDrive/global_cnn_model_partial_avg.h5'
scaler_path = '/content/drive/MyDrive/client4_cnn/scaler.pkl'

test_data = pd.read_csv(test_csv)
y_test = test_data['label']
X_test_raw = test_data.drop(columns=['label'], errors='ignore')
X_test_numeric = X_test_raw.select_dtypes(include=[np.number])

scaler = joblib.load(scaler_path)
expected_columns = scaler.feature_names_in_
X_test = X_test_numeric.reindex(columns=expected_columns, fill_value=0)
X_test_scaled = scaler.transform(X_test)
X_test_scaled = X_test_scaled[..., np.newaxis]

model = tf.keras.models.load_model(model_path, compile=False)

y_pred_probs = model.predict(X_test_scaled)
y_pred = (y_pred_probs > 0.5).astype(int).flatten()

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=[0, 1], yticklabels=[0, 1])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix Heatmap")
plt.show()

print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=4))
```

Fig. 3.9 Testing of CNN global model

3.2.3 DNN

```
import numpy as np
import joblib
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
csv_folder = "/content/drive/MyDrive/abc"
csv_files = [
    "processed_balanced_final_split_1_processed.csv",
    "processed_balanced_final_split_2_processed.csv",
    "processed_balanced_final_split_3_processed.csv",
    "processed_balanced_final_split_4_processed.csv"
]
def build_model(input_dim):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(input_dim,)),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
for idx, file in enumerate(csv_files, 1):
    print(f"\nTraining model for: {file}")
    data = pd.read_csv(os.path.join(csv_folder, file))
    X = data.drop(columns=['label', 'type'])
    y = data['label']
    X_train, X_val, y_train, y_val = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
    X_train = X_train.select_dtypes(include=[np.number])
    X_val = X_val.select_dtypes(include=[np.number])
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)
    save_dir = f"/content/client{idx}_dense"
    os.makedirs(save_dir, exist_ok=True)
    joblib.dump(scaler, os.path.join(save_dir, 'scaler.pkl'))
    model = build_model(X_train_scaled.shape[1])
    early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
    model.fit(X_train_scaled, y_train,
              validation_data=(X_val_scaled, y_val),
              epochs=15,
              batch_size=128,
              callbacks=[early_stop],
              verbose=2)

    model.save(os.path.join(save_dir, 'model.h5'))
    print(f"✅ Saved model for Client {idx} to {save_dir}")
```

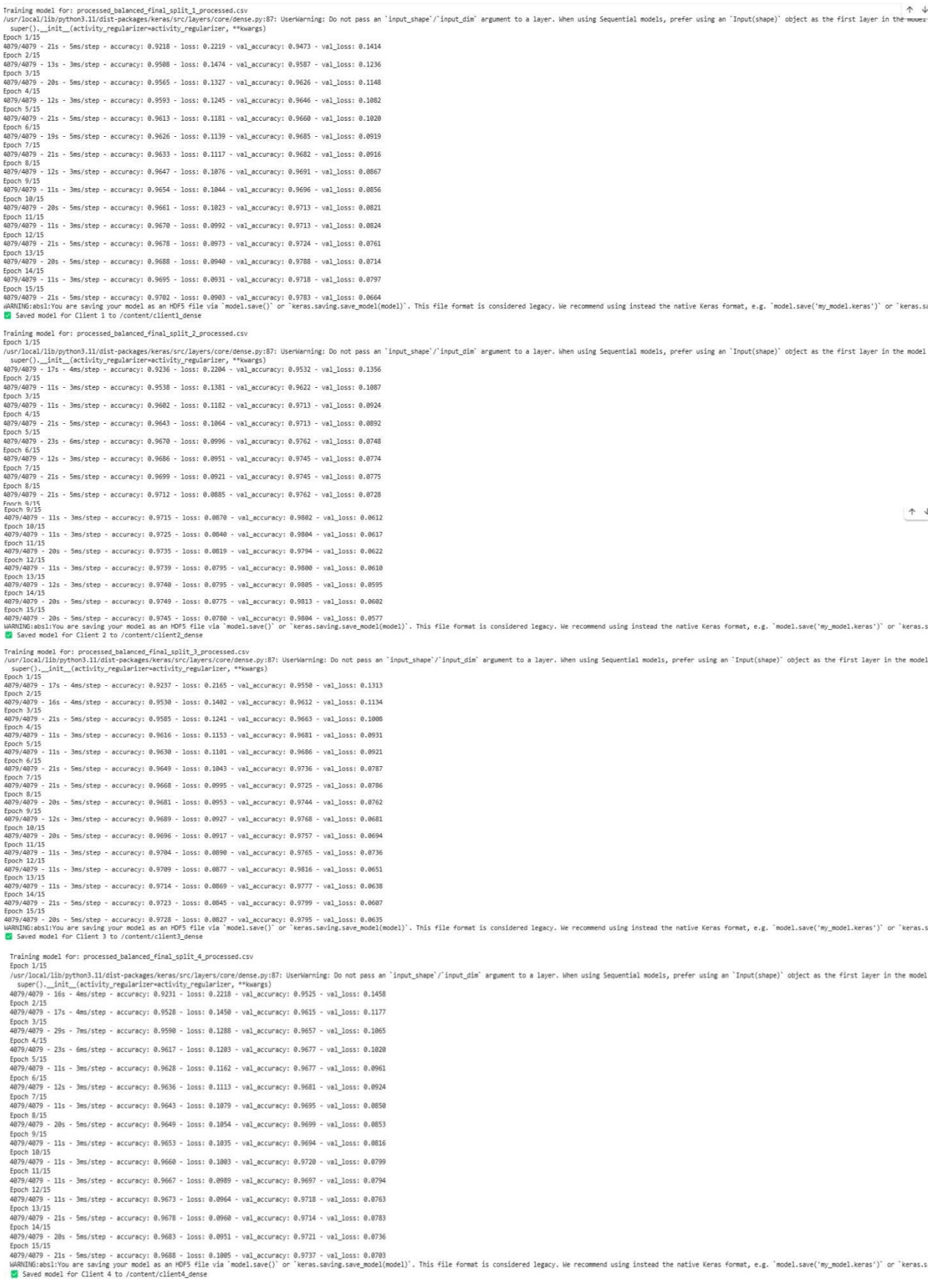
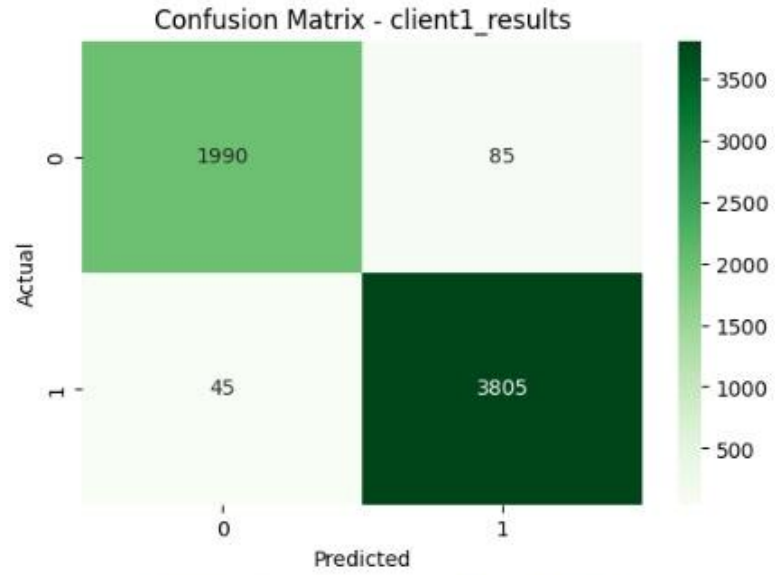


Fig. 3.10 Training of DNN local models

```python
import os
import pandas as pd
import numpy as np
import joblib
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

test_csv = '/content/drive/MyDrive/testdataset.csv'
test_data = pd.read_csv(test_csv)

y_test = test_data['label']
X_test_raw = test_data.drop(columns=['label'], errors='ignore')
X_test_numeric = X_test_raw.select_dtypes(include=[np.number])

base_path = '/content/drive/MyDrive/keras'

for client_folder in sorted(os.listdir(base_path)):
    if not client_folder.startswith("client") or not client_folder.endswith("_results"):
        continue

    client_path = os.path.join(base_path, client_folder)
    model_path = os.path.join(client_path, f"{client_folder.replace('_results', '')}_weights.h5")
    scaler_path = os.path.join(client_path, 'scaler_client1.pkl')

    try:
        print(f"\n📊 Evaluating {client_folder}")

        scaler = joblib.load(scaler_path)
        expected_columns = scaler.feature_names_in_
        X_test = X_test_numeric.reindex(columns=expected_columns, fill_value=0)
        X_test_scaled = scaler.transform(X_test)

        model = tf.keras.models.load_model(model_path, compile=False)

        y_pred_probs = model.predict(X_test_scaled)
        y_pred = (y_pred_probs > 0.5).astype(int).flatten()

        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(6, 4))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Greens", xticklabels=[0, 1], yticklabels=[0, 1])
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title(f"Confusion Matrix - {client_folder}")
        plt.show()

        print(classification_report(y_test, y_pred, digits=4))

    except Exception as e:
        print(f"❌ Error evaluating {client_folder}: {e}")
```

Evaluating client1_results

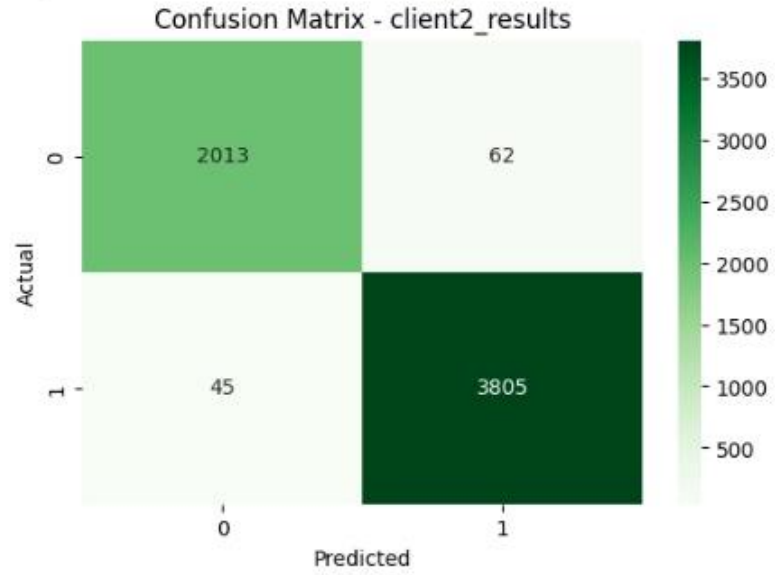
186/186 ——— 2s 7ms/step



	precision	recall	f1-score	support
0	0.9779	0.9590	0.9684	2075
1	0.9781	0.9883	0.9832	3850
accuracy			0.9781	5925
macro avg	0.9780	0.9737	0.9758	5925
weighted avg	0.9781	0.9781	0.9780	5925

Evaluating client2_results

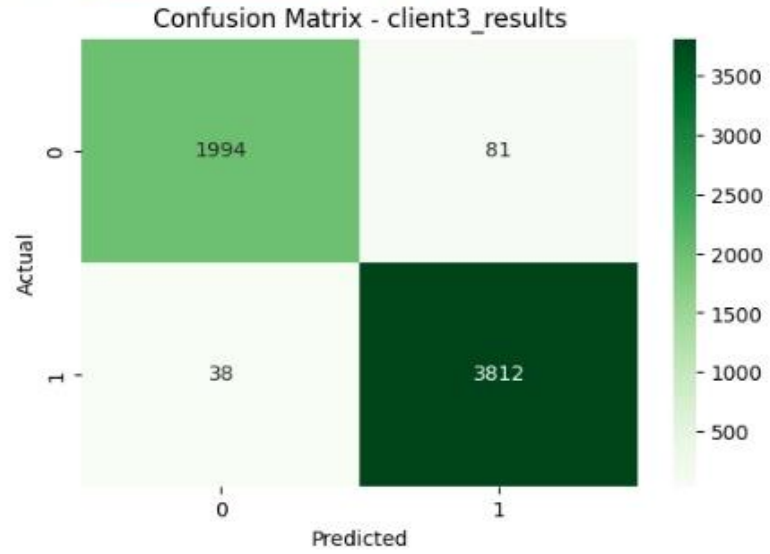
186/186 ——— 2s 7ms/step



	precision	recall	f1-score	support
0	0.9781	0.9701	0.9741	2075
1	0.9840	0.9883	0.9861	3850
accuracy			0.9819	5925
macro avg	0.9811	0.9792	0.9801	5925
weighted avg	0.9819	0.9819	0.9819	5925

Evaluating client3_results

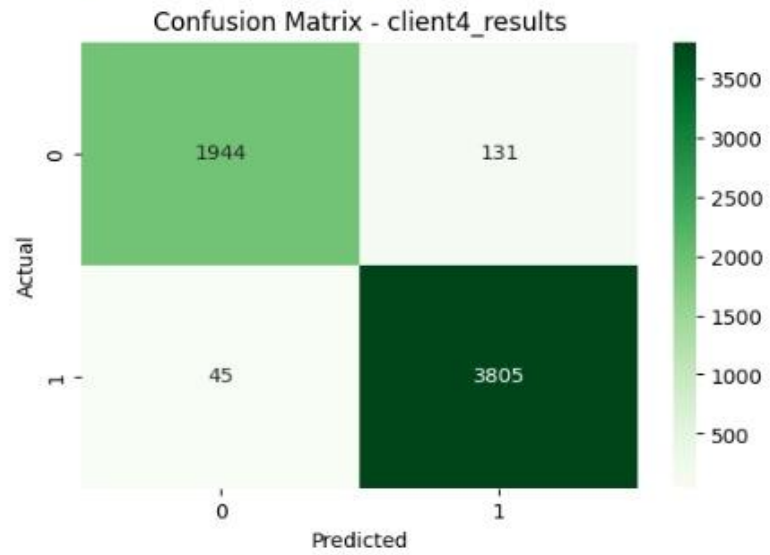
186/186 ——————— 1s 5ms/step



	precision	recall	f1-score	support
0	0.9813	0.9610	0.9710	2075
1	0.9792	0.9901	0.9846	3850
accuracy			0.9799	5925
macro avg	0.9802	0.9755	0.9778	5925
weighted avg	0.9799	0.9799	0.9799	5925

Evaluating client4_results

186/186 ——————— 2s 6ms/step



	precision	recall	f1-score	support
0	0.9774	0.9369	0.9567	2075
1	0.9667	0.9883	0.9774	3850
accuracy			0.9703	5925
macro avg	0.9720	0.9626	0.9670	5925
weighted avg	0.9705	0.9703	0.9701	5925

Fig. 3.11 Testing of DNN local models

```python
import os
import numpy as np
from tensorflow.keras.models import load_model, clone_model

client_dirs = [
    '/content/drive/MyDrive/keras/client1_results',
    '/content/drive/MyDrive/keras/client2_results',
    '/content/drive/MyDrive/keras/client3_results',
    '/content/drive/MyDrive/keras/client4_results'
]

models = [load_model(os.path.join(dir, f'client{i+1}_weights.h5')) for i, dir in enumerate(client_dirs)]

base_model = models[0]
global_model = clone_model(base_model)
global_model.build(input_shape=(None, 17))
global_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

all_weights = [model.get_weights() for model in models]
base_weights = base_model.get_weights()

averaged_weights = []
for i in range(len(base_weights)):
    try:
        layer_weights = [weights[i] for weights in all_weights]
        if all(w.shape == layer_weights[0].shape for w in layer_weights):
            avg = np.mean(np.array(layer_weights), axis=0)
            averaged_weights.append(avg)
        else:
            print(f"Layer {i} updated.")
            averaged_weights.append(base_weights[i])
    except Exception as e:
        print(f"layer {i} updated.")
        averaged_weights.append(base_weights[i])

global_model.set_weights(averaged_weights)
global_model.save('/content/global_model.keras')
print("Global model saved successfully.")
```

```
Layer 0 updated.
Layer 1 updated.
Layer 2 updated.
Layer 3 updated.
Layer 4 updated.
Layer 5 updated.
layer 6 updated.
layer 7 updated.
Global model saved successfully.
```

Fig. 3.12 Updating the DNN global model

```
import pandas as pd
import numpy as np
import joblib
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.metrics import confusion_matrix, classification_report

test_csv = '/content/drive/MyDrive/testdataset.csv'
test_data = pd.read_csv(test_csv)

y_test = test_data['label']
X_test_raw = test_data.drop(columns=['label'], errors='ignore')
X_test_numeric = X_test_raw.select_dtypes(include=[np.number])

scaler = joblib.load('/content/drive/MyDrive/keras/client4_results/scaler_client1.pkl')
model = tf.keras.models.load_model('/content/drive/MyDrive/global_model.keras')

expected_columns = scaler.feature_names_in_
X_test = X_test_numeric.reindex(columns=expected_columns, fill_value=0)
X_test_scaled = scaler.transform(X_test)


y_pred_probs = model.predict(X_test_scaled)
y_pred = (y_pred_probs > 0.5).astype(int).flatten()


cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens", xticklabels=[0, 1], yticklabels=[0, 1])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Global Dense Model")
plt.show()


print(classification_report(y_test, y_pred, digits=4))
```

Fig. 3.13 Testing of DNN global model

3.3 BLOCKCHAIN

```python
import hashlib
import json
import time

class Blockchain:
    def __init__(self):
        self.chain = []
        self.transactions = []
        self.create_block(proof=1, previous_hash="0")  # Genesis block

    def create_block(self, proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time.time(),
            'transactions': self.transactions,
            'proof': proof,
            'previous_hash': previous_hash
        }
        self.transactions = []  # Reset transactions
        self.chain.append(block)
        return block

    def get_previous_block(self):
        return self.chain[-1]

    def proof_of_work(self, previous_proof):
        new_proof = 1
        while True:
            hash_value = hashlib.sha256(str(new_proof**2 - previous_proof**2).encode()).hexdigest()
            if hash_value[:4] == "0000":  # PoW condition
                return new_proof
            new_proof += 1

    def hash(self, block):
        return hashlib.sha256(json.dumps(block, sort_keys=True).encode()).hexdigest()

    def add_transaction(self, sender, receiver, model_hash):
        self.transactions.append({'sender': sender, 'receiver': receiver, 'model_hash': model_hash})
        return self.get_previous_block()['index'] + 1
```

```python
import requests

PINATA_API_KEY = "fd7db91eb025757d72af"
PINATA_SECRET_API_KEY = "1921318a8d1e0c3f0799e9fd05417d978d412e62988c03f7939994d8d30df2d9"

def upload_to_ipfs(file_path):
    url = "https://api.pinata.cloud/pinning/pinFileToIPFS"
    headers = {
        "pinata_api_key": PINATA_API_KEY,
        "pinata_secret_api_key": PINATA_SECRET_API_KEY
    }
    with open(file_path, "rb") as file:
        response = requests.post(url, files={"file": file}, headers=headers)
    ipfs_hash = response.json()["IpfsHash"]
    return ipfs_hash

file_path = "/content/global_model.json"  # Change this to your actual file
ipfs_hash = upload_to_ipfs(file_path)
print("Stored on IPFS with hash:", ipfs_hash)
```

```
Stored on IPFS with hash: QmU7heh2eVwgfNyaYZuAst744cuyQN1nQMpKL2a1VnjFNr
```

Fig. 3.14 Storing on IPFS

```python
from web3 import Web3

# Use your Infura Sepolia RPC URL
infura_url = "https://sepolia.infura.io/v3/e89eff990e004123b9d421a585f34a4b"
web3 = Web3(Web3.HTTPProvider(infura_url))

if web3.is_connected():
    print("✅ Connected to Ethereum Sepolia Testnet")
else:
    print("❌ Connection Failed")
```

```
✅ Connected to Ethereum Sepolia Testnet
```

Fig. 3.15 Connecting sepolia testnet

```python
# Your Ethereum account (Metamask address)
account = "0x953C0A043188C1E9C0AA337FD8B657202E779584"

# Your private key (DO NOT SHARE, use in a secure way)
private_key = "clevertinytransfermusicinfantnicefestivalcapablesheriffdepartchapterempty"

contract_abi = [
    {
        "inputs": [{"internalType": "string", "name": "_modelHash", "type": "string"}],
        "name": "submitUpdate",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [{"internalType": "uint256", "name": "index", "type": "uint256"}],
        "name": "getUpdate",
        "outputs": [
            {"internalType": "address", "name": "", "type": "address"},
            {"internalType": "string", "name": "", "type": "string"},
            {"internalType": "uint256", "name": "", "type": "uint256"}
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "getTotalUpdates",
        "outputs": [{"internalType": "uint256", "name": "", "type": "uint256"}],
        "stateMutability": "view",
        "type": "function"
    }
]

from web3 import Web3

# Convert contract address to checksum format
contract_address = Web3.to_checksum_address("0x7888d98a7533f6e4e7881cd891ae8a653f434e4b")

# Connect to your contract
contract = web3.eth.contract(address=contract_address, abi=contract_abi)
```

Fig. 3.16 Deploying smart contract

```python
import os
import requests
from web3 import Web3

# Pinata API Keys (keep these secure!)
PINATA_API_KEY = "fd7db91eb025757d72af"
PINATA_SECRET_API_KEY = "1921318a8d1e0c3f0799e9fd05417d978d412e62988c03f7939994d8d30df2d9"

# Ethereum Setup
infura_url = "https://sepolia.infura.io/v3/e89eff990e004123b9d421a585f34a4b"
web3 = Web3(Web3.HTTPProvider(infura_url))
account = Web3.to_checksum_address("0x953C0A0431B8C1E9C0AA337FD8B657202E779584")
private_key = "59a131e909a55e4fdeaf20cfccfc33e7959a4f3df0afa2cd456d8740a50728b5"

# Smart contract setup
contract_address = Web3.to_checksum_address("0x7888d98a7533f6e4e7881cd891ae8a653f434e4b")
contract_abi = [
    {
        "inputs": [{"internalType": "string", "name": "_modelHash", "type": "string"}],
        "name": "submitUpdate", "outputs": [], "stateMutability": "nonpayable", "type": "function"
    },
    {
        "inputs": [{"internalType": "uint256", "name": "index", "type": "uint256"}],
        "name": "getUpdate", "outputs": [
            {"internalType": "address", "name": ""}, {"internalType": "string", "name": ""},
            {"internalType": "uint256", "name": ""}
        ], "stateMutability": "view", "type": "function"
    },
    {
        "inputs": [], "name": "getTotalUpdates",
        "outputs": [{"internalType": "uint256", "name": ""}], "stateMutability": "view", "type": "function"
    }
]
contract = web3.eth.contract(address=contract_address, abi=contract_abi)

def upload_to_ipfs(file_path):
    url = "https://api.pinata.cloud/pinning/pinFileToIPFS"
    headers = {
        "pinata_api_key": PINATA_API_KEY,
        "pinata_secret_api_key": PINATA_SECRET_API_KEY
    }
    with open(file_path, "rb") as file:
        response = requests.post(url, files={"file": file}, headers=headers)
        return response.json()["IpfsHash"]

def store_hash_on_chain(ipfs_hash):
    tx = contract.functions.submitUpdate(ipfs_hash).build_transaction({
        "from": account,
        "nonce": web3.eth.get_transaction_count(account),
        "gas": 2000000,
        "gasPrice": web3.to_wei("10", "gwei")
    })
    signed_tx = web3.eth.account.sign_transaction(tx, private_key)
    tx_hash = web3.eth.send_raw_transaction(signed_tx.raw_transaction)
    print("✅ Transaction sent:", web3.to_hex(tx_hash))

models_folder = "/content/drive/MyDrive/block"
for filename in os.listdir(models_folder):
    if filename.endswith(".h5"):
        model_path = os.path.join(models_folder, filename)
        print(f"\nUploading model: {filename}")
        ipfs_hash = upload_to_ipfs(model_path)
        print(f"IPFS Hash: {ipfs_hash}")
        store_hash_on_chain(ipfs_hash)
```

```
Uploading model: client1_weights.h5
IPFS Hash: QmakPTdF7yRHTNnLfZ4tugznvEJ7LhsjJGeYv6jQDfXZtP
✅ Transaction sent: 0x21eb01a14b35726f26f25f16e467eac9503ec9fe447d52f583a5b16737b493dd

Uploading model: client2_weights.h5
IPFS Hash: Qmc4FDpDGtAKNNnsA754hXrmhiJqcsJtbZtQd2dsM24L99
✅ Transaction sent: 0xd08d09acc8638a95cc42a998e77a46c28f2ca95e8f09f78d0a809980c5f70533

Uploading model: client3_weights.h5
IPFS Hash: QmZ1qbnazE6rdVGKqMZa1FaTZDLQUiPyRePCTLTNwZ3kmB
✅ Transaction sent: 0x9d239d091c25b7f98fd50d4f0d66330fda1fb179cdf541b6f37732e84aa0fac7

Uploading model: client4_weights.h5
IPFS Hash: QmRN3U7pYYYDrx4MKPiiqtTeiwVAM4StRbFmeFQVV3rfDk
✅ Transaction sent: 0x6f8207d93394b1cb00bae688b19fb6209b816954c5f7a57fad7a532a96106e09
```

Fig. 3.17 Making transaction on Ethereum blockchain

```python
import os
import requests
import tempfile
import numpy as np
import tensorflow as tf
from web3 import Web3
from keras.models import load_model
from keras.saving import save_model
from scipy.optimize import minimize

infura_url = "https://sepolia.infura.io/v3/e89eff990e004123b9d421a585f34a4b"
web3 = Web3(Web3.HTTPProvider(infura_url))

contract_address = Web3.to_checksum_address("0x7888d98a7533f6e4e7881cd891ae8a653f434e4b")
contract_abi = [
    {"inputs": [], "name": "getTotalUpdates", "outputs": [{"internalType": "uint256", "name": "", "type": "uint256"}], "stateMutability": "view", "type": "function"},
    {"inputs": [{"internalType": "uint256", "name": "index", "type": "uint256"}],
     "name": "getUpdate",
     "outputs": [
         {"internalType": "address", "name": "", "type": "address"},
         {"internalType": "string", "name": "", "type": "string"},
         {"internalType": "uint256", "name": "", "type": "uint256"}
     ],
     "stateMutability": "view", "type": "function"}
]
contract = web3.eth.contract(address=contract_address, abi=contract_abi)
def download_ipfs_model(ipfs_hash):
    url = f"https://gateway.pinata.cloud/ipfs/{ipfs_hash}"
    response = requests.get(url)
    if response.status_code != 200:
        raise Exception("IPFS fetch failed")

    temp_file = tempfile.NamedTemporaryFile(delete=False, suffix=".h5")
    temp_file.write(response.content)
    temp_file.close()
    return temp_file.name

def fetch_valid_models():
    valid_model_paths = []
    total = contract.functions.getTotalUpdates().call()
    for i in range(total):
        try:
            _, ipfs_hash, _ = contract.functions.getUpdate(i).call()
            if ipfs_hash.endswith('.json'): continue
            path = download_ipfs_model(ipfs_hash)
            _ = load_model(path, compile=False)
            print(f"✅ Model {i} loaded: {ipfs_hash}")
            valid_model_paths.append(path)
        except Exception as e:
            print(f"❌ Skipping model {i}: {e}")
    return valid_model_paths
```

```python
def geometric_median(points, eps=1e-5):
    def aggregate(x):
        return sum(np.linalg.norm(x - p) for p in points)

    x0 = np.mean(points, axis=0)
    res = minimize(aggregate, x0, method='COBYLA')
    return res.x

def average_weights(model_paths):
    loaded_models = [load_model(p, compile=False) for p in model_paths]
    base_model = loaded_models[0]
    base_weights = base_model.get_weights()
    total_layers = len(base_weights)

    averaged_weights = []
    for i in range(total_layers):
        layer_weights = [m.get_weights()[i] for m in loaded_models if len(m.get_weights()) > i and m.get_weights()[i].shape == base_weights[i].shape]

        if len(layer_weights) == 0:
            print(f"Skipping layer {i}: shape mismatch in all models.")
            averaged_weights.append(base_weights[i])
        elif len(layer_weights[0].shape) > 1:

            layer_median = np.median(layer_weights, axis=0)
            averaged_weights.append(layer_median)
        else:

            median = geometric_median(np.array(layer_weights))
            averaged_weights.append(median)

    return averaged_weights, base_model

model_paths = fetch_valid_models()
averaged_weights, base_model = average_weights(model_paths)

base_model.set_weights(averaged_weights)
final_path = "/content/global_model_geomedian.h5"
base_model.save(final_path)
print(f"✅ Final global model saved with geometric median averaging: {final_path}")
```

```
✅ Model 1 loaded: QmakPTdF7yRHTNnLfZ4tugznvEJ7LhsjJGeYv6jQDfXZtP
✅ Model 2 loaded: Qmc4FDpDGtAKNNnsA754hXrmhiJqcsJtbZtQd2dsM24L99
✅ Model 3 loaded: QmZ1qbnazE6rdVGKqMZa1FaTZDLQUiPyRePCTLTNwZ3kmB
✅ Model 4 loaded: QmRN3U7pYYYDrx4MKPiiqtTeiwVAM4StRbFmeFQVV3rfDk
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`.
✅ Final global model saved with geometric median averaging: /content/global_model_geomedian.h5
```

Fig. 3.18 Updating the global model

```python
import pandas as pd
import numpy as np
import joblib
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf


test_data = pd.read_csv('/content/drive/MyDrive/testdataset.csv')

y_test = test_data['label']

X_test_raw = test_data.drop(columns=['label'])

X_test_numeric = X_test_raw.select_dtypes(include=[np.number])

scaler = joblib.load('/content/drive/MyDrive/keras/client4_results/scaler_client1.pkl')
model = tf.keras.models.load_model('/content/drive/MyDrive/final_global_model.h5')

expected_columns = scaler.feature_names_in_

common_columns = list(set(expected_columns).intersection(X_test_numeric.columns))

X_test = X_test_numeric[common_columns]
X_test = X_test.reindex(columns=expected_columns, fill_value=0)

X_test_scaled = scaler.transform(X_test)


y_pred_probs = model.predict(X_test_scaled)
y_pred = (y_pred_probs > 0.5).astype(int).flatten()

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=4))
```

```
WARNING:absl:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
186/186 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Confusion Matrix:
[[2044   31]
 [ 263 3587]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8860    0.9851    0.9329      2075
           1     0.9914    0.9317    0.9606      3850

    accuracy                         0.9504      5925
   macro avg     0.9387    0.9584    0.9468      5925
weighted avg     0.9545    0.9504    0.9509      5925
```

Fig. 3.19 Global model testing

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 MODEL TESTING

The proposed work focused on three models such as DNN, CNN and LR The implementation results are depicted in Fig 4.1, Fig. 4.2, Fig 4.3 and Fig 4.4. Among these three models, DNN achieves high accuracy. Therefore, blockchain technology was integrated into the DNN to ensure secure, transparent, and tamper-proof data and model operations.

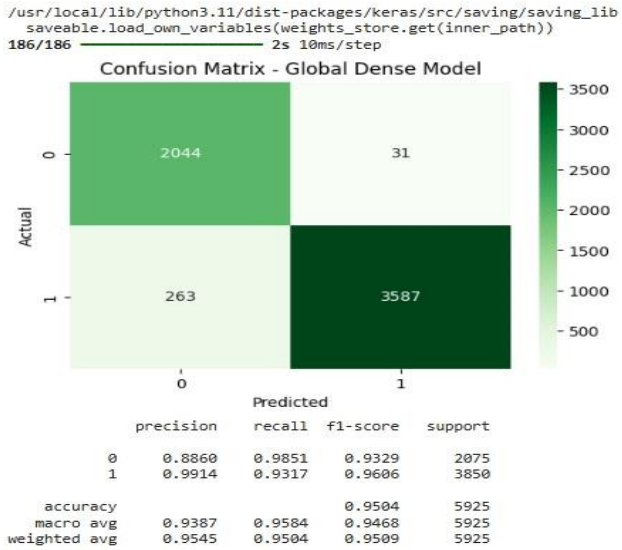


Fig 4.1 DNN Accuracy for Global Model

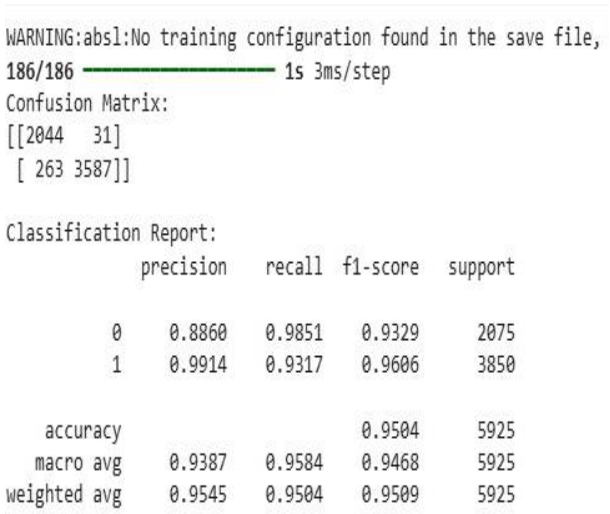


Fig 4.2 Blockchain Integrated DNN Accuracy for Global Model

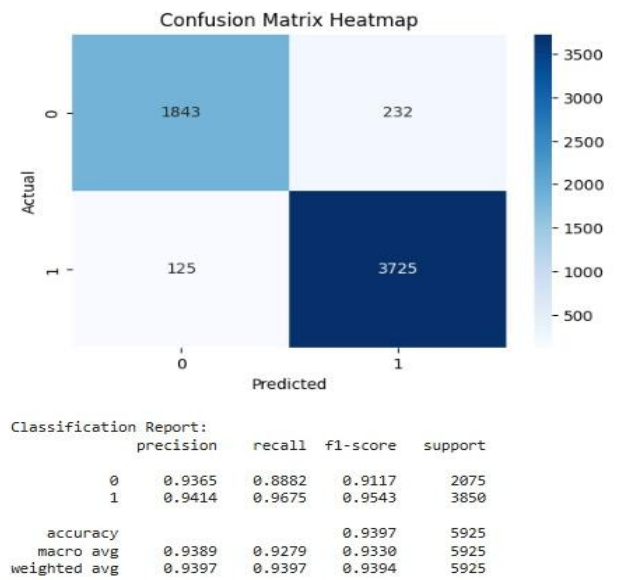


Fig 4.3 CNN Accuracy for Global Model

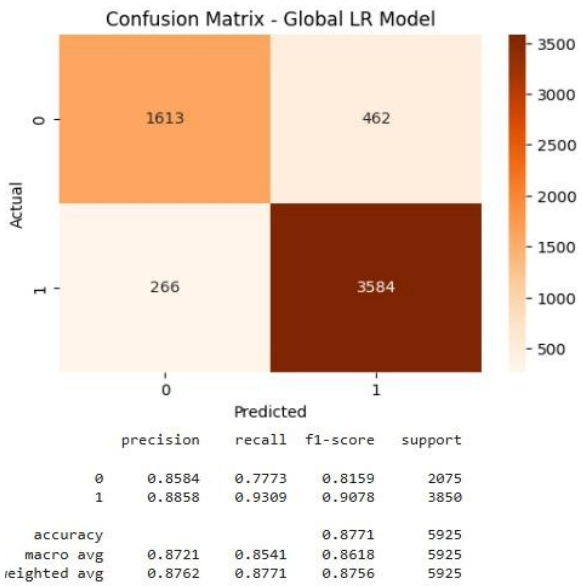
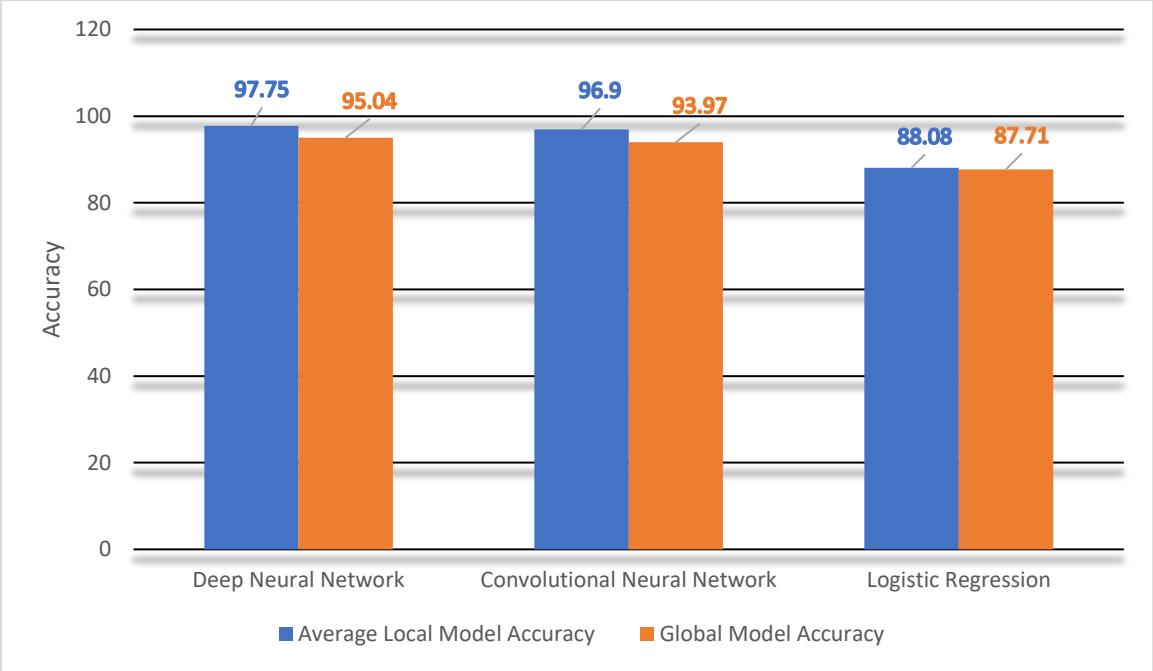


Fig 4.4 LR Accuracy for Global Model

4.2 MODEL COMPARISON



Model Name	Local Model 1 Accuracy	Local Model 2 Accuracy	Local Model 3 Accuracy	Local Model 4 Accuracy	Average Local Model Accuracy	Global Model Accuracy
DNN	97.81	98.19	97.99	97.03	97.75	95.04
CNN	97.43	97.18	97.06	95.95	96.90	93.97
LR	88.19	87.88	88.24	88.02	88.08	87.71

Fig 4.5 Model accuracy comparison

In Fig 4.5 presents a comparison of model accuracies across local and global environments for DNN, CNN, and LR models. The Deep Neural Network (DNN) achieved the highest average local accuracy of 97.75% and a corresponding global accuracy of 95.04%, indicating strong generalization. This comparison highlights DNN's superior performance in federated learning settings. This hybrid DNN-blockchain framework enhances the system's reliability, making it suitable for applications requiring robust security.

CHAPTER 5
CONCLUSION AND FUTURE PLAN

5.1 CONCLUSION

In this project, A secure and efficient attack detection system using Federated Learning with Deep Neural Networks, CNN, and Logistic Regression were implemented and multiple local models were trained on separate datasets and aggregated them to form a global model. Blockchain is integrated to store and retrieve model updates securely using IPFS and a smart contract. Additionally, a web-based UI was developed for uploading CSV files, predicting attacks, and securely transmitting only the normal data with features like encryption and user authentication.

5.2 FUTURE PLANS

- A custom consensus mechanism can be implemented in the blockchain to enhance security and efficiency over Proof of Work.
- Aim to reduce model size and latency for deployment in real-time IoT edge environments using model pruning and quantization.
- The current system can be extended to multi-class attack classification instead of binary classification (normal vs. attack).
- Plan to add role-based access control to the user authentication module to ensure secure decryption and data transmission.
- Enhancing the UI to visualize model training progress, real-time predictions, and blockchain transactions is under consideration.
- Integration with cloud services like AWS or GCP is planned for scalable storage and deployment of encrypted data.

REFERENCES

[1] Archana D. Wankhade, Kishor P. Wagh - Implementation of Attack Detection and Mitigation for Securing Cloud-Based IoT Network Journal of Theoretical and Applied Information Technology, Vol. 102, No. 9, 2024

[2] Maryam Anwer, Muhammad Umer Farooq, Shariq Mahmood Khan, Waseemullah - Attack Detection in IoT Using Machine Learning Engineering, Technology & Applied Science Research, Vol. 11, No. 3, 2021

[3] Zhiqiang Feng - IoT Data Sharing Technology Based on Blockchain and Federated Learning Algorithms Intelligent Systems with Applications, Vol. 22, 2024

[4] Hui Chen, Zhendong Wang, Shuxin Yang, Xiao Luo, Daojing He, Sammy Chan - Intrusion Detection Using Synaptic Intelligent Convolutional Neural Networks for Dynamic IoT Environments Alexandria Engineering Journal, Vol. 111, 2025

[5] Bhukya Madhu, M. Venu Gopala Chari, Ramdas Vankdothu, Arun Kumar Silivery, Veerender Aerranagula - Intrusion Detection Models for IoT Networks via Deep Learning Approaches Measurement: Sensors, Vol. 25, 2023

[6] Mahmudul Hasan, Md. Milon Islam, Md Ishrak Islam Zarif, M.M.A. Hashem - Attack and Anomaly Detection in IoT Sensors Using Machine Learning Approaches Internet of Things, Vol. 7, 2019

[7] Khawla Shalabi, Qasem Abu Al-Haija, Mustafa Al-Fayoumi - A Blockchain-Based Intrusion Detection/Prevention Systems in IoT Network: A Systematic Review Procedia Computer Science, Vol. 236, 2024

[8] Sowmya T, Dr. Mary Anita E A - A Novel Stable Feature Selection Algorithm for Machine Learning-Based Intrusion Detection System Procedia Computer Science, Vol. 252, 2025

[9] C. Malathi, I. Naga Padmaja - Identification of Cyber Attacks Using Machine Learning in Smart IoT Networks Materials Today: Proceedings, Vol. 80, 2023

CHAPTER 6
APPENDIX - BASE PAPER

Journal of Parallel and Distributed Computing 172 (2023) 69–83

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

journal homepage: www.elsevier.com/locate/jpdc

A blockchain-orchestrated deep learning approach for secure data transmission in IoT-enabled healthcare system

Prabhat Kumar [a,*], Randhir Kumar [b], Govind P. Gupta [c], Rakesh Tripathi [c], Alireza Jolfaei [d], A.K.M. Najmul Islam [e]

[a] Department of Software Engineering, LUT School of Engineering Science, LUT University, 53850 Lappeenranta, Finland
[b] Department of Computer Science and Engineering, SRM University AP, AP 522240, India
[c] National Institute of Technology, Raipur, India
[d] College of Science and Engineering, Flinders University, Adelaide, Australia
[e] LUT School of Engineering Science, LUT University, 53850 Lappeenranta, Finland

ARTICLE INFO

Article history:
Received 26 May 2022
Received in revised form 18 August 2022
Accepted 6 October 2022
Available online 17 October 2022

Keywords:
Blockchain
Deep learning
Healthcare systems
Internet of Things
Zero Knowledge Proof

ABSTRACT

The integration of the Internet of Things (IoT) with traditional healthcare systems has improved quality of healthcare services. However, the wearable devices and sensors used in Healthcare System (HS) continuously monitor and transmit data to the nearby devices or servers using an unsecured open channel. This connectivity between IoT devices and servers improves operational efficiency, but it also gives a lot of room for attackers to launch various cyber-attacks that can put patients under critical surveillance in jeopardy. In this article, a Blockchain-orchestrated Deep learning approach for Secure Data Transmission in IoT-enabled healthcare system hereafter referred to as "BDSDT" is designed. Specifically, first a novel scalable blockchain architecture is proposed to ensure data integrity and secure data transmission by leveraging Zero Knowledge Proof (ZKP) mechanism. Then, BDSDT integrates with the off-chain storage InterPlanetary File System (IPFS) to address difficulties with data storage costs and with an Ethereum smart contract to address data security issues. The authenticated data is further used to design a deep learning architecture to detect intrusion in HS network. The latter combines Deep Sparse AutoEncoder (DSAE) with Bidirectional Long Short-Term Memory (BiLSTM) to design an effective intrusion detection system. Experiments on two public data sources (CICIDS-2017 and ToN-IoT) reveal that the proposed BDSDT outperformed state-of-the-arts in both non-blockchain and blockchain settings and have obtained accuracy close to 99% using both datasets.

1. Introduction

The Internet of Things (IoT) has revolutionized the traditional healthcare systems into intelligent system by allowing remote access and continuous monitoring of patient data. For example, the wearable, and various IoT-based medical devices are used to collect real-time physiological data from patients, such as body temperature, blood glucose levels, and other pertinent data [5]. Indeed, the IoT can help the healthcare industry in terms of fast diagnosis and effective clinical treatment remotely [1]. However, the IoT nodes in an IoT-enabled Healthcare System (HS) are connected round the clock through an open insecure public channel making the entire network vulnerable to eavesdropping, data manipulation and other security related issues [30]. More specifically, security concern arises due to debilitated security considerations in communication protocols, and through the fast advancement in hacking techniques, wherein attackers attempt to compromise the availability, integrity, and reliability of IoT data and devices [31]. These attacks are not only carried out on healthcare network components using malware or malicious software but can be launched on actual IoT devices with aim to temper its functionality [17]. Privacy issues in IoT network mean compromising sensitive, private data by performing passive and active (e.g., data poisoning attack) attacks. The passive attack includes sniffing substantial public data content, while active attack aims to obtain access, infer and/or alter private data [16]. This can lead to poor resource utilization by the participating devices in the network, as well as disruptions in their normal communication flow.

To handle the aforementioned challenges, blockchain and Deep Learning (DL) can be integrated to provide a prominent solution in

* Corresponding author.
E-mail addresses: prabhat.kumar@lut.fi (P. Kumar), randhir.honeywell@ieee.org (R. Kumar), gpgupta.it@nitrr.ac.in (G.P. Gupta), rtripathi.it@nitrr.ac.in (R. Tripathi), alireza.jolfaei@flinders.edu.au (A. Jolfaei), najmul.islam@lut.fi (A.K.M. Najmul Islam).

https://doi.org/10.1016/j.jpdc.2022.10.002

IoT-enabled HS. First blockchain consists of a block (a type of digital information) and chain (a kind of open database). As soon as the data is recorded in the chain of immutable blocks, it becomes impossible to change it (i.e., prevents data poisoning attacks) [3]. Owing to the immutability and decentralized system exhibited by blockchain, smart contracts can further foster the confidence between the parties involved in data transmission by self-enforcing and self-executing the terms of the agreement. Moreover, the integrity of the distributed data recorded in the blockchain is supported by the consensus procedures [25]. As a result, the patient's medical information can be considered to be secure and trustworthy during transmission in IoT-enabled HS. On the other hand, to mitigate the attacks of HS, DL-based Intrusion Detection System (IDS) is most commonly used to detect abnormal network behaviors over machine learning methods [28]. However, most of the IDS designed in the literature uses data directly from the network and have poor performance with specific attack types in terms of detection rate and false alarm rate [20,14]. Additionally, scalability is one of the additional significant challenging issues in IoT-enabled HS. The rationale is that as there are more IoT devices, more storage space will be needed to accommodate the exponential growth in data created [13].

1.1. Attack model

The widely accepted "Dolev-Yao threat (DY) model" is taken into consideration in designing BDSDT framework [8]. The end entities participating in the communication (IoT devices and Edge servers) are not completely trusted according to this concept. Furthermore, it is presumed that they communicate through an unsecure, open, and public channel. The trusted verifier ($\mathcal{V}$) is completely trusted, whereas the IoT and edge servers are considered semi-trusted. An attacker, say $\mathcal{A}$, can modify the contents of message which is transmitted between two entities. Moreover, $\mathcal{A}$ can get the sensitive information, and can perform data poisoning attack. As a result, it is critical to verify the participating entities prior to secret communication in order to prevent them from gaining access to the data. Therefore, it becomes important to consider this model to analyze the security of IoT data [1].

1.2. Key contribution

We develop and implement the BDSDT, which combines blockchain and deep learning approaches, in order to address the aforementioned issues. The following are this paper's main contributions:

- A new secure data transmission mechanism named BDSDT is proposed by combining blockchain and deep learning approach.
- A blockchain-enabled security architecture is proposed. The underlying scheme registers, verifies IoT devices using zero knowledge proof. This approach validates data records, presents a generalized way to transmit healthcare data in the HS network and prevents data from poisoning attacks.
- A DL-enabled security architecture is proposed. This includes a Deep Sparse AutoEncoder (DSAE) approach to encode the initial data into a new format (i.e., feature extraction). Further, using encoded data, an intrusion detection system based on Bidirectional Long Short-Term Memory (BiLSTM) technique is designed.
- We use an InterPlanetary File System (IPFS) as a distributed file system to distribute and store the entire IoT data due to the limited storage capacity available on each blockchain node. As a result, BDSDT can handle enormous amounts of IoT data and scales well in this direction.
- The performance of the proposed BDSDT framework is accessed on two network datasets, ToN-IoT and CICIDS-2017. Finally, results are compared with several recent approaches in both blockchain and non-blockchain environment.

The remainder of this article is structured as follows. The relative background is covered in Section 2. We describe our suggested framework and its useful elements in Section 3. Analysis of security and privacy issues is presented in Section 4. The effectiveness of our framework in terms of numerical results is evaluated in Section 5. This article is concluded with the future work in Section 6.

2. Background and related work

This section presents relevant work in three different areas particularly related with privacy-preservation (i.e., deep learning, blockchain and smart contracts), IDS and state-of-the-arts in non-blockchain and blockchain environment.

2.1. Privacy-preserving methods

The primary goal of privacy-preserving strategies is to use a preprocessing step before running data mining algorithms (such as IDS/IPS) to prevent information leakage yet maintaining crucial details [3], [2]. These techniques are categorized into six types; authentication-based [17], Differential Privacy (DP)-driven [12], perturbation-driven [15], encryption-driven [16], deep learning-driven [2] and blockchain and smart contracts based mechanisms [3], [17].

The blockchain-based privacy-preserving technique applies the principle of blockchain technology. The underlying approach is a decentralized, distributed ledger of data blocks created by cryptographic techniques. In blockchain, the data block consist a series of transactions accepted by the majority of the participants in the network. The series of blocks are linked or chained together in chronological order with the previous hash of the subsequent block for unique identification. The block in blockchain can be found through the corresponding hash value of the block [30]. Essentially, these series of linked block must be disseminated and replicated over the peer-to-peer network. The distributed consensus algorithm is applied on transactions ordering to place into a new block and disseminate consensus (proof) over the network. The block creation process is carried out in the network by distinguished nodes (full nodes) namely miners (i.e., by applying distributed consensus algorithm) [17]. The traditional blockchain uses proof-of-work (PoW) consensus for data authentication and for unique block hash creation. However, this approach is computationally more intensive, consumes huge amount of resources, and violated to malicious behavior of the miners (51%) attack [3].

In traditional blockchain system, data privacy can be compromised due to distributed consensus approach i.e., PoW malicious activities (51% attack). The smart contract enabled consensus mechanism can resolve this issue owing to its existence across the decentralized peers of network. The smart contracts is a programming code that hat executes on the infrastructure of the blockchain. It automatically allows the transparent execution of predefined terms of an agreement, without the intervention of a trusted third party [13,6]. It accepts transactions as an input and performs operation on data and provides the desired output [30].

2.2. Intrusion detection techniques

An Intrusion Detection System (IDS) can be a software or hardware, that aims to automate the attack detection process [1]. In order to mitigate the threat consequences, IDS tracks systems operation or actions, identify when attack occurs and then triggers

warning. Further, based on attack detection approach, IDS can be of signature- or anomaly-based. In signature-based IDS (SIDS), the patterns of observed events are compared against a database of already known attacks to identify threats. Several issues exist in SIDS such as, low Detection Rate (DR), due to its ability to detect only already-known attacks [16]. Whereas, in anomaly-based IDS (AIDS), the systems normal behavior is modeled using observed sequence of incoming events in attack free mode [17].

In the literature most of the attack detection process are based on traditional machine learning and data mining techniques, rules-based models, and statistical technique [18,9]. However, due to the overlap between benign and abnormal events, these techniques also suffer from low Detection Rate (DR) and high False Alarm Rate (FAR). In this research, we design an effective IDS for IoT network, that uses DL techniques to resolve the drawbacks of DR and FAR. Due to the fact that, the DL models can automatically analyze incoming events to proficiently detect abnormal operations [19]. A DL technique is efficient, as it can handle high dimension of incoming events and can evaluate the latent structure from unlabeled data [17]. DL supports both discriminative and generative designs. In contrast to the discriminative architecture, which calculates subsequent class distributions based on seen data, the generative architecture estimates joint probability distributions for its classes from observed data. The BiLSTM is a useful generative technique that has good capability to learn time series data of IoT network and hence is used in this paper [3].

2.3. Security in non-blockchain environment

Qiao et al. [23] designed an IDS based on ML-based approaches. The underlying scheme first provides privacy using two linear projected transformations techniques i.e., Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). The extracted features was used by k-nearest neighbors algorithm (k-NN) to detect and report intrusion. The approach was tested using UNSW-NB15 dataset and was proven to perform well in terms of DR and FAR. However, UNSW-NB15 dataset is outdated dataset and do not include the specific features of IoT/IoT [4]. Hasan et al. [14] compared the performance of various ML techniques for attack detection in IoT sensors in IoT sites. The IDS based on Random Forest (RF) outperformed and obtained 99.04% accuracy over other techniques. However, one of the important evaluation metrics, i.e., FAR was not considered in this experiment. Moreover, RF constructs several Decision Trees (DTs), which makes it less intuitive, requires more computational resources and therefore, it may be impractical to apply RF in specific online sites that demands large training datasets. Ghulam et al. [22] designed an IDS, that used stacked AutoEncoder (AE) to extract features and Deep Neural Network (DNN) for attack detection in IoT network. The model was evaluated using three different data sources, AWID, KDDCup99 and NSL-KDD datasets. However, all above datasets are outdated and do not contain either IoT network traffic or IoT telemetry data.

Keshk et al. [15] designed a privacy-preserving-based IDS for CPS. This model used projection-based transformation technique i.e., Independent Component Analysis (ICA) to protect sensitive information and compared IDS performance using different ML techniques. The model was evaluated using CPS power system dataset and outperformed using DT technique. Alsaedi et al. [4] proposed an updated, and representative IoT/IoT dataset, i.e., ToN-IoT, and evaluated its performance using various ML and DL algorithms. The proposed model outperformed using Classification and Regression Trees (CART), however, model obtained very low accuracy, precision, recall and F1 score in both binary (i.e., 88%, 90%, 88%, 88%) and multi-class (i.e., 77%, 77%, 77%, 75%) attack detection approaches. Keshk et al. [16] designed a PPAD-CPS, a privacy-preserving-based attack detection framework. A data preprocessing and Gaussian mixture technique was used to protect CPS data. The encoded data was used by Kalman filter to detect attacks. The framework was evaluated using power system and UNSW-NB15 datasets and obtained accuracy of 97.27% and 93.70% using both datasets, respectively. However, when operating IDS, the proposed model cannot guarantee integrity of the data against poisoning and inference attacks.

2.4. Security in blockchain environment

Various researches have been carried out to demonstrate confidentiality and integrity of IoT data by integrating blockchain and ML/DL in IoT/IoT network. For instance, Zhao et al. [30] reviewed various ways of integrating blockchain with CPS. The authors presented the possibility of improving the reputation of IDS by using blockchain techniques. In [1] authors surveyed the security vulnerabilities in IoT/IoT architecture and significance of using blockchain-based solutions to secure IoT network. Qiu et al. [24] created a consortium blockchain-based spectrum trading system for unmanned aerial vehicles (UAVs). The suggested architecture solves two critical security and privacy vulnerabilities coming from malevolent UAVs' unlawful spectrum exploitation and privacy leakages caused by key sharing with centralized third parties.

Rathore et al. [25] presented a Software Defined Networking (SDN), fog, and mobile edge computing based decentralized security architecture that combines blockchain with DL techniques in IoT. This model used memory-hardened PoW to authenticate IoT, and edge devices and to avoid single point of failure. The performance was evaluated using NSL-KDD dataset and achieved 91% accuracy. However, NSL-KDD dataset is considered to be outdated and does not contain threats of a IoT. Liang et al. [20] designed a security framework, SESS, that uses multi-agent system based on DL and blockchain to secure IoT data. The performance was measured using NSL-KDD dataset and DNN-based IDS obtained 98% accuracy. However, this model lacked specific blockchain implementation and moreover, used outdated dataset for evaluation. Alkadi et al. [3] proposed a Deep Blockchain Framework (DBF) that integrates DL with blockchain to provide data security and privacy in IoT network. This framework was evaluated using UNSW-NB15 and BoT-IoT datasets and has obtained better detection rate. However, this framework lacks specific blockchain implementation and results. Keshk et al. [17] proposed a privacy-based IDS to protect and secure data of smart power network. This framework provides privacy using an ePoW and Variational AutoEncoder (VAE) technique and Long Short Term Memory (LSTM) to detect attack. The underlying framework was evaluated using Power System and UNSW-NB15 datasets and has shown good results on both datasets. However, authors have not discussed block creation and access time taken by their ePoW by varying transactions in smart power network. Ferrag et al. [10] designed DeliveryCoin to provide security and privacy for drone-delivered services. In privacy scheme model used Strong Diffie–Hellman combined with hash functions and short signatures. In security schemes, various ML and DL techniques were used. The model outperformed using RNN technique and obtained 98.71% highest accuracy. Weng et al. [29] designed DeepChain, a framework that ensures privacy of local gradients using DL and blockchain. The IDS was designed using Convolution Neural Network (CNN) and obtained highest accuracy of 97.32% using MNIST dataset.

Singh et al. [28] designed an security system using blockchain and DL-based deep Boltzmann machine to protect SDN-based industrial applications. The blockchain concept was used to register and validate SDN switches using a voting-based blockchain consensus mechanism. This framework was evaluated using KDD-CUP'99 dataset and obtained accuracy of 88.59%. Ferrag et al. [11] designed DeepCoin, a DL and blockchain based energy framework for smart

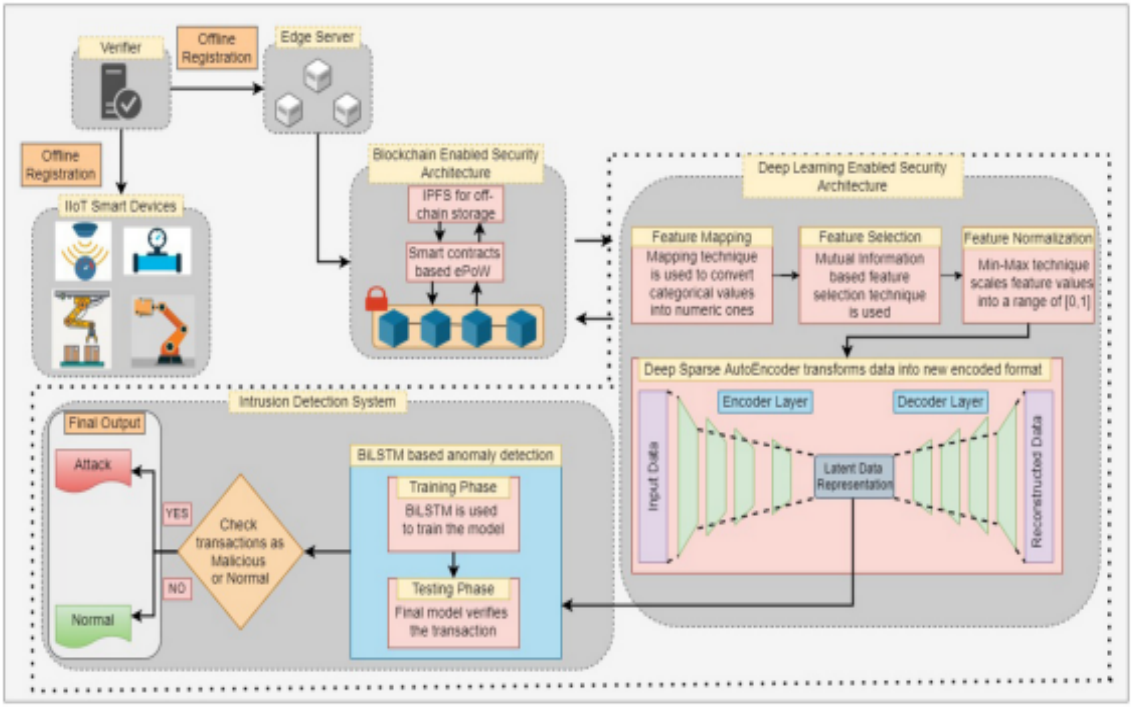


Fig. 1. Proposed blockchain-orchestrated deep learning approach for secure data transmission in IoT-enabled healthcare system.

grids. It uses Byzantine fault tolerance consensus mechanism. The proposed IDS was designed based on truncated BackPropagation Through Time (BPTT)-based Recurrent Neural Network (RNN) algorithm, and was evaluated using power system, CICIDS-2017, and BoT-IoT datasets, and achieved accuracy of 96.52%, 98.23% and 98.20%, respectively. However, they have not performed security analysis of proposed framework. Derhab et al. [7] designed a security architecture using blockchain and SDN for Industrial Control Systems (ICS). The proposed model obtained 96.73% and 91.07% accuracy under binary and multi-class detection task. However, the proposed framework lacks blockchain specific analysis.

3. The proposed BDSDT for IoT-enabled healthcare system

3.1. Working of proposed BDSDT

The working of proposed BDSDT framework is shown in Fig. 1, which depicts the communication occurring among different parties. This framework includes various communicating entities, such as IoT devices ($\mathcal{S}_{\mathcal{U}_i}$), edge servers ($\mathcal{EDGE}$), verifier ($\mathcal{V}$). The responsibility of $\mathcal{V}$ is to register all participating entities prior to their placement in the network. The $\mathcal{S}_{\mathcal{U}}$ have limited resources, and computing power. This includes pressure, water quality, proximity sensors and so on. They are used to measure equipment leakage, to identify non-regulatory water quality and to sense the presence of objects, respectively. Each $\mathcal{S}_{\mathcal{U}_i}$ is connected with the Internet, and can also send and receive the information using the Internet. $\mathcal{EDGE}$ includes industrial computer, data analysis server, and so on. One or more $\mathcal{S}_{\mathcal{U}_i}$ are connected to $\mathcal{EDGE}$ to perform mining operations. The systematic architecture of the BDSDT includes two main components; (*i*) blockchain enabled security architecture, (*ii*) deep learning enabled security architecture, as discussed below and illustrated in Fig. 1. The blockchain technology is used to register IoT devices and to provide secure data transmission. Additionally, this design offers insights on network activity and associated performance. Accordingly, this level aids in keeping track of the data source, owner, final destination, alternative routes, security measures, security enabling authority, and protects against data poisoning attacks. In the DL enabled security architecture, a DSAE technique is applied to convert original data into a new format that considerably reduces the dimension of the datasets. The working of these architectures is explained in detail below.

3.2. Blockchain enabled security architecture

In the first level of security, the BDSDT is divided in six distinct phases: 1) Initialization phase, 2) Registration and verification phase, and 3) Encryption and Decryption Phase, 4) Block Creation and validation phase, 5) Data Generation and Block Updation Phase, 6) Consensus Phase. The detailed working of all the phases is given below. The Table 1 shows notation used in the different phase of security and privacy in the proposed model.

3.2.1. Initialization phase

To bootstrap the framework parameters, trusted verifier ($\mathcal{V}$) evaluates this phase. The $\mathcal{V}$ registers the IoT sensor node ($\mathcal{S}_{\mathcal{U}}$) in proposed framework. The steps involved in generation and calculation of framework parameters such as public and private key are discussed below:

Step-1: The verifier ($\mathcal{V}$) chooses an appropriate largest prime value $\mathcal{P}_{\mathcal{N}}$, in elliptic curve with non-singular representation $\mathcal{E}_{\mathcal{P}_{\mathcal{N}}}(a, b)$: $y^2 = x^3 + ax + (b \bmod \mathcal{P}_{\mathcal{N}})$. Next, $\mathcal{V}$ selects additive group $\mathcal{G}_1$ with infinity point $\mathcal{O}$ and multiplicative group $\mathcal{G}_2$ with identity 1 of prime number $\mathcal{P}_{\mathcal{N}}$. It picks $\mathcal{G}$ a random generator for $\mathcal{G}_1$ and e as bilinear mapping $\mathcal{G}_1 \times \mathcal{G}_1 \rightarrow \mathcal{G}_2$ with three different properties [9]:

- Bilinearity: $\forall\ \mathcal{M}, \mathcal{N}, \mathcal{Y} \in \mathcal{G}_1$, $e(\mathcal{M} + \mathcal{N}, \mathcal{Y}) = e(\mathcal{M}, \mathcal{Y})\, e(\mathcal{N}, \mathcal{Y})$ and $e(\mathcal{M}, \mathcal{N} + \mathcal{Y}) = e(\mathcal{M}, \mathcal{N})\, e(\mathcal{M}, \mathcal{O})$. Thus, all $a, b \in \mathcal{Z}_{\mathcal{P}_{\mathcal{N}}} = \{0, 1, 2, \ldots, \mathcal{P}_{\mathcal{N}} - 1\}$, $e(a\mathcal{M}, b\mathcal{N}) = e(\mathcal{M}, \mathcal{N})^{a.b}$.
- Non-Degeneracy: $e(\mathcal{M}, \mathcal{M}) \neq 1_{\mathcal{G}_1}$ for all $\mathcal{M} \in \mathcal{G}_1$ where $1_{\mathcal{G}_1}$ denotes identity of $\mathcal{G}_1$.

Table 1
Notations used in proposed BDSDT.

Symbol	Explanation
$\mathcal{V}$	Verifier
$\mathcal{P}_{\mathcal{N}}$	Large prime number
$\mathcal{S}_{\mathcal{U}}$	IoT Sensor nodes
$\mathcal{G}_1$	Additive Group
$\mathcal{G}_2$	Multiplicative Group
$\mathcal{PR}_{\mathcal{V}\mathcal{K}}$	Verifier Private Key
$\mathcal{PB}_{\mathcal{V}\mathcal{K}}$	Verifier Public Key
$\mathcal{S}_{\mathcal{U}_i}$	i^{th} sensor node
$\mathcal{ID}_{\mathcal{S}_{\mathcal{U}_i}}$	i^{th} sensor node $\mathcal{ID}$
$\mathcal{T}_{\mathcal{S}i}$	Timestamp of i^{th} sensor node
$\mathcal{M}_{\mathcal{S}_{\mathcal{U}_i}}$	Mac Address of i^{th} sensor nodes
$\mathcal{Q}_1, \mathcal{Q}_2$	Question
$\mathcal{A}_1, \mathcal{A}_2$	Answer
$\mathcal{G}$	Generator
$\mathcal{ZKP}$	Zero Knowledge Proof
$\mathcal{MSG}_{\mathcal{S}_{\mathcal{U}_i}}$	Message from i^{th} Sensor node
$\mathcal{PB}_{\mathcal{S}_{\mathcal{U}_i}}$	Public key of i^{th} sensor node
$\mathcal{PR}_{\mathcal{S}_{\mathcal{U}_i}}$	Private key of i^{th} sensor node

- Computability: There exists efficient algorithm that can evaluate the $e(\mathcal{M}, \mathcal{N})$ for all $\mathcal{M}, \mathcal{N} \in \mathcal{G}_1$.

Step-2: Verifier randomly chooses the $\mathcal{PR}_{\mathcal{V}\mathcal{K}}$ (private key) $\in Z_{\mathcal{P}_{\mathcal{N}}}$, $\mathcal{R}_{\mathcal{K}} \in \{0,1\}^{Z_{\mathcal{P}_{\mathcal{N}}}}$ and sets $\mathcal{PR}_{\mathcal{V}\mathcal{K}}$ as a private key. Next, $\mathcal{PB}_{\mathcal{V}\mathcal{K}}$ (public key) is generated using $\mathcal{PB}_{\mathcal{V}\mathcal{K}} = \mathcal{PR}_{\mathcal{V}\mathcal{K}}.\ \mathcal{G}$, where $k.\mathcal{G}$ denotes multiplication points on elliptic curve, here $k \in Z_{\mathcal{P}_{\mathcal{N}}}$ that defines repeated addition points on curve $k\ .\ \mathcal{G} = \mathcal{G} + \mathcal{G} + \mathcal{G}$, k times.

Step-3: Next, $\mathcal{V}$ chooses one-way cryptographic hash function $H(.)$ and publishes the necessary elements $\{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}, a, b, e, k, Z_{\mathcal{P}_{\mathcal{N}}}, \mathcal{P}, Q, \mathcal{R}, \mathcal{PR}_{\mathcal{V}\mathcal{K}}, \mathcal{PB}_{\mathcal{V}\mathcal{K}}, H(.)\}$ for further access.

3.2.2. Registration and verification phase

In the registration phase, IoT sensor node $\mathcal{S}_{\mathcal{U}_i}$ request the verifier $\mathcal{V}$ to join the blockchain network. The $\mathcal{V}$ registers $\mathcal{S}_{\mathcal{U}_i}$ with the following steps given below:

- The provisional key $\mathcal{PK}$ is created by $\mathcal{S}_{\mathcal{U}_i}$ which consists of two major components (i) sensor identity (i.e., model number) ($\mathcal{ID}_{\mathcal{S}_{\mathcal{U}_i}}$) and ($ii$) mac address ($\mathcal{M}_{\mathcal{S}_{\mathcal{U}_i}}$) of the sensor node.
- Once the $\mathcal{PK}$ is generated successfully, timestamp ($\mathcal{T}_{\mathcal{S}i}$) is stored for verification of the $\mathcal{S}_{\mathcal{U}_i}$ registration.
- The $\mathcal{PK}$ is composed of $\mathcal{ID}_{\mathcal{S}_{\mathcal{U}_i}}$ and $\mathcal{M}_{\mathcal{S}_{\mathcal{U}_i}}$ which is sent to verifier $\mathcal{V}$ including the timestamp.
- The verifier receives the $\mathcal{PK}$ and corresponding $\mathcal{ID}_{\mathcal{S}_{\mathcal{U}_i}}$ and $\mathcal{M}_{\mathcal{S}_{\mathcal{U}_i}}$. Further, $\mathcal{M}_{\mathcal{S}_{\mathcal{U}_i}}$ is checked against the blacklisted mac address by verifier. If it found the in the list then request gets immediately terminated. Otherwise the response is given to the $\mathcal{S}_{\mathcal{U}_i}$ that $\mathcal{PK}$ can be successfully processed and the $\mathcal{PB}_{\mathcal{V}\mathcal{K}}$ is provided to $\mathcal{S}_{\mathcal{U}_i}$ for further access.

The $\mathcal{ZKP}$ approach is applied in the BDSDT framework to verify the registered sensor nodes after the $\mathcal{PB}_{\mathcal{V}\mathcal{K}}$ (public key) has been received by the IoT nodes. The primary aim of $\mathcal{ZKP}$ is to authenticate the $\mathcal{S}_{\mathcal{U}_i}$ without revealing any secret information. In this approach prover and verifier based challenge techniques is proposed. In BDSDT, $\mathcal{S}_{\mathcal{U}_i}$ becomes a prover and $\mathcal{V}$ is the verifier entities. The verifier asks certain question to prover and check the response, to identify the correctness of prover. The detailed steps are given below:

- A value $\mathcal{W}$ is calculated with large prime number $\mathcal{R}$ and generator $\mathcal{L}$. The prover ($\mathcal{S}_{\mathcal{U}_i}$) has to prove that timestamp ($\mathcal{T}_{\mathcal{S}i}$) is known to it, such that $\mathcal{W} = \mathcal{L}^{\mathcal{T}_{\mathcal{S}i}} \bmod \mathcal{R}$, without revealing the timestamp ($\mathcal{T}_{\mathcal{S}i}$). Here, knowing $\mathcal{T}_{\mathcal{S}i}$ denotes the proof of identity and it must be known to $\mathcal{S}_{\mathcal{U}_i}$ only. Next the computed value $\mathcal{W} = \mathcal{L}^{\mathcal{T}_{\mathcal{S}i}}$ is provided to $\mathcal{V}$, further proving the $\mathcal{T}_{\mathcal{S}i}$ knowledge is similar to proving the identity of $\mathcal{S}_{\mathcal{U}_i}$.
- $\mathcal{S}_{\mathcal{U}_i}$ picks the random value $\mathcal{R}\mathcal{V}$ to calculate $Q = \mathcal{L}^{\mathcal{R}\mathcal{V}} \bmod \mathcal{R}$ and sent Q to $\mathcal{V}$ by encrypting the Q value using $\mathcal{PB}_{\mathcal{V}\mathcal{K}}$ over the secure channel.
- By Receiving the encrypted value $\mathcal{PB}_{\mathcal{V}\mathcal{K}}(Q)$, $\mathcal{V}$ decrypts and extracts the value of Q and further the value of $\mathcal{L}^{\mathcal{R}\mathcal{V}} \bmod \mathcal{R}$ gets extracted.
- Next $\mathcal{V}$ randomly asks two question ($\mathcal{Q}_1, \mathcal{Q}_2$) with the probability of disclose of value $\mathcal{T}_{\mathcal{S}i+\mathcal{D}}$ or ($\mathcal{T}_{\mathcal{S}i} + \mathcal{T}_{\mathcal{S}i+\mathcal{D}}$) mod ($\mathcal{R}$-1).
- Once the $\mathcal{S}_{\mathcal{U}_i}$ gives the answer of ($\mathcal{A}$), $\mathcal{V}$ checks with ($\mathcal{A}_1, \mathcal{A}_2$). If $\mathcal{S}_{\mathcal{U}_i}$ is asked the question ($\mathcal{Q}_1$) i.e., $\mathcal{T}_{\mathcal{S}i+\mathcal{D}}$ it evaluates the value $\mathcal{L}^{\mathcal{T}_{\mathcal{S}i+\mathcal{D}}} \bmod \mathcal{R}$ and matches with Q. If question ($\mathcal{Q}_2$) is asked then the value is evaluated with ($\mathcal{L}^{\mathcal{T}_{\mathcal{S}i+\mathcal{D}}} \bmod (\mathcal{R}-1)$) mod $\mathcal{R}$, and matched with $Q.\mathcal{W} \bmod \mathcal{R}$.
- If the correct answer is found from the $\mathcal{S}_{\mathcal{U}_i}$ for the question ($\mathcal{Q}_1, \mathcal{Q}_2$) by $\mathcal{V}$ then it assigns a permanent id $\mathcal{ID}_i$ to $\mathcal{S}_{\mathcal{U}_i}$ and same is added to the blockchain network.

All the $\mathcal{S}_{\mathcal{U}_i}$ which are part of the blockchain network can add the $\mathcal{ID}_{\mathcal{S}_{\mathcal{U}_i}}$ for receiving a new block from the $\mathcal{V}$. The process of the registration and verification is discussed in the Table 2.

3.2.3. Encryption and decryption of IoT generated data

Once the IoT device ($\mathcal{S}_{\mathcal{U}_i}$) is registered by verifying authority i.e. $\mathcal{V}$ successfully, a public key $\mathcal{PB}_{\mathcal{S}_{\mathcal{U}_i}}$ and private key $\mathcal{PR}_{\mathcal{S}_{\mathcal{U}_i}}$ gets generated. Next, for a authenticated IoT device i.e. sensor node $\mathcal{S}_{\mathcal{U}_i}$, secret key $\mathcal{SKEY}'_{\mathcal{S}_{\mathcal{U}_i}}$ is computed over the infinite field $Z_{\mathcal{P}_{\mathcal{N}}}$ and random chosen point $\mathcal{P}_{\mathcal{N}}$ over elliptic curve. The $\mathcal{SKEY}'_{\mathcal{S}_{\mathcal{U}_i}}$ gets computed using summation of $\mathcal{PB}_{\mathcal{S}_{\mathcal{U}_i}}$, $\mathcal{PR}_{\mathcal{S}_{\mathcal{U}_i}}$, and $\mathcal{P}_{\mathcal{N}}$ which is shown in Eq. (1),

$$\mathcal{SKEY}'_{\mathcal{S}_{\mathcal{U}_i}} = \sum(\mathcal{PB}_{\mathcal{S}_{\mathcal{U}_i}}, \mathcal{PR}_{\mathcal{S}_{\mathcal{U}_i}}, \mathcal{P}_{\mathcal{N}}) \tag{1}$$

Here $\mathcal{SKEY}'_{\mathcal{S}_{\mathcal{U}_i}}$ is a secret key. After computation of secret key, the data gained from $\mathcal{S}_{\mathcal{U}_i}$ gets encrypted. The encrypted data are divided into two different ciphertexts that are illustrated in Eq. (2) and Eq. (3),

$$\mathcal{CP}1 = (\mathcal{P}_{\mathcal{N}}1 * \mathcal{P}_{\mathcal{N}}) + \mathcal{SKEY}'_{\mathcal{S}_{\mathcal{U}_i}} \tag{2}$$

$$\mathcal{CP}2 = \mathcal{MSG}_{\mathcal{S}_{\mathcal{U}_i}} + (\mathcal{P}_{\mathcal{N}}1 * \mathcal{PB}_{\mathcal{S}_{\mathcal{U}_i}}) + \mathcal{SKEY}'_{\mathcal{S}_{\mathcal{U}_i}} \tag{3}$$

Here $\mathcal{CP}1$ and $\mathcal{CP}2$ denote the ciphertext, $\mathcal{P}_{\mathcal{N}}1$ denotes random $\in Z_{\mathcal{P}_{\mathcal{N}}}$, and $\mathcal{MSG}_{\mathcal{S}_{\mathcal{U}_i}}$ is the actual message generated from a IoT device. Finally, the message gets decrypted using Eq. (4).

$$\mathcal{MSG}_{\mathcal{S}_{\mathcal{U}_i}} = ((\mathcal{CP}2 - \mathcal{PB}_{\mathcal{S}_{\mathcal{U}_i}}) * \mathcal{CP}1) - \mathcal{SKEY}'_{\mathcal{S}_{\mathcal{U}_i}} \tag{4}$$

3.2.4. Block creation and validation phase

After successful registration of $\mathcal{S}_{\mathcal{U}_i}$, block creation and validation process starts which is shown in Table 3. The steps involved for communication between $\mathcal{S}_{\mathcal{U}_i}$ and $\mathcal{EDGE}$ is summarized below:

Step 1: The initial stage includes, key pairs of $\mathcal{S}_{\mathcal{U}_i}$ i.e., ($\mathcal{PB}_{\mathcal{S}_{\mathcal{U}_i}}$, $\mathcal{PR}_{\mathcal{S}_{\mathcal{U}_i}}$), where $\mathcal{PB}_{\mathcal{S}_{\mathcal{U}_i}}$ is a public key and $\mathcal{PR}_{\mathcal{S}_{\mathcal{U}_i}}$ is a private key for IoT sensor node ($\mathcal{S}_{\mathcal{U}_i}$). Further, process of $\mathcal{EDGE}$ is performed.

Step 2: The $\mathcal{EDGE}$ creates signature ($\mathcal{EDGE}_{sig}$) and sent it for validation to $\mathcal{S}_{\mathcal{U}_i}$.

Step 3: The $\mathcal{S}_{\mathcal{U}_i}$ verify signature for the further process of communication. If signature $\mathcal{EDGE}_{sig}$ matches successfully, then $\mathcal{S}_{\mathcal{U}_i}$ request to $\mathcal{EDGE}_{sig}$ join the blockchain network with its credential i.e., $\mathcal{PB}_{\mathcal{S}_{\mathcal{U}_i}}$ and $\mathcal{ID}_{\mathcal{S}_{\mathcal{U}_i}}$.

Table 2
Registration and verification phase.

IoT sensor Node (S_{id_i})	Verifier (V)
Input: $\mathcal{REQ}_{S_{id_i}}$ Request to join blockchain network Output: $ID_{S_{id_i}}$ $PK \rightarrow (ID_{S_{id_i}}, M_{S_{id_i}})$ Store $T_{Si} \leftarrow$ Timestamp of PK Transmit $(PK(\mathcal{REQ}_{S_{id_i}}) = (ID_{S_{id_i}}, M_{S_{id_i}}))$ to V using open channel	
	Extract $PK(\mathcal{REQ}_{S_{id_i}}) \rightarrow ID_{S_{id_i}}, M_{S_{id_i}}$ Check $M_{S_{id_i}}$ exists or not if exist then terminate PK Else share PB_{VK} to $ID_{S_{id_i}}$ using open channel
ZKP challenge response protocols Evaluate $W = \mathcal{L}^{T_{Si}} \bmod \mathcal{R}$ Evaluate $r \leftarrow T_{Si+\Phi}$ Evaluate $d = \mathcal{L}^r \bmod \mathcal{R}$ Encrypt d with $PB_{VK} \rightarrow PB_{VK}(d)$ Transmit $PB_{VK}(d)$ to V using secure channel	
	Decrypt using $PR_{VK}(d)$ Extract $d = \mathcal{L}^r \bmod \mathcal{R}$ Ask question $Q_1 \rightarrow T_{Si+\Phi}$ or $Q_2 \rightarrow (T_{Si} + T_{Si+\Phi}) \bmod (\mathcal{R}-1)$ such that $PB(T_{Si+\Phi}) =$ $PB(T_{Si} + T_{Si+\Phi}) \bmod (\mathcal{R}-1)$ Transmit $Q \in (Q_1, Q_2)$ using secure channel
If $Q = Q_1$ Then send $A_1 \leftarrow T_{Si+\Phi}$ Else send $A_2 \leftarrow (T_{Si} + T_{Si+\Phi}) \bmod (\mathcal{R}-1)$ Transmit $A \in (A_1, A_2)$ Encrypt $PB_{VK}(A)$ to (V) using secure channel	
	Decrypt using $PR_{VK}(A)$ Extract A if $A = A_1$ then verify $\mathcal{L}^{T_{Si+\Phi}} \bmod \mathcal{R} = d$ else Compute $(\mathcal{L}^{T_{Si+\Phi}} \bmod (\mathcal{R}-1)) = d$. $W \bmod \mathcal{R}$ Verified successfully Assign $S_{id_i} \rightarrow ID_{S_{id_i}}$ Add S_{id_i} to Blockchain Network Disseminate $ID_{S_{id_i}}$ to all peer of blockchain network

Table 3
Block validation and creation process.

IoT sensor nodes (S_{id_i})	Edge ($EDGE$)	Edge Peer ($EDGE_{Peer}$)
INPUT: $ID_{S_{id_i}}$ OUTPUT: $ID_{S_{id_i}}^{BLOCK}$ Create key pair $(PB_{S_{id_i}}, PR_{S_{id_i}})$ of $ID_{S_{id_i}}$ Sent by secure channel $PB_{S_{id_i}}$		
	Create $SIG_{S_{id_i}}$ Transmit the signature $SIG_{S_{id_i}}$	
Validate ($SIG_{S_{id_i}}$) Send ($PB_{S_{id_i}}$) Request to join $PB_{S_{id_i}}$, $ID_{S_{id_i}}$		
	Send $PB_{S_{id_i}}$ to $EDGE_{Peer}$ peer nodes and send the $PB_{S_{id_i}}$ using secure channel	
		Check $PB_{S_{id_i}}$ Return True/False as a status
	Verify $PB_{S_{id_i}}$ If matches, perform block creation $ID_{S_{id_i}}^{BLOCK}$ Append $ID_{S_{id_i}}^{BLOCK}$ into blockchain and preserve data into IPFS layer Distribute $ID_{S_{id_i}}^{BLOCK}$ to peer nodes $EDGE_{Peer}$ Send the $ID_{S_{id_i}}^{BLOCK}$ for further access.	

Table 4
Data creation and block updation.

IoT sensor Node ($\mathcal{S}_{d_i}$)	Edge ($\mathcal{EDGE}$)	Edge Peer Nodes (EDGE_{Peer})
INPUT: $ID_{\mathcal{S}_{d_i}^{TX}}$ OUTPUT: Update $ID_{\mathcal{S}_{d_i}^{BLOCK}}$ read ($ID_{\mathcal{S}_{d_i}^{TX}}$, $ID_{\mathcal{S}d_i}$) Sign ($SIG_{\mathcal{S}d_i}$)←($ID_{\mathcal{S}_{d_i}^{TX}}$, $\mathcal{PB}_{\mathcal{S}d_i}$) Create $ID_{\mathcal{S}_{d_i}^{NEWTX}}$ = ($ID_{\mathcal{S}_{d_i}^{TX}}$, $\mathcal{PB}_{\mathcal{S}d_i}$, $SIG_{\mathcal{S}d_i}$, $ID_{\mathcal{S}d_i}$) Send $ID_{\mathcal{S}_{d_i}^{NEWTX}}$ using secure channel	Validate $ID_{\mathcal{S}d_i}$, $\mathcal{PB}_{\mathcal{S}d_i}$ Check $ID_{\mathcal{S}_{d_i}^{NEWTX}}$, $SIG_{\mathcal{S}d_i}$ Validate $SIG_{\mathcal{S}d_i}$ Add $ID_{\mathcal{S}_{d_i}^{NEWTX}}$ to $ID_{\mathcal{S}_{d_i}^{BLOCK}}$ Disseminate to $\mathcal{EDGE}_{Peer}$ peer	Validate $SIG_{\mathcal{S}d_i}$ and $ID_{\mathcal{S}_{d_i}^{BLOCK}}$ Synchronized Blockchain

Step 4: Further, $SIG_{\mathcal{S}d_i}$ is sent by the $\mathcal{PB}_{\mathcal{S}d_i}$ to peer nodes ($\mathcal{EDGE}_{Peer}$) to validate the public key $\mathcal{PB}_{\mathcal{S}d_i}$.

Step 5: The public key including signature $SIG_{\mathcal{S}d_i}$ is validated by the peer nodes ($\mathcal{EDGE}_{Peer}$) and sends an acknowledgment as Successful/Unsuccessful.

Step 6: For the Successful acknowledgment, new block ($ID_{\mathcal{S}_{d_i}^{BLOCK}}$) is produced and forward it for addition into blockchain with credentials $\mathcal{PB}_{\mathcal{S}d_i}$ and $ID_{\mathcal{S}d_i}$. Finally, actual data is preserved into IPFS storage layer.

3.2.5. Data generation and block updation

This phase describes data generation and respective block updation (how $\mathcal{S}_{d_i}$ generates data ($ID_{\mathcal{S}_{d_i}^{TX}}$) and makes updation in block). The entire process of data generation and block updation is illustrated in the Table 4. The working steps are detailed below.

Step 1: At first step, transaction (TX) is generated by the $ID_{\mathcal{S}_{d_i}^{TX}}$ and is signed ($SIG_{\mathcal{S}d_i}$) using $\mathcal{PB}_{\mathcal{S}d_i}$ of $ID_{\mathcal{S}d_i}$, once the $ID_{\mathcal{S}_{d_i}^{TX}}$ is signed, new transaction ($ID_{\mathcal{S}_{d_i}^{NEWTX}}$) is created along with $SIG_{\mathcal{S}d_i}$, $\mathcal{PB}_{\mathcal{S}d_i}$, $ID_{\mathcal{S}d_i}$ of $\mathcal{S}_{d_i}$. Next $ID_{\mathcal{S}_{d_i}^{NEWTX}}$ is sent to the $\mathcal{EDGE}_{Peer}$ for validation and further updation in $ID_{\mathcal{S}_{d_i}^{BLOCK}}$.

Step 2: Further, records are validated $\mathcal{PB}_{\mathcal{S}d_i}$ for respective $ID_{\mathcal{S}d_i}$ along with $ID_{\mathcal{S}_{d_i}^{TX}}$ and $SIG_{\mathcal{S}d_i}$. If all are valid then, $ID_{\mathcal{S}_{d_i}^{NEWTX}}$ is appended in a block $ID_{\mathcal{S}_{d_i}^{BLOCK}}$ and distributed into the blockchain network.

Step 3: Furthermore, $ID_{\mathcal{S}_{d_i}^{BLOCK}}$ gets updated and successfully appended to the network of blockchain.

3.2.6. Consensus phase

After the successful verification of ZKP, the $ID_{\mathcal{S}d_i}$ gets generated and handover to respective IoT sensor node and updated into blockchain network. The ePoW consensus approach is performed for transaction verification and addition into blockchain network i.e., $ID_{\mathcal{S}_{d_i}^{NEWTX}}$ by $ID_{\mathcal{S}d_i}$. The verification of transaction is done by legitimate peers $\mathcal{EDGE}_{peer}$, where various credentials are matched, i.e., $ID_{\mathcal{S}d_i}$, and $SIG_{\mathcal{S}d_i}$ of respective $\mathcal{S}_{d_i}$. After successful verification of transactions a nonce gets created along with difficulty level 1 for respective transactions $ID_{\mathcal{S}_{d_i}^{NEWTX}}$. Here, difficulty level is preserved minimum as all the nodes are legitimate in the network. The hash of transaction is computed by SHA-512. The block consists of various parameters, i.e., $ID_{\mathcal{S}d_i}$, $SIG_{\mathcal{S}d_i}$, hash of $ID_{\mathcal{S}_{d_i}^{NEWTX}}$, $\mathcal{PB}_{\mathcal{S}d_i}$, nonce, and timestamp $\mathcal{T}_{Si}$ and successfully appended into the blockchain network. Next, the actual information is preserved in IPFS storage layer. The entire consensus approach is illustrated in Table 5.

3.3. Deep learning enabled security architecture

3.3.1. Deep Sparse AutoEncoder (DSAE) for feature extraction

In general, a Sparse AutoEncoder (SAE) is made up of two parts: an encoder and a decoder. The SAE is trained on a $\mathcal{D}$-dimensional training set of $\mathcal{X} \in \left\{\mathcal{X}_k^{(\mathcal{D})}\right\}$, (where k denotes number of samples), such that, the $\mathcal{D}$-dimensional sample set is transformed into $\mathcal{D}'$-dimensional encoder vector $\mathcal{H} \in \left\{\mathcal{H}^{(\mathcal{D}')}\right\}$. Then, the obtained encoded vector, $\mathcal{D}'$-dimensional is decoded into initial $\mathcal{D}$-dimensional space to obtain $\mathcal{Y} \in \left\{\mathcal{Y}_k^{(\mathcal{D})}\right\}$. To be more generalized, the process of encoder is expressed as [19];

$$\mathcal{H} = \mathcal{J}_\theta(\mathcal{D}) = \mathcal{E}_\mathcal{F}(\mathcal{W}_\mathcal{D} + \mathcal{B}) \tag{5}$$

Where, $\theta = \{\mathcal{W}, \mathcal{B}\}$ denotes parameter sets of encoder, in which $\mathcal{W}$ represents weight matrix and $\mathcal{B}$ denotes the offset vector. Encoder activation function is represented by $\mathcal{E}_\mathcal{F}$. Similarly, the process of decoder is expressed as

$$\mathcal{Y} = \mathcal{G}_{\theta'}(\mathcal{H}) = \mathcal{E}_\mathcal{G}\left(\mathcal{W}'_\mathcal{H} + \mathcal{B}'\right) \tag{6}$$

Where, $\theta' = \left\{\mathcal{W}', \mathcal{B}'\right\}$ denotes parameter sets of decoder, in which $\mathcal{W}'$ represents weight matrix and $\mathcal{B}'$ denotes the offset vector. Decoder activation function is represented by $\mathcal{E}_\mathcal{G}$. Thus, AE cost function is formulated as

$$\mathcal{JF} = min\frac{1}{k}\sum_{i=1}^{k}\|\mathcal{Y}_i - \mathcal{D}_i\| \tag{7}$$

Let us assume that $\mathcal{J}^{th}$ cell in the hidden layer is denoted by $\mathcal{A}_\mathcal{J}(\mathcal{D})$, then $\mathcal{J}^{th}$ cell average activation amount is calculated as

$$\hat{\mathcal{P}}_\mathcal{J} = \frac{1}{k}\sum_{i=1}^{k}\mathcal{A}_\mathcal{J}(\mathcal{D}(i)) \tag{8}$$

The number of samples is indicated by k. The $\hat{\mathcal{P}}_\mathcal{J}$ should be equivalent to a constant that is similar to 0 in order to make the maximum of neurons "inactive." A sparse penalty term is applied to the AE cost function to penalize $\hat{\mathcal{P}}_\mathcal{J}$. As the expression of the penalty term ($\mathcal{PT}$), the Kullback–Leibler (KL) divergence is used.

$$\mathcal{PT} = \sum_{\mathcal{J}=1}^{\mathcal{C}_2} KL\left(\mathcal{P} \,\|\, \hat{\mathcal{P}}_\mathcal{J}\right) \tag{9}$$

The number of cells in the hidden layer is denoted by $\mathcal{C}_2$. The KL divergence is denoted by $KL\left(\mathcal{P} \,\|\, \hat{\mathcal{P}}_\mathcal{J}\right)$ and is expressed as

Table 5
Consensus (ePoW) phase between edge ($EDGE$) and edge peer ($EDGE_{peer}$).

$EDGE$	$EDGE_{peer}$
Input: HD_{SA_i}	
$HD_{SA_i}^{RT}$ PB_{SA_i} SIG_{SA_i}	
Validate HD_{SA_i} and PB_{SA_i}	
$BLOCK_{NORSN}$ = $DIGEST(HD_{SA_i}, HD_{SA_i}^{BLOCKID}$,	
$HD_{SA_i}^{RT}$, T_{SI},	
$PROOF$, $SHA512$ $PREVIOUS_{NORSN}$,	
$CURRENT_{NORSN}$)	
/* Process for $PROOF$ creation */	
Validate $LAST_{PROOF}$	
$PROOF \leftarrow LAST_{PROOF}$	
Repeat (($PROOF$ + (int)	
($LAST_{PROOF}$ / 7)) & 7) == 0)	
// difficulty level 1 is set // and block hash gets created	
$BLOCK_{NORSN}$	
$PROOF$ ++	
Share HD_{SA_i}, $PROOF$, $BLOCK_{NORSN}$	
PB_{SA_i}, SIG_{SA_i}	
to $EDGE_{peer}$ using secure channel	
	Received HD_{SA_i}, $PROOF$, $BLOCK_{NORSN}$
	PB_{SA_i}, SIG_{SA_i}
	Validate HD_{SA_i}, $PROOF$,
	$BLOCK_{NORSN}$ PB_{SA_i}, SIG_{SA_i}
	After successful match return True
	Else return False
	through secure channel
Store HD_{SA_i} and $HD_{SA_i}^{RT}$ into IPFS	
Append $BLOCK_{NORSN}$ into a block	

$$KL(\mathcal{P} \parallel \hat{\mathcal{P}}_j) = \mathcal{P} \log \frac{\mathcal{P}}{\hat{\mathcal{P}}_j} + (1-\mathcal{P}) \log \frac{(1-\mathcal{P})}{(1-\hat{\mathcal{P}}_j)} \tag{10}$$

Then, we add sparse penalty term and calculate the optimized objective function of SAE as

$$\mathcal{JF}_{sparse}(\mathcal{W}, \mathcal{B}) = \frac{1}{k} \sum_{i=1}^{k} \left\| \mathcal{D}(i) - \hat{\mathcal{D}}(i) \right\|^2 + \frac{\lambda}{2} \sum_{l=1}^{nl-1} \sum_{i=1}^{C_l} \sum_{j=1}^{C_{l+1}} \left(W_{ij}(l)^2 \right) \tag{11}$$

The weight decay coefficient is indicated by λ, while the neural network layers are indicated by nl. C_l denotes the number of neurons in the l layer. The number of neurons in the $l+1$ layer is represented by $C+1$. Finally, the cost function with sparse penalty term is computed as follows:

$$\mathcal{JF}_{sparse}(\mathcal{W}, \mathcal{B}) = (\mathcal{W}, \mathcal{B}) + \beta \mathcal{PT} \tag{12}$$

Where, sparse penalty coefficient is denoted by β. SAE latent distributions may reduce the likelihood of data variations. The model is set up to transform original data into a new format using the L2 regularization approach. The extracted low dimensional features are used by BiLSTM for intrusion detection.

3.3.2. Bidirectional Long Short-Term Memory (BiLSTM) for anomaly detection

To verify the effectiveness of DL based feature extraction approach i.e., DSAE technique, a BiLSTM algorithm is designed and applied to detect threats. The bidirectional LSTM (BiLSTM) technique employs two different hidden layers i.e., a forward LSTM layer and a backward LSTM layer to handle sequences of the IoT data in two directions. The two different arrow symbols $\rightarrow$ and $\leftarrow$ represents forward and backward process, respectively. The operating theory is as follows: the forward layer gathers the sequence past information $\overrightarrow{h_t}$; the backward layer gathers the sequence's future information $\overleftarrow{h_t}$. The same output layer $y(t)$ connected to both layers reiterates top to the bottom (forwards) from $t = 1$ to t_f, bottom to the top (backwards) from $t = t_f$ to 1. For the BiLSTM, the transition functions at time step t are calculated as follows [3]:

$$\overrightarrow{h_t} = \mathcal{F}\left(\overrightarrow{d_t}, \overrightarrow{h_{t-1}}; \overrightarrow{\Theta_{LSTM}}\right) = \begin{cases} \overrightarrow{f_t} = \sigma\left(\overrightarrow{W_f}\overrightarrow{d_t} + \overrightarrow{\mathcal{H}_f}\overrightarrow{h_{t-1}} + \overrightarrow{\mathcal{B}_f}\right), \\ \overrightarrow{i_t} = \sigma\left(\overrightarrow{W_i}\overrightarrow{d_t} + \overrightarrow{\mathcal{H}_i}\overrightarrow{h_{t-1}} + \overrightarrow{\mathcal{B}_i}\right), \\ \overrightarrow{z_t} = \tanh\left(\overrightarrow{W_z}\overrightarrow{d_t} + \overrightarrow{\mathcal{H}_z}\overrightarrow{h_{t-1}} + \overrightarrow{\mathcal{B}_z}\right), \\ \overrightarrow{c_t} = \overrightarrow{z_t} \odot \overrightarrow{i_t} + \overrightarrow{c_{t-1}} \odot \overrightarrow{f_t}, \\ \overrightarrow{o_t} = \sigma\left(\overrightarrow{W_o}\overrightarrow{d_t} + \overrightarrow{\mathcal{H}_o}\overrightarrow{h_{t-1}} + \overrightarrow{\mathcal{B}_o}\right), \\ \overrightarrow{h_t} = \tanh\left(\overrightarrow{c_t}\right) \odot \overrightarrow{o_t}. \end{cases} \tag{13}$$

$$\overleftarrow{h_t} = \mathcal{F}\left(\overleftarrow{d_t}, \overleftarrow{h_{t+1}}; \overleftarrow{\Theta_{LSTM}}\right) = \begin{cases} \overleftarrow{f_t} = \sigma\left(\overleftarrow{W_f}\overleftarrow{d_t} + \overleftarrow{\mathcal{H}_f}\overleftarrow{h_{t+1}} + \overleftarrow{\mathcal{B}_f}\right), \\ \overleftarrow{i_t} = \sigma\left(\overleftarrow{W_i}\overleftarrow{d_t} + \overleftarrow{\mathcal{H}_i}\overleftarrow{h_{t+1}} + \overleftarrow{\mathcal{B}_i}\right), \\ \overleftarrow{z_t} = \tanh\left(\overleftarrow{W_z}\overleftarrow{d_t} + \overleftarrow{\mathcal{H}_z}\overleftarrow{h_{t+1}} + \overleftarrow{\mathcal{B}_z}\right), \\ \overleftarrow{c_t} = \overleftarrow{z_t} \odot \overleftarrow{i_t} + \overleftarrow{c_{t+1}} \odot \overleftarrow{f_t}, \\ \overleftarrow{o_t} = \sigma\left(\overleftarrow{W_o}\overleftarrow{d_t} + \overleftarrow{\mathcal{H}_o}\overleftarrow{h_{t+1}} + \overleftarrow{\mathcal{B}_o}\right), \\ \overleftarrow{h_t} = \tanh\left(\overleftarrow{c_t}\right) \odot \overleftarrow{o_t}. \end{cases} \tag{14}$$

Where $\overrightarrow{\Theta_{LSTM}}$ and $\overleftarrow{\Theta_{LSTM}}$ are the parameters for the forward and backward phases shared during each time steps and learnt by model training. For the forward and backward processes, $\overrightarrow{W_i}, \overrightarrow{W_f}, \overrightarrow{W_z}, \overrightarrow{W_o}$ and $\overleftarrow{W_i}, \overleftarrow{W_f}, \overleftarrow{W_z}, \overleftarrow{W_o}$ represent the input weight matrix from the input layer to the hidden layer, which is related to the current input $\overrightarrow{x_t}$ and $\overleftarrow{x_t}$. $\overrightarrow{\mathcal{H}_z}, \overrightarrow{\mathcal{H}_i}, \overrightarrow{\mathcal{H}_f}, \overrightarrow{\mathcal{H}_o}$ and $\overleftarrow{\mathcal{H}_z}, \overleftarrow{\mathcal{H}_i}, \overleftarrow{\mathcal{H}_f}, \overleftarrow{\mathcal{H}_o}$ represent the recurrent weight matrix among two successive hidden states, $\overrightarrow{h_{t-1}}$, previous recurrent input, and $\overleftarrow{h_{t+1}}$, future recurrent input. The forward and backward processes' bias weights are

indicated by the letters $\overrightarrow{R_z}$, $\overrightarrow{R_i}$, $\overrightarrow{R_f}$, $\overrightarrow{R_o}$ and $\overleftarrow{R_z}$, $\overleftarrow{R_i}$, $\overleftarrow{R_f}$, $\overleftarrow{R_o}$, respectively. A tanh, or hyperbolic tangent, is a point-wise non-linear activation function, and $\odot$ is point-wise multiplication of two vectors. The final output vector, $y(t)$, is then calculated as follows:

$$y(t) = \sigma_y(\overrightarrow{h_t}, \overleftarrow{h_t}). \tag{15}$$

where the σ_y function concatenates the output neuron sequences in hidden layers and can do any of the four operations: add, concatenate, multiply, and average.

4. Security and privacy analysis

This section presents formal security and privacy verification for the proposed framework.

4.0.1. Impersonation attack

The attacker, $\mathcal{A}$ perform activities as a legitimate $\mathcal{S}_{d_i}$ by sharing (i) temporary credential and identification ($ID_{\mathcal{S}_{d_i}}$) and (ii) sensor MAC address ($\mathcal{M}_{\mathcal{S}_{d_i}}$), to $\mathcal{V}$ for obtaining a provisional key $PK_{\mathcal{S}_{d_i}}$. Next, timestamp $\mathcal{T}_{\mathcal{S}i}$ gets created for request of $ID_{\mathcal{S}_{d_i}}$. Further, $\mathcal{V}$ checks existing credential such as $\mathcal{M}_{\mathcal{S}_{d_i}}$ with records and also legitimate the $\mathcal{T}_{\mathcal{S}i}$. If it matches successful, further ZKP verification is performed for identity creation $ID_{\mathcal{S}_{d_i}}$. If mentioned timestamp $\mathcal{T}_{\mathcal{S}i}$ not matched successfully then, immediately work flow gets terminated. As a result, the framework is secure from impersonation attack.

4.0.2. Insider attack

The attacker $\mathcal{A}$ might be legitimate and can have all the necessary credential information of $\mathcal{S}_{d_i}$ such as (i) temporary credential or identification ($ID_{\mathcal{S}_{d_i}}$) and (ii) sensor MAC address ($\mathcal{M}_{\mathcal{S}_{d_i}}$). But all above actual identity $ID_{\mathcal{S}_{d_i}}$ cannot be computed owing to timestamp verification $\mathcal{T}_{\mathcal{S}i}$ over ZKP prover and challenge response protocols. Thus, it prevents from insider attack.

4.0.3. MITM and replay attack

The attacker $\mathcal{A}$ can get a message from insecure channel such as (i) temporary credential or identification ($ID_{\mathcal{S}_{d_i}}$) and (ii) sensor MAC address ($\mathcal{M}_{\mathcal{S}_{d_i}}$) to execute a MITM and Replay attack. Next, from obtained information from insecure channel $\mathcal{S}_{d_i}$ has to compute possible timestamp using brute-force techniques $\mathcal{T}_{\mathcal{S}i}$. The operation takes several computation power to obtain the correct timestamp over ZKP. This ZKP process is almost difficult to predict with correct answer. Thus, the process is safe from MITM and Replay attack.

4.0.4. Privacy analysis

In the proposed framework, we have also performed privacy analysis. Let $\mathcal{A}$ try to get the existing information using the short signature techniques and wish to modify the information obtained from the secure channel. In the proposed BDSDT framework, two major approaches have been considered for the privacy-preservation. Firstly, the $ID_{\mathcal{S}_{d_i}}$ is created based on ZKP verification scheme by using prover ($\mathcal{S}_{d_i}$) and verifier ($\mathcal{V}$) challenge response protocols. The signature is verified during the block creation in the blockchain network with the associated $ID_{\mathcal{S}_{d_i}}$. Thus, the information is stored into the block, in form of transaction hash and each block is attached with subsequent block as a chain. Thus, modifying (poisoning) the information is almost impossible as altering one block hash needs alternation is all subsequent blocks.

5. Experimental results evaluation

This section evaluates the performance of the BDSDT framework through simulations. A Tyrone PC having 128 GB RAM with Python 3 and TensorFlow library Keras were used to develop deep learning approaches. The private blockchain was built using Ganache and smart contracts scripts were written and executed using Ethereum blockchain (public). These two separate interfaces were connected using "WEB3 Provider interface of Ethereum". We implement IPFS version 0.4.19 for off-chain storage making BDSDT framework scalable. We followed few pre-processing steps, mentioned in [19] to make datasets useful and understandable format. In order to assess the performance, datasets were divided into testing (30%) and training sets (70%), respectively.

5.1. Description of dataset

- ToN-IoT ($\mathcal{D}1$) [4], [21]: It includes normal, and 9 abnormal observations mostly found in IoT/IoT environment such as Backdoor (0), DDoS (1), DoS (2), Injection (3), MITM (4), Normal (5), Password (6), Ransomware (7), Scanning (8), XSS (9). It consists of 43 labeled features and has 1498334 attack and 79053 normal instances.
- CICIDS-2017 ($\mathcal{D}2$) [26], [27]: It contains various updated attack observations including Ransomware, SSH-Patator, FTP-Patator, DoSHulk, DoS-Slowhttptest, DoS-Goldeneye, Injection and MITM. It consists of 78 labeled features and has 390222 attack and 2035505 benign observations.

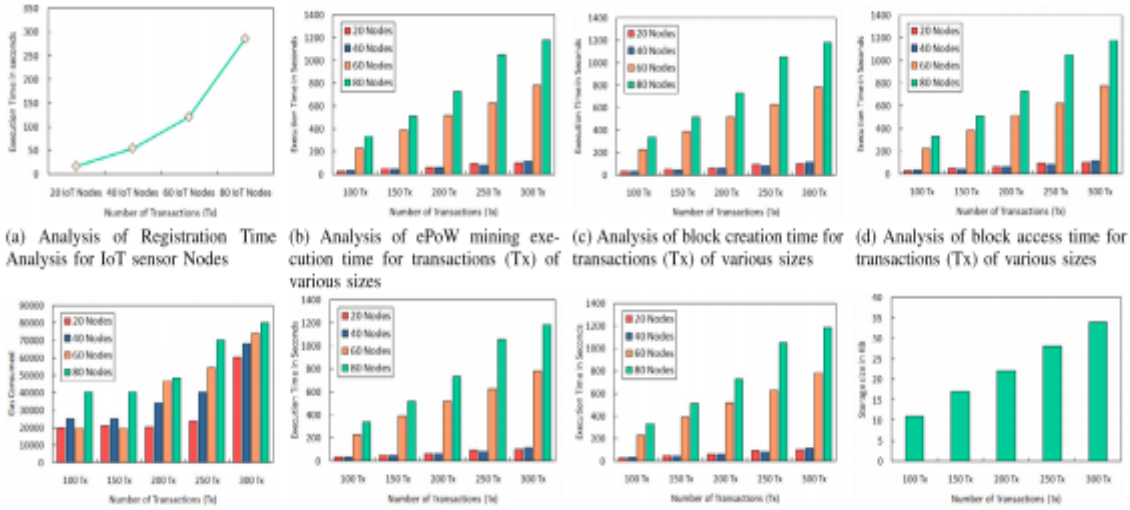
5.2. Description of evaluation metrics

In this article, we have used False Positive (β), False Negative (δ), True Positive (α), True Negative (γ) parameters to calculate Accuracy (AC) = $\frac{\alpha+\gamma}{\gamma+\delta+\alpha+\beta}$, Detection Rate (DR) = $\frac{\alpha}{\delta+\alpha}$, Precision (PR) = $\frac{\alpha}{\alpha+\beta}$, F1 Score = $2 * \frac{PR*RC}{PR+RC}$ and False Alarm Rate (FAR) = $\frac{\beta}{\beta+\gamma}$ [1].

5.3. Numerical analysis of blockchain enabled security architecture

The security and privacy evaluation of the proposed blockchain architecture is discussed below. The suggested blockchain architecture is validated with respect to various criteria i.e., First with increase in the number of nodes and transactions registration time of IoT nodes in BDSDT framework is determined. The registration time of the participating IoT nodes in the BDSDT framework is depicted in Fig. 2a. The time it takes to register progressively increases as the number of nodes grows. Similarly, Fig. 2b shows the time taken by the proposed ePoW consensus algorithm. The BDSDT framework is evaluated for maximum of 80 IoT nodes and 300 Tx. It has been noticed that as the number of transactions with nodes rises, the amount of time spent on each transaction increases. Fig. 2c and Fig. 2d illustrate block creation and access time. We can see that, up to 40 nodes and with 300 Tx, time is stabilized. However, when the number of nodes rises from 60 to 80, the time it takes to create a block and access them increases. Fig. 2e shows the gas price consumption for smart contract access and its deployment. It is evident from Fig. 2e that price rises steadily with increase in IoT nodes and Tx. In order to ensure non-repudiation in BDSDT framework signing time taken by participating IoT nodes is calculated. Fig. 2f describes the signing time taken by the participating IoT nodes by varying Tx. With an increase of IoT nodes and Tx, a similar trend in time is observed. Fig. 2g shows time taken to deploy the proposed smart contracts in the network. It is seen that up to 40 nodes and 300 Tx, there is marginal increase in time. But as number of nodes increases with Tx, there is increase in time. Fig. 2h illustrates storage size in KB taken by the Tx to store data into IPFS-based off-chain storage. With increase in Tx size, the storage size in KB increases.

We also tested the suggested ePoW and PoW with difficulty level 1 for a range of transaction (Tx) numbers. As illustrated in



(a) Analysis of Registration Time Analysis for IoT sensor Nodes

(b) Analysis of ePoW mining execution time for transactions (Tx) of various sizes

(c) Analysis of block creation time for transactions (Tx) of various sizes

(d) Analysis of block access time for transactions (Tx) of various sizes

(e) Analysis of Gas price consumption for varying size of Transactions (Tx) on Ethereum blockchain network

(f) Execution Time Analysis of Signing of varying sizes of Transaction (Tx) for Non-Repudiation

(g) Execution Time Analysis of Contract Deployment time for varying sizes of Transaction (Tx) and Varying Number of Peers

(h) Analysis of IPFS Secure Storage Layer off-chain Storage utilization (in KB) for different Transaction Sizes (Tx)

Fig. 2. Result interpretation for blockchain-enabled security architecture.

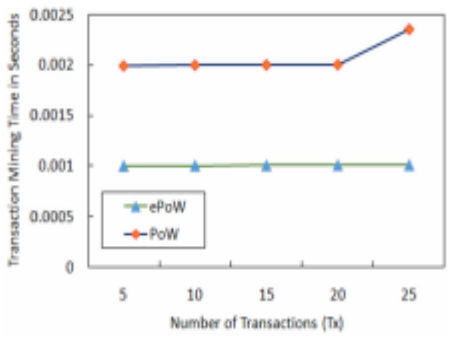


Fig. 3. Block mining time comparison of proposed ePoW with existing PoW at difficulty level 1.

Fig. 3, the ePoW takes the least amount of time to execute than the PoW. The proposed smart contract enabled ePoW outperform compare to existing PoW algorithm. Thus, we have used the smart contracts enabled ePoW algorithm for mining the transactions in proposed BDSDT framework.

5.4. Numerical analysis of deep learning enabled security architecture

Before extracting features, the dataset categorical values are mapped to numeric and normalized to build final model. Next, to evaluate the performance of feature extraction technique, the proposed DSAE technique is applied. The hyperparameters for executing feature extraction is listed in Table 6. The findings in Fig. 4 and, Fig. 5 show the efficiency of the feature extraction approach based on *accuracy* and *loss* acquired from $\mathcal{D}1$ and $\mathcal{D}2$ datasets, respectively. It's worth mentioning that the feature extraction approach learnt from both datasets efficiently. The goal of the proposed feature extraction approach is to transform data into a new format and extract key low-dimensional characteristics, rather than to identify these threat observations. Thus, the obtained datasets can be used to evaluate BiLSTM threat detection capability. The hyperparameters for attack detection using proposed IDS is listed in Table 7. With feature extraction technique (transformed), the proposed IDS achieved 0.0076% *validation loss* and 99.03% *validation accuracy* using $\mathcal{D}1$ dataset. Similarly, with $\mathcal{D}2$ dataset proposed IDS obtained 0.0235% *validation loss* and 99.04% *validation accuracy*. As a consequence of the accuracy and loss findings obtained, we can infer that proposed IDS performed well on both intrusion datasets.

A valuable technique to interpret the output of multi-class vectors available in the dataset is the Receiver Operating Characteristic (ROC). Specifically, ROC graphs are used to test the outcomes of classifiers as a visual evaluation tool [15]. Typically, the

Table 6
Hyper-parameters for feature extraction.

Settings	Hyperparameters
Input Layer	44 features of ($\mathcal{D}1$) and 78 from ($\mathcal{D}2$) datasets
Encoder	4 hidden layers; Hidden Nodes (64,32,16,10), Elu function; regularizers.l1(10e-5)
Decoder	4 hidden layers; Hidden Nodes (10,16,32,64), Elu function (for 3 layers); sigmoid (for last layer) regularizers.l1(10e-5)
DSAE Model	loss='categorical_crossentropy', optimizer='adam', epochs=100, batch_size=50

Table 7
Hyper-parameters for intrusion detection.

Settings	Hyperparameters
Input Layer	Extracted features from DSAE method
Hidden layers	4 hidden layers: 1) Layer 1: 100 HN, a tanh function, a 0.2 Dropout Rate (DR). 2) Later 2: 50 HN, a tanh function, a 0.2 DR.
Output layer	10 units (1 benign and 9 threats for ($\mathcal{D}1$)), 11 units (1 benign and 10 threats for ($\mathcal{D}2$)), a softmax function
BiLSTM Model	loss='categorical_crossentropy', optimizer='adam', epochs=20, batch_size=60

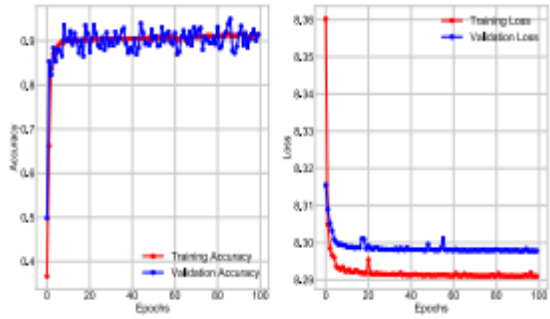


Fig. 4. The accuracy vs loss obtained from feature extraction approach using $\mathcal{D}1$ dataset.

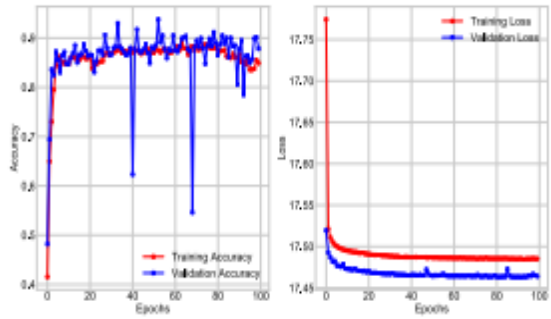


Fig. 5. The accuracy vs loss obtained from feature extraction approach using $\mathcal{D}2$ dataset.

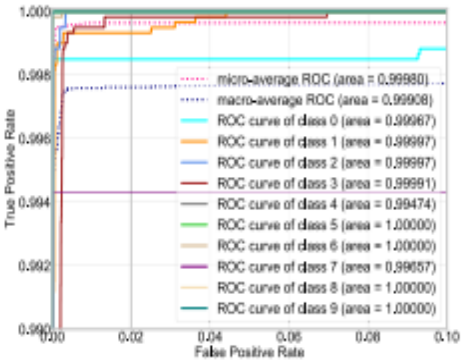


Fig. 6. ROC curve obtained from proposed IDS using $\mathcal{D}1$ dataset.

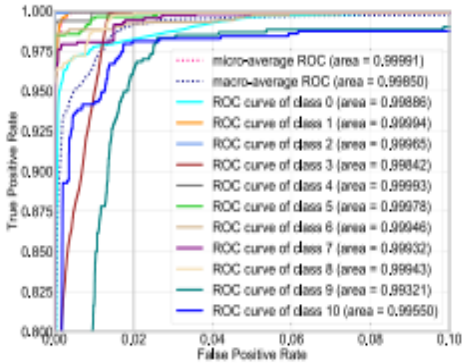


Fig. 7. ROC curve obtained from proposed IDS using $\mathcal{D}2$ dataset.

α and β axes are extended along the boundaries of the two-dimensional ROC space. The classifier performance is shown by the area under the ROC curve (AUC) for the complete range of cut-off points. The following information is interpreted using $\mathcal{D}1$ and $\mathcal{D}2$ datasets:

- Fig. 6 shows the expediency of the proposed IDS using $\mathcal{D}1$ dataset. It can be noted that the proposed IDS has reported micro-average AUC i.e., 0.99995 and macro-average AUC values as 0.99988. Moreover, for normal and attack vectors the AUC values are close to 1.00.
- With the $\mathcal{D}2$, Fig. 7 displays the ROC curve and AUC value for proposed IDS. The model has obtained 0.99967 micro-average AUC and 0.99248 macro-average AUC. Furthermore, the AUC values for normal and attack vector are between 0.94-1.00.

The class-wise prediction outcomes in terms of PR, DR, F1 score, and FAR are employed in the following assessment criteria. Table 8 and Table 9 show the PR, DR, F1 score, and FAR computed for each class using the converted and original $\mathcal{D}1$ and $\mathcal{D}2$ datasets, respectively. When compared to the original $\mathcal{D}1$ dataset, the suggested IDS produced exceptional results. In both situations, the PR, DR, and F1 numbers are between 97 and 100%. Furthermore, in both cases, proposed IDS decreased FAR to near-zero for all vectors in the dataset. Similarly, the class-wise prediction results for the converted and original $\mathcal{D}2$ datasets demonstrate the usefulness of the proposed methodology. The proposed IDS, on the other hand, did not perform well against Bot and Web threats in the $\mathcal{D}2$ sample. This is owing to the fact that there are less instances for these two attack groups in this dataset. Furthermore, for all attack and normal vectors in the $\mathcal{D}2$ dataset, FAR is close to 0%.

5.5. Comparison with traditional approaches

The comparison of proposed IDS with three different state-of-the-art ML techniques i.e., NB, DT and RF and with BiLSTM in

Table 8
Specific class scores (%) using $\mathcal{D}1$ dataset.

Dataset	Parameters	Backdoor	DDoS	DoS	Injection	MITM	Normal	Password	Ransomware	Scanning	XSS
With Transformed	PR	99.91	98.96	97.87	99.25	98.05	100.00	100.00	99.80	99.89	99.93
	DR	99.90	99.36	99.66	97.12	98.37	99.99	99.68	100.00	100.00	99.93
	F1	99.90	99.16	98.76	98.17	98.21	99.99	99.84	99.90	99.94	99.93
	FAR	0.000037	0.000468	0.000974	0.000332	0.000043	0.0	0.0	0.000090	0.000045	0.000030
With Original	PR	99.89	98.82	99.39	99.48	100.00	100.00	99.98	99.86	99.94	99.88
	DR	99.89	99.66	99.71	98.35	97.25	100.00	99.98	99.83	100.00	99.94
	F1	99.89	99.24	99.55	98.91	98.60	100.00	99.98	99.85	99.97	99.91
	FAR	0.000045	0.000536	0.000279	0.000226	0.0	0.0	0.000007	0.000060	0.000022	0.000052

Table 9
Specific class scores (%) using $\mathcal{D}2$ dataset.

Dataset	Parameters	BENIGN	DoS Hulk	DDoS	PortScan	DoS GoldenEye	FTPPatator	DoS slowloris	DoS Slowhttptes	SSHPatator	Bot	Web Attack
With Transformed	PR	98.76	87.11	99.63	90.06	90.70	98.80	98.12	88.25	99.67	98.62	18.16
	DR	99.09	95.28	84.27	80.82	98.15	75.85	93.95	98.06	96.51	37.39	52.50
	F1	98.93	91.01	91.31	85.19	94.28	85.82	95.99	92.89	98.07	54.22	27.53
	FAR	0.064554	0.010736	0.000174	0.002131	0.000420	0.000020	0.000038	0.000287	0.000004	0.0000041	0.002074
With Original	PR	99.19	96.86	99.96	93.72	95.46	95.37	98.82	94.94	99.78	97.57	97.72
	DR	99.54	99.40	99.46	79.95	98.64	99.63	97.49	98.56	96.62	34.95	06.52
	F1	99.37	98.11	99.71	86.31	97.03	97.45	98.15	96.72	98.17	51.47	12.23
	FAR	0.041944	0.002490	0.000020	0.001280	0.000195	0.000108	0.000024	0.000115	0.000002	0.000006	0.000001

Table 10
Comparison with other techniques using $\mathcal{D}1$ dataset.

Techniques	Backdoor	DDoS	DoS	Injection	MITM	Normal	Password	Ransomware	Scanning	XSS
BiLSTM	99.89	99.66	99.71	98.35	97.25	100.00	99.98	99.83	100.00	99.94
DT	100.00	100.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00	100.00
RF	99.98	90.40	91.97	93.53	0.00	100.00	97.81	99.40	95.74	85.47
NB	99.22	26.80	91.70	92.96	95.11	100.00	75.32	79.98	96.91	19.02
Proposed IDS	99.90	99.36	99.66	97.12	98.37	99.99	99.68	100.00	100.00	99.93

Table 11
Comparison with other techniques using $\mathcal{D}2$ dataset.

Techniques	BENIGN	DoS Hulk	DDoS	PortScan	DoS GoldenEye	FTPPatator	DoS slowloris	DoS Slowhttptes	SSHPatator	Bot	Web Attack
BiLSTM	99.54	99.40	99.46	79.95	98.64	99.63	97.49	98.56	96.62	34.95	06.52
DT	100.00	90.00	99.00	97.00	66.00	99.00	35.00	0.00	97.00	0.00	0.00
RF	100.00	95.00	100.00	97.00	50.00	72.00	0.00	55.00	0.00	0.00	0.00
NB	55.00	89.00	98.00	50.00	99.00	100.00	60.00	77.00	97.00	76.00	08.00
Proposed IDS	99.09	95.28	84.27	80.82	98.15	75.85	93.95	98.06	96.51	37.39	52.50

terms of multi-class DR is illustrated in Table 10 and Table 11 based on $\mathcal{D}1$ and $\mathcal{D}2$ datasets, respectively. We see that the BiLSTM with $\mathcal{D}1$ dataset has obtained in an average of 97%-100% DR and the proposed IDS has attained DR in an average of 97%-100%. Overall, Table 10 shows that the model can attain higher DR compared to traditional ML models that too without having significant loss in DR. Similarly, with $\mathcal{D}2$ datasets, the proposed IDS has shown higher values, when compared with BiLSTM and other ML techniques. However, Table 11 shows that IDS has achieved higher DR values for Bot and Web attack, when compared with other techniques.

The overall PR, AC, F1 score and DR or Recall obtained using $\mathcal{D}1$ and $\mathcal{D}2$ datasets is compared with the other three ML-based baseline techniques and BiLSTM as demonstrated in Fig. 8 and Fig. 9. With $\mathcal{D}1$ dataset, it is seen that RF has attained PR of 87.55%, AC of 97.81%, 86.41% F1 score and 85.43% DR. On the other hand, for DT the obtained values are 74.42% PR, 95.34% AC, F1 score is 76.33% and DR is 80.00%. Similarly, using NB model, PR is 77.68%, AC is 90.62%, 72.43% is F1 score and 77.70% is DR. Finally, BiLSTM achieved PR of 99.93%, 99.86% AC, F1 score is 99.32% and DR is 99.64%. Similarly, proposed IDS achieved 99.01% PR, 99.03% of AC, F1 score is 97.10% and DR is 96.76%. The results with $\mathcal{D}2$ dataset

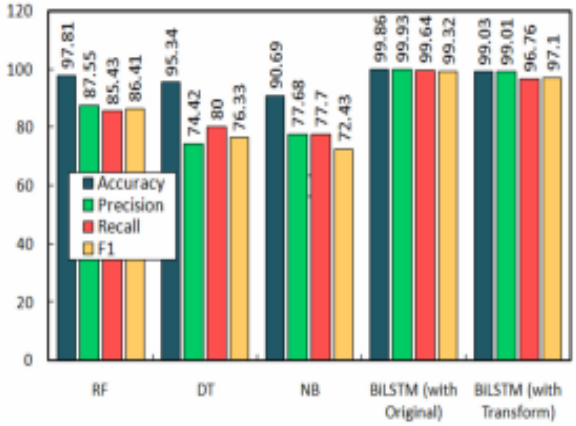


Fig. 8. Performance comparison using $\mathcal{D}1$ dataset.

for RF are as follows: 62.01% PR, AC of 98.62%, 86.41% F1 score and 51.77% DR. The DT model achieved 77.40% PR, 98.34% AC, F1 score of 64.58% and DR 62.16%. The NB model has obtained 45.75%

Table 12
Comparison of proposed approach with some recent approaches.

Authors	Year	Method	Dataset	Blockchain	Accuracy
Keshk et al. [15]	2018	DT	Power Data	No	97.43%
Ferrag et al. [10]	2019	DeliveryCoin	CICIDS-2018	Yes	98.71%
Weng et al. [29]	2019	CNN	MNIST	Yes	97.32%
Derhab et al. [7]	2019	RSL-KNN	Power System	Yes	91.07%
Hasan et al. [14]	2019	RF	DS2OS	No	99.04%
Rathore et al. [25]	2019	DL	NSL-KDD	Yes	91.00%
Liang et al. [20]	2019	DNN	NSL-KDD	Yes	98.00%
Ferrag et al. [11]	2019	DeepCoin	Power System $\mathcal{D}2$ BoT-IoT	Yes	96.52% 98.20% 98.23%
Keshk et al. [16]	2019	PPAD	Power System	No	97.27%
Ghulam et al. [22]	2020	DNN	KDD CUP 99	No	94.20%
Alsaedi et al. [4]	2020	CART	$\mathcal{D}1$	No	77.00%
Keshk et al. [17]	2020	LSTM	Power System	Yes	96.27%
Qiao et al. [23]	2020	DPCA	UNSW-NB15	No	65.21%
Singh et al. [28]	2021	BM	KDD CUP 99	Yes	88.59%
Alkadi et al. [3]	2021	DBF	$\mathcal{D}1$ $\mathcal{D}2$	Yes	98.86% 98.92%
Proposed Work	2022	BDSDT	$\mathcal{D}1$ $\mathcal{D}2$	Yes	99.03% 99.04%

Terms & Abbreviations: DT: Decision Tree, CNN: Convolutional Neural Network, RSL-KNN: Random Subspace learning and K-Nearest Neighbor, RF: Random Forest, PPAD: Privacy-Preserving Anomaly Detection, DL: Deep Learning, DNN: Deep Neural Network, CART: Classification and Regression Trees, BiLSTM: Bidirectional Long Short-Term Memory, LSTM: Long Short Term Memory, BM:Boltzmann Machine, DPCA: Discriminative Principal Component Analysis, DBF: Deep Blockchain Framework.

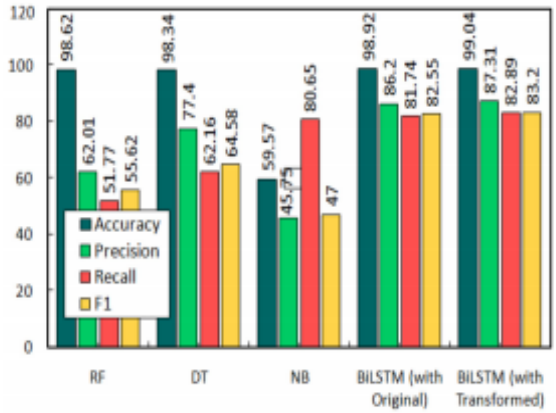


Fig. 9. Performance comparison using $\mathcal{D}2$ dataset.

PR, AC of 59.57%, 47.00% F1 score and 80.65% DR. The BiLSTM with $\mathcal{D}2$ dataset achieved PR of 86.20%, AC of 98.92%, F1 score is 82.55% and DR is 81.74%. Similarly, the proposed IDS has obtained 87.31% PR, 99.04% AC, F1 score is 83.20% and DR is 82.89%. Thus, we can conclude the proposed IDS has shown significant results and performed well compared with other peer intrusion detection approaches.

5.6. Comparisons with state-of-the-art approaches

We have compared the proposed BDSDT performance with some recent state-of-the-art techniques. It is concluded from the Table 12 that these woks mostly used outdated power, NSL-KDD and KDD CUP 99 datasets. Moreover, these data sources has very less influence in designing a modern IDS. Therefore, the proposed approach used two recent publicly available data sources named $\mathcal{D}2$ and $\mathcal{D}1$ for evaluation. It can be seen that the proposed approach has achieved high accuracy with blockchain compared with [15], [10], [29], [7], [14], [16], [25], [20], [11], [22], [4], [17], [28], [23], [3]. In Table 13, the proposed BDSDT framework is compared to several recent research [15], [14], [16], [4], [23], [22], [28], [3] in both non-blockchain and blockchain settings. Security, privacy, scalability and IDS are most important parameters that should be included in the design of modern security frameworks. The proposed BDSDT framework provides security and privacy by integrating blockchain and DL techniques. To make the framework scalable, the IPFS distributed file storage system (also known as off-chain storage) is employed. The second assessment measure is ledger distribution, which refers to each IoT node's ability to reproduce and have the same copy of information in the IoT network's ledger. The proposed ePoW is automatically executed using smart contracts. It generates a block's hash and links it to the next block to preserve the network's transparency and trust. The blockchain network maintains the suggested model's i.e. BDSDT decentralized structure.

6. Conclusion

In this paper, we designed BDSDT, a new secure data transmission application for IoT-enabled healthcare system by integrating blockchain and deep learning techniques. Specifically, BDSDT provides two level architecture to ensure security. A blockchain architecture was presented at the first level, where all IoT devices

Table 13
Comparison of BDSDT and other competing strategies in blockchain and non-blockchain settings.

Authors	Year	1	2	3	4	5	6	7	8	9	10
Keshk et al. [15]	2018	✓	✓	×	✓	×	×	×	×	×	×
Hasan et al. [14]	2019	✓	×	×	✓	×	×	×	×	×	×
Keshk et al. [16]	2019	✓	✓	×	✓	×	×	×	×	×	×
Alsaedi et al. [4]	2020	✓	×	×	✓	×	×	×	×	×	×
Qiao et al. [23]	2020	✓	×	×	✓	×	×	×	×	×	×
Ghulam et al. [22]	2020	✓	×	×	✓	×	×	×	×	×	×
Singh et al. [28]	2021	✓	×	×	✓	✓	×	✓	✓	✓	×
Alkadi et al. [3]	2021	✓	✓	×	✓	✓	✓	✓	✓	✓	×
Proposed (BDSDT)	2022	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

1: Security; *2*: Privacy; *3*: Scalability; *4*: Intrusion Detection System; *5*: Ledger Distribution; *6*: Smart Contracts; *7*: Transparency; *8*: Decentralized; *9*: Trust; *10*: Off-Chain.

were registered, verified (using zero knowledge proof), and then added in the blockchain network utilizing a smart contract-based ePoW consensus. The second level incorporates a deep learning architecture in which the features from original data were extracted using a deep sparse autoencoder technique and were used by the bidirectional long short-term memory to identify intrusions in the network. Furthermore, we use IPFS-based off-chain storage to make BDSDT more scalable. The findings show that the proposed framework outperforms competing strategies in both a blockchain and non-blockchain setting. Future study will comprise deploying a prototype of the proposed model in an IoT-based healthcare context to formally test the framework's efficiency.

CRediT authorship contribution statement

All the authors have participated sufficiently, and take responsibility of the content of the manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] S. Aggarwal, R. Chaudhary, G.S. Aujla, N. Kumar, K.-K.R. Choo, A.Y. Zomaya, Blockchain for smart communities: applications, challenges and opportunities, J. Netw. Comput. Appl. 144 (2019) 13–48.

[2] R.M. Alguliyev, R.M. Aliguliyev, F.J. Abdullayeva, Privacy-preserving deep learning algorithm for big personal data analysis, J. Ind. Inf. Integr. 15 (2019) 1–14.

[3] O. Alkadi, N. Moustafa, B. Turnbull, K.-K.R. Choo, A deep blockchain framework-enabled collaborative intrusion detection for protecting iot and cloud networks, IEEE Int. Things J. 8 (12) (2021) 9463–9472.

[4] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, A. Anwar, Ton_iot telemetry dataset: a new generation dataset of iot and iiot for data-driven intrusion detection systems, IEEE Access 8 (2020) 165130–165150.

[5] G.S. Aujla, A. Jindal, A decoupled blockchain approach for edge-envisioned iot-based healthcare monitoring, IEEE J. Sel. Areas Commun. 39 (2) (2021) 491–499.

[6] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, V. Sassone, Pbft vs proof-of-authority: applying the cap theorem to permissioned blockchain, 2018.

[7] A. Derhab, M. Guerroumi, A. Gumaei, L. Maglaras, M.A. Ferrag, M. Mukherjee, F.A. Khan, Blockchain and random subspace learning-based ids for sdn-enabled industrial iot security, Sensors 19 (14) (2019) 3119.

[8] D. Dolev, A. Yao, On the security of public key protocols, IEEE Trans. Inf. Theory 29 (2) (1983) 198–208.

[9] A. Dua, N. Kumar, A.K. Das, W. Susilo, Secure message communication protocol among vehicles in smart city, IEEE Trans. Veh. Technol. 67 (5) (2017) 4359–4373.

[10] M.A. Ferrag, L. Maglaras, Deliverycoin: an ids and blockchain-based delivery framework for drone-delivered services, Computers 8 (3) (2019) 58.

[11] M.A. Ferrag, L. Maglaras, Deepcoin: a novel deep learning and blockchain-based energy exchange framework for smart grids, IEEE Trans. Eng. Manag. 67 (4) (2019) 1285–1297.

[12] M. Gong, K. Pan, Y. Xie, A.K. Qin, Z. Tang, Preserving differential privacy in deep neural networks with relevance-based adaptive noise imposition, Neural Netw. 125 (2020) 131–141.

[13] R. Gupta, S. Tanwar, F. Al-Turjman, P. Italiya, A. Nauman, S.W. Kim, Smart contract privacy protection using ai in cyber-physical systems: tools, techniques and challenges, IEEE Access 8 (2020) 24746–24772.

[14] M. Hasan, M.M. Islam, M.I.I. Zarif, M. Hashem, Attack and anomaly detection in iot sensors in iot sites using machine learning approaches, Int. Things 7 (2019) 100059.

[15] M. Keshk, N. Moustafa, E. Sitnikova, B. Turnbull, Privacy-preserving big data analytics for cyber-physical systems, Wirel. Netw. (2018) 1–9.

[16] M. Keshk, E. Sitnikova, N. Moustafa, J. Hu, I. Khalil, An integrated framework for privacy-preserving based anomaly detection for cyber-physical systems, IEEE Trans. Sustain. Comput. (2019) 1.

[17] M. Keshk, B. Turnbull, N. Moustafa, D. Vatsalan, K.R. Choo, A privacy-preserving-framework-based blockchain and deep learning for protecting smart power networks, IEEE Trans. Ind. Inform. 16 (8) (2020) 5110–5118.

[18] P. Kumar, G.P. Gupta, R. Tripathi, A distributed ensemble design based intrusion detection system using fog computing to protect the internet of things networks, J. Ambient Intell. Humaniz. Comput. (2020) 1–18.

[19] R. Kumar, P. Kumar, R. Tripathi, G.P. Gupta, T.R. Gadekallu, G. Srivastava, Sp2f: a secured privacy-preserving framework for smart agricultural unmanned aerial vehicles, Comput. Netw. 187 (2021) 107819.

[20] C. Liang, B. Shanmugam, S. Azam, M. Jonkman, F. De Boer, G. Narayansamy, Intrusion detection system for internet of things based on a machine learning approach, in: 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), IEEE, 2019, pp. 1–6.

[21] N. Moustafa, Ton_iot datasets, online, available: https://doi.org/10.21227/fesz-dm97, 2019. (Accessed 10 February 2020).

[22] G. Muhammad, M.S. Hossain, S. Garg, Stacked autoencoder-based intrusion detection system to combat financial fraudulent, IEEE Int. Things J. (2020) 1.

[23] H. Qiao, J.O. Blech, H. Chen, A machine learning based intrusion detection approach for industrial networks, in: 2020 IEEE International Conference on Industrial Technology (ICIT), IEEE, 2020, pp. 265–270.

[24] J. Qiu, D. Grace, G. Ding, J. Yao, Q. Wu, Blockchain-based secure spectrum trading for unmanned-aerial-vehicle-assisted cellular networks: an operator's perspective, IEEE Int. Things J. 7 (1) (2019) 451–466.

[25] S. Rathore, B.W. Kwon, J.H. Park, Blockseciotnet: blockchain-based decentralized security architecture for iot network, J. Netw. Comput. Appl. 143 (2019) 167–177.

[26] I. Sharafaldin, Cic-ids2017 datasets, online, available: http://205.174.165.80/CICDataset/CIC-IDS-2017/Dataset/, 2017. (Accessed 15 March 2019).

[27] I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, in: ICISSp, 2018, pp. 108–116.

[28] M. Singh, G.S. Aujla, A. Singh, N. Kumar, S. Garg, Deep-learning-based blockchain framework for secure software-defined industrial networks, IEEE Trans. Ind. Inform. 17 (1) (2021) 606–616.

[29] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang, W. Luo, Deepchain: auditable and privacy-preserving deep learning with blockchain-based incentive, IEEE Trans. Dependable Secure Comput. (2019) 1.

[30] W. Zhao, C. Jiang, H. Gao, S. Yang, X. Luo, Blockchain-enabled cyber-physical systems: a review, IEEE Int. Things J. (2020).

[31] M. Zolanvari, M.A. Teixeira, L. Gupta, K.M. Khan, R. Jain, Machine learning-based network vulnerability analysis of industrial internet of things, IEEE Int. Things J. 6 (4) (2019) 6822–6834.