Use Case Experiment Results

DataBytes: PTFI
Privacy Technologies for Financial Intelligence

Trimester 2, 2024

The Data Science Team

DataBytes
Data Powered Platforms

PTFI

Contents

DataBytes Data Powered Platforms PTFI

1

Overview

1.1. Introduction to the PTFI Data Science Stream

The primary objective of the Data Dcience stream in the PTFI project during Trimester 2 of 2024 was to design and execute experimental use cases targeting financial intelligence scenarios. Under the direction of the product owner, the team concentrated on two critical areas:

- Suspicious Persons List Matching
- Credit Card Fraud Detection

Both areas utilized advanced methodologies such as homomorphic encryption and federated learning to enhance privacy and security. This document outlines the proposed experiments and their outcomes.

Below is a summary of the key experiments conducted:

1.1.1. Privacy-Preserving Techniques for Suspicious Person List Matching

This experiment explored various privacy-preserving techniques such as Homomorphic Encryption (HE), Secure Multiparty Computation (SMPC), Differential Privacy (DP), and Hash Matching for SPL scenarios. The goal was to enable secure data comparison between public and private entities without revealing sensitive information.

1.1.2. Full Homomorphic Encryption (FHE) for Fraud Detection

This experiment implemented Full Homomorphic Encryption (FHE) with a Random Forest model to detect credit card fraud while securing customer transaction data.

1.1.3. Federated Learning for Fraud Detection Using Synthetic Credit Card Transaction Data

Federated Learning (FL) was employed to train a global fraud detection model without sharing raw data, allowing multiple financial institutions to collaborate while maintaining data privacy.

1.1.4. Federated Learning with Ray for Parallel Computing

This experiment extended the FL setup by integrating the Ray framework for parallel computation, aiming to simulate multiple clients in parallel on a single machine for improved efficiency.

1.1.5. Federated Learning as a Service (FLaaS) for Credit Card Fraud Detection

In this experiment, a client-server architecture using Flask was developed to implement Federated Learning as a Service (FLaaS). A logistic regression model was used for local and central training, with synthetic transaction data generated for fraud detection.

2

Homomorphic Encryption Experiments

2.1. Experiment on Privacy-Preserving Techniques for Sensitive Data Matching in SPL Scenarios

2.1.1. Overview

In an increasingly connected world, organizations often need to compare sensitive data against external lists, such as Suspicious Persons Lists (SPL), without compromising privacy. Privacy-preserving techniques such as Homomorphic Encryption (HE), Secure Multiparty Computation (SMPC), Differential Privacy (DP), and Hash-based Matching enable entities to collaborate securely on sensitive data without revealing the actual data. This experiment explores the effectiveness of these techniques in ensuring privacy and trust, particularly when public and private entities must collaborate.

As the demand for data privacy increases, particularly in contexts like national security, fraud prevention, and financial crime investigations, it's critical to strike a balance between data sharing and safeguarding individuals' sensitive information.

Experiment contact: Mathew Serek

2.1.2. Scenario (Use Case): Suspicious Person List (SPL) Comparison

This use case involves comparing publicly held SPLs, which include identifiers such as passport numbers or national IDs, with customer lists held by private entities. The goal is to identify potential threats while ensuring the privacy of both private companies and individuals.

2.1.3. Privacy Considerations

- **High Trust Situations:** Data is encrypted in transit and at rest, with secure system operations.
- **Medium Trust Situations:** Data is compared while encrypted using privacy-preserving technologies, with governance arrangements between parties.
- **Low Trust Situations:** Data is encrypted throughout the process, with results only accessible to the public entity. Private parties do not learn the outcome unless further consent is obtained.

Use Case Example: Public to Private - Low Trust
Public entities push SPL data to private entities for matching against their customer lists. The results are only accessible to the public party, ensuring that private entities gain no insights. Strong encryption methods such as HE and DP maintain confidentiality throughout the process.

2.1.4. Technology Involved

- **Homomorphic Encryption (HE):** Enables encrypted data comparison without revealing underlying data.

- **Symmetric Encryption:** Ensures secure sharing of results post-matching.
- **Differential Privacy (DP):** Adds noise to ensure individual data points cannot be identified in the results.
- **Secure Multiparty Computation (SMPC):** Allows multiple parties to jointly compute the result without revealing their inputs.

2.1.5. Experimental Setup

Data Generation:

Synthetic data simulating matching between bank customers and suspicious persons:

- **Bank Customers:** 5,000 records with a unique TFN, name, date of birth, and address.
- **Suspicious Persons:** 50 records, with a 10% overlap in TFNs for realistic matching.

Privacy-Preserving Techniques:

- **Hash Matching:** TFNs are hashed and compared.
- **HE Matching:** TFNs are encrypted using HE, and matches are found by comparing encrypted values.
- **SMPC Matching:** TFNs are secret-shared between parties, and matches are determined based on shared secrets.
- **DP Matching:** Laplace noise is added to the match count for privacy preservation.

2.1.6. Performance Metrics

Performance was measured based on:

- **Number of Matches Identified:** Customer records matching with suspicious persons.
- **Execution Time:** Time taken to perform the matching process.

2.1.7. Findings

Results Summary:

Method	Number of Matches	Time Taken (seconds)	Remarks
Hash Matching	5 matches	0.0418	Efficient, but weak privacy guarantees
HE Matching	5 matches	560.12	Extremely secure, but computationally expensive
SMPC Matching	0 matches	0.2970	Fast, but implementation issues caused no matches
DP Matching	Est. 11.5 matches	0.0006	Highly efficient, noisy results due to privacy guarantees

Table 2.1: Results Summary of Privacy-Preserving Techniques

Detailed Observations:

- **Hash Matching:** This method is highly efficient and best suited for scenarios where speed and exactness are critical. However, the privacy guarantees are weak, as hashed values can be vulnerable to reverse engineering.
- **HE Matching:** Despite being secure, HE Matching was computationally expensive, taking over 500 seconds to complete. It is highly accurate and useful in environments where security is paramount, but its high overhead suggests that further optimization is necessary for large-scale deployments.
- **SMPC Matching:** This method demonstrated fast performance but failed to produce any matches. The result may indicate an issue with either the implementation or data handling. However, SMPC remains a strong option when privacy is a top priority and multi-party collaboration is required.

- **DP Matching:** Differential Privacy provided an estimated number of matches (11.5), reflecting the noise added to the results for privacy preservation. Although highly efficient, this method sacrifices some accuracy in exchange for formal privacy guarantees. DP is ideal when privacy is more critical than exact results.

2.1.8. Conclusions

The experiment demonstrated that different privacy-preserving techniques suit different use cases, with trade-offs between efficiency, accuracy, and security. Organizations must assess their privacy needs and operational requirements to select the most appropriate technique.

- **Hash Matching:** Ideal for fast, low-cost matching with limited privacy guarantees.
- **HE:** Strong privacy protection but requires optimization for large-scale use.
- **SMPC:** Promising for multi-party collaboration, but accuracy improvements are needed.
- **DP:** Provides strong privacy but at the cost of result accuracy.

2.1.9. Further Considerations

- **Legal and Regulatory Implications:**

  In high-stakes environments like financial institutions or law enforcement, compliance with regulations such as GDPR or CCPA is paramount. The use of advanced encryption techniques such as HE and SMPC can help ensure data privacy, but organizations should also consider the legal requirements surrounding data-sharing practices.

- **Hybrid Approaches:**

  A hybrid approach combining techniques such as HE with DP could be explored in future iterations. This may provide both strong security guarantees and efficiency while minimizing the trade-offs between privacy and accuracy.

- **Performance Optimization:**

  For methods like Homomorphic Encryption, performance optimization through better algorithms or hardware advancements (e.g., leveraging GPUs or quantum computing) could make these techniques more feasible in real-world scenarios.

- **Future Outlook:**

  As privacy-preserving techniques evolve, it is likely that methods like HE and SMPC will become more accessible and efficient. Further research should focus on improving their computational overhead, especially for large datasets, while maintaining strong privacy guarantees.

Full code for experiment location: *SPL Matching Experiment Code*

2.2. Full Homomorphic Encryption (FHE) Experiment

2.2.1. Overview

Fraud detection is a critical task in the banking sector, where large amounts of sensitive transaction data must be processed to identify suspicious activity. The goal of this experiment was to build a machine learning model capable of identifying fraudulent transactions based on features such as transaction amount, credit score, and other synthetic features. In addition, we explored the use of Full Homomorphic Encryption (FHE) to ensure the privacy of sensitive data while still allowing for meaningful analysis and predictions to be made on encrypted data.

This experiment provides insights into how privacy-preserving techniques like FHE can be applied to fraud detection, allowing secure communication between banks and customers while keeping the underlying data secure.

Experiment contact: Akash Verma

2.2.2. Scenario (Use Case)

Fraud Detection in Banking: The use case involves detecting fraudulent banking transactions based on the features of the transaction. A machine learning model is trained to classify transactions as either fraudulent or non-fraudulent. The results are then encrypted using Full Homomorphic Encryption (FHE) to ensure that predictions can be securely transmitted between financial institutions and customers without revealing sensitive data.

2.2.3. Technology Involved

- **Random Forest Classifier:** A supervised learning algorithm used to classify transactions as either fraudulent or non-fraudulent.
- **Full Homomorphic Encryption (FHE):** CKKS scheme from the TenSEAL library, used to encrypt predictions and allow operations on encrypted data without decrypting it.
- **SMOTE (Synthetic Minority Over-sampling Technique):** A technique used to handle the class imbalance in the dataset by generating synthetic samples of the minority class.
- **Faker Library:** Used to generate a synthetic dataset that simulates realistic banking transactions.

2.2.4. Experimental Setup

Data Generation: A synthetic dataset was generated using the Faker library, simulating 1000 banking transactions.
Features included:

- *Transaction Amount (AUD):* Random amounts between AUD 100 and 10,000.
- *Credit Score:* Random values between 300 and 850.
- *Fraud Label (IsFraud):* Indicating whether the transaction is fraudulent (15% of the dataset was labeled as fraudulent).

Handling Class Imbalance: Since fraudulent transactions were underrepresented, SMOTE was used to balance the classes by generating synthetic samples of fraudulent transactions.

Modeling: A Random Forest Classifier was used to predict fraudulent transactions. Features such as Transaction Amount and Credit Score were standardized using StandardScaler. The model was trained on the balanced dataset and evaluated on a separate test set.

Privacy-Preserving Techniques: Predictions made by the model were encrypted using the Full Homomorphic Encryption (FHE) CKKS scheme. This allowed for secure transmission of sensitive predictions without exposing the underlying data. Both sum and product operations were performed on the encrypted data to demonstrate the capabilities of FHE.

2.2.5. Model Evaluation

After training, the model was evaluated on key metrics:

- **ROC AUC Score:** The ROC AUC score of 0.74 indicates the model's reasonable ability to distinguish between fraudulent and non-fraudulent transactions. While this is a significant improvement from initial experiments, there's still room for improvement.
- **Confusion Matrix:**
  - The model correctly identified 216 non-fraudulent transactions but misclassified 126 as fraudulent.
  - It also correctly identified 245 fraudulent transactions, but 89 were misclassified as non-fraudulent.
- **Classification Report:**
  - Precision for non-fraudulent transactions (class 0) was 0.71, with a recall of 0.63.
  - Precision for fraudulent transactions (class 1) was 0.66, with a recall of 0.73.

2.2.6. Findings

Results Summary:

Metric	Value
ROC AUC	0.74
Precision (Class 0)	0.71
Precision (Class 1)	0.66
Recall (Class 0)	0.63
Recall (Class 1)	0.73
Overall Accuracy	68%

Table 2.2: Results Summary of Full Homomorphic Encryption (FHE) Experiment

Detailed Observations:

- **Class Imbalance Handling:** Applying SMOTE was effective in improving the detection of fraudulent transactions, but further refinement may be required to reduce false positives and false negatives.
- **Model Performance:** The Random Forest Classifier demonstrated moderate accuracy, but further hyperparameter tuning or exploring more advanced models like XGBoost could improve the results.
- **FHE Performance:** Full Homomorphic Encryption proved to be secure and allowed for operations on encrypted data, although it introduced computational overhead. In a real-world implementation, performance optimizations would be necessary for large-scale deployment.

2.2.7. Future Improvements

Model Optimization:

- Apply more advanced models like XGBoost or Gradient Boosting to improve classification accuracy.
- Conduct hyperparameter tuning to further refine the Random Forest model.

Feature Engineering:

- Include additional features such as transaction location, time of transaction, and customer spending habits to provide more context for the fraud detection task.

Performance Optimizations for FHE: While FHE provided strong privacy guarantees, it introduced a performance hit. Future work should explore optimizations for handling large datasets with homomorphic encryption.

Class Imbalance: Continue experimenting with techniques like class weighting or undersampling to mitigate the effects of class imbalance.

2.2.8. Conclusion

This experiment demonstrated the viability of applying privacy-preserving techniques to a fraud detection system. While the Random Forest Classifier achieved a ROC AUC score of 0.74, further improvements are needed to increase precision and recall for both classes. Additionally, Full Homomorphic Encryption successfully protected sensitive transaction data, ensuring privacy while allowing meaningful operations on encrypted data.

Full code for experiment location: *Full Homomorphic Encryption FHE Experiment Code*

2.3. Suspicious Person List using HE - Public to public

2.3.1. Use Case

Public to public – international trusted group

- Medium trust
- Encryption in transit, operation, rest
- Results summary available to all parties

Compare SPL to other SPLs held by trusted international public parties. SPL is push only – a public party compares the SPL to one or more international public parties. Results are available in summary only, with full details on the application to the public party holding the information.

Experiment contact: VENKATA SAI PREETHI GODDI

Privacy

This use case enables medium trust comparison by maintaining data encryption during transit, operation, and at rest, combined with trusted international security and governance arrangements. It maintains privacy by ensuring results are only available in summary to trusted international parties and requiring the consent of a data holder before full details are made available.

2.3.2. Technology

- **Homomorphic encryption (comparison):** Allows encrypted data to be compared without decrypting it first, protecting data privacy during the comparison process.
- **Symmetric encryption (sharing summary and full details):** Used to encrypt the summary results and ensure that sensitive information is secure when shared or stored.

2.3.3. Demonstrating the Experiment

Data Generation: Create synthetic SPL data for demonstration purposes. This simulates the SPLs held by different parties.

Encryption: Use homomorphic encryption to securely encrypt the SPL scores, allowing for secure comparison.

Comparison: Perform the comparison on encrypted data to identify matches without revealing the actual scores.

Results Summary: Encrypt the comparison results (summary) to ensure it remains confidential during sharing. Decrypt the results to display them.

This approach demonstrates how privacy-preserving technologies can be applied in practical scenarios where multiple parties need to collaborate while protecting sensitive information.

2.3.4. Code Explanation

Required Packages:

- TenSEAL: For Homomorphic Encryption.
- Cryptography: For Symmetric Encryption (e.g., AES).
- Pandas: For handling data.

Synthetic Data Generation: The `generate_synthetic_spl()` function generates synthetic data for the Suspicious Person List (SPL). The `person_id` is a unique identifier, and the `suspicious_score` is a numerical value indicating the suspicion level. This data simulates real-world SPL data.

Homomorphic Encryption Context Initialization: The `initialize_he_context()` function creates a TenSEAL CKKS context for homomorphic encryption. CKKS allows for encrypted computations on floating-point numbers, ideal for operations like comparisons. The context is configured with specific parameters, including the polynomial modulus and global scale. Galois keys are generated to enable operations like rotations.

Encrypt SPL Data: The `encrypt_spl_data()` function encrypts the `suspicious_score` column of the SPL using CKKS encryption. Each score is encrypted separately and stored in a list of encrypted vectors.

Compare SPL Data: The `compare_spl_data()` function compares two lists of encrypted SPL data. It subtracts corresponding encrypted scores to check for matches. If the decrypted result of the subtraction is close to zero, it indicates a match. The results are stored as a list of boolean values.

Symmetric Encryption (AES): The `symmetric_encrypt()` function encrypts data using AES encryption in CBC mode. The function generates a random initialization vector (IV) for each encryption operation, ensuring that the same plaintext encrypts to different ciphertexts. The data is padded to ensure compatibility with the AES block size.

The `symmetric_decrypt()` function decrypts the AES-encrypted data, reversing the encryption process and removing the padding to retrieve the original plaintext.

Main Function: The `main()` function demonstrates the entire process:

- Synthetic SPL data is generated for two public parties.
- The SPL data is encrypted using homomorphic encryption.
- The encrypted data is compared to find matches.
- A summary of the results (e.g., the number of matches) is created and encrypted using symmetric encryption.
- The summary is decrypted for demonstration.

2.3.5. Experimental Setup

Data Generation: Synthetic data was generated to simulate the matching process between two Suspicious Person Lists (SPLs):

- **SPL1 and SPL2:** Each list contains 5000 records, each with a unique identifier and a suspicious score. The records were randomly generated, creating a realistic scenario where two separate lists need to be compared for matching entries.

Privacy-Preserving Techniques:

- **Homomorphic Encryption (HE) Matching:** Suspicious scores were encrypted using HE, allowing comparison of encrypted values without decryption.
- **Secure Multiparty Computation (SMC) Matching:** Suspicious scores were split into shares, with matches identified by comparing the sums of these shares using secret-sharing techniques.
- **Differential Privacy (DP) Matching:** Laplace noise was added to the data before comparison to preserve privacy, resulting in a noisy match count.

2.3.6. Performance Metrics

The performance of each method was measured based on:

- **Number of Matches Identified:** How many records from SPL1 matched with records from SPL2.
- **Execution Time:** The time taken to perform the matching process.

2.3.7. Detailed Observations

Homomorphic Encryption (HE) Matching: HE allows computations on encrypted data, ensuring that sensitive information remains protected. However, in this experiment, the HE method identified zero matches, which could be due to the complexity of the encryption and comparison process. The exact execution time was 526.164 seconds, indicating that HE was computationally intensive. HE provides a high level of privacy and security, making it suitable for environments where data confidentiality is paramount, but its computational overhead requires optimization for practical use.

Secure Multiparty Computation (SMC) Matching: SMC successfully identified 24 matching records between the two lists. The execution time was 0.0655 seconds, showcasing its efficiency. SMC is ideal

for scenarios requiring privacy and multi-party collaboration, offering an effective way to compare data without revealing sensitive information.

Differential Privacy (DP) Matching: Differential Privacy added noise to the data, resulting in zero identified matches. The comparison was extremely fast, taking only 0.0299 seconds. While this method ensures strong privacy, the noise introduced significantly impacted the accuracy of the matching process. DP is best suited for situations where privacy is prioritized over precise matching.

2.3.8. Findings

Results Summary

Method	Number of Matches	Time Taken (seconds)	Remarks
Homomorphic Encryption (HE)	0 matches	526.164	Highly secure, but the encryption and comparison process was complex and time-consuming, resulting in zero matches.
Secure Multiparty Computation (SMC)	24 matches	0.0655	Fast and effective, providing accurate matches. Ideal for scenarios requiring privacy-preserving data comparison.
Differential Privacy (DP)	0 matches	0.0299	Highly efficient, but the added noise affected the accuracy, leading to zero matches. Provides strong privacy guarantees.

Table 2.3: Results Summary for Different Privacy-Preserving Techniques

2.3.9. Conclusions

This experiment highlights the trade-offs between different privacy-preserving techniques:

- **Homomorphic Encryption (HE):** Offers robust privacy protection by enabling operations on encrypted data. However, HE's high computational complexity led to zero matches in this experiment. HE is best suited for use cases demanding high security where performance can be optimized.
- **Secure Multiparty Computation (SMC):** Provided accurate matches with a relatively quick execution time. It is effective for scenarios where multi-party data comparison is needed while maintaining privacy.
- **Differential Privacy (DP):** Ensures robust privacy by adding noise to the data, leading to zero matches in this experiment. DP is suitable for scenarios that prioritize privacy over accuracy.

Organizations must carefully evaluate their privacy and performance requirements when selecting an appropriate privacy-preserving technique.

2.3.10. Further Considerations

Legal and Regulatory Implications Regulatory compliance (e.g., GDPR or CCPA) is critical in sensitive domains like finance or law enforcement. Advanced encryption methods like HE and SMC support data privacy while ensuring compliance with data-sharing regulations. Legal constraints and data-sharing practices must be thoroughly understood before implementing these techniques.

Hybrid Approaches Exploring hybrid approaches, such as combining HE with DP, could leverage the strengths of both methods, offering robust security and improved efficiency.

Performance Optimization Performance optimization, especially for HE, is crucial for enhancing computational efficiency and enabling real-world application scalability. Future advancements in hardware (e.g., leveraging GPUs or quantum computing) may make these techniques more feasible.

Future Outlook As privacy-preserving technologies advance, methods like HE and SMC are expected to become more accessible and efficient. Research should continue to focus on reducing computational overhead while maintaining strong privacy guarantees to enable large-scale, secure data applications.

Full code for experiment location: *Suspicious Person List using HE GitHub Repo.*

2.4. Suspicious Person List using HE - Private to Private

2.4.1. Use Case: Private to Private – Domestic Trusted Group

Medium trust
Encryption in transit, operation, rest
Results summary available to all parties

2.4.2. Overview

This use case involves comparing Suspicious Person Lists (SPLs) between two trusted domestic private parties. The objective is to identify common suspicious individuals while maintaining a high level of data privacy and security. The results of the comparison are shared with all involved parties in the form of a summary, with full details available only by mutual agreement.

2.4.3. Privacy and Trust

In this scenario, a medium level of trust is established between the parties involved. To ensure data privacy and security, encryption is utilized at every stage of data handling:

- **In Transit:** Data is encrypted during transmission to prevent unauthorized access.
- **In Operation:** Data is processed in its encrypted form to maintain confidentiality.
- **At Rest:** Data is stored in an encrypted state to protect it from unauthorized access.

The comparison process focuses solely on matching personal information of suspicious customers that is common between the two parties. This approach minimizes the amount of personal data shared and ensures that only the necessary information is disclosed.

2.4.4. Governance and Compliance

To ensure effective management of this use case, it is critical to implement robust governance frameworks among the parties involved. Considering the sensitive characteristics of the data, compliance with privacy regulations and applicable legislation is paramount. This could require the modification of privacy policies and terms of service to acquire customer consent for the proposed data sharing. The parties are unlikely to depend on current exceptions in privacy laws concerning law enforcement, which highlights the necessity of establishing clear agreements and consent mechanisms.

2.4.5. Technologies

Homomorphic Encryption: A sophisticated encryption method that enables operations to be conducted on encrypted data without requiring decryption. In SPL comparison, this guarantees sensitive information is safeguarded throughout the comparison procedure.

Symmetric Encryption: Used to safeguard the summary of the comparison results before distribution. Ensures the summary is confidential and only available to authorized individuals. Also applied to detailed results, shared only upon mutual agreement.

2.4.6. Code Explanation

Required Packages: `phe, pycryptodome, faker`
Synthetic Data Generation: The `generate_synthetic_spl` function generates synthetic SPL data using the Faker library. The data simulates sensitive information used in SPL comparison.
Encryption and Comparison: Risk scores are encrypted using Paillier's public key, and encrypted data is compared between two institutions without decryption, maintaining privacy.

2.4.7. Experimental Setup

Data Generation: Synthetic data was generated for two SPLs, each containing 5000 records, with unique identifiers, names, DOBs, SSNs, addresses, transaction activities, and suspicious risk scores. The `Faker` library created realistic entries, and `risk_score` was randomly assigned between 0 and 100.

2.4.8. Privacy-Preserving Technique

Homomorphic Encryption (HE) Matching:

- **Key Generation:** Paillier public-private key pair generated.
- **Data Encryption:** SPL risk scores were encrypted using the Paillier public key.
- **Encrypted Data Comparison:** Encrypted risk scores from both lists were compared without decryption.
- **Match Detection:** Iteration through risk scores detected privacy-preserving matches.

2.4.9. Performance Metrics

The performance of each method was measured based on the number of matches identified and the time taken for comparison.

Method	Number of Matches	Time Taken (seconds)	Remarks
Homomorphic Encryption (HE)	0 matches	966.352	Maintained high security and privacy during the comparison process. However, the process is complex and computationally intensive.
Hash-Based Comparison	18 matches	535.565	Efficient and quick, providing secure comparisons without revealing sensitive information.
Secure Multi-Party Computation (SMPC)	22 matches	232.478	Effective for privacy-preserving multi-party data comparison, with a balance between security and performance.
Differential Privacy (DP)	0 matches	720.568	Ensured strong privacy with added noise, but impacted the accuracy of matching. Fast and privacy-centric.

Table 2.4: Results Summary for Privacy-Preserving Techniques

2.4.10. Detailed Observations

Homomorphic Encryption (HE) Matching: While secure, the HE method identified zero matches due to its computational complexity.
Hash-Based Comparison Matching: Identified 18 matches, balancing privacy and efficiency.
Secure Multi-Party Computation (SMPC): The most effective method, with 22 matches and a reasonable execution time.

Differential Privacy (DP) Matching: Added noise, resulting in zero matches but ensured strong privacy.

2.4.11. Conclusions

This experiment illustrates the trade-offs between different privacy-preserving techniques:

- **Homomorphic Encryption (HE):** Offers the highest privacy level but has high computational costs.
- **Hash-Based Comparison:** Provides an efficient and secure comparison method.
- **Secure Multi-Party Computation (SMPC):** Offers the best balance between accuracy and privacy, ideal for multi-party scenarios.
- **Differential Privacy (DP):** Ensures strong privacy but impacts accuracy.

Future research should focus on optimizing these methods for broader deployment while ensuring compliance with legal frameworks like GDPR and CCPA.

Full code for experiment location: *Suspicious Person List using HE GitHub Repo.*

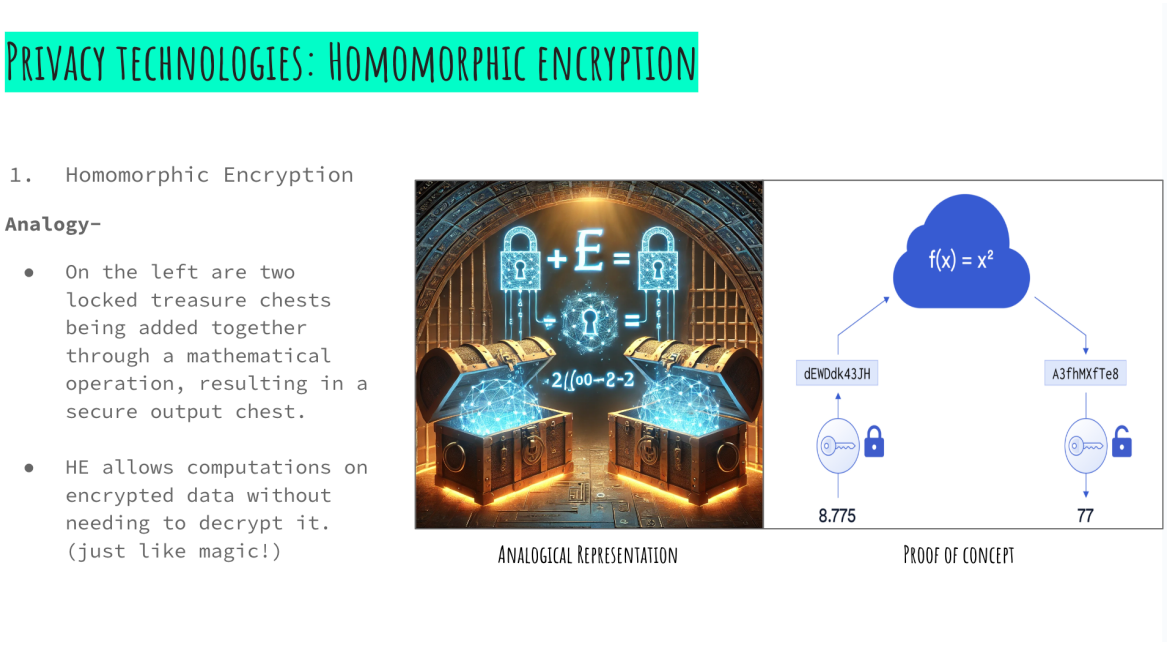

Figure 2.1: Homomorphic Encryption Simplified Analogy Explanation

3

Federated Learning Experiments

3.1. Experiment on Federated Learning for Fraud Detection Using Synthetic Credit Card Transaction Data

3.1.1. Overview

This document outlines an experiment comparing centralized and federated learning approaches for detecting fraudulent transactions in synthetic credit card data. The experiment leverages federated learning (FL), a privacy-preserving technique that enables multiple clients to train a global model without sharing their private data. Various privacy and performance metrics are compared between the federated model and a centralized model to evaluate their respective effectiveness in fraud detection.

Experiment contact: Mathew Serek

3.1.2. Scenario (Use Case)

Fraud Detection in Credit Card Transactions

In this scenario, multiple financial institutions aim to detect fraudulent transactions by collaboratively training a machine learning model. The institutions maintain privacy by using federated learning, ensuring no raw transaction data is shared across parties. Each institution holds private credit card transaction data and contributes to the training process by updating a global model with its local data.

Privacy Considerations

Centralized Approach: All data is pooled and processed on a central server, raising privacy concerns but potentially improving model performance.

Federated Learning Approach: Individual datasets remain on local servers, and only model updates (e.g., weights) are shared to build the global model, minimizing privacy risks.

Use Case Example: Banks collaboratively train a fraud detection model using federated learning. Each bank retains its transaction records but contributes to a global model that aims to detect fraudulent activities. This enables the sharing of insights without exposing sensitive customer data.

3.1.3. Technology Involved

Federated Learning: A decentralized approach to training machine learning models where each client trains locally and contributes updates to a central server.

Logistic Regression: A simple but effective model used for binary classification to identify fraudulent transactions.

Synthetic Data: Simulated credit card transaction data generated using the Faker library to mimic real-world fraud detection scenarios.

DataBytes PTFI

3.1.4. Experimental Setup

Data Generation

Number of records: 50,000

Fraud cases: 1% (500 cases marked as fraudulent)

Client datasets: 7 clients, each receiving a subset of the data for training.

Each transaction contains fields such as Transaction Amount, Customer Age, and Transaction Type, which are used to train the model. Fraudulent transactions are identified with a binary label (Is Fraud), set at a rate of 1%.

Privacy-Preserving Technique

Federated Learning (FL): The dataset is divided among 7 clients, each training a local model without sharing raw data. A central server aggregates the clients' updates to build the global fraud detection model.

Centralized Model

For comparison, a centralized logistic regression model is trained on one client's dataset to evaluate performance in a non-privacy-preserving scenario.

3.1.5. Performance Metrics

Accuracy: The accuracy of both the federated and centralized models in detecting fraudulent transactions.

Privacy Implications: Qualitative assessment of privacy preservation using federated learning versus centralized data pooling.

3.1.6. Findings

The results compare the performance of the federated learning model and a centralized model trained on the same synthetic data.

Method	Accuracy	Remarks
Federated Learning Model	0.9895	The federated model achieved high accuracy despite distributed training.
Centralized Model	0.9897	The centralized model performed slightly better, but with no privacy benefits.

Table 3.1: Results Summary for Federated vs Centralized Models

Observations

Federated Learning Accuracy: The federated model achieved 98.95% accuracy on the remaining test dataset. This demonstrates that federated learning can perform competitively while maintaining data privacy.

Centralized Model Accuracy: The centralized model achieved a slightly higher accuracy of 98.97%, which is expected since all data is available for training in this setup.

Performance Gap: The difference in accuracy between the federated and centralized models is minimal, demonstrating that federated learning can approximate centralized learning performance while preserving privacy.

3.1.7. Conclusions

The experiment shows that federated learning is a viable approach for fraud detection in scenarios where data privacy is critical. Despite being distributed and privacy-preserving, the federated model performed comparably to the centralized model, with only a slight reduction in accuracy (0.9895 vs. 0.9897).

Key conclusions:

- Federated Learning offers significant privacy benefits, making it suitable for sensitive applications like fraud detection in finance, healthcare, or insurance.
- The accuracy trade-off between federated and centralized learning is minimal, suggesting that federated learning can be used effectively in real-world applications.
- Centralized Learning is marginally better in performance but at the cost of privacy.

3.1.8. Further Considerations

The experiment can be extended and improved in the following ways:

- **Advanced Models**: Incorporating more complex models like neural networks or decision trees to test how federated learning handles models with greater capacity.
- **Privacy-Enhancing Techniques**: Adding secure aggregation methods or differential privacy to further enhance privacy during the federated learning process.
- **Extended Metrics**: Evaluating other performance metrics such as precision, recall, F1-score, and training time to get a broader perspective on model performance.
- **Real-World Data**: Applying federated learning to real-world transaction data (subject to privacy laws) would provide more concrete insights into the practical viability of this approach.

The results of this experiment highlight that federated learning can be a powerful tool for privacy-preserving fraud detection in financial transactions, making it ideal for industries where privacy concerns and data security are paramount.

Full code for experiment location: *FL for CC Fraud Detection Demonstration*

3.2. Fraud Detection Using Neural Networks and Ray Parallel Computing in FL

3.2.1. Overview

This experiment aimed to evaluate the efficiency and accuracy of a Federated Learning (FL) setup using Neural Networks for detecting fraudulent transactions in synthetic credit card data. Two approaches were compared: a traditional sequential federated learning setup and a parallelized federated learning setup using Ray to simulate multiple clients concurrently. The objective was to compare execution times and assess whether Ray offers performance improvements at different client scales. **Experiment contact:** Pratham Shelar (s224144968)

3.2.2. Scenario (Use Case)

The use case involved fraud detection in credit card transactions across multiple financial institutions. Federated Learning allowed these institutions to train a global machine learning model collaboratively without sharing raw transaction data, thereby preserving privacy.

3.2.3. Federated Learning Setup

- **Clients:** 50 clients
- **Rounds:** 10
- **Batch Size:** 32
- **Features:** 5 generated features simulating credit card transactions
- **Model:** A simple neural network (FraudNet) with three fully connected layers:
  - Input Layer: 5 features to 64 neurons
  - Hidden Layer: 64 neurons to 32 neurons
  - Output Layer: 32 neurons to 2 output classes (fraud/no fraud)

3.2.4. Technology Involved

- **Flower Framework:** For federated learning simulation.
- **Torch:** For training the neural network model.
- **Ray Framework:** For distributed parallel execution.

3.2.5. Performance Metrics

- **Accuracy:** Measured for both the federated learning (FL) and centralized learning models.
- **Execution Time:** Compared between simulations run with and without Ray.

3.2.6. Results

The experiment ran simulations using 10 to 50 clients, and results were recorded for accuracy and execution time:

- **Without Ray (Sequential Execution):**
  - Execution time increased linearly with the number of clients, ranging from 30 seconds for 10 clients to 60 seconds for 50 clients.
  - Accuracy of the FL model with neural networks was **98.92%**, slightly lower than the centralized model (98.97%).
- **With Ray (Parallel Execution):**
  - Ray showed higher execution times initially for fewer clients (32 seconds for 10 clients), with the gap increasing as more clients were added.
  - For 50 clients, the execution time reached 65 seconds due to overhead from parallel task management.
  - The accuracy of the FL model with Ray integration remained consistent at **98.92%**.

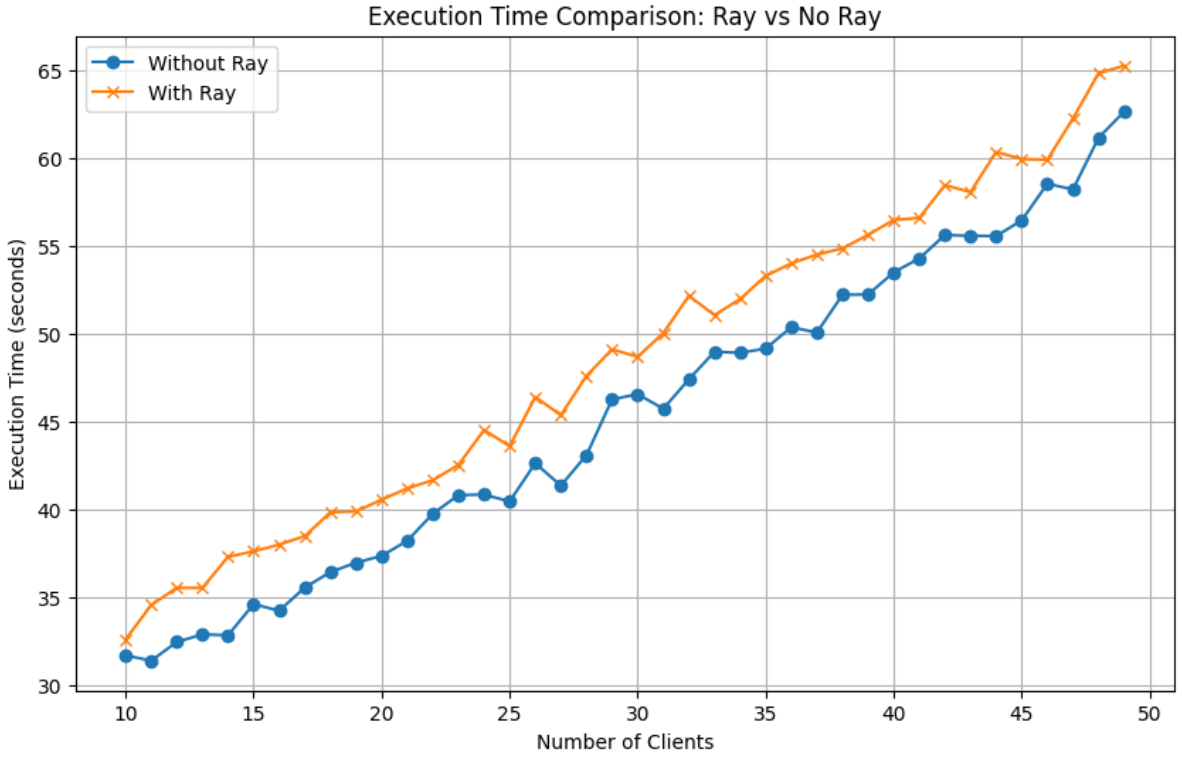

Figure 3.1: Execution-time comparison (Ray v/s no Ray)

3.2.7. Observations

- **Sequential Execution (Without Ray):** Execution time increased linearly with the number of clients, showing predictable scaling behavior.
- **Parallel Execution (With Ray):** Ray introduced additional overhead, making it less efficient than the sequential approach in this setup due to task management overhead.
- **Accuracy Comparison:** The accuracy remained consistent between both approaches (98.92% with Ray, 98.92% without Ray), and was only marginally lower than the centralized model's accuracy (98.97%).

3.2.8. Conclusion

While Ray offers potential benefits for distributed workloads, the additional overhead in this small to medium-scale setup outweighed its advantages. However, as the number of clients or the complexity of the model increases, Ray's parallelism could provide significant performance improvements in larger-scale FL scenarios.

3.2.9. Recommendations

- **Scalability Considerations:** Ray is more suitable for larger FL setups with more clients or more complex models. For small-to-medium workloads, its overhead may outweigh performance gains.
- **Optimization:** Future experiments should focus on optimizing Ray's resource management or selectively parallelising certain tasks to minimize overhead.
- **Hybrid Approach:** Consider using a hybrid approach where only a subset of clients uses Ray, to reduce unnecessary overhead for smaller datasets.

3.2.10. Future Scope

1. **Advanced Neural Networks in FL:** The experiment utilized a simple neural network (FraudNet). Future experiments can incorporate more complex neural network architectures, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), especially for more intricate fraud detection models. These architectures would enable FL models to handle larger, more complex datasets, improving accuracy and robustness in fraud detection.
2. **Federated Learning with Hyperparameter Tuning:** A potential improvement in future experiments would be the inclusion of automated hyperparameter tuning for the neural networks during the federated learning process. This would help optimize model performance across clients, especially in scenarios where data distribution differs.
3. **Scaling Ray for Large-Scale Federated Learning:** Although Ray introduced overhead in this small-scale experiment, future tests on larger datasets and a higher number of clients could explore how Ray scales in distributed FL. For example, running experiments with 100–500 clients and more complex models would provide a clearer understanding of Ray's potential for reducing training time at scale.
4. **Exploring Differential Privacy with Neural Networks:** Privacy-preserving techniques like differential privacy can be applied to FL neural networks. This would add a layer of security by introducing noise to the gradient updates, further protecting sensitive transaction data from being inferred during the training process.
5. **GPU Utilization with Ray:** Ray's parallelization capabilities can be combined with GPU acceleration to handle more computationally intensive tasks, especially when scaling neural networks in federated learning. This would allow for faster model training while maintaining distributed privacy-preserving mechanisms.
6. **Real-World Data Application:** Applying FL with neural networks to real-world credit card transaction data, subject to privacy laws, would offer more concrete insights into the practicality of using Ray for large-scale fraud detection. Additionally, integrating real-world data would allow researchers to evaluate the real-world performance of these privacy-preserving technologies.

Full code for experiment location: *FL Fraud Detection Using Neural Networks and Ray Parallel Computing*

DataBytes PTFI

3.3. Federated Learning as a Service (FLaaS) for Credit Card Fraud Detection

3.3.1. Overview

Federated Learning (FL) is a decentralized approach to machine learning where data remains localized, and models are trained collaboratively across multiple devices or servers without sharing raw data. This project focuses on implementing a Federated Learning as a Service (FLaaS) system for credit card fraud detection, utilizing a logistic regression model to identify fraudulent transactions. The experiment involves generating a custom transaction list using algorithms designed for this specific use case.

3.3.2. Objectives

- Develop a client-server architecture using Flask for distributed training.
- Use logistic regression as the model for both local and central training.
- Enable secure and scalable weight aggregation at the central server through API calls.
- Implement a prediction system via an API that takes input files from clients.
- Generate synthetic transaction data using custom algorithms for credit card fraud detection.

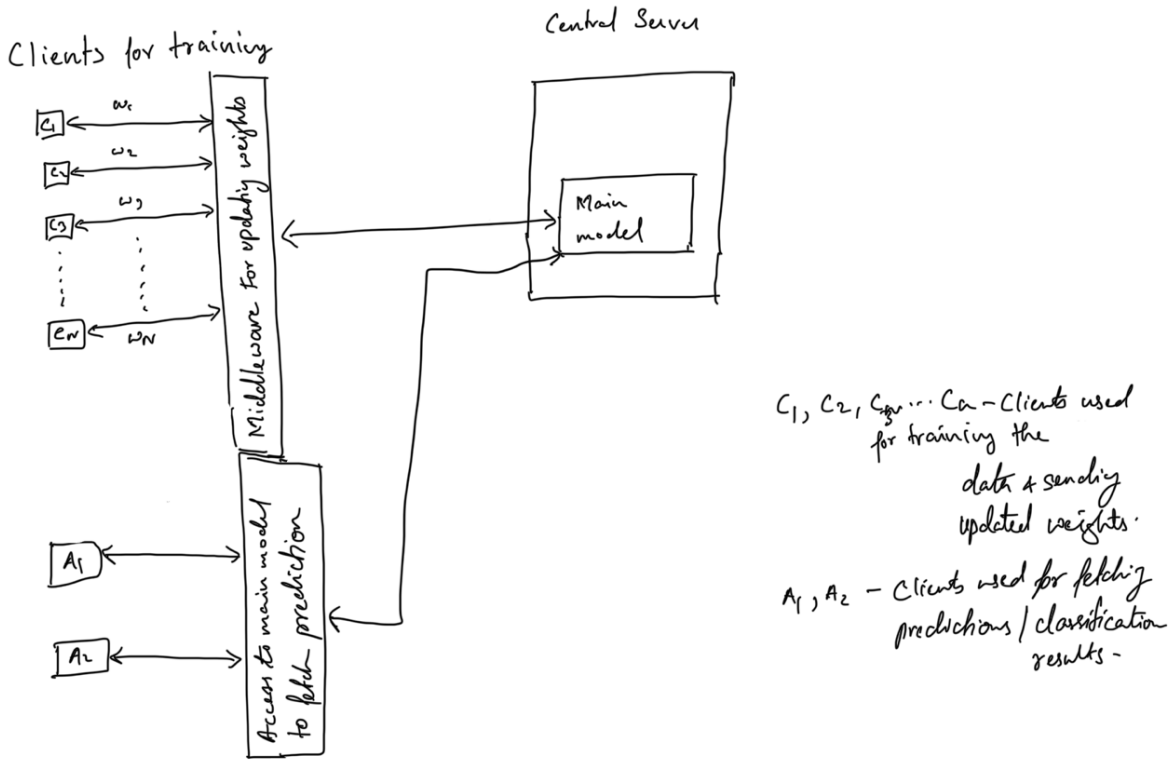

Figure 3.2: Federated Learning as a Service Representation Diagram

3.3.3. Architecture

1. Client Server Built using Flask, the client server hosts a logistic regression model that performs local training on synthetic transaction data.

- **Local Training:** The logistic regression model trains on custom generated transaction data at each client.
- **API for Training Update:** After training, updated model weights are sent to the central server through an API call.

- **Prediction API:** The client exposes an API for fraud prediction, where users can upload a transaction file to receive fraud likelihood scores.

2. Central Server The central server, also built with Flask, aggregates model weights from multiple clients.

- **API for Weight Aggregation:** Receives and aggregates model weights from various clients, updating the global logistic regression model.
- **Model Updates:** Weights from clients are periodically aggregated using techniques like weighted averaging to update the global model.

3. Transaction List Generation Custom algorithms generate synthetic credit card transaction data, ensuring that the dataset captures the nuances of real-world fraud detection scenarios. The transaction list includes both fraudulent and non-fraudulent data points to train the models on a realistic and balanced dataset.

4. Federated Learning Process

- **Local Model Training:** Clients use their own generated transaction lists for training logistic regression models locally.
- **Weight Sharing:** Updated model weights are sent to the central server through an API.
- **Global Model Update:** The central server aggregates weights to maintain and improve the global model.
- **Prediction:** Fraud detection predictions are handled on the client side through a separate API.

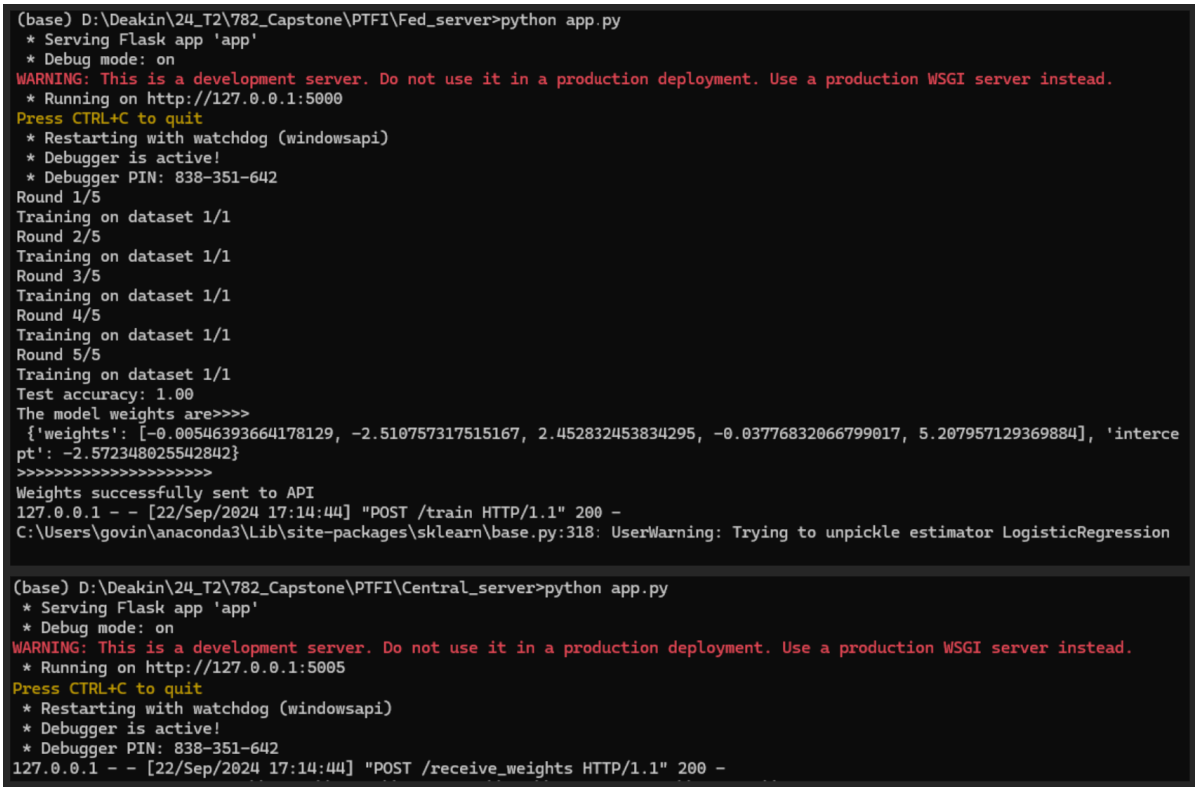

Figure 3.3: Federated Learning as a Service Server Response

Full code for experiment location: *Federated Learning as a Service GitHub Repo*

3.3.4. System Components

- **Flask Framework:** Flask was chosen for both client and central servers due to its simplicity and ability to handle lightweight RESTful APIs.
- **Logistic Regressor:** Logistic regression, ideal for binary classification tasks like fraud detection, serves as the core model for both local training and the global model at the central server.
- **Custom Transaction Generator:** A custom algorithm was designed to simulate credit card transactions. This algorithm creates realistic fraud and non-fraud data for training purposes.
- **API Communication:**
  - **Client to Central Server:** Model weights are sent to the central server after local training.
  - **Prediction API:** Clients expose an API endpoint that accepts transaction files and returns a fraud prediction.

3.3.5. Key Implementation Steps

1. **Local Training on Clients:** The custom generated transaction list is used to train a logistic regression model on each client. Clients only share the updated model weights with the central server, ensuring data privacy.
2. **Weight Aggregation at Central Server:** The central server aggregates the model weights from multiple clients, updating its global model by combining weights in a privacy-preserving manner.
3. **Prediction via API:** Clients provide an API for fraud predictions, where transaction data can be uploaded for real-time fraud detection.

3.3.6. Challenges and Solutions

- **Data Privacy:** As no raw transaction data is shared between clients and the central server, privacy is maintained.
- **Transaction Data Generation:** Custom algorithms were designed to create realistic transaction datasets that include both fraudulent and non-fraudulent transactions, ensuring robust model training.
- **Scalability:** The API-driven architecture using Flask ensures that the system is scalable, with the potential to add more clients or central servers as needed.

3.3.7. Future Directions

1. **Advanced Model Support:** Expanding the model portfolio to include more advanced classifiers such as SGD Classifiers or deep learning models could further improve accuracy.
2. **Improved Transaction Generation:** Enhancing the custom transaction generation algorithms to better capture rare fraud patterns could increase the system's predictive power.
3. **Real-time Model Monitoring:** Adding real-time monitoring of model performance and transaction patterns could help detect anomalies faster.
4. **Cloud Deployment:** Deploying the system in a cloud environment could enhance its scalability and enable it to handle large numbers of clients and transactions.

FLaaS Proposal PDF Link (by Pratham): *FLaaS Proposal for PTFI*

3.3.8. Conclusion

The Federated Learning as a Service (FLaaS) system provides a privacy-preserving, scalable solution for credit card fraud detection using logistic regression. By leveraging custom algorithms to generate transaction data and enabling decentralized model training through APIs, the system demonstrates a robust approach to tackling fraud in a federated setting.