National College of Ireland

Configuration Manual

MSc Research Project
MSC_DAD_A

Aryan Nagnath Salge
Student ID: x23268018

School of Computing
National College of Ireland

Supervisor: Dr. David Hamill

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	ARYAN NAGNATH SALGE
Student ID:	X23268018
Programme:	MSC_DAD_A **Year:** 2024-2025
Module:	RESEARCH PRACTICUM
Lecturer:	PROF. DAVID HAMILL
Submission Due Date:	11/08/2025
Project Title:	Federated Transfer Learning for Cross-Institution Fraud Detection in Finance
Word Count:	547 **Page Count:** 8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ARYAN SALGE

Date: 10/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aryan Nagnath Salge
Student ID: X23268018

1 Introduction

This document provides a detailed, step-by-step guide to the environment, tools, datasets, and execution process required to replicate the research thesis of Federated Transfer Learning for Cross-Institution Fraud Detection in Finance. The complete project is implemented in Python, using libraries such as scikit-learn, pandas, and matplotlib, and is executed within a virtual environment using Visual Studio Code and Jupyter Notebook. This manual outlines the exact system configuration, dataset descriptions, preprocessing techniques, model design, and commands used to run the code. It is intended to ensure full reproducibility, enabling anyone to set up the required environment, follow the execution steps, and obtain near-identical results.

2 System Specification

This section gives basic hardware and operating system requirements for this project:

2.1 Hardware

Machine: Apple MacBook Air (M1, 2020)
Processor: Apple M1 chip (ARM architecture, 8-core CPU)
RAM: 8 GB Unified Memory

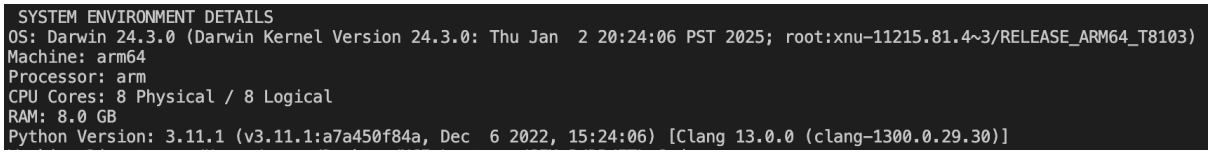

Fig 1: System Information

2.2 Software

Operating System: macOS Sequoia 15.3.1 (Darwin 24.3.0)
Python Version: 3.11.1 (Clang 13.0.0)
IDE: Visual Studio Code
Virtual Environment: Python venv
Dependencies:

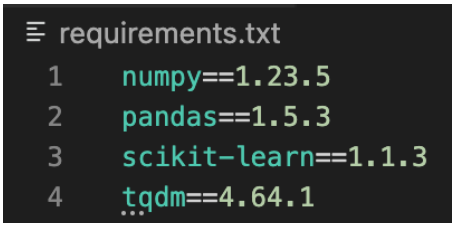

Fig 2: Python Dependencies

3 Development Environment

Python language was utilized to make the research experimentation and below are the details of the environments used:

3.1 Visual Studio Code

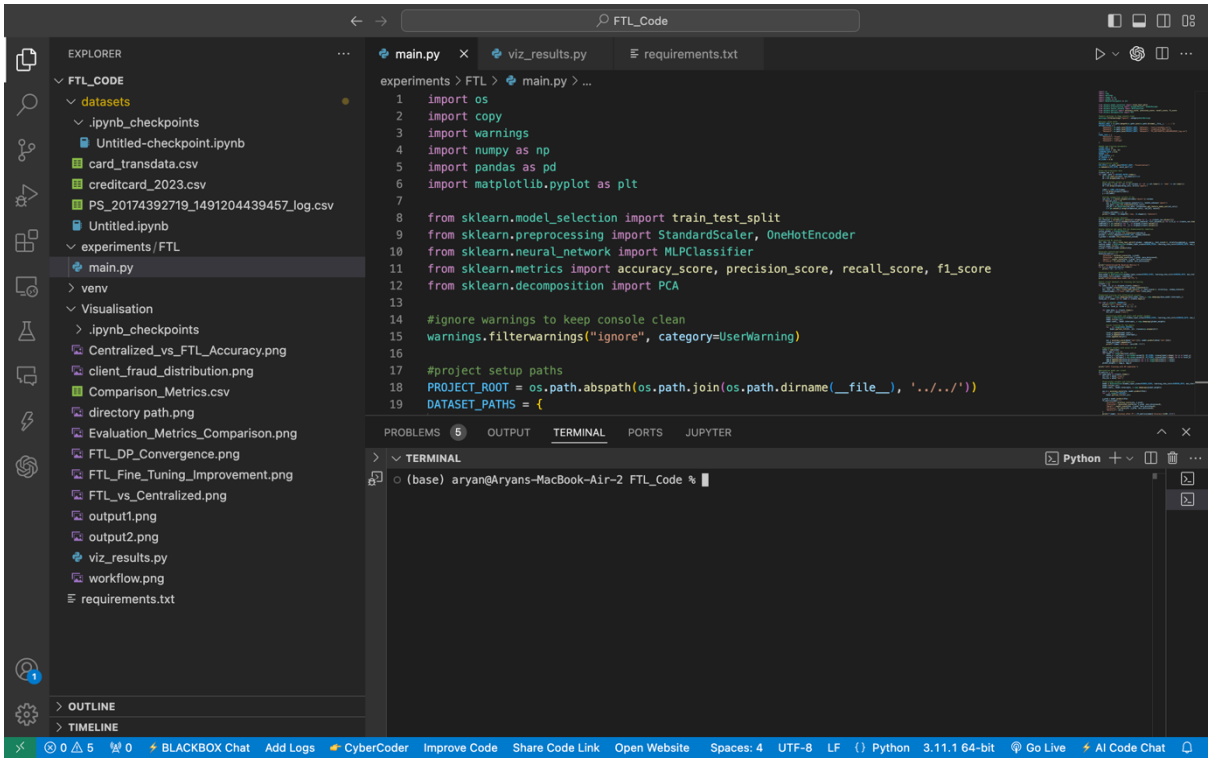

Fig 3: Visual Studio Code

4 Repository Structure

Folder Structure Diagram:

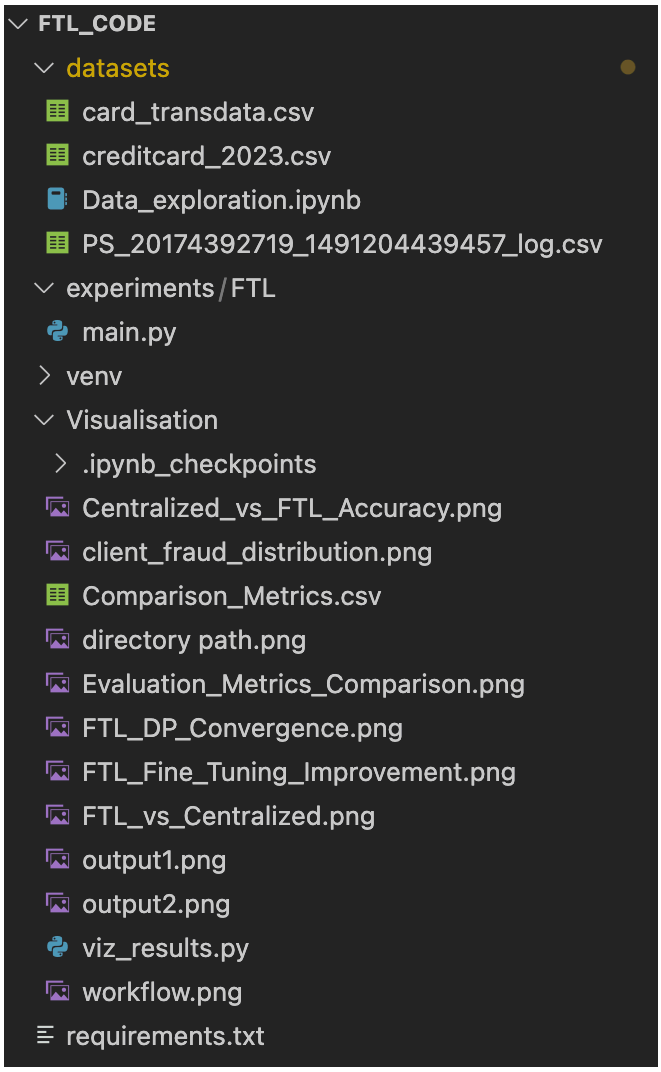

Fig 4: File Directory Path

5 Datasets

5.1 Dataset Exploration

Dataset exploration was done on Jupyter Notebook with code

```python
import pandas as pd

dataset_paths = {
    "card_transdata.csv": "card_transdata.csv",
    "creditcard_2023.csv": "creditcard_2023.csv",
    "PS_20174392719_1491204439457_log.csv": "PS_20174392719_1491204439457_log.csv"
}

for name, path in dataset_paths.items():
    print(f"\n{name}")
    try:
        df = pd.read_csv(path, low_memory=False)
        print("Shape:", df.shape)

        dtype_groups = df.dtypes.groupby(df.dtypes).apply(lambda x: list(x.index))
        print("\nColumn Groups by Data Type:")
        for dtype, cols in dtype_groups.items():
            print(f"{dtype}: {cols}")

        missing_counts = df.isnull().sum()
        no_missing = missing_counts[missing_counts == 0].index.tolist()
        missing_some = missing_counts[missing_counts > 0]
        if len(missing_some) > len(df.columns) / 2:
            print("\nMost columns have missing values.")
        if no_missing:
            print("Columns with No Missing Values:", no_missing)
        if not missing_some.empty:
            print("Top Columns with Missing Values:", missing_some.sort_values(ascending=False).head(5).to_dict())

        print("\nDuplicate Rows:", df.duplicated().sum())
        print("Unique Values (first 10 columns):", df.nunique().head(10).to_dict())
        print("\nSample Data:\n", df.head(3))

    except FileNotFoundError:
        print(f"File '{path}' not found.")
```

Dataset 1:

```
card_transdata.csv
Shape: (1000000, 8)

Column Groups by Data Type:
float64: ['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price', 'repeat_retailer', 'used_chip', 'used_pin
_number', 'online_order', 'fraud']
Columns with No Missing Values: ['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price', 'repeat_retailer',
'used_chip', 'used_pin_number', 'online_order', 'fraud']

Duplicate Rows: 0
Unique Values (first 10 columns): {'distance_from_home': 1000000, 'distance_from_last_transaction': 1000000, 'ratio_to_median_purchase_pric
e': 1000000, 'repeat_retailer': 2, 'used_chip': 2, 'used_pin_number': 2, 'online_order': 2, 'fraud': 2}

Sample Data:
    distance_from_home  distance_from_last_transaction  \
0           57.877857                        0.311140
1           10.829943                        0.175592
2            5.091079                        0.805153

   ratio_to_median_purchase_price  repeat_retailer  used_chip  \
0                        1.945940              1.0        1.0
1                        1.294219              1.0        0.0
2                        0.427715              1.0        0.0

   used_pin_number  online_order  fraud
0              0.0           0.0    0.0
1              0.0           0.0    0.0
2              0.0           1.0    0.0
```

Dataset 2:

```
creditcard_2023.csv
Shape: (568630, 31)

Column Groups by Data Type:
int64: ['id', 'Class']
float64: ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount']
Columns with No Missing Values: ['id', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16',
'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Class']

Duplicate Rows: 0
Unique Values (first 10 columns): {'id': 568630, 'V1': 552035, 'V2': 552035, 'V3': 552035, 'V4': 552035, 'V5': 552035, 'V6': 552035, 'V7': 55
2035, 'V8': 552035, 'V9': 552035}

Sample Data:
    id        V1        V2        V3        V4        V5        V6        V7  \
0   0 -0.260648 -0.469648  2.496266 -0.083724  0.129681  0.732898  0.519014
1   1  0.985100 -0.356045  0.558056 -0.429654  0.277140  0.428605  0.406466
2   2 -0.260272 -0.949385  1.728538 -0.457986  0.074062  1.419481  0.743511

         V8        V9  ...       V21       V22       V23       V24       V25  \
0 -0.130006  0.727159  ... -0.110552  0.217606 -0.134794  0.165959  0.126280
1 -0.133118  0.347452  ... -0.194936 -0.605761  0.079469 -0.577395  0.190090
2 -0.095576 -0.261297  ... -0.005020  0.702906  0.945045 -1.154666 -0.605564

        V26       V27       V28    Amount  Class
0 -0.434824 -0.081230 -0.151045  17982.10      0
1  0.296503 -0.248052 -0.064512   6531.37      0
2 -0.312895 -0.300258 -0.244718   2513.54      0

[3 rows x 31 columns]
```

Dataset 3:

```
PS_20174392719_1491204439457_log.csv
Shape: (1048562, 11)

Column Groups by Data Type:
float64: ['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', 'isFraud', 'isFlaggedFraud']
object: ['type', 'nameOrig', 'nameDest']

Most columns have missing values.
Top Columns with Missing Values: {'step': 1038562, 'type': 1038562, 'amount': 1038562, 'nameOrig': 1038562, 'oldbalanceOrg': 1038562}

Duplicate Rows: 1038561
Unique Values (first 10 columns): {'step': 7, 'type': 5, 'amount': 9954, 'nameOrig': 10000, 'oldbalanceOrg': 7276, 'newbalanceOrig': 5914, 'n
ameDest': 6397, 'oldbalanceDest': 4317, 'newbalanceDest': 1501, 'isFraud': 2}

Sample Data:
    step      type   amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
0   1.0   PAYMENT  9839.64  C1231006815       170136.0       160296.36
1   1.0   PAYMENT  1864.28  C1666544295        21249.0        19384.72
2   1.0  TRANSFER   181.00  C1305486145          181.0            0.00

      nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
0  M1979787155             0.0             0.0      0.0             0.0
1  M2044282225             0.0             0.0      0.0             0.0
2   C553264065             0.0             0.0      1.0             0.0
```

6 Modelling

main.py viz_results.py requirements.txt

experiments > FTL > main.py > ...

```python
import os
import copy
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.decomposition import PCA

#ignore warnings to keep console clean
warnings.filterwarnings("ignore", category=UserWarning)

#project setup paths
PROJECT_ROOT = os.path.abspath(os.path.join(os.path.dirname(__file__), '../../'))
DATASET_PATHS = {
    "Dataset1": os.path.join(PROJECT_ROOT, "datasets", "card_transdata.csv"),
    "Dataset2": os.path.join(PROJECT_ROOT, "datasets", "creditcard_2023.csv"),
    "Dataset3": os.path.join(PROJECT_ROOT, "datasets", "PS_20174392719_1491204439457_log.csv")
}
LABEL_COLS = {
    "Dataset1": "fraud",
    "Dataset2": "Class",
    "Dataset3": "isFraud"
}

#model and training parameters
LATENT_DIM = 20
HIDDEN_SIZES = (64, 32)
LEARNING_RATE = 0.01
ROUNDS = 8
LOCAL_EPOCHS = 3
FT_EPOCHS = 2
DP_SIGMA = 0.01
```

Fig. 5: Project bootstrap: imports, dataset paths, label columns, and core hyperparameters.

```python
#visualization folder
VIS_PATH = os.path.join(PROJECT_ROOT, "Visualisation")
os.makedirs(VIS_PATH, exist_ok=True)

#load and preprocess data
clients_raw = {}
for name, path in DATASET_PATHS.items():
    df = pd.read_csv(path, low_memory=False)
    df = df.dropna(how='any')

    #drop id/name columns if present
    drop_cols = [col for col in df.columns if 'id' in col.lower() or 'name' in col.lower()]
    df = df.drop(columns=drop_cols, errors='ignore')

    label = LABEL_COLS[name]
    X = df.drop(columns=[label])
    y = df[label]

    #encode categorical columns if any
    cat_cols = X.select_dtypes(include=['object']).columns
    if len(cat_cols) > 0:
        enc = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
        cat_data = enc.fit_transform(X[cat_cols])
        cat_df = pd.DataFrame(cat_data, columns=enc.get_feature_names_out(cat_cols))
        X = pd.concat([X.drop(columns=cat_cols), cat_df], axis=1)

    clients_raw[name] = (X, y)
    print(f"{name}: {X.shape[0]} rows, {X.shape[1]} features")

#align features across datasets
all_features = sorted(set().union(*[X.columns for X,_ in clients_raw.values()]))
aligned_clients = {n:(X.reindex(columns=all_features, fill_value=0),y) for n,(X,y) in clients_raw.items()}
combined_X = pd.concat([X for X,_ in aligned_clients.values()])
combined_y = pd.concat([y for _,y in aligned_clients.values()])

#scale features and apply PCA for dimensionality reduction
scaler_global = StandardScaler()
X_scaled = scaler_global.fit_transform(combined_X)
```

Fig. 6: Data loading, cleansing, encoding, and cross-client feature alignment.

```python
encoder = PCA(n_components=LATENT_DIM, random_state=42)
Z_global = encoder.fit_transform(X_scaled)

#centralized ML baseline
Xtr, Xte, ytr, yte = train_test_split(Z_global, combined_y, test_size=0.2, stratify=combined_y, random_state=42)
central_model = MLPClassifier(hidden_layer_sizes=HIDDEN_SIZES, learning_rate_init=LEARNING_RATE, max_iter=20)
central_model.fit(Xtr, ytr)
y_pred = central_model.predict(Xte)

#evaluate centralized model
baseline_metrics = {
    "Accuracy": accuracy_score(yte, y_pred),
    "Precision": precision_score(yte, y_pred, zero_division=0),
    "Recall": recall_score(yte, y_pred, zero_division=0),
    "F1-Score": f1_score(yte, y_pred, zero_division=0)
}
print("\nCentralized ML Baseline Metrics:")
for k,v in baseline_metrics.items():
    print(f"{k}: {v:.4f}")

#pretrain global model for FTL
base_model = MLPClassifier(hidden_layer_sizes=HIDDEN_SIZES, learning_rate_init=LEARNING_RATE, max_iter=20)
base_model.fit(Z_global, combined_y)
print("\nPretrained base model for FTL.")

#split client datasets for training and testing
clients = {}
for name, (X, y) in aligned_clients.items():
    Z = encoder.transform(scaler_global.transform(X))
    Xtr, Xte, ytr, yte = train_test_split(Z, y, test_size=0.2, stratify=y, random_state=42)
    clients[name] = {'train':(Xtr,ytr),'test':(Xte,yte)}

#federated training with differential privacy
global_weights = (copy.deepcopy(base_model.coefs_), copy.deepcopy(base_model.intercepts_))
round_acc = {name: [] for name in clients.keys()}

for rnd in range(1, ROUNDS+1):
    print(f"\n---- Round {rnd} ----")
```

Fig.7: Global scaling/PCA, centralized baseline training, and global model pretraining before FL

```python
    local_w, local_b, sizes = [], [], []

    for name,data in clients.items():
        Xtr,ytr = data['train']

        #initialize model and start with global weights
        model = MLPClassifier(hidden_layer_sizes=HIDDEN_SIZES, learning_rate_init=LEARNING_RATE, max_iter=1, warm_start=True)
        model.fit(Xtr,ytr)
        model.coefs_, model.intercepts_ = copy.deepcopy(global_weights)

        #local training for few epochs
        for _ in range(LOCAL_EPOCHS):
            model.partial_fit(Xtr, ytr, classes=np.unique(ytr))

        local_w.append(model.coefs_)
        local_b.append(model.intercepts_)
        sizes.append(len(ytr))

        acc = accuracy_score(data['test'][1], model.predict(data['test'][0]))
        round_acc[name].append(acc)
        print(f"{name} accuracy: {acc*100:.2f}%")

    #aggregate weights with noise for DP
    total = sum(sizes)
    agg_w, agg_b = [], []
    for layer in range(len(local_w[0])):
        noise_w = [w[layer] + np.random.normal(0, DP_SIGMA, size=w[layer].shape) for w in local_w]
        noise_b = [b[layer] + np.random.normal(0, DP_SIGMA, size=b[layer].shape) for b in local_b]
        agg_w.append(sum(noise_w[i]*sizes[i] for i in range(len(sizes))) / total)
        agg_b.append(sum(noise_b[i]*sizes[i] for i in range(len(sizes))) / total)
    global_weights = (agg_w, agg_b)

print("\nFTL Training with DP completed.")

#personalize model per client
ft_metrics = {}
for name,data in clients.items():
    Xtr,ytr = data['train']
```

Fig. 8: Federated training rounds with per-client local updates and differentially private aggregation.

```
    Xte,yte = data['test']

    #load global weights and fine-tune
    model = MLPClassifier(hidden_layer_sizes=HIDDEN_SIZES, learning_rate_init=LEARNING_RATE, max_iter=1, warm_start=True)
    model.fit(Xtr,ytr)
    model.coefs_, model.intercepts_ = copy.deepcopy(global_weights)

    acc_b = accuracy_score(yte, model.predict(Xte))
    for _ in range(FT_EPOCHS):
        model.partial_fit(Xtr,ytr)

    y_pred = model.predict(Xte)
    ft_metrics[name] = {
        "Accuracy": accuracy_score(yte, y_pred),
        "Precision": precision_score(yte, y_pred, zero_division=0),
        "Recall": recall_score(yte, y_pred, zero_division=0),
        "F1-Score": f1_score(yte, y_pred, zero_division=0),
        "Before_FT": acc_b
    }
    print(f"{name}: Accuracy after FT = {ft_metrics[name]['Accuracy']*100:.2f}%")

#save results and plot graphs
metrics_df = pd.DataFrame(ft_metrics).T
metrics_df.loc['Centralized_ML'] = baseline_metrics
metrics_df = metrics_df.fillna('-')
metrics_df.to_csv(os.path.join(VIS_PATH, "Comparison_Metrics.csv"))
print("\nMetrics saved to Visualisation/Comparison_Metrics.csv")
print(metrics_df)

#plot accuracy convergence per round
plt.figure(figsize=(8,5))
for name, accs in round_acc.items():
    plt.plot(range(1,ROUNDS+1), np.array(accs)*100, marker='o', label=name)
plt.xlabel("Round")
plt.ylabel("Accuracy (%)")
plt.title("FTL with DP - Convergence per Client")
plt.legend()
plt.savefig(os.path.join(VIS_PATH,"FTL_DP_Convergence.png"), dpi=300)
```

Fig. 9: Per-client fine-tuning (FT) of the global model and metric collection.

```
plt.close()

#plot final accuracy comparison
plt.figure(figsize=(6,4))
labels = list(clients.keys()) + ["Centralized_ML"]
acc_values = [metrics_df.loc[n,"Accuracy"]*100 for n in clients] + [baseline_metrics["Accuracy"]*100]
plt.bar(labels, acc_values)
plt.ylabel("Accuracy (%)")
plt.title("Centralized vs FTL (DP) Accuracy Comparison")
plt.savefig(os.path.join(VIS_PATH,"FTL_vs_Centralized.png"), dpi=300)
plt.close()

print("Plots saved in Visualisation folder.")
```

Fig. 10: Final accuracy comparison: each client (after FT) vs centralized baseline.

7 Implementation

1. **Imports & Setup** - Loads required Python libraries, sets dataset paths and label columns, and defines model training hyperparameters.
2. **Data Preprocessing** - Reads datasets, drops irrelevant id/name columns, encodes categorical features, aligns feature sets across clients, and prepares combined data.
3. **Global Scaling & PCA** - Standardizes features, applies PCA for dimensionality reduction, trains a centralized baseline MLP model, and pretrains a global model for FL.
4. **Federated Training with DP** - Performs multiple FL rounds where each client trains locally, adds differential privacy noise to weights, and aggregates updates.

5. **Personalization (Fine-Tuning)** - Fine-tunes the global model on each client's local data, evaluates metrics, and stores results.
6. **Visualization** - Saves comparison metrics to CSV and generates convergence and accuracy comparison plots.

8 Execution

8.1 1. Activate virtual environment

```
# macOS/Linux
source venv/bin/activate

# Windows
venv\Scripts\activate
```

8.2 Run the main training pipeline:

```
python experiments/FTL/main.py
```

This script will:

- Load and preprocess all datasets.
- Train the centralized baseline model.
- Execute the Federated Transfer Learning (FTL) process with Differential Privacy.
- Perform fine-tuning for each client.
- Save results (Comparison_Metrics.csv) and core plots to the Visualization.

8.3 Generate visualizations from results:

```
python Visualisation/viz_results.py
```

This script will:

- Read Comparison_Metrics.csv.
- Create and save additional plots, including:
    1. Centralized vs FTL Accuracy Comparison.
    2. Fine-Tuning Improvement per Client.
    3. Evaluation Metrics Comparison.
    4. FTL Convergence per Round.
    5. Confusion matrices (if prediction files are available).

8.4 Deactivate the virtual environment when finished:

```
deactivate
```

9 Outputs

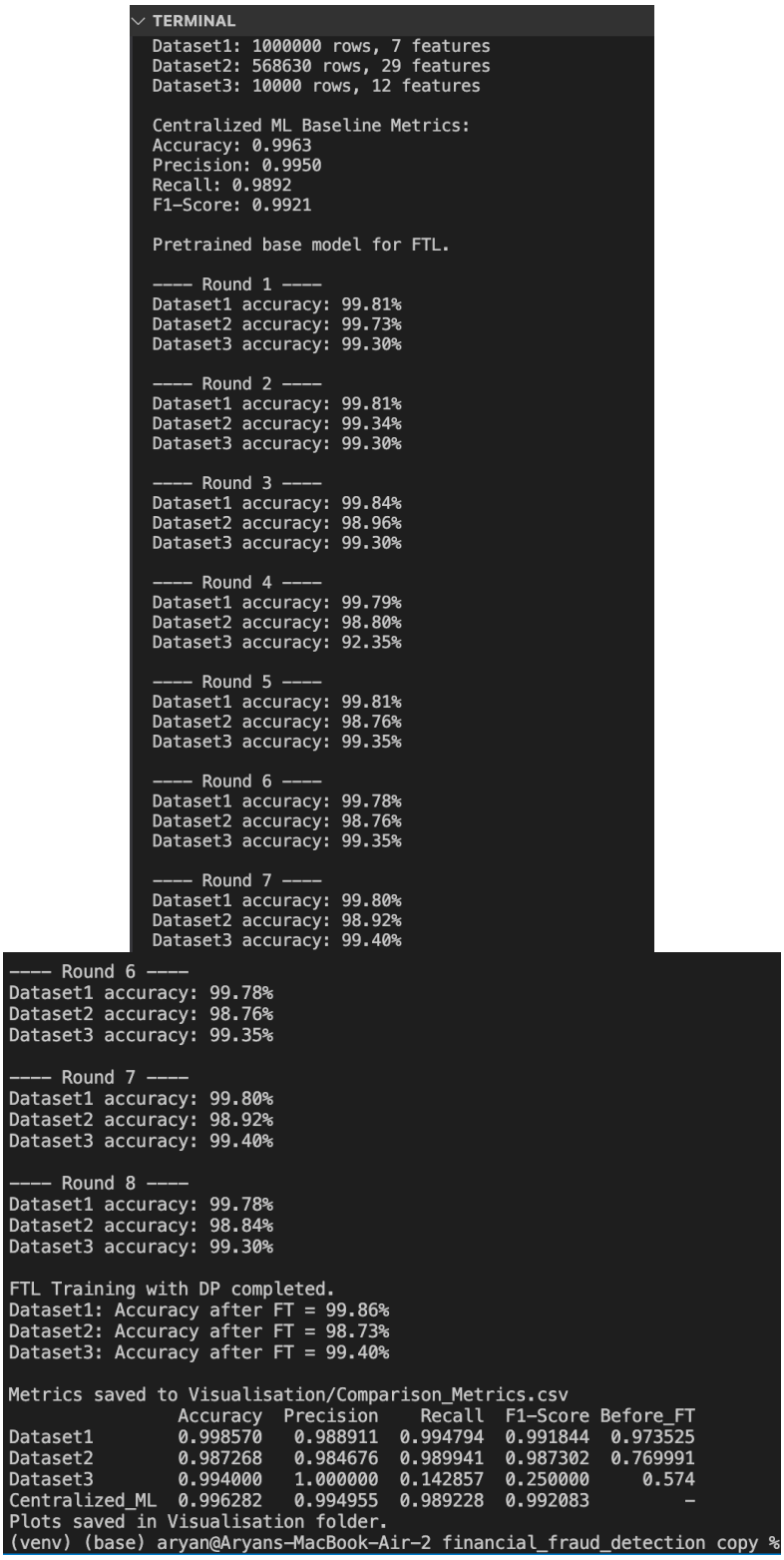

Fig. 11: Round wise dataset specific Accuracy

	Accuracy	Precision	Recall	F1-Score	Before_FT
Dataset1	0.99857	0.988911	0.99479435	0.99184395	0.973525
Dataset2	0.98726764	0.98467647	0.98994073	0.98730159	0.76999103
Dataset3	0.994	1	0.14285714	0.25	0.574
Centralized_	0.99628159	0.99495462	0.98922765	0.99208287	-

Fig. 12: Comparison_Metrics.csv output