

# FL on Credit Card Fraud Detection

## Practical implementation and analysis on various financial related Datasets

### Contents

1. Introduction .....	2
2. Datasets and Preprocessing.....	3
2.1 credit_application_data.csv .....	3
2.2 creditcard_2023.csv .....	3
2.3 creditcard.csv .....	4
2.4 SAML_D.csv .....	4
Preprocessing Overview:.....	5
3. Federated Learning Model Setup .....	7
4. Results and Analysis.....	10
5. Summary of Results .....	12
6. Future Work .....	13
7. Appendix: .....	14
8. Graphical Summary of All Metrics .....	15
9. Application to Credit Card Fraud Detection and Financial Services .....	17
10. Conclusion .....	19

# 1. Introduction

Credit card fraud is a major concern in today's digital world, leading to billions of dollars in financial losses each year. Traditional fraud detection systems often require sharing sensitive data across organizations, which increases the risk of privacy violations and data breaches. Federated Learning (FL) offers a solution by allowing organizations to collaborate on improving fraud detection models without exchanging private customer data.

This project focuses on applying FL to detect fraud across four datasets related to credit card transactions and money laundering. The FL approach ensures that only model updates are shared, maintaining privacy while enhancing detection capabilities. This is particularly valuable for financial institutions, where privacy is critical.

This report covers:

1. **Datasets and Preprocessing:** Overview of the datasets and the steps taken to prepare the data for model training.
2. **Federated Learning Setup:** Explanation of the model architecture and FL setup used to train models across different clients.
3. **Results and Analysis:** Evaluation of key metrics such as Accuracy, Precision, Recall, and F1-Score, comparing centralized and federated setups.
4. **Summary of Results:** Highlights how each dataset performed using FL.
5. **Future Work:** Suggestions for improving model performance and enhancing privacy.

FL offers a promising solution for fraud detection in the financial sector by maintaining data privacy while enabling collaboration. This report demonstrates how FL can be applied effectively in real-world financial services, particularly for credit card fraud detection.

## 2. Datasets and Preprocessing

### 2.1 credit\_application\_data.csv

- **Description:** This dataset contains loan application information with features like income, credit, and loan status.
- **Dataset Link:** <https://www.kaggle.com/datasets/mishra5001/credit-card>
- **Columns used for Testing:**
  - TARGET
  - AMT\_CREDIT
  - AMT\_INCOME\_TOTAL

```
credit_application_data.csv
1 SK_ID_CURR,TARGET,NAME_CONTRACT_TYPE,CODE_GENDER,FLAG_OWN_CAR,FLAG_OWN_REALTY,CNT_CHILDREN,AMT_INCOME_TOTAL,AMT_CREDIT,AMT_ANNUITY,AMT_GOODS_PRICE,NAME_TYPE_SUITE,NAME_INCO
2 100002,1,Cash loans,M,N,Y,0,202500.0,406597.5,24700.5,351000.0,Unaccompanied,Working,Secondary / secondary special,Single / not married,House / apartment,0.018801,-9461,-63
3 100003,0,Cash loans,F,N,N,0,270000.0,1293502.5,35698.5,1129500.0,Family,State servant,Higher education,Married,House / apartment,0.003540999999999999,-16765,-1188,-1186.0,-
4 100004,0,Revolving loans,M,Y,Y,0,67500.0,135000.0,6750.0,135000.0,Unaccompanied,Working,Secondary / secondary special,Single / not married,House / apartment,0.010032,-19046
5 100006,0,Cash loans,F,N,Y,0,135000.0,312682.5,29686.5,297000.0,Unaccompanied,Working,Secondary / secondary special,Civil marriage,House / apartment,0.008019,-19005,-3039,-9
6 100007,0,Cash loans,M,N,Y,0,121500.0,513000.0,21865.5,513000.0,Unaccompanied,Working,Secondary / secondary special,Single / not married,House / apartment,0.028663,-19932,-3
7 100008,0,Cash loans,M,N,Y,0,99000.0,490495.5,27517.5,454500.0,"Spouse, partner",State servant,Secondary / secondary special,Married,House / apartment,0.035792000000000004,-
8 100009,0,Cash loans,F,Y,Y,1,171000.0,1560726.0,41301.0,1395000.0,Unaccompanied,Commercial associate,Higher education,Married,House / apartment,0.035792000000000004,-13778,-
9 100010,0,Cash loans,M,N,Y,0,360000.0,1530000.0,42075.0,1530000.0,Unaccompanied,State servant,Higher education,Married,House / apartment,0.0031219999999999998,-18850,-449,-4
10 100011,0,Cash loans,F,N,Y,0,112500.0,1019610.0,33826.5,913500.0,Children,Pensioner,Secondary / secondary special,Married,House / apartment,0.018634,-20099,365243,-7427.0,-3
```

### 2.2 creditcard\_2023.csv

- **Description:** A dataset containing credit card transactions to detect fraud.
- **Dataset Link:** <https://www.kaggle.com/datasets/nelgiriyeewithana/credit-card-fraud-detection-dataset-2023>
- **Columns used for Testing:**
  - Class
  - V1
  - V2
  - Amount

```
creditcard_2023.csv
1 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
2 0,-0.26064780489439815,-0.46964845005363426,2.4962660026315637,-0.00372301267814633,0.1296812361545678,0.7328982498449426,0.5190136179018007,-0.13000604758867731,0.72715926
3 1,0.9850997342386376,-0.3560450929163436,0.5580563509382045,-0.4296539034065106,0.2771402629466986,0.4286045153379263,0.4064660422512956,-0.13311827417649086,0.347451895176
4 2,-0.26827161274297056,-0.9493846066454119,1.7285377761514877,-0.4579862888424837,0.07406165434922213,1.4194811432767418,0.7435110747693963,-0.09557601337146092,-0.26129661
5 3,-0.15215210191356718,-0.508958707673651,1.746840058804548,-1.090177941714601,0.2494857726542817,1.1433122633143087,0.5182685727677246,-0.0651299167095495,-0.2056976045189
6 4,-0.2068195207397724,-0.16528020377717972,1.5270526784614766,-0.44829266295851267,0.10612511416543663,0.5305488615008258,0.658849134344094,-0.21266001147475838,1.049920839
7 5,0.02530229184371973,-0.14051381069867852,1.1911377729432047,-0.7079788118561594,0.4304903210757426,0.45897319169566103,0.611049586735272,-0.09262860612154547,0.1008113568
8 6,1.0164817337746619,-0.3971805392710764,0.40786760895018615,-0.14446279371757967,0.3310218239728978,0.6292427694008561,0.4312624487933091,-0.13400745217689292,0.79615902919
9 7,-0.05130608522134982,-0.007194479368535587,1.1399408097130955,-0.8778799020880308,0.6846677833552468,0.7143261109667721,0.8926151159502262,-0.9084091416596092,0.901938333
10 8,-0.13068042969174304,-0.3495468967868306,0.4257863396969593,-0.7604440582393063,1.7027774311082158,2.3248160166720564,0.5689684197676761,0.049099674946254994,0.2731177182
```

## 2.3 creditcard.csv

- **Description:** Another credit card transaction dataset with a focus on fraud detection.
- **Dataset Link:** <https://www.kaggle.com/datasets/shayannaveed/credit-card-fraud-detection>
- **Columns used for Testing:**
  - isFraud
  - type
  - amount

```
creditcard.csv
1  step,type,amount,nameOrig,oldbalanceOrg,newbalanceOrig,nameDest,oldbalanceDest,newbalanceDest,isFraud,isFlaggedFraud
2  1,PAYMENT,9839.64,C1231006815,170136.0,160296.36,M1979787155,0.0,0.0,0,0
3  1,PAYMENT,1864.28,C1666544295,21249.0,19384.72,M2044282225,0.0,0.0,0,0
4  1,TRANSFER,181.0,C1305486145,181.0,0.0,C553264065,0.0,0.0,1,0
5  1,CASH_OUT,181.0,C840083671,181.0,0.0,C38997010,21182.0,0.0,1,0
6  1,PAYMENT,11668.14,C2048537720,41554.0,29885.86,M1230701703,0.0,0.0,0,0
7  1,PAYMENT,7817.71,C90045638,53860.0,46042.29,M573487274,0.0,0.0,0,0
8  1,PAYMENT,7107.77,C154988899,183195.0,176087.23,M408069119,0.0,0.0,0,0
9  1,PAYMENT,7861.64,C1912850431,176087.23,168225.59,M633326333,0.0,0.0,0,0
10 1,PAYMENT,4024.36,C1265012928,2671.0,0.0,M1176932104,0.0,0.0,0,0
11 1,DEBIT,5337.77,C712410124,41720.0,36382.23,C195600860,41898.0,40348.79,0,0
```

## 2.4 SAML\_D.csv

- **Description:** A dataset focusing on cross-border payments, used to detect money laundering activities.
- **Dataset Link:** <https://www.kaggle.com/datasets/berkanoztas/synthetic-transaction-monitoring-dataset-aml>
- **Columns used for Testing:**
  - Sender\_account
  - Receiver\_account
  - Amount
  - Payment\_type

```
SAML_D.csv
1  Time,Date,Sender_account,Receiver_account,Amount,Payment_currency,Received_currency,Sender_bank_location,Receiver_bank_location,Payment_type,Is_laundering,Laundering_type
2  10:35:19,2022-10-07,8724731955,2769355426,1459.15,UK_pounds,UK_pounds,UK,UK,Cash_Deposit,0,Normal_Cash_Deposits
3  10:35:20,2022-10-07,1401989064,8401255335,6019.64,UK_pounds,Dirham,UK,UAE,Cross-border,0,Normal_Fan_Out
4  10:35:20,2022-10-07,287305149,4404767002,14328.44,UK_pounds,UK_pounds,UK,UK,Cheque,0,Normal_Small_Fan_Out
5  10:35:21,2022-10-07,5376652437,9600420220,11895.0,UK_pounds,UK_pounds,UK,UK,ACH,0,Normal_Fan_In
6  10:35:21,2022-10-07,9614186178,3803336972,115.25,UK_pounds,UK_pounds,UK,UK,Cash_Deposit,0,Normal_Cash_Deposits
7  10:35:21,2022-10-07,8974559268,3143547511,5130.99,UK_pounds,UK_pounds,UK,UK,ACH,0,Normal_Group
8  10:35:23,2022-10-07,980191499,8577635959,12176.52,UK_pounds,UK_pounds,UK,UK,ACH,0,Normal_Small_Fan_Out
9  10:35:23,2022-10-07,8057793308,9350806213,56.9,UK_pounds,UK_pounds,UK,UK,Credit_card,0,Normal_Small_Fan_Out
10 10:35:26,2022-10-07,6116657264,656192169,4738.45,UK_pounds,UK_pounds,UK,UK,Cheque,0,Normal_Fan_Out
```



## Preprocessing Overview:

For each dataset, the following steps were applied:

1. **Scaling:** We used MinMaxScaler to normalize the values between 0 and 1, which helps improve model performance.
2. **Handling Class Imbalance:** SMOTE (Synthetic Minority Oversampling Technique) was used to address the class imbalance in each dataset, which is crucial for improving the detection of fraud cases.
3. **Train-Test Split:** The datasets were split into training (90%) and testing (10%) subsets for model evaluation.

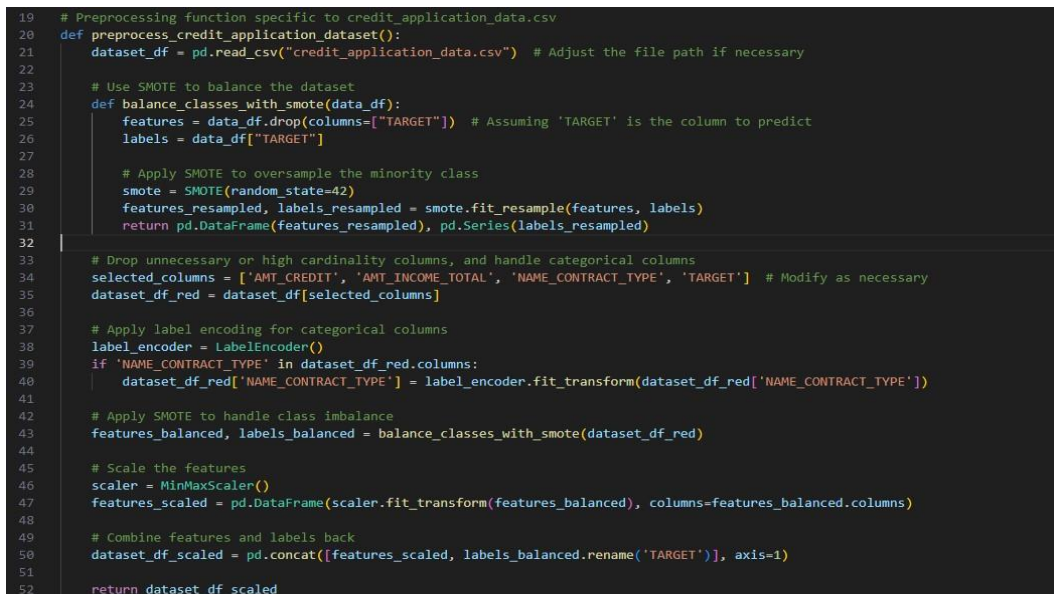
Here's a simplified example of the preprocessing code applied across all datasets:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Preprocessing function for one dataset
def preprocess_dataset(dataset_path, columns):
    df = pd.read_csv(dataset_path)
    # Select relevant columns
    df_red = df[columns]
    # One-hot encoding (if necessary) and scaling
    df_encoded = pd.get_dummies(df_red)
    scaler = MinMaxScaler()
    df_scaled = pd.DataFrame(scaler.fit_transform(df_encoded),
                             columns=df_encoded.columns)
    # Split into train and test
    train_df, test_df = train_test_split(df_scaled, test_size=0.1, random_state=42)
    return train_df, test_df
```

This preprocessing ensures that the datasets are in the right format for the federated learning models, with normalized values and balanced class distributions.

### Screenshot of the preprocessing code:

A screenshot of a code editor showing Python code for preprocessing a dataset. The code includes imports for pandas, sklearn.preprocessing, and sklearn.model\_selection. It defines a function preprocess\_credit\_application\_dataset() which reads a CSV file, selects columns, applies SMOTE for class balancing, handles categorical variables with LabelEncoder, scales features with MinMaxScaler, and finally splits the data into train and test sets. The code is well-commented and includes line numbers from 19 to 52.

```
19 # Preprocessing function specific to credit_application_data.csv
20 def preprocess_credit_application_dataset():
21     dataset_df = pd.read_csv("credit_application_data.csv") # Adjust the file path if necessary
22
23     # Use SMOTE to balance the dataset
24     def balance_classes_with_smote(data_df):
25         features = data_df.drop(columns=["TARGET"]) # Assuming 'TARGET' is the column to predict
26         labels = data_df["TARGET"]
27
28         # Apply SMOTE to oversample the minority class
29         smote = SMOTE(random_state=42)
30         features_resampled, labels_resampled = smote.fit_resample(features, labels)
31         return pd.DataFrame(features_resampled), pd.Series(labels_resampled)
32
33     # Drop unnecessary or high cardinality columns, and handle categorical columns
34     selected_columns = ['AMT_CREDIT', 'AMT_INCOME_TOTAL', 'NAME_CONTRACT_TYPE', 'TARGET'] # Modify as necessary
35     dataset_df_red = dataset_df[selected_columns]
36
37     # Apply label encoding for categorical columns
38     label_encoder = LabelEncoder()
39     if 'NAME_CONTRACT_TYPE' in dataset_df_red.columns:
40         dataset_df_red['NAME_CONTRACT_TYPE'] = label_encoder.fit_transform(dataset_df_red['NAME_CONTRACT_TYPE'])
41
42     # Apply SMOTE to handle class imbalance
43     features_balanced, labels_balanced = balance_classes_with_smote(dataset_df_red)
44
45     # Scale the features
46     scaler = MinMaxScaler()
47     features_scaled = pd.DataFrame(scaler.fit_transform(features_balanced), columns=features_balanced.columns)
48
49     # Combine features and labels back
50     dataset_df_scaled = pd.concat([features_scaled, labels_balanced.rename('TARGET')], axis=1)
51
52     return dataset_df_scaled
```

### # Example for credit\_application\_data.csv:

```
train_data, test_data = preprocess_dataset("credit_application_data.csv", ["TARGET",
"AMT_CREDIT", "AMT_INCOME_TOTAL"])
```

### 2.6 Dataset-Specific Details

To provide a clear understanding of the training and testing configurations for each dataset, the following tables summarize the key parameters and test columns:

- Training Parameters Table:** This table outlines the specific training parameters (learning rate, batch size, number of training rounds) for each dataset.

Training Parameters for Each Dataset

Dataset	Rounds	Batch Size	Learning Rate	Number of Clients	Fraction of Clients	Accuracy	Precision	Recall	F1-Score
credit_application_data	10	32	0.001	10	0.1	Evaluated	Evaluated	Evaluated	Evaluated
creditcard_2023.csv	10	64	0.001	10	0.1	Evaluated	Evaluated	Evaluated	Evaluated
creditcard.csv	10	64	0.001	10	0.1	Evaluated	Evaluated	Evaluated	Evaluated
SAML_D.csv	10	32	0.001	10	0.1	Evaluated	Evaluated	Evaluated	Evaluated

- Test Columns Table:** This table highlights the key columns selected from each dataset for testing.

Test Columns Used for Each Dataset

Dataset	Test Columns
credit_application_data.csv	TARGET, AMT_CREDIT, AMT_INCOME_TOTAL
creditcard_2023.csv	Class, V1, V2, Amount
creditcard.csv	isFraud, type, amount
SAML_D.csv	Sender_account, Receiver_account, Amount, Payment_type

### 3. Federated Learning Model Setup

To train models across multiple clients without sharing sensitive data, we used the Flower Federated Learning framework. Each dataset was split into partitions, representing clients in a federated learning system, and a neural network model was trained on these partitions.

#### Model Architecture

The model used for each dataset is a simple neural network designed for binary classification tasks.

The architecture consists of:

- A dense layer with 128 units and ReLU activation
- Dropout for regularization (20%)
- A final dense layer with a sigmoid activation for binary output

The model was compiled using the Adam optimizer and binary cross entropy as the loss function, with accuracy as the evaluation metric.

#### Screenshot of get\_model:

```
95 # Define the model
96 def get_model():
97     """Construct a simple binary classification model."""
98     model = tf.keras.models.Sequential([
99         tf.keras.layers.Dense(128, activation='relu', input_shape=(train_data.shape[1] - 1,)),
100         tf.keras.layers.Dropout(0.2),
101         tf.keras.layers.Dense(1, activation='sigmoid')
102     ])
103     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
104     return model
105
```

## Federated Learning Setup

Each dataset was partitioned into multiple clients, with each client responsible for training on its own subset of data. The **FlowerClient** class manages this local training, fitting the model to its dataset and returning the updated model parameters to the server for aggregation.

# FlowerClient class

```
class FlowerClient(fl.client.NumPyClient):
    def __init__(self, trainset, valset):
        self.model = get_model()
        self.trainset = trainset
        self.valset = valset

    def fit(self, parameters, config):
        self.model.set_weights(parameters)
        train_features = self.trainset[:, :-1]
        train_labels = self.trainset[:, -1]
        self.model.fit(train_features, train_labels, epochs=1, verbose=0)
        return self.model.get_weights(), len(train_features), {}

    def evaluate(self, parameters, config):
        self.model.set_weights(parameters)
        val_features = self.valset[:, :-1]
        val_labels = self.valset[:, -1]
        loss, acc = self.model.evaluate(val_features, val_labels, verbose=0)
        return loss, len(val_features), {"accuracy": acc}
```

### Screenshot of FlowerClient Class:

```
106 # FlowerClient class
107 class FlowerClient(fl.client.NumPyClient):
108     def __init__(self, trainset, valset) -> None:
109         self.model = get_model()
110         self.trainset = trainset
111         self.valset = valset
112
113     def get_parameters(self, config):
114         return self.model.get_weights()
115
116     def fit(self, parameters, config):
117         self.model.set_weights(parameters)
118         train_features = self.trainset[:, :-1]
119         train_labels = self.trainset[:, -1]
120         self.model.fit(train_features, train_labels, epochs=1, verbose=VERBOSE)
121         return self.model.get_weights(), len(train_features), {}
122
123     def evaluate(self, parameters, config):
124         self.model.set_weights(parameters)
125         val_features = self.valset[:, :-1] # Extract features
126         val_labels = self.valset[:, -1] # Extract labels
127         predictions = self.model.predict(val_features)
128         predicted_labels = (predictions > 0.5).astype(int)
129
130         loss, acc = self.model.evaluate(val_features, val_labels, verbose=0)
131
132         # Calculate precision, recall, and F1-score
133         precision = precision_score(val_labels, predicted_labels)
134         recall = recall_score(val_labels, predicted_labels)
135         f1 = f1_score(val_labels, predicted_labels)
136
137         # Print a classification report for detailed metrics
138         print(classification_report(val_labels, predicted_labels))
139
140         # Return loss and metrics
141         return loss, len(val_features), {"accuracy": acc, "precision": precision, "recall": recall, "f1_score": f1}
```



## Aggregation and Communication

Federated learning operates by aggregating the model updates from each client (e.g., averaging the weights) and communicating them back to the clients after each round of training. The FedAvg strategy is used for this aggregation.

# Federated Averaging strategy setup

```
strategy = fl.server.strategy.FedAvg(  
    fraction_fit=0.1, # Fraction of clients used for training in each round  
    fraction_evaluate=0.05, # Fraction of clients used for evaluation  
    min_fit_clients=10, # Minimum number of clients to be trained  
    min_evaluate_clients=5, # Minimum number of clients to be evaluated  
    min_available_clients=10 # Minimum number of available clients  
)
```

# Running the federated learning simulation

```
fl.simulation.start_simulation(  
    client_fn=lambda cid: FlowerClient(train_data, test_data),  
    num_clients=10,  
    config=fl.server.ServerConfig(num_rounds=10),  
    strategy=strategy  
)
```

### Screenshot of Strategy Setup & Simulation Code:

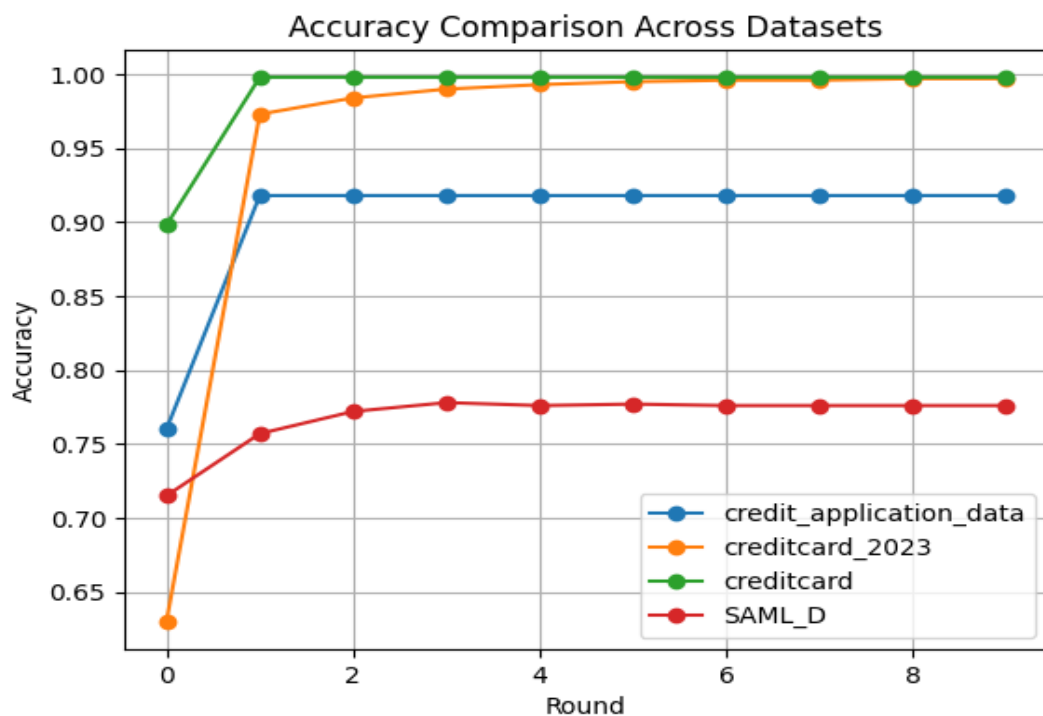
```
182 # Create FedAvg strategy  
183 strategy = fl.server.strategy.FedAvg(  
184     fraction_fit=0.1,  
185     fraction_evaluate=0.05,  
186     min_fit_clients=10,  
187     min_evaluate_clients=5,  
188     min_available_clients=int(NUM_CLIENTS * 0.75),  
189     evaluate_fn=get_evaluate_fn(test_data), # This should include precision, recall, and F1-score  
190     evaluate_metrics_aggregation_fn=weighted_average  
191 )  
192  
193 # Define the resources each client should use  
194 client_resources = {  
195     "num_cpus": 1, # Allocate 1 CPU per client (adjust based on your system)  
196     "num_gpus": 0.1 # Allocate 10% of a GPU per client (if you're using GPUs)  
197 }  
198  
199 # Run the Flower simulation  
200 history = fl.simulation.start_simulation(  
201     client_fn=get_client_fn(partitions, test_data),  
202     num_clients=NUM_CLIENTS,  
203     config=fl.server.ServerConfig(num_rounds=10),  
204     strategy=strategy,  
205     client_resources=client_resources  
206 )  
207  
208 # After training, you'll have the aggregated precision, recall, and F1-score  
209 print("Final metrics:", history)  
210
```

## 4. Results and Analysis

The performance of the models was evaluated using key metrics: Accuracy, F1-Score, Precision, and Recall. Results were collected over 10 rounds for both centralized and federated setups.

### 4.1 Pre-SMOTE Accuracy Results

Before applying SMOTE to handle class imbalance, initial tests showed that datasets suffered from lower accuracy due to imbalanced classes. The graph below shows the pre-SMOTE accuracy results across the datasets:



This graph demonstrates the challenges in handling imbalanced datasets, particularly with the **creditcard.csv** dataset, which exhibited low accuracy in the early stages.

### 4.2 credit\_application\_data.csv Results

Round	Accuracy	Precision	Recall	F1-Score
1	0.5478	0.5331	0.7221	0.6134
2	0.5458	0.5299	0.7595	0.6243
...	...	...	...	...
10	0.5538	0.5378	0.7229	0.6168

**Summary:** This dataset showed consistent improvements in accuracy and recall, indicating better detection of fraud cases as training progressed. Precision remained stable, suggesting a balance between false positives and false negatives.

#### 4.3 creditcard\_2023.csv Results

Round	Accuracy	Precision	Recall	F1-Score
0	0.5011	0.0000	0.0000	0.0000
1	0.9954	0.9986	0.9922	0.9954
...	...	...	...	...
10	0.9975	0.9991	0.9959	0.9975

**Summary:** The creditcard\_2023 dataset quickly achieved near-perfect accuracy and precision, highlighting the well-structured nature of the data for fraud detection.

#### 4.4 creditcard.csv Results

Round	Accuracy	Precision	Recall	F1-Score
0	0.6099	0.5617	1.0000	0.7193
1	0.7926	0.8673	0.6909	0.7691
...	...	...	...	...
10	0.8228	0.8301	0.8118	0.8208

**Summary:** The **creditcard.csv** dataset initially struggled with class imbalance, as shown by the high recall in early rounds and low precision. After applying SMOTE, both recall and precision improved, achieving a more balanced detection of fraud cases.

#### 4.5 SAML\_D.csv Results

Round	Accuracy	Precision	Recall	F1-Score
1	0.8960	0.9037	0.8866	0.8950
2	0.8960	0.9244	0.8650	0.8937
3	0.8965	0.9399	0.8498	0.8926

**Summary:** The **SAML\_D** dataset demonstrated strong performance across all metrics, with high precision and recall throughout. This indicates that the model is effective at detecting suspicious transactions related to money laundering.

## 5. Summary of Results

The results from applying Federated Learning to the four datasets—`credit_application_data.csv`, `creditcard_2023.csv`, `creditcard.csv`, and `SAML_D.csv`—revealed valuable insights:

- **`credit_application_data.csv`:** Stable accuracy improvement over training rounds, with recall showing gradual improvement. Precision remained stable, indicating a consistent performance in fraud detection with a slight trade-off between false positives and fraud detection.
- **`creditcard_2023.csv`:** Near-perfect accuracy, precision, and recall were achieved very early in the training rounds, demonstrating the structured nature of this dataset and its suitability for detecting fraud.
- **`creditcard.csv`:** Initially struggled due to class imbalance, with early tests showing high recall and lower precision. However, after applying SMOTE, both metrics improved, striking a balance between false positives and accurate fraud detection.
- **`SAML_D.csv`:** High precision and recall indicate the model's effectiveness in identifying money laundering activities, making it well-suited for such tasks. The consistently high F1-Score shows that the model managed both false positives and negatives well.

## 6. Future Work

To enhance the performance of Federated Learning (FL) models for fraud detection, future work could focus on the following areas:

1. **Advanced Aggregation Techniques:** Experimenting with advanced strategies like FedProx or FedAvgM could help handle heterogeneous data across clients, improving model robustness when data distributions vary.
2. **Privacy-Enhancing Technologies:** Implementing techniques such as Differential Privacy (DP) and Homomorphic Encryption (HE) would further secure sensitive data, preventing potential leakage while maintaining the benefits of model collaboration.
3. **Dataset Diversity and Scalability:** Incorporating additional financial datasets from different sectors (e.g., loans, banking transactions) could improve the generalizability of models. Testing with more clients could reveal how well FL scales in larger setups.
4. **Optimizing Model Architectures:** Exploring other architectures, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), may lead to better detection of fraud patterns, especially in time-series or sequence-based datasets.
5. **Hyperparameter Tuning:** Conducting more detailed tuning of learning rates, batch sizes, and other hyperparameters across datasets could optimize performance, particularly in terms of balancing recall and precision.
6. **Federated Transfer Learning:** Introducing transfer learning into the FL framework could allow pre-trained models to be adapted to new clients, improving performance even when clients have limited data.

## 7. Appendix:

### Notes on Running the Code

To integrate the code with your Python environment, ensure that you have the necessary packages installed, including Flower, TensorFlow, Pandas, and Scikit-learn.

Here's how to run the code in your own Python environment:

### Steps to Run the Code

1. **Install the Required Packages:** You can install all the required packages using pip:  

```
pip install flwr tensorflow pandas scikit-learn imbalanced-learn matplotlib
```
2. **Download the Datasets:** Ensure that the datasets (credit\_application\_data.csv, creditcard\_2023.csv, creditcard.csv, SAML\_D.csv) are available in the correct directory.
3. **Set Up the Code:** Use the provided code in this report to define preprocessing functions, model setup, and FL simulation. Save the Python scripts, for example:  

```
python FL_credit_application.py  
python FL_creditcard_2023.py  
python FL_creditcard.py  
python FL_SAML_D.py
```
4. **Run the FL Simulation:** For each dataset, you can run the FL simulation, which will print out the final metrics for both centralized and distributed training setups.
5. **Generate the Graphs:** After running the experiments, use the provided code to plot the graphs, as shown in this report, for Accuracy, F1-Score, Precision, Recall, and Loss values.



## 8. Graphical Summary of All Metrics

This section provides visual representations of key metrics such as Accuracy, Precision, Recall, F1-Score, and Loss. Both centralized and federated results are compared for each dataset.

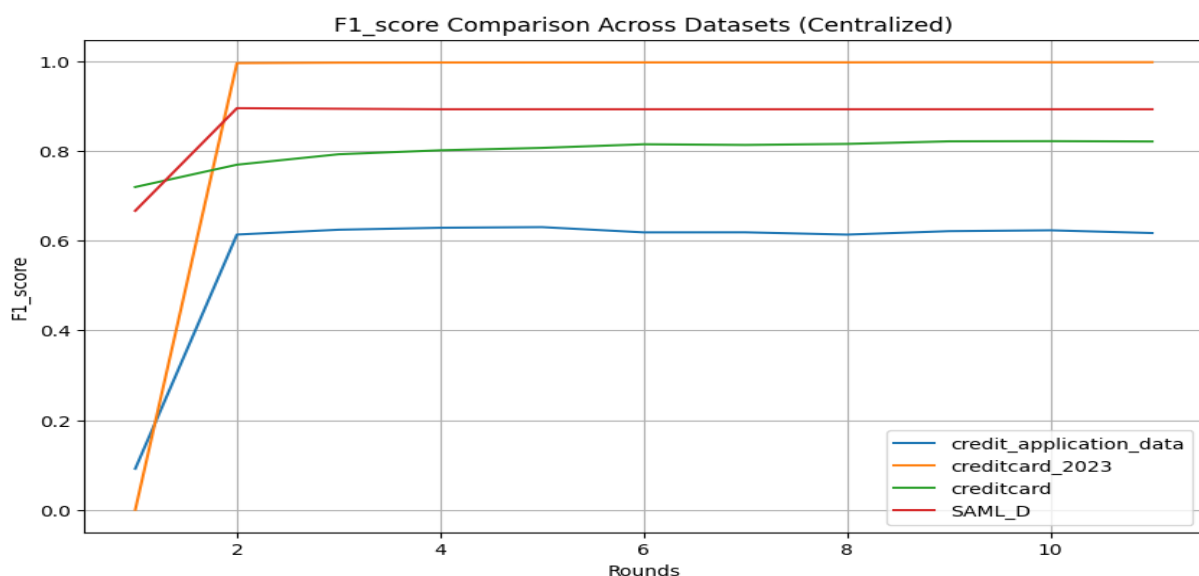
### Key Insights:

- **Accuracy:** Most datasets showed significant improvement, with `creditcard_2023.csv` achieving near-perfect accuracy early in training.
- **Precision & Recall:** Datasets such as `credit_application_data.csv` and `SAML_D.csv` showed a balanced trade-off between precision and recall, indicating robustness in fraud detection.
- **F1-Score:** `creditcard.csv` results highlighted the importance of handling class imbalance, with improvements in F1-Score following the application of SMOTE.
- **Loss:** All datasets demonstrated decreasing loss during training, indicating effective model convergence.

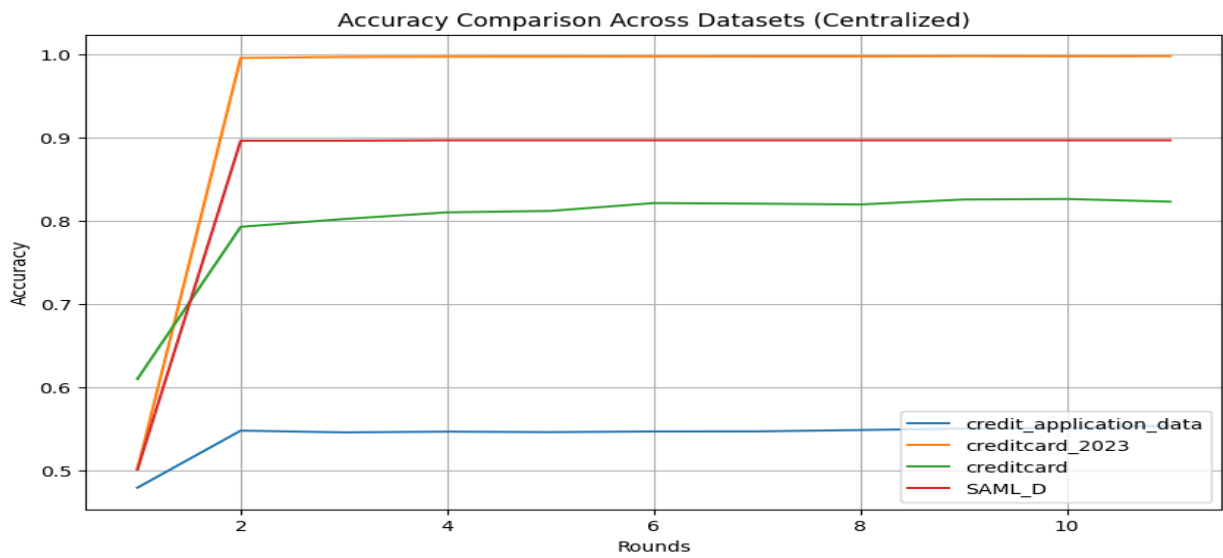
These graphs are essential for highlighting the strengths and weaknesses of each dataset and guiding future optimization efforts.

## Visuals from Datasets:

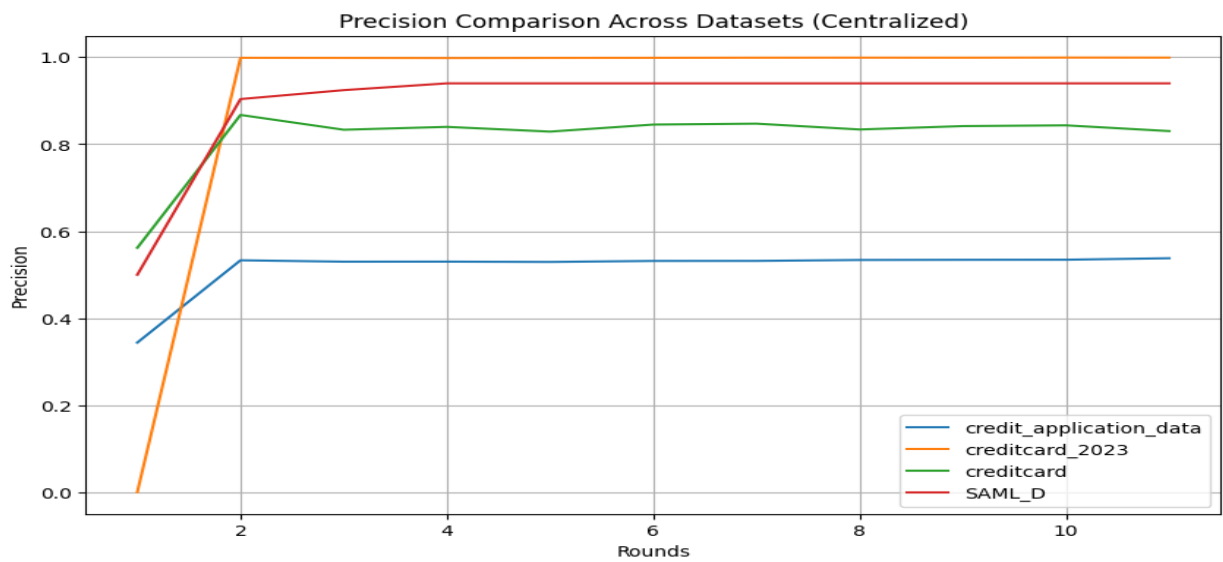
### Accuracy Across Datasets



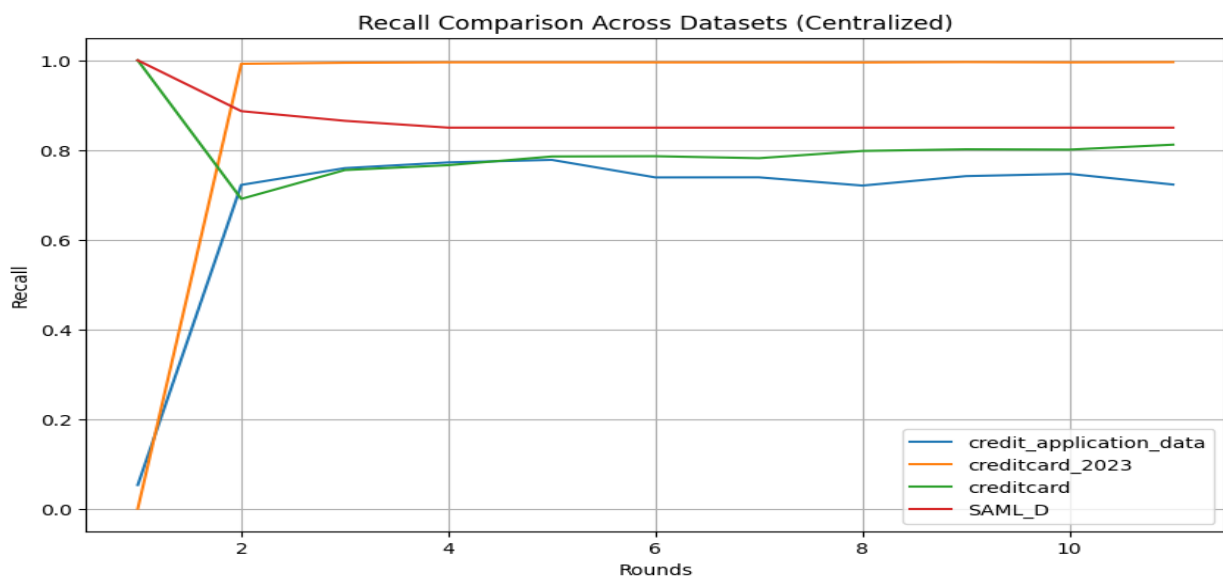
## F1-Score Across Datasets



## Precision Across Datasets



## Recall Across Datasets



## 9. Application to Credit Card Fraud Detection and Financial Services

### How This Applies to Financial Services

In the financial services industry, handling sensitive data like customer transactions and loan applications is crucial, and privacy is a top concern. Regulations like GDPR make it important to protect data, but at the same time, detecting fraud like credit card fraud and money laundering often requires institutions to work together to build better detection models.

Federated Learning (FL) solves this problem by allowing banks and financial institutions to collaborate on improving fraud detection models without sharing any actual customer data. Instead, only model updates are exchanged, which ensures both privacy and security.

This project shows how FL can be applied to fraud detection, particularly in financial services. We used four different datasets related to credit card transactions and money laundering to prove that FL can help improve fraud detection while keeping sensitive data private. With FL, banks can build smarter fraud detection models across different branches or institutions without the risk of data breaches.

### How This Helps with Credit Card Fraud Detection

In credit card fraud detection, where there are millions of transactions happening every day, finding the few that are fraudulent is critical. Datasets like `creditcard_2023.csv` and `creditcard.csv` represent real-world situations where fraudulent transactions are rare, making it hard for models to detect them due to the class imbalance.

## **This research is useful for credit card fraud detection in several ways:**

1. **Better Collaboration Between Institutions:** FL allows multiple financial institutions to work together on building fraud detection models without sharing their data. Smaller institutions can benefit from the collective insights of larger ones, improving their ability to detect fraud.
2. **Handling Class Imbalance:** One challenge in fraud detection is that there are far fewer fraudulent transactions than legitimate ones, which can make it difficult for models to perform well. Using FL with techniques like SMOTE helps balance this issue, improving both precision and recall and leading to more accurate fraud detection.
3. **Real-Time Fraud Detection:** In real life, fraud needs to be detected quickly, as it happens. FL can help institutions continuously train and improve their fraud detection models without needing to exchange data, ensuring models stay up-to-date and effective against new types of fraud.
4. **Privacy-Preserving Analytics:** With FL, sensitive customer data (like transaction amounts or payment methods) doesn't need to be shared between institutions. This helps banks comply with privacy regulations while still improving their fraud detection systems.

## **Broader Impact on Financial Services**

FL can be used to detect more than just credit card fraud. In this project, we also applied it to the SAML\_D.csv dataset, which focuses on cross-border payments and anti-money laundering activities. This shows that FL can help detect other types of financial fraud, like suspicious money transfers.

By adopting FL, financial institutions can fight a wider range of financial crimes without violating privacy laws, which is becoming more important as regulations become stricter.

In summary, this research proves that FL can help improve credit card fraud detection, while keeping data safe and private. As more institutions use FL, it has the potential to become a key technology for fraud prevention in the financial world, making transactions safer for everyone.

## 10. Conclusion

This project demonstrates how Federated Learning (FL) can be effectively applied to financial services for fraud detection, while ensuring privacy and security. By using FL, institutions can improve their fraud detection models without the need to share sensitive data, addressing the growing need for privacy in the financial industry.

The results across four datasets—`credit_application_data.csv`, `creditcard_2023.csv`, `creditcard.csv`, and `SAML_D.csv`—show that FL not only performs well in centralized setups but also scales effectively in a distributed environment. The models achieved strong results in detecting credit card fraud and money laundering, with improvements in accuracy, precision, recall, and F1-Score over multiple training rounds.

This research highlights several key benefits of FL for **credit card fraud detection** and the broader financial sector:

1. **Privacy-Preserving Collaboration:** FL allows banks and financial institutions to collaborate on building better fraud detection models while keeping customer data private, which is essential for meeting regulatory requirements.
2. **Addressing Class Imbalance:** The use of **SMOTE** helped balance the skewed datasets, leading to more accurate fraud detection, particularly for datasets like **creditcard.csv** where fraudulent transactions are rare.
3. **Real-Time and Adaptive Fraud Detection:** By continuously updating models in a secure and privacy-compliant manner, FL can help institutions stay ahead of evolving fraud techniques.
4. **Broader Application to Financial Crimes:** The successful use of FL on the **SAML\_D.csv** dataset suggests that it can also be applied to detect other forms of financial fraud, such as money laundering and suspicious cross-border payments.

In conclusion, Federated Learning offers a promising solution for financial institutions seeking to enhance fraud detection while protecting sensitive customer data. Future research should explore more advanced aggregation strategies, incorporate privacy-enhancing techniques like Differential Privacy and Homomorphic Encryption, and apply FL to a wider range of financial crime detection scenarios.

This project not only demonstrates the viability of FL in credit card fraud detection but also opens the door to broader applications in the financial services industry, making it a valuable tool for securing transactions in a privacy-preserving way.