Coventry University | London

Transaction Monitoring with Machine Learning & Homomorphic Encryption.

FinTech MSc Dissertation

13460222

10.12.2023

Contents

Abstract

This paper explores using privacy-preserving machine learning with fully homomorphic encryption (FHE) for improving anti-money laundering (AML) transaction monitoring programs. It leverages a recent framework, Privacy-Preserving Tree-Based Inference with TFHE, to enable secure inference on encrypted data. The classification model XGBoost is tested on its FHE equivalent ConcreteXGBoost using the Concrete-ML library. Experiments are conducted on a robust synthetic financial transaction dataset modeling an entire virtual ecosystem. The IBM Transactions for Anti Money Laundering dataset models 5+ million transactions with 3,600 cases of laundering across banks, individuals, and companies. The dataset provides a rare comprehensive view across a financial system. Results demonstrate feasibility of encrypted XGBoost delivering 90% accuracy in detecting money laundering patterns. Exact predictions are produced after FHE computations. However further improvements on the model are needed, i.e. more advanced parameter tuning to minimize performance drop from model quantization. Overall, this study makes a case for collaborative AML frameworks where institutions share encrypted transactional data to train more powerful detection models.

1. Introduction

Money laundering refers to the process of concealing illicit funds by disguising their illegal origin. While the global scale of money laundering is unknown, United Nations estimates 2-5% of global GDP or up to $2.0 trillion dollars are laundered annually (UN, 2022). The volume continues to grow as laundering schemes gets even more sophisticated. This increases the pressure on regulators and financial institutions (FI). In 2019, it was revealed that Estonian Branch of Danske Bank alone is included in the laundering of 230 billion USD, which resulted in a resignation of the CEO (US DOJ, 2022). In another case, NatWest faced criminal prosecution by Financial Conduct Authority (FCA) for money laundering failings resulting in £264.8 million fines, discounted from £397 (FCA, 2021). Furthermore, FCA issued £577 million in fines in 2021 and half of it were due to compliance failures (Rach, 2021), %400 increase compared to previous year. Increasing volumes and involvement of large banks, either voluntarily or unknowingly, in these launderings increase the pressure on both financial institutions and regulators. To stay compliant, banks have been investing in new technologies. The average annual cost of compliance in the UK is has reached to £194 million per bank. Retail and commercial banks seem to be investing most amongst these firms, with estimated £264 million each (LexisNexis, 2023). This has caused significant decreases in Bank's profits, leading them to find more efficient solutions to reduce costs while staying compliant.

Machine Learning (ML) models can provide more sophisticated algorithms for pattern recognition that can work efficiently on Big Data and get insights from it. ML has already improved AML programs significantly, especially the transaction monitoring processes where banks are failing to detect suspicious events and report high false-negative rates. (McKinsey & Company, 2022). They are now heavily investing in ML to comply with money laundering regulations in a cost-efficient way. However, they are struggling with leveraging this technology efficiently.

Sophisticated laundering schemes involves multiple accounts through multiple banks, using different transaction types. Criminals can easily hide themselves from the ML algorithms when they are trained on single bank's data. As they will not be able to learn the complete patterns. This is why in order to ML algorithms to capture such complex patterns, complete view of transactions across entire financial network is required. However, this cannot be achieved due to sensitive nature of the financial data, unless the security and privacy of the data is well established. Fortunately, recent developments provided efficient Privacy-Preserving Machine Learning (PPML) frameworks. They use secure encryption schemes like Fully Homomorphic Encryption (FHE) for advanced machine learning algorithms such as Gradient Boosting and Neural Networks.

This study explores whether advanced ML algorithms can be efficiently used on encrypted data in a secure way. To achieve that, we utilize a recent Privacy-Preserving Machine Learning (PPML) framework by Frery et al. (2023). PPML leverages TFHE scheme, a secure cryptographic Fully Homomorphic Encryption (FHE) technique, which allows computing directly on encrypted data without decryption. Furthermore, we use a synthetic robust and realistic dataset with a simulation of financial transactions, both legitimate and fraudulent, across an entire virtual ecosystem (Altman, 2023). This data enabled us to test the efficiency on ML algorithms when transactions from entire financial network is available. As a result, this study provides a simple demonstration of the proposed solution for transaction monitoring programs.

In this dissertation, we first introduce notable recent development in this field, classification ML algorithms and FHE schemes. First, we compare classification algorithms and select the most suited one for our task, which turns out to be Gradient Boosting Algorithm, XGBoost. Then we introduce and model ConcreteXGBoost from Concrete-ML library, to run the model on encrypted data. After hyperparameter tuning, we evaluate the empirical results and compare XGBoost with its FHE equivalent.

2. Our contribution

In 2023, we've witnessed significant improvements in the field. Frery et al. (2023) introduced Privacy-Preserving Tree-Based Inference with TFHE, which provided an efficient framework for using Tree-Based algorithms with FHE. On the other hand, Altman et al. (2023) provided a realistic synthetically generated AML (Anti-Money Laundering) datasets, simulating the entire financial ecosystem. This was crucial for our research as there was no reliable financial data publicly available, and any access to real data is almost impossible.

This dissertation presents a new application for Privacy-Enhancing Machine Learning with FHE. It aims to combine these recent works and demonstrate that using machine learning algorithms on encrypted data, we can overcome long standing challenges for AML programs. This study doesn't provide a final framework ready to use by financial institutions, but it provides a path that in the end, financial institutions and regulators can work together to leverage machine learning with FHE to fight against money laundering. FHE plays a critical role since it will allow sensitive information sharing amongst institutions and regulators, which will further enhance predictive performance of ML algorithms.

First, we compared several classification algorithms and tested them on our data. A powerful ensemble technique, XGBoost (eXtreme Gradient Boosting) algorithm, is selected for its state-of-art performance and usefulness in financial datasets. Then we utilized concrete-ml library in python to compare the results from XGBoost with its FHE equivalent, ConcreteXGBoost.

3. Literature Review

3.1 Related Works

The paper by Frery et al. (2023) explores the application of fully homomorphic encryption (FHE) to tree-based ML models for privacy-preserving machine learning (PPML). FHE enables complex computations on encrypted data, making it well-suited for PPML. Prior FHE research has relatively underexplored tree algorithms compared to neural networks, even though on tabular datasets decision tree models deliver state-of-the-art accuracy while being more robust and accessible than neural networks. The work addresses that gap by implementing Private Decision-Tree Evaluation (PDTE) based on the TFHE, an FHE scheme that allows arbitrary depth computation circuits, and associated Concrete-ML library. The same FHE formulation adopted in our research.

The core technical contribution involves modifying TFHE to best represent decision trees in an encrypted parameterizable integer format amenable for homomorphic calculations. Custom integer encoding and TFHE's powerful programmable bootstrapping (PBS) mechanism are leveraged to efficiently evaluate tree models on encrypted data. First, they use ciphertexts to store multi-bit integers instead of single bits. And the crypto-system is parameterized to allow accurate homomorphic accumulation over these integers without message corruption. Secondly, quantization is applied on individual data features to constrain the integer range. This controls message space needed in the ciphertexts. By leveraging tensorized computation on integers and PBS, private decision tree evaluation is realized. Finally, an automated optimization strategy is used to determine crypto-system parameters. The size of the message space can be easily changed by adjusting the quantization of the input data, which enables us to find the optimal point that maximizes FHE inference speed while retaining the target performance. Further information on FHE, TFHE, PBS and tree-based models is given in the methodology part of this dissertation.

They tested their technique with three compatible models: decision trees, random forests, and gradient boosted trees. Experiments performed on benchmark datasets demonstrate encrypted models achieving accuracy, f1-score, and average precision extremely close to original counterparts. Furthermore, their solution provides easy tuning to find optimal quantization vs inference speed trade-off. The security of the encrypted computation is handled in the background where crypto-system parameters are determined automatically based on the target tree model. This auto-tuning ensures leveled homomorphic operations correctly aggregate without overflow or any other errors. In summary, the tailored TFHE scheme for tree algorithms facilitates secure and efficient privacy-preserving ML without compromising on model performance. Their work offers significant advancements in the field which we build our model upon, using these techniques with financial transaction data.

Like the previous work by Frery et al., this paper by Stoian et al. (2023) leverages TFHE and its fast PBS mechanism for homomorphic encryption. However, their contribution is tailored to evaluating Deep Neural Networks (DNN) in a privacy-preserving manner. They demonstrate how to construct DNN that are compatible with the constraints of TFHE using the Concrete-ML library, an open-source implementation of TFHE.

The constraints imposed by TFHE, chiefly integer-only operations and restrictions around conditional statements, also apply for neural networks. The authors use Quantization-Aware Training (QAT) methodology that add quantizers to network activations and weights during training. QAT enables quantization with less than 4-bit weights and activations without losing the accuracy, where in other quantization methods, like per-channel quantization or logarithmic quantization, as few as 4-bits are required. Reducing number of required bid-with is crucial as computational costs of PBS mechanisms depends on the number of bits of the encrypted value to be bootstrapped. Further details of their model are irrelevant to our research, but the results are significant as it is important to understand the application range of this revolutionary method.

They implemented their networks in PyTorch with Brevitas (Pappalardo, 2023) and converted to FHE with Concrete-ML [17]. Their approach supports Convolutional Neural Network (CNN) architectures of arbitrary depth and complexity to be converted to an encrypted equivalent with negligible accuracy loss. Evaluations on image classification datasets demonstrate encrypted models with ~2-6% drop in accuracy compared to originals. Furthermore, A 6-layer CNN model retained 98.7% accuracy on MNIST dataset with encrypted inferencing. This shows the potential to handle models of arbitrary depth, and that the optimizations made in the Concrete-ML library creates the path for not just tree algorithms but also deep neural networks to be homomorphically evaluated. The core ideas around integer encoding, bootstrapping-based function representation and automated parameter tuning extend nicely to neural network architectures as well. This paper makes a compelling case for using TFHE within the library to realize high-fidelity yet private deep learning applications.

Another work by Egressy et al. (2023) focuses on detecting money laundering transactions in financial networks. The authors highlight that existing solutions struggle with sophisticated criminal techniques like layering illicit money flows across accounts and institutions. Instead of tree ensembles, they propose using Graph Neural Networks (GNNs) to uncover money laundering patterns in the financial transaction graph. Their goal is to demonstrate the promise GNNs for learning representations. Even though we're using a tree-based model in our research, this paper provides significant improvement in the field which can be combined with FHE in the future. (Note: Our current time limitations prevented us to try both models with FHE and provide a comparison.)

GNNs have emerged as highly effective models for relational reasoning across domains like biology, physics, social networks, and weather forecasting. More recently, there is growing interest in applying GNNs for financial crime detection use cases like anti-money laundering (Cardoso, Saleiro, and Bizarro 2022; Kanezashi et al. 2022; Weber et al. 2019, 2018; Nicholls, Kuppa, and Le-Khac 2021). The underlying premise is that financial crimes manifest as subgraph patterns in the transaction network topology. They provided some established money laundering pattern examples in Fig. 1 (from Egressy et. al. 2023), showing that the problem fits nicely to the use of GNNs.

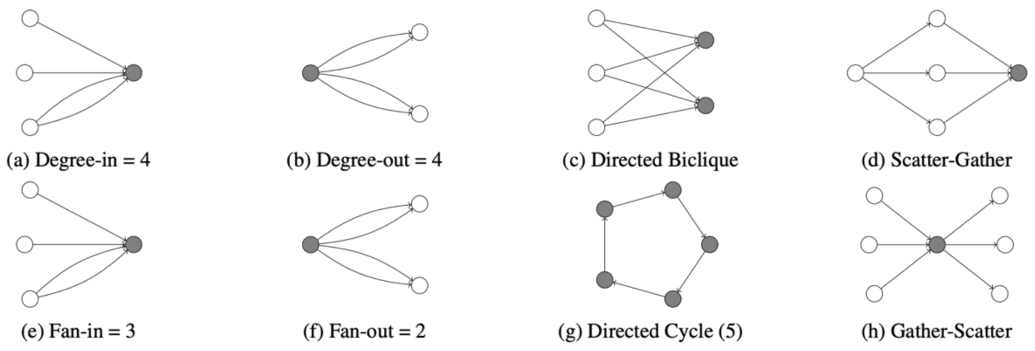


Figure 1: Money Laundering Patterns. The gray fill indicates the nodes to be detected by the synthetic pattern detection tasks. The exact degree/fan pattern sizes here are for illustrative purposes only.

However, previous GNN architectures were unable to efficiently handle financial networks which are in fact directed multigraphs where edges, representing transactions, are directed and can be found in multiple between nodes, representing accounts. Moreover, GNNs cannot identify certain type of subgraph patterns like cycles. Previous works focusing on solving this problem were not successful, even on undirected simple graphs (You et al. 2021; Huang et al. 2022; Papp and Wattenhofer 2022; Zhang and Li 2021; Loukas 2019; Sato, Yamada, and Kashima 2019).

Egressy et al. (2023) contributes both challenges by introducing the first GNN architecture specifically tailored for directed multigraphs that can detect any subgraph patterns. Proposed architecture can transform message-passing GNN structure into a directed multigraph neural network, by cleverly combining following individual adaptations from previous works and adjusting them to directed multigraphs: Reverse Message Passing (Jaume et al. 2019), Port Numbering (Sato, Yamada, and Kashima 2019), and Ego IDs (You et al. 2021). They provide theories and their proofs for these individual adaptations.

Message Passing Neural Networks (MPNNs) are most prominent family of GNNs (Egressy et al. 2023). They work in three steps to constitute a layer of the GNN: Every node sends its current state to all neighboring nodes, each node gathers all messages from neighbors, and each node updates its own state using the neighbor messages to produce a new state. These 3 steps can be repeated to connect further reaches of the graph. When run on directed edges such as money transfer networks, a limitation is that nodes only receive messages from accounts they get money from, not accounts they send money to. So, they cannot count the outgoing transactions and cannot analyze important patterns of money flowing out like the number of different recipients. To address this, the authors propose using separate steps to collect incoming and outgoing money flows, adding Reverse Message Passing. This lets nodes compare money coming in and going out. The authors provide theoretical proof to the following proposition: MPNN with sum aggregation and reverse MP can solve degree-out. Then they provide empirical proof with using the synthetic pattern detection tasks, which uses randomly created anti-money laundering (AML) subgraph patterns seen in Fig. 1.

Banks often notice multiple transfers happening between same account pairs over time, which is represented as parallel edges in transaction networks. For reliable fan-in and fan-out pattern detection, a model should identify if the edges are from the same neighbor or a different one. To overcome this challenge and increase the expressivity of their model, Egressy et al. (2023) adapt Port Numbering (Sato, Yamada, and Kashima 2019) to directed multigraphs. They assign each directed edge an incoming and an outgoing port number, so edges receive the same incoming/outgoing if they are coming from or going to the same node. Furthermore, they use transaction timestamps to order the neighbors. This is crucial in financial fraud detection as timestamps can define the meanings of the patterns. They calculate port numbers in advance for the train, validation, and test sets to avoid longer training and inference times. The authors provide both theoretical and empirical proof for the following proposition: An MPNN with max aggregation, multigraph Port Numbering, and reverse MP can solve fan-in and fan-out.

While the combination of Reverse MP and Port Numbering is helpful to detect some patterns, they found not to be sufficient for detecting directed cycles, scatter-gather patterns, and directed bicliques (Egressy et al. 2023). To detect cycles, the authors further adapt Ego IDs (You et al. 2021) to their architecture, which defines a center node with a distinct (binary) feature to recognize when a sequence of messages cycles around to it, therefore identifying cycles that includes that node. Even though Ego IDs are helpful for detecting short cycles, they do not enable detection of longer cycles alone (Egressy et al. 2023; Huang et al. 2022). However, when combined with reverse MP and Port Numbering, Ego IDs can detect cycles and all patterns found in Fig. 1. Most importantly, these adaptations can be used to assign unique IDs to each node in the graph. The authors proves that an MPNN with these adaptations are known to be universal and can theoretically identify any directed subgraph patterns.

The authors designed synthetic pattern detection tasks for testing their propositions, using AML subgraph patterns seen in Fig. 1. They introduced random circulant graph generator to randomly create patterns. Due to the extreme difficulty of finding real-financial data, they use synthetic financial transactions dataset generated by IBM to test their model. Their selection of IBM dataset is crucial as the results can provide insights for our research, as we use the same dataset with tree-based models instead of GNNs. They also test their model on real-word Ethereum transaction data for cryptocurrencies and Real-World Directed Graph Datasets. Money laundering datasets has significantly low ratio of laundering transactions, which makes them very imbalanced. Therefore, accuracy and other scoring metrics would not be meaningful. Instead, the authors use F1 score to evaluate their model.

The authors use GIN by Hu et al. (2019) as their main GNN base model with integrating three adaptations to it. The propositions for three adaptations are proved using the synthetic graphs, which has already mentioned above. All the GNNs that are equipped with reverse MP score above 98%. Furthermore, for the fan-in and fan-out task, the combination of reverse MP and Port Numbering score above 99%. The combination of reverse MP, Port Numbering, and Ego IDs, performs well on all of the subtasks. Significant improvements in F1 scores are observed when combination of the three used together. For the scatter-gather detection, the most extreme case, the minority class F1 score jumps from 67.84% to 97.42% when ego IDs added in alongside with reverse MP and Port Numbers. The results clearly show the need for using all three adaptions together. The authors further tested their GIN model with the three datasets. Using the Small-HI IBM dataset, the minority class F1 score of GIN jumped from 28.7% to 59.2% with the adaptations, a gain of more than 30%. Similarly impressive results are observed on a real-world dataset, where the model outperformed all baselines with an 15% F1 score improvement. We will further compare this result with our work based on the extreme Gradient Boosting classifier, to observe how different models performed for financial crime detection.

This paper by Altman et al. (2023) introduces AMLworld, a detailed multi-agent simulator modeling financial transactions between individuals, corporate entities, and banking intermediaries in a virtual economy. The synthetic financial data includes legal and illicit funds labeled for money laundering. The key motivation is generating a realistic and standardized datasets at scale to train anti-money laundering (AML) models, overcoming limitations of privacy-restricted real data. They use relational tables and graphs to store and represent financial transaction data. A graph-based representation enables extracting complex patterns, such as cycles mentioned previously, by exposing the connections between underlying data objects. We already mentioned that financial transaction networks are in fact directed multigraphs, and this graph structure underlies many financial crime analysis techniques. They further contributed to the field by calibrating the generator to match real transactions as closely as possible and by making six AML datasets with varying size publicly available, for development and comparison of new AML models. Although evaluation of the models is beyond this paper’s scope, initial experiments for testing the datasets are conducted using GNNs and Gradient Boosted Trees (GBT). In a keyway, the measurements showed better results compared to real data. This is reasonable considering there is no reliable labeling in real financial data where most of the money laundering is not detected. They further demonstrate that financial institutions can achieve higher F1 scores if they share the transactions information. Significant improvements are observed when shared machine learning model used with complete information sharing. This supports the objective of our dissertation and highlights the value proposition of developing privacy-preserving machine learning model for AML practices. Furthermore, the authors also provide open source GNN code, which can be used and compared with the previously explained model of Egressy et al. (2023).

This paper by Ali et al. (2023) focuses on financial statement fraud (FSF) detection. They utilize a powerful ensemble technique called XGBoost or Extreme Gradient Boosting which is the same machine learning algorithm we use in our model. Such accounting manipulation is difficult to spot but causes significant investor losses annually. The main goal is developing a classification model leveraging published financial data to reliably identify fraudulent accounting practices. They use a dataset of 950 Middle Eastern and North African (MENA) corporations spanning 8 years and various sectors. The dataset contains 26 quantitative features extracted from balance sheets and income summaries. This includes specific ratios connected by prior research to conditions encouraging earnings manipulation like aggressive revenue recognition, cost capitalization etc. So in essence, the features comprise reported monetary amounts from annual filings as well as derived metrics that reveal potential accounting distortions - an established diagnostic approach. The issue of class imbalance in the dataset is addressed by applying the Synthetic Minority Oversampling Technique (SMOTE) algorithm. Experiments on this imbalanced dataset reveals their optimized XGBoost solution achieving over 96% overall accuracy and 0.83 F1 score in detecting minority illicit class. Comparisons to alternative machine learning algorithms like Support Vector Machine (SVM), AdaBoost, Random Forest (RF), Logistic Regression (LR) and Decision Tree (DT) validate the superiority of properly tuned gradient tree ensembles. While not handling sensitive commercial data directly, demonstrations on open MENA corpora lends credibility regarding tackling opaque financial crimes in practice. The performance of the model demonstrated XGBoost's capability of dealing with imbalanced financial data. It is important for our research as we also use highly imbalanced financial dataset. While Egressy et al. (2023) uses GNNs to handle such dataset, now this study shows that XGBoost with SMOTE can handle the issue of class imbalance as well.

The paper by Chillotti et al. (2020) conducts first experiments on the new library developed at Zama, which implements a variant of the TFHE fully homomorphic encryption scheme that leverages efficient bootstrapping. They explore using FHE to enable private deep neural network inferencing, by introducing programmable bootstrapping (PBS) which enables fine control of the noise. As we use the same library in our model, details of the TFHE scheme and PBS are provided in the methodology section. They conducted the experiments using MNIST dataset on neural networks with depth 20,50 and 100. The networks include dense and convolution layers with activation functions, with each layer having at least 92 active neurons. Preliminary results show 5% drop in accuracy, but there is an opportunity for improvement with better parameter tuning. In fact, the authors introduced fully automated parameter selection since, which is used in recent works mentioned above.

Several alternatives exist for fully homomorphic encryption (FHE) schemes for machine learning applications. This paper by Sun et al. (2020) proposes an improved FHE scheme suitable for private machine learning classification algorithms. Proposed FHE scheme is built upon HElib, a FHE library implementation based on Brakerski's FHE scheme (BGV12). They introduce three key efficiency enhancements. HElib's implementation of BGV12 uses the modulus switching technique first, which is applied on three ring elements, and then use the relinearization technique to reduce the ciphertext size. To improve efficiency, the authors use relinearization technique first, reducing the ciphertext size from three ring elements to two, and then apply modulus switching technique to decrease modulus and decryption noise. Another improvement is that there is no need for relinearization and modulus switching if there is additive homomorphic or no homomorphic operation, while both techniques are used in HElib's every implementation of BGV12 homomorphic. Finally, they leverage simple instruction multiple data (SIMD) to increase the speed of homomorphic operations. Comparisons made by simulation results show that the efficiency of their FHE scheme with SIMD is more efficient than the FHE library HElib with SIMD and Khedr's FHE scheme. Furthermore, considering the implementation times and efficiency, proposed FHE scheme performed better with private decision tree classification than the other two schemes. This work provides an alternative to TFHE, another FHE scheme we use in this dissertation. Unfortunately, we were not able to test it as HElib library is not compatible with the Python.

There are other approaches encouraging application of machine learning to sensitive data. Hunt et al. (2018) introduces a new system called Chiron for privacy-preserving machine learning as a service. Chiron enables organizations to leverage cloud-based machine learning (ML) services without revealing their sensitive training data. It uses hardware-isolated trusted execution environments offered by Intel SGX enclaves to isolate client data and code during model training. A client provides encrypted training data that is decrypted within the enclave sandbox. Core technical innovation is the use of a Ryoan sandbox to further prevent the ML code from leaking. This code defines model architecture, loss functions etc. and drives training. Strict data confinement prevents leakage but retains flexibility for proprietary implementations unlike alternatives requiring code disclosure. Orchestration uses fixed size messages to prevent timing channels. Empirical results on image classification datasets demonstrate ability to train deep neural networks with around 20% runtime overhead and less than 6% drop in accuracy compared to baseline due to roundtrip latency. This establishes practical viability especially given commercial adoption drivers. Query response after deployment sees negligible impact.

Another paper by Mohassel, Zhang (2017) presents privacy-preserving for machine learning protocols on collaborative data, using the stochastic gradient descent method. They focus on machine learning algorithms for training linear regression, logistic regression, and neural network training. They adopt a two-server model where data owners process, encrypt and/or secret-share their data between non-colluding servers. These servers then employ secure two-party computation (2PC) to train ML models over the joint dataset without leaking information beyond the final parameters. They propose multiparty computation (MPC) alternatives to nonlinear functions such as softmax and sigmoid, which provide a significant enhancement by allowing different parties to train their models on combined data without revealing any information. This is the first implementation of privacy preserving machine learning for training neural networks. Empirical results validate that their system is significantly more efficient than the state-of-the-art solutions for the same problem. This paper provides valuable insights for future works as TFHE scheme is currently unable to perform training on encrypted data.

3.2 Classification Algorithms

Fraud detection is known as a binary classification problem, which is identifying the classes of new observations based on a training set of data containing observations whose class is known. The financial transactions dataset for training is already labeled, indicating if a transaction is fraudulent or not. This binary output feature makes our problem a supervised classification task. We will first introduce several Supervised Machine Learning (ML) classification techniques, then compare their results to find the most effective one based on our dataset. Following Supervised ML algorithms are found to be useful for our problem: Logistic Regression, Bayesian Networks, Decision Trees, Random Forest, Gradient Boosting and Neural networks.

Logistic regression is a classification function that uses a single multinomial logistic regression model with a single estimator (Cramer, 2002). It predicts class probabilities based on the distance from estimated decision boundary between the classes. The points very far from the boundary are very likely or unlikely to be in a class. Probabilities approach the extremes more rapidly (0 and 1) when dataset is larger. It is an approach to probabilistic prediction, like Ordinary Least Squares (OLS) regression, but with a dichotomous outcome. It is commonly used for applied statistics and discrete data analysis. However, it can provide inaccurate predictions if model assumptions are not strong.

Bayesian Networks are graphical models for probability relationships among a set of features. They are well-known representative of statistical learning algorithms, and they are most likely to consider the structural relationship among the features compared to Decision Trees and Neural Networks. Naïve Bayesian (NB) networks are simple demonstration of Bayesian networks which are composed of directed acyclic graphs (DAG) with one parent and several children nodes (features), with a strong assumption of independence between child nodes (Akinsola et. al, 2017). They use Bayes' theorem to estimate class probabilities. NB classifiers require short computational time for training, which is its major advantage. However, they are not suitable for datasets with many features since constructing such large network is not feasible. They also don’t perform well with correlated features.

Decision Trees (DT) are tree-like models, with nodes and branches, that classify instances by sorting them based on feature values. Each Node represents tests on a feature or attribute, each branch represents the outcome of the test, and each leaf node at the end of the branches represents the final classification decision. Several advantages of DT’s are their interpretability, simplicity, and ability to easily handle interactions between features (Ali et al., 2023). They can handle both numerical and categorical features, and there is no need for scaling or other preprocessing steps. However, they are not efficient when dealing with high dimensional data since they can be sensitive to noise which can lead to overfitting.

Ensemble methods, such as Random Forests (RF), AdaBoost and Gradient Boosted Trees, can be used to improve the performance of decision trees by combining the predictions of multiple trees. This helps to avoid overfitting and increase robustness. Random Forest (RF) is a technique that combines multiple decision trees (DTs) to address the high variance and overfitting tendencies of single DT models (Ho, 1995). It works by training each DT on a randomly selected subset of the training data and features, then returns the class with majority over all the trees in the ensemble. Randomness is introduced with two ways, Random Sampling of Data and Random Subset of Features. By combining predictions from multiple trees and introducing randomness, RFs tends to be more robust to noise and overfitting.

Boosting is an ensemble strategy with the key goal of reducing bias and variance. The first successful boosting algorithm developed for binary classification was AdaBoost, short for Adaptive Boosting (Freund & Shapire, 1997). The key idea is to minimize the error of a weak learning algorithm, and combine multiple ‘weak classifiers’ into a single ‘strong classifier’ (Sreedharan et al., 2020). Weak learner refers to any simple model that can classify examples slightly better than random guessing. It works by sequentially adding and training weak models on weighted training data, until either a desired number of models are trained or there is no further improvement observed. On the other hand, Gradient Boosting builds decision trees sequentially, where each tree corrects the errors of the previous ones. This process enables it to capture complex relationships in the data. It also uses gradient descent optimization to minimize the loss function. Gradient Boosting has more hyperparameters to tune compared to Random Forests, so they require careful hyperparameter tuning but can yield state-of-art results. Gradient Boosting is often used when high predictive accuracy is needed. They are also effective in handling imbalanced datasets and skewed class distributions, which makes it a perfect candidate for our model.

Finally, we would like to introduce Neural Networks (NN). Even if we were not able to test our dataset with NNs, previous papers working with Deep Neural Networks and Graph Neural Networks (Stoian et al. 2023; Egressy et al. 2023) proved their effectiveness in handling such problems on the same dataset we're using. NNs are inspired by the structure and function of the human brain. They consist of interconnected nodes, also known as neurons or units, organized into layers. A neural network consists of an input layer, one or more hidden layers, and an output layer. The input layer receives the initial input data, the hidden layers process information, and the output layer produces the final result. During training, the network goes through a process of forward propagation where input is passed through the layers to produce an output. Backpropagation is used to update the weights by computing the gradient of the loss function with respect to the weights and adjusting them accordingly. Such optimization algorithms are used minimize the loss function. They can automatically learn hierarchical representations of features, which allows them to capture complex patterns. They can capture many kinds of relationships since they are not limited by assumptions of normality, linearity, variable independence and so on. However, their success often requires careful tuning of hyperparameters and substantial computational resources. Neural networks have shown remarkable success in various fields, including image and speech recognition, natural language processing, and even for bank's AML programs. Deep learning, which involves training deep neural networks with multiple hidden layers, has become very popular in machine learning due to its ability to automatically learn complex representations from data.

3.3 Homomorphic Encryption

The need for performing computations on encrypted data led to the emergence of homomorphic encryption (HE). HE characterized by its homomorphic properties, which means it supports operations on encrypted data. For an encryption scheme to be homomorphic, there must be two operations '·' and '' such that

$$f(a \cdot b) = f(a)\ f(b),$$

for all valid messages a and b. These operations can be additive or multiplicative.

The concept of HE traces back to 1978 but gained its first practical success in 2009 with Craig Gentry's lattice-based construction (Gentry, 2009), supporting both additions and multiplications on ciphertexts. However, lattice-based HE schemes introduce noise in ciphertexts, which limits the number of homomorphic operations can be made since too much noise will affect its accuracy (Ramalho,2023).

Fully Homomorphic Encryption (FHE) schemes address this limitation by implementing bootstrapping technique. Bootstrapping works by homomorphically evaluating the decryption circuit to periodically refresh the noise inside ciphertexts. It Reduces the noise, therefore allows us to increase the size of encrypted programs. Gentry showed a theoretic pathway for performing unlimited number of consecutive operations over the same ciphertext, but it had limitations in efficiency that prevented practical applications (Ramalho,2023). Since then, the field of FHE cryptogaphy advanced rapidly through several generations of improved schemes.

The first second-generation FHE scheme, Brakerski-Gentry-Vaikuntanathan (BGV) scheme, is introduced in 2011. Following by another notable scheme, Brakerski/Fan-Vercauteren (BFV) scheme, in 2014. They employed Ring Learning With Errors (RLWE) to improve efficiency and control noise growth, and they offered faster computations to be used in real-word applications without the need of bootstrapping. BFV is a scale-invariant scheme, meaning it works with a single modulus for ciphertexts, and it outperforms predecessors by linear noise growth during multiplication. In contrast, BGV uses a small set of moduli associated with levels, adapting ciphertexts as computations progress (Brakerski et al., 2011; Brakerski et al. 2014).

The quantum computers pose a threat to existing public-key cryptographies since they are able to break existing encryption techniques in days (Ramalho, 2023). Modern HE schemes use hardness assumptions related to lattice problems, rather than directly working with lattices, for their security. The Learning With Errors (LWE) problem (Regev, 2005) is proposed as a quantum-secure alternative to previous public-key methods. LWE provides the hardness assumption for many FHE cryptosystems. It works by introducing small-bounded noise into linear operations to hide secret key information. RLWE (Lyubashevsky, 2010), the LWE problem applied to Polynomial Rings instead of integer lattices, provides more compact ciphertexts and faster performance for FHE schemes. Placing plaintext and ciphertext in separate rings, the schemes utilize encoding methods like constant polynomial representation and integer encoding.

A third-generation scheme FHEW (Ducas & Micciancio, 2015) is introduced with converting a new cryptosystem (Gentry et al., 2013) into an efficient ring variant. TFHE scheme (Chillotti et al., 2020) was proposed as an enhancement over FHEW, shifts from traditional ring structures to the torus, offering greater efficiency for non-polynomial operations. TFHE stands out as a most notable third-generation scheme, which built on RLWE problem further and achieved slower noise management and bootstrapping optimizations. However, limitations exist in TFHE's integer-only computations as it doesn't allow using Single Instruction/Multiple Data (SIMD) computations unlike other schemes BFV, BGV and CKKS. It has two versions: the leveled version and the bootstrapped version. The TFHE scheme is utilized in this dissertation and is explained further in the methodology part.

The latest fourth generation began with the Cheon-Kim-Kim-Song (CKKS) scheme (2017) that diverged from previous generations in its use of approximate arithmetics instead of exact arithmetics. Even though there are still efficiency issues preventing the large-scale FHE adoption, practical solutions exist for privacy-preserving machine learning after a decade of cryptographic research into FHE.

4. Methodology

A schematic representation of the stages involved in the rest of this study is shown in Figure 2. First stage is the data preprocessing where we applied feature engineering. After splitting the data, we used SMOTE oversampling technique only on training data. This will allow us to deal with highly imbalanced dataset. Then in classifier modeling phase, we evaluated classifier algorithms in terms of their accuracy and F1 score. After selecting the most suitable classifier for our dataset, XGBoost, the modeling phase is reapplied for its FHE equivalent, ConcreteXGBoost. Then hyper parameter tuning is applied for both models, followed by the evaluation of the results where we investigate if there is a performance drop in the Concrete model.

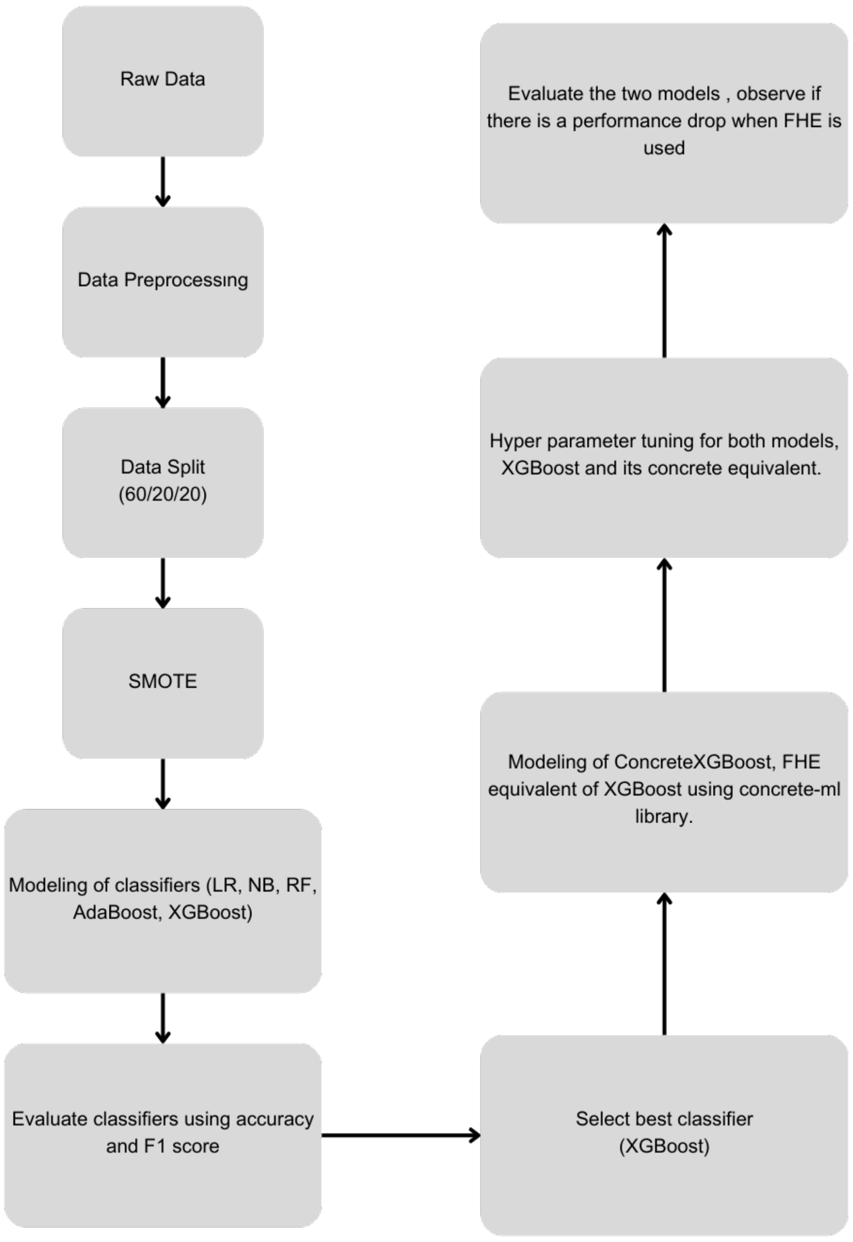

Raw Data
Data Preprocessing
Data Split (60/20/20)
SMOTE
Modeling of classifiers (LR, NB, RF, AdaBoost, XGBoost)
Evaluate classifiers using accuracy and F1 score
Select best classifier (XGBoost)
Modeling of ConcreteXGBoost, FHE equivalent of XGBoost using concrete-ml library.
Hyper parameter tuning for both models, XGBoost and its concrete equivalent.
Evaluate the two models , observe if there is a performance drop when FHE is used

Figure 2. Stages involved in this study.

4.1 SMOTE

We used Synthetic Minority Over-sampling Technique (SMOTE) to re-balance our data. SMOTE works randomly picking a point from the minority class and computing the k-nearest neighbors for it (McKinney, 2023). New synthetic points are added between the selected point and its minority neighbors. Unlike Random Oversampling (ROS), SMOTE does not create exact copies of observations, but creates new, synthetic samples that are quite similar to the existing observations in the minority class. It is therefore slightly more realistic and sophisticated than ROS, and it avoids duplicating observations. However, it only works well if the minority case features are similar. So, if launderings are spread through the data and not distinct, using nearest neighbors introduces noise into the data since the nearest neighbors are not fraud cases. In our data, laundering patterns includes many consequent transactions spread through multiple days, so we expect this technique to work well. Finally, SMOTE is only applied to training data as our goal is not to predict the synthetic samples, but to train a better model by using balanced data.

4.2 Extreme Gradient Boosting

XGBoost (eXtreme Gradient Boosting) has emerged as a powerful ensemble technique, particularly in our area of interest, financial transaction monitoring. In Anti Money Laundering (AML) programs, model's accuracy is crucial as false detections cannot be tolerated, and XGBoost offers a robust solution. The success of XGBoost lies in its ability to handle imbalanced data, making it well-suited for scenarios with missing values and high-class imbalance, characteristics often found in financial datasets. The algorithm's flexibility in tuning hyperparameters further enhances its efficiency.

A notable study by Ali et al. (2023) showcased the effectiveness of XGBoost in detecting Financial Statement Fraud (FSF). The study utilized the XGBoost algorithm to identify fraud in a dataset with class imbalance. Addressing the challenges of imbalanced data, XGBoost outperformed other algorithms, including Logistic Regression, Decision Tree, Support Vector Machine, AdaBoost, and Random

Forest. Through optimization, the XGBoost algorithm achieved a final accuracy of 96.05% in FSF detection. In another extensive study, Jones et al. (2015) conducted a comprehensive comparison of classifier' predictive performance, ranging from traditional ones like Logistic Regression and Linear Discriminant Analysis (LDA) to fully nonlinear counterparts such as Support Vector Machines and Neural Networks, and recent statistical learning techniques like Generalized Boosting, Random Forests and AdaBoost. In their results, recent classifiers showed superior performance over their counterparts. Furthermore, addressing financial distress, Zieba et al. (2016) examined the performance of various machine learning approaches designed to solve classification problems. Their comparison among statistical hypothesis testing, statistical modeling like generalized linear models, and recent machine learning methods like Neural Networks, Support Vector Machines, and Decision Trees revealed that Extreme Gradient Boosting delivered the best results. Finally, Carmona et al. (2019) leverage XGBoost to predict bank failure. And their results suggests that this method is well-suited to the banking industry.

XGBoost is an optimized distributed gradient boosting library created by Chen in 2014, a PhD student at the University of Washington. The algorithm utilizes the "weighted quantile sketch algorithm" to focus on misclassified data, continuously improving its accuracy. The weighted quantile sketch algorithm enables XGBoost to concentrate on areas where improvement is needed, contributing to its overall success. After achieving state-of-the-art results in several machine learning competitions, XGBoost gained popularity and used widely by data scientists in the field (Chen & Guestrin, 2016). Details about the algorithm are available on GitHub (https://github.com/dmlc/xgboost).

While single decision trees are simple, extreme gradient boosting (XGBoost) produces complex models comprising hundreds or thousands of trees (Al Wakil, 2018; Capatina et al., 2017). This poses interpretability challenges. However, XGBoost models can be summarized to provide insight into their predictions, and they tend to be more accurate than most conventional methods (Carmona et al., 2019). They have become widely used by data scientists, underpinning over half the winning solutions in machine learning challenges on Kaggle, a platform for data science competitions (He, 2016).

According to Chen and Benesty (2016), XGBoost represents an efficient and scalable implementation of the gradient boosting framework initially proposed by Friedman (2001, 2002). In comparison to the traditional gradient boosting methodology (GBM), XGBoost employs a more regularized model formalization to control overfitting, resulting in enhanced performance (Carmona et al., 2019). It is called regularized gradient boosting technique since it allows controlling variable weights, setting it apart from traditional GBM implementations. Regularization performs variable selection by applying penalty term on variable weights, pushing them to zero. This is crucial when dealing with high-dimensional problems.

Boosting minimizes overall errors by sequentially training models based on previous mistakes and then combining them (Chambers & Dinsmore, 2015). Tree ensemble methods like XGBoost sum the outputs of multiple trees since individual trees tend to perform poorly alone (Kuhn & Johnson, 2013). Friedman’s random sampling scheme, the stochastic gradient boosting, is introduced in 2002 into boosting algorithms with using a random subset of data to fit each new tree. This improved the final model by increasing performance and decreasing variance. XGBoost builds on this by fitting thousands of small decision trees in sequence, adjusting each one to the residuals of the previous model and updating the residuals thereafter. The final model becomes a linear combination of hundreds or thousands of trees, which improves the model (James et al., 2017). Elith et al (2008) resembles it to a regression model where each term is a tree. Moreover, boosting models can efficiently handle missing data, nonlinearity, and various input types such as categorical variables.

XGBoost relies on a set of key parameters for the learning phase. These parameters play a pivotal role in shaping its performance, and they should be determined carefully to optimize and improve the model. The key parameters according to Chen and Benesty (2016) explained below.

The optimal number of trees (iterations), often determined through cross-validation, required to minimize validation error. More trees allow fitting complex patterns but also increase overfitting risk. Another significant parameter is the "Maximum Depth," which controls size and complexity of individual trees. XGBoost grows trees to max depth then prunes redundant splits. This depth control serves the purpose of mitigating overfitting, allowing the model to learn relationships specific to the training data. The "Learning Rate", typically set between 0.01-0.3, Determines the contribution or "weight" of each new tree added. Smaller learning rates make boosting more conservative and prevent overfitting by slowing model adaptation. "Gamma," specifying the minimum loss reduction for further partitioning, introduces a conservative element to the algorithm. The "Column and Observation Sample" parameter, defining the subsample ratio during tree construction, acts as a preventive measure against overfitting while expediting computations. "Minimum Child Weight" is a critical determinant, indicating minimum subsample size needed to create a new tree leaf/node. Larger values prevent highly specific tree structures. Useful for imbalanced classes. The "Regularization" parameter, acting as a penalty term on weights, holds significance in reducing overfitting. As regularization increases, variable weights decrease, reducing variance at the cost of increased bias. As mentioned, this distinct feature sets XGBoost apart from other boosting models.

Hyperparameter tuning through cross-validation ensures the model's performance is fine-tuned for the specific task at hand. Their collective impact determines the model's complexity, resilience against overfitting, and ability to predict effectively. Careful consideration and calibration of these parameters contribute significantly to reaching the full potential of the XGBoost algorithm in predictive modeling scenarios.

XGBC’s robustness, adaptability, and ability to handle the intricacies of financial datasets position it as a reliable tool in the fight against financial crimes.

4.3 Privacy-Preserving Tree-Based Inference with TFHE

We utilize Privacy-Preserving Tree-Based Inference with TFHE by Frery et al. (2023) for testing XGBoost algorithm on encrypted data using its FHE equivalent. Their key contributions are applying quantization on individual data features by applying tensorized computation on integers and programmable boostrapping (PBS), using ciphertexts for multi-bit integers and employing automated crypto-system parameter tuning. It should be emphasized that their method only focuses on secure inference, so secure training is beyond the scope of this dissertation.

Fully Homomorphic Encryption over the Torus (TFHE) is a lattice-based cryptosystem that operates over a torus, and it allows for FHE, meaning computations can be performed directly on encrypted data without the need for decryption. It introduces two key innovations - noise flooding and bootstrapping over the torus instead of ring structure - to construct a leveled FHE scheme with very efficient bootstrapping. This allows to refresh the noise in ciphertexts to perform operations much more efficiently. Additionally, the torus foundation provides faster evaluation of non-polynomial functions compared to prior ring-based schemes like BFV and BGV. For privacy-preserving machine learning with integers, TFHE offers a practical fully homomorphic encryption solution after appropriately transforming models via quantization. With configurable precision levels, Private Decision-Tree Evaluation (PDTE) methods can enable accurate, secure classifiers.

Frery et al. (2023) utilizes approach from Bourse et al. (2018) to represent integers with TFHE, splitting the torus into slices to encode multi-bit numbers. TFHE scheme supports both leveled operations. It employs a Programmable Bootstrapping Scheme (PBS), which can perform a Table Look-Up (TLU) operation on input ciphertexts while reducing noise. TLU provides a mechanism to perform a range of computations and logical operations on encrypted data without compromising the security.

However, TFHE has some constraints that impact the design. All input and intermediate values within the model must be of integer type, so using it for machine learning requires quantization. This eliminates the risk of noise corrupting the message during accumulation. In addition, TFHE does not support control-flow operations (branching). This limitation is addressed using the PBS, a tool unique to TFHE, to compute arbitrary univariate non-linear functions.

In machine learning, quantization extends to not only converting the learned parameters of a model but also conducting model inference with quantized intermediary values. This overall quantization process produces an integer decision tree model compatible with TFHE. The tradeoff is reduced model accuracy due to quantization. However, prior works have shown 6-8 bits is often sufficient to match full precision models. The precision parameters balance accuracy vs compute time - higher bits improve accuracy but slow down homomorphic inference.

To enable privacy-preserving inferences on tree-based machine learning models such as XGBoost, Frery et al. (2023) uses following techniques in their model. First, they quantize the real-valued tree model into an integer version using uniform and asymmetric quantization on the input features, decision thresholds, and leaf node values. This converts the model into an FHE-compatible form. Then they implement integer decision tree inference homomorphically by evaluating all branches simultaneously using polynomial batching and table look ups. This replaces explicit tree traversal which is not possible in FHE. Finally, they optimize crypto parameters based on an analysis of the FHE operations graph, consisting mainly of accumulations and look ups. The parameters are selected to ensure sufficient space for the quantized inputs and intermediate values. A key aspect is converting the regular tree traversal into equivalent matrix computations using specialized encoding tensors. Quantization is also necessary to map the model to low-precision integers supported by FHE schemes. The output is a methodology to take any decision tree model, transform and evaluate it securely on encrypted data using FHE for privacy-preserving prediction, without compromising accuracy (Frery et al., 2023).

This method, Privacy-Preserving Tree-Based Inference with TFHE, plays crucial role in our dissertation by providing a secure and efficient framework for privacy-preserving machine learning. It enables us to test our learning algorithm XGBoost on encrypted data, and further compare the results with the original algorithm. Concrete-ML library is used to implement this method into our model. Concrete-ML is a PPML open-source set of tools built on top of Concrete by Zama. It aims the simplify the use of FHE with machine learning algorithms. Concrete ML models are based on scikit-learn, a machine learning python library. It uses bootstrap version of the TFHE scheme, and it has three types of models: Linear Models, Treebased Models, and Neural Networks. Further information is available on Zama’s website (https://docs.zama.ai/concrete-ml).

5. Data

Detecting money laundering has significant technical challenges. Existing algorithmic approaches struggle with false positives, flagging legal money transfers as illegal, and false negatives, missing identifying actual laundering. Part of the problem lies in criminals' highly sophisticated money laundering schemes across multiple accounts and banks.

The IBM Transactions for Anti Money Laundering (Altman,2023) synthetic dataset provides a robust and realistic simulation of financial transactions, both legitimate and fraudulent, across an entire virtual ecosystem. Using synthetic data avoids the significant privacy and proprietary restrictions faced when trying to obtain real transactional data from financial institutions.

The dataset models financial interactions between individuals, companies, and banks within the multi-agent virtual world. Individuals and companies engage in both legal forms like salary payments, loan repayments, and consumer purchases as well as criminal activities like smuggling, illegal gambling, extortion and more. The criminals then attempt to launder their illegal funds to hide the original source. The financial transactions are usually conducted through banks using checking accounts, credit cards and even bitcoin.

The IBM agent-based generator models the full lifecycle of money laundering- from placement of illegal funds into the system, to layering, and finally to integration by spending the illicit money. It should be noted that it can track laundered money across any number of intermediary transactions back to the original illicit source even when mixed with legitimate transactions. The generator adopts well-established money laundering patterns to create realistic money laundering transactions.

The dataset generator labels each transaction as either legitimate or laundering. This avoids the difficult task of correctly labeling real transaction data, providing unambiguous truth. Internal labeling of transactions as laundering or clean tends to be unreliable when relying only on limited transactions visible to a single institution. Having accurate labels is crucial for effectively training and evaluating machine learning models.

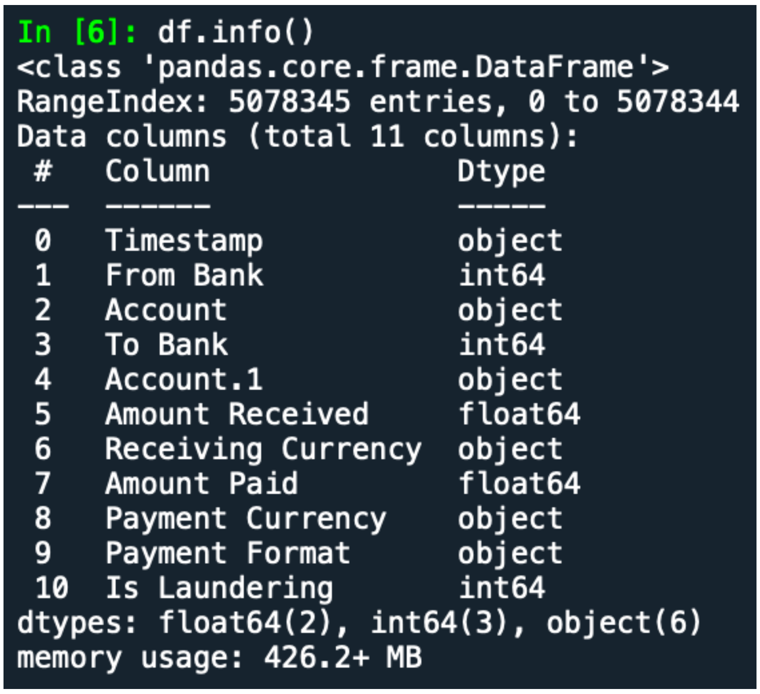
The dataset contains a complete financial ecosystem, which provides a significant advantage over real data in which a bank or regulator may only see a subset of transactions, missing money flows outside of the institution. The comprehensive view allows developing more holistic detection models.

Among 6 available dataset options, we choose smallest dataset HI-Small due to resource limitations. The dataset offers a reasonable sample size for proof-of-concept modeling. With 10 days of transaction data including over 5 million transactions from 515,000 accounts, it provides 3,600 laundering transactions with 1:981 ratio of laundering transactions.

In the previous papers (Egressy et al., 2023; Altman et al., 2023) working with the same dataset, the performance measurements were found to be even better than using real data. Money laundering patterns and further details can be found in the dataset paper (Altman et al., 2023).

5.1 Data Preparation

I've worked on the same dataset in the previous semester for Machine Learning Coursework #2 (Karabay, 2023), so in this dissertation I use similar methods for data analysis and preparation, but with a few key changes. First, I've imported required libraries including sklearn and concrete.ml.sklearn, which will enable using machine learning algorithms and their FHE equivalents. Then I uploaded our dataset "HI-Small_Trans.csv". After uploading, I observed that the dataset has 5,078,345 rows and 11 columns. The column heads are: 'Timestamp', 'From Bank', 'Account', 'To Bank', 'Account.1', 'Amount Received', 'Receiving Currency', 'Amount Paid', 'Payment Currency', 'Payment Format', 'Is Laundering']. Further information regarding the dataset is given below.



```
In [6]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5078345 entries, 0 to 5078344
Data columns (total 11 columns):
 #   Column              Dtype
---  ------              -----
 0   Timestamp           object
 1   From Bank           int64
 2   Account             object
 3   To Bank             int64
 4   Account.1           object
 5   Amount Received     float64
 6   Receiving Currency  object
 7   Amount Paid         float64
 8   Payment Currency    object
 9   Payment Format      object
 10  Is Laundering       int64
dtypes: float64(2), int64(3), object(6)
memory usage: 426.2+ MB
```

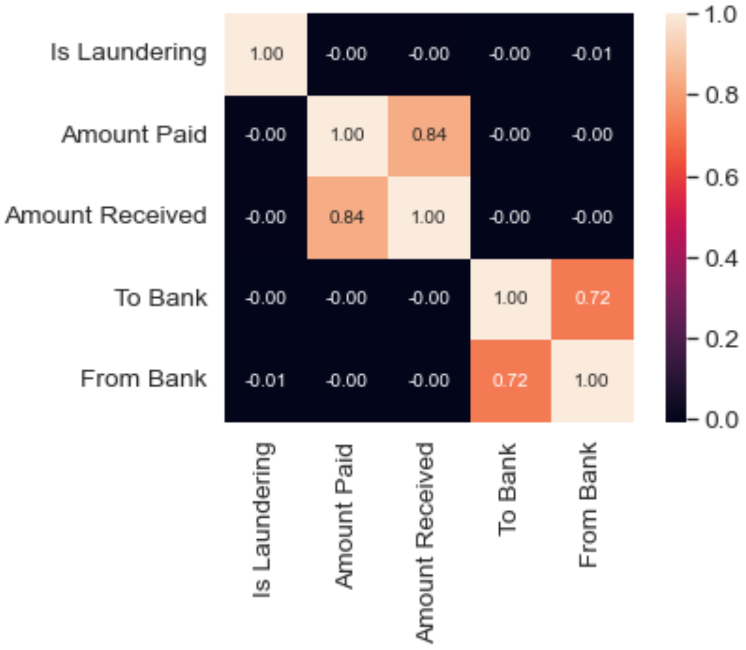
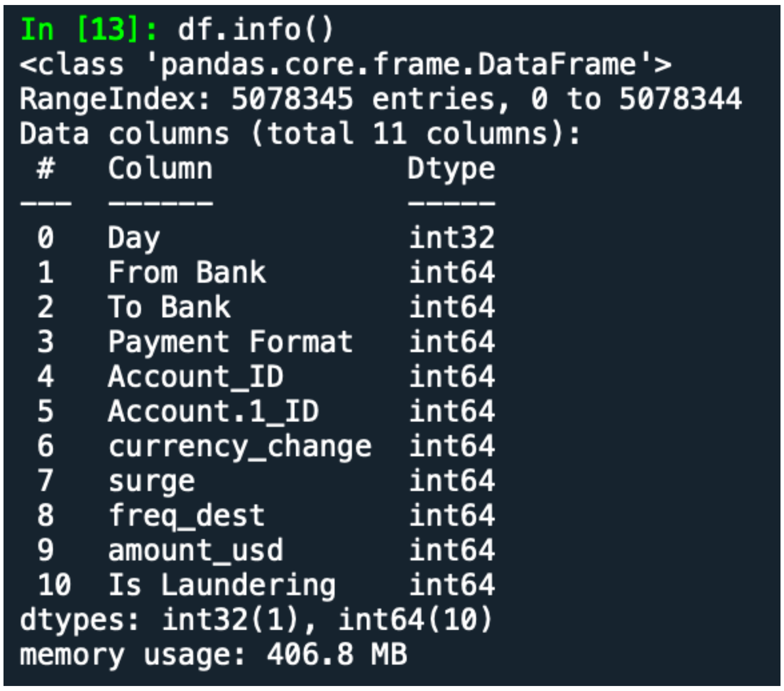


Fig. 3. Data information and correlation matrix

The data types found in columns are object, integer, and float. I need to fix these “object” columns and convert them to be able to run calculations on the data. I started with the “Timestamp” column, which has the following format “yyyy/mm/dd hh:mm”. The laundering patterns in the IBM data take multiple days to be completed (Altman, 2023). This is why I use only the date information, which also helped simplifying the column with converting it to integer. Moving to “Account” and “Account.1”, their original format includes both numerical and alphabetic values. As a key feature to detect patterns, we need to assign unique IDs to each account, and make sure that same accounts in the different columns get the same ID number. This is crucial considering the laundering cycles includes many transformations throughout days. Among several encoding options such as one-hot encoding, label-encoding or tokenization, I choose Feature Hashing as it avoided collisions between two columns and successfully assigned unique IDs. This is validated by counting unique values in each column and further checking if an account has same IDs in both columns.

Moving to other object type columns “Amount Received” and “Amount Paid”. These two columns have different values in 72,158 rows, which is caused by the currency changes. The correlation matrix in Fig.1 indicates there are high correlation between “Amount Paid” and “Amount Received”, and also between “To Bank” and “From Bank”. Considering the contents, these features are expected to be highly correlated. However, using these features together may cause overfitting and unnecessary burden regarding the dimensionality. I still kept “To Bank” and “From Bank” columns as they are one of the key indicators to identify laundering patterns. On the other hand, to avoid complexity and for the future functions, all currencies are fixed to USD. This allowed me to use only one column for transaction amounts. However, before implementing this change, I created a new binary column to indicate the rows with currency changes. As a result, while fixing the object type columns, we have also decreased related column number to half without losing any information. For the last object type column “Payment Format”, I used factorization to assign each payment format a unique integer. Finally, I added two new binary columns “surge” and “freq_dest”. Surge takes 1 if the transaction amount is greater than 10,000 USD, while freq_dest takes 1 if the receiver account receives money from more than 30 different accounts. As seen below, after these adjustments, there are now 11 columns all with integer values.



```
In [13]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5078345 entries, 0 to 5078344
Data columns (total 11 columns):
 #   Column           Dtype
---  ------           -----
 0   Day              int32
 1   From Bank        int64
 2   To Bank          int64
 3   Payment Format   int64
 4   Account_ID       int64
 5   Account.1_ID     int64
 6   currency_change  int64
 7   surge            int64
 8   freq_dest        int64
 9   amount_usd       int64
 10  Is Laundering    int64
dtypes: int32(1), int64(10)
memory usage: 406.8 MB
```

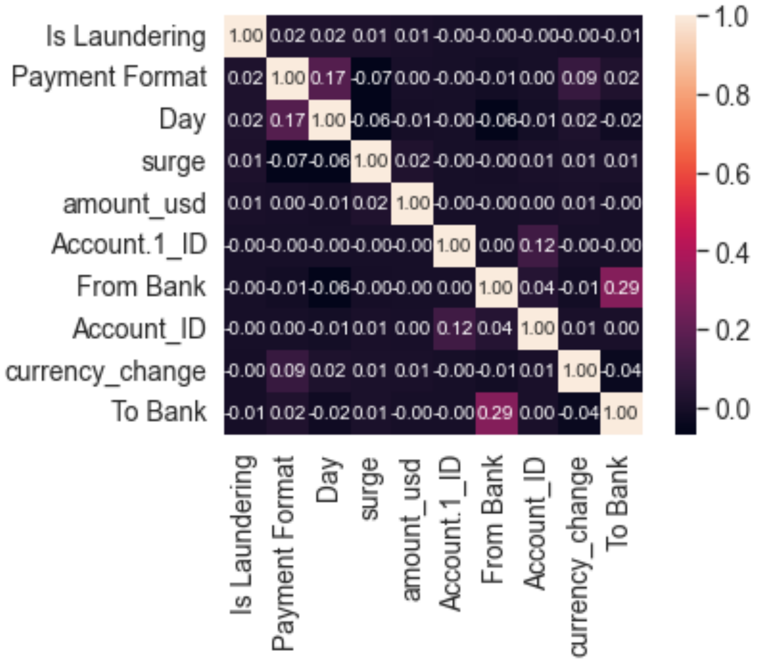
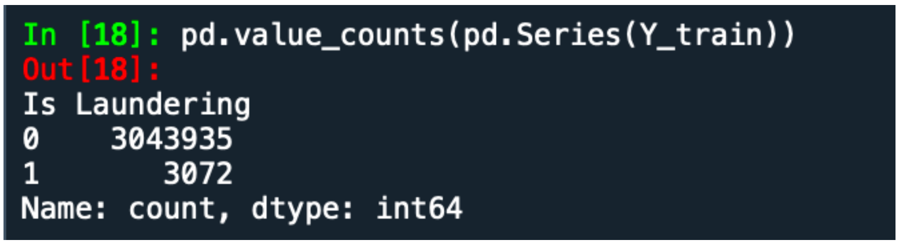


Fig 4. Correlation matrix after feature engineering

5.2 Data Split & Oversampling

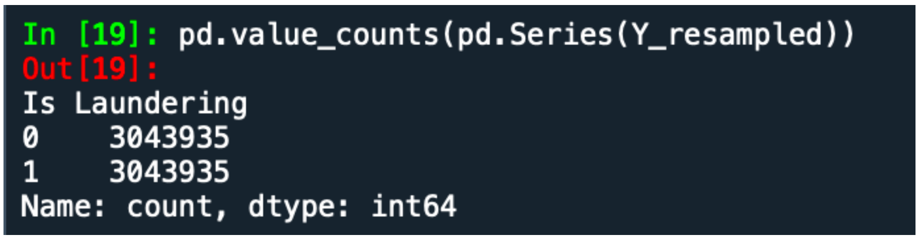
In their work, where they introduced the dataset generator, Altman et. al (2023) found the division of 60/20/20 to be effective. For dividing our dataset into training, validation, and testing datasets, we follow their suggestion and use the 60/20/20 proportion accordingly. However, there are some concerns needs to be highlighted regarding this division. Previously we mentioned that our dataset includes 10 days of transactions. Actually, there are transactions observed in the 11th and 12th days, which are included in the dataset for completing the money laundering schemes. So, these two days only have fraudulent transactions. This can manipulate the test results, but I ignored this in this dissertation as I'm not aiming to get the best model performance. But I want to show that current AML programs can be replaced by PPML models which enables complete information sharing amongst financial institutions.

AML datasets are often extremely imbalanced and unfortunately machine learning algorithms learns better when both classes in the dataset are more or less equally present. This class imbalance is one of the main challenges of fraud detection (McKinney, n.d.). Resampling methods can help model performance in cases of imbalanced datasets. This can be done either oversampling the minority class, or undersampling the majority class. Since some of the key information will be lost in undersampling, I used an oversampling method called SMOTE for my model. The selection of oversampling method is also crucial as we're training our model on duplicates. The details of SMOTE is given in the methodology part of this dissertation. Finally, I only used SMOTE on the training dataset, so the test results are not affected by these duplicates.



```
In [18]: pd.value_counts(pd.Series(Y_train))
Out[18]:
Is Laundering
0    3043935
1       3072
Name: count, dtype: int64
```

Before using the oversampling technique SMOTE, our training data shows 3,043,935 legal transactions and 3,071 fraudalent transactions.



```
In [19]: pd.value_counts(pd.Series(Y_resampled))
Out[19]:
Is Laundering
0    3043935
1    3043935
Name: count, dtype: int64
```

It can be seen that after SMOTE, our resampled training data has equal cases for both classes.

6. Implementation & Results

6.1 Implementation

We used Python 3.9.18 within Spyder for implementation. Computations are made on Mac M1. Several essential libraries are used in Python. First, we imported pandas library, which enables handling of datasets through its DataFrame structure. It facilitates efficient data manipulation and analysis, enabling us to implement feature engineering techniques on our data. Visualizations such as correlation matrix were created through seaborn and matplotlib.pyplot libraries, which provided clear understanding for the data. Hashlib library is used for creating a hash function to generate unique IDs in feature engineering. Time is imported for tracking computing time for FHE calculations, and numpy is imported for algebra calculations. The sklearn library played a pivotal role in model development, providing tools for main tasks such as data splitting (train-test-split), hyperparameter tuning (GridSearchCV), data standardization (StandardaScaler), and performance evaluation metrics like f1-score, accuracy, and confusion matrix. For addressing class imbalance, SMOTE with default 5 k_neighbors is implemented from the imblearn library. All models were implemented using the Scikit-Learn library and evaluated using five-fold cross-validation. Concrete-ml library provided FHE equivalent of XGBoost classifier.

All classifiers were modelled using scikit-learn library. LR was implemented using the LogisticRegression method with liblinear solver and ND was implemented using the GaussianNB. For the ensemble classifiers, RF was modelled using RandomForestClassifier with default 100 n_estimators. AdaBoost was modelled using AdaBoostClassifier with default DecisionTreeClassifier as the base estimator and default n_estimators and learning_rate. XGBoost was modelled using XGBClassifier with default parameters, then the hyperparameters were fine tuned in the subsequent phase. From Zama’s Concrete-ml library, ConcreteXGBClassifier is used for modelling and tuning FHE equivalent of XGBoost.

Overall, this comprehensive set of libraries enabled us to conduct data preprocessing, descriptive data analysis, model development and evaluation.

6.2 Performance Measures

To measure predictive performance of machine learning classifiers, we use common evaluation metrics: accuracy, precision, recall, F1 score and confusion matrix. However, since we have very imbalanced datasets, accuracy and other metrics can be deceiving. This is why we will consider minority class F1 score along with other metrics. This aligns well with what banks and regulators use in real-world scenarios (Egressy et al., 2023).

Confusion Matrix:

- True Positive (TP): Correctly predicted positive instances.
- True Negative (TN): Correctly predicted negative instances.
- False Positive (FP): Incorrectly predicted positive instances.
- False Negative (FN): Incorrectly predicted negative instances.

$$\text{Accuracy} = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} = \frac{TP + TN}{TP + TN + FP + FN} \qquad (1)$$

$$\text{Precision} = \frac{True\ Positive}{Total\ Predictive\ Positive} = \frac{TP}{TP + FP} \qquad (2)$$

$$\text{Recall} = \frac{True\ Positive}{Total\ Actual\ Positive} = \frac{TP}{TP + FN} \qquad (3)$$

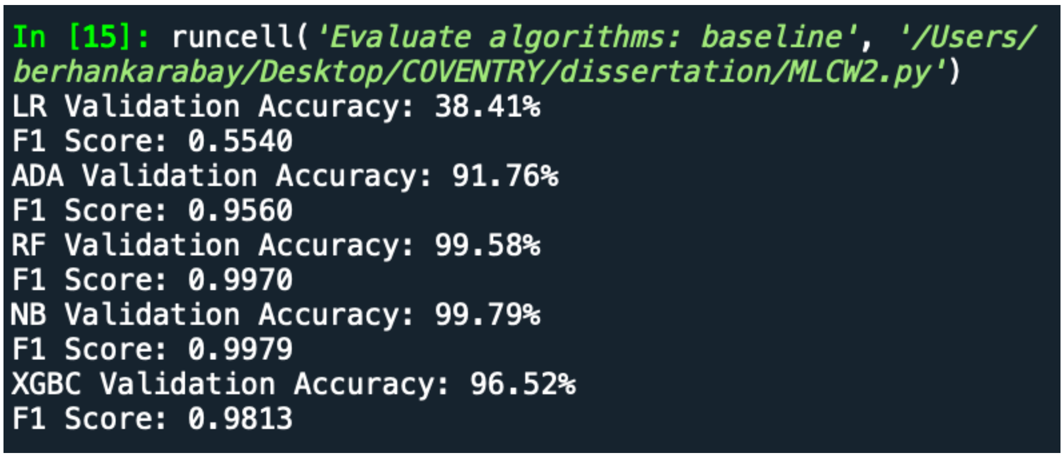
$$\text{F1 Score} = 2 \times \frac{Precision\ \ x\ Recall}{Precision + Recall} \qquad (4)$$

Accuracy gives the overall correctness of the classifier. Precision is the ratio of correctly predicted positive observations to the total predicted positives. Recall is the ratio of correctly predicted positive observations to all observations in the actual positive class. F1 score is the harmonic mean of precision and recall, balancing false positives and false negatives.

6.3 Model Analysis

Money Laundering detection is a binary classification problem. In order to the find most efficient solution, we'll compare several classification machine learning algorithms including boosting ensemble techniques. These algorithms use different techniques for their predictions and their results varied based on the problem on hand, as well as on the integrity and complexity of the training dataset. First selection is made based on the technical features and experimental results from previous' works. We included Logistic Regression (LR) as it's one of the most used ML algorithms in binary classification (McKinney, n.d.). And the Naïve Bayes (NB) as it was the best performing model in another Fraud Detection study in Kaggle. (Roshan, 2021). However, a comprehensive comparison of classifiers by Jones et al. (2015) showed that Boosting algorithms have superior predictive performance over other classifiers such as LR, LDA and SVM. Further studies (Ali et al., 2023; Zreba et al. 2016) also showed effectiveness of XGBoost algorithm, especially when there is a high class-imbalances. Based on these and further information, we decided to evaluate XGBoost, AdaBoost and RF along with LR, NB classifiers. Details of these models are explained previously in this dissertation.

The prediction performance of all five models on IBM dataset are listed below. These results are achieved after applying SMOTE on the training data.



```
In [15]: runcell('Evaluate algorithms: baseline', '/Users/
berhankarabay/Desktop/COVENTRY/dissertation/MLCW2.py')
LR Validation Accuracy: 38.41%
F1 Score: 0.5540
ADA Validation Accuracy: 91.76%
F1 Score: 0.9560
RF Validation Accuracy: 99.58%
F1 Score: 0.9970
NB Validation Accuracy: 99.79%
F1 Score: 0.9979
XGBC Validation Accuracy: 96.52%
F1 Score: 0.9813
```

The accuracy and F1 score results for five models are listed above. First of all, poor performance of LR classifier can be observed from these results. Even this can be significantly improved using standardization, we will not focus on this algorithm since it is generally considered less prone to overfitting compared to other models especially when dealing with high-dimensional datasets. Moving forward, highest accuracy and F1 score is achieved by RF and NB. However, they are very close to 100% and this may be a sign for overfitting. We already mentioned that our dataset has high class-imbalance, which is one of the main challenges of dealing with financial datasets. This is why we disregard the almost perfect performance of RF and NB algorithms. Amongst the two remaining boosting algorithms, XGBoost and ADABoost, the highest performance is observed with XGBC. The 96.52% accuracy and 98.13% weighted F1-score are promising for our model. AdaBoost displayed accuracy of 91.76% and 95.6% F1 score, close to XGBoost. However, XGBoost has more room for improvement as it enables more advanced parameter tuning, which can yield to state-of-art results if handled carefully.

It is evident from these results that XGBoost delivers promising performance on our SMOTE-applied IBM dataset. It scored above 95% both for accuracy and weighted f1-score, which creates some skepticism as it may also be a sign for overfitting. There is no significant performance gap between models in our experimental results. However, technical analysis and previous works dictates XGBoost as the perfect candidate for our model, considering its advanced parameter tuning and effectiveness in handling imbalanced datasets. Finally, XGBoost can be useful to capture the complex laundering patterns found in our dataset.

6.4 XGBoost Parameter Tuning

After model evaluations, we selected XGBoost algorithm for fraud detection on the IBM financial dataset. We’ve already mentioned that parameter tuning is essential for XGBoost algorithm. With optimized parameters, it can achieve state-of-art accuracy while allowing efficient computations. For hyperparameter tuning, we use GridSearchCV cross-validation technique in scikit-learn since it considers all possible combinations of hyperparameter values provided in the grid. While it might be computationally expensive, it can provide a more detailed exploration of the hyperparameters which is useful in transaction monitoring, where accuracy has outmost importance.

GridSearchCV is used with roc-auc (receiver operating characteristic - area under the curve) scoring. Furthermore, it utilizes ShuffleSplit with 5 splits and 0.3 test size. ShuffleSplit is a cross-validation iterator that randomly shuffles and splits the data into train/test sets for multiple iterations. Parameter grid is defined as below. Rather than working on all parameters to search combinations, we focused only on the selected important parameters, and their ranges are chosen small to keep the FHE execution time per inference relatively low.

```
param_grid = {
  "max_depth": list (range(1, 10)),
  "n_estimators": list (range(1, 5)),
  "learning_rate": [0.01, 0.1, 1],
}
```

Hyperparameter tuning is performed where GridSearchCV tries all combinations in the grid. The best performing parameters in our model are listed below:

```
Best Parameters: {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 4}
```

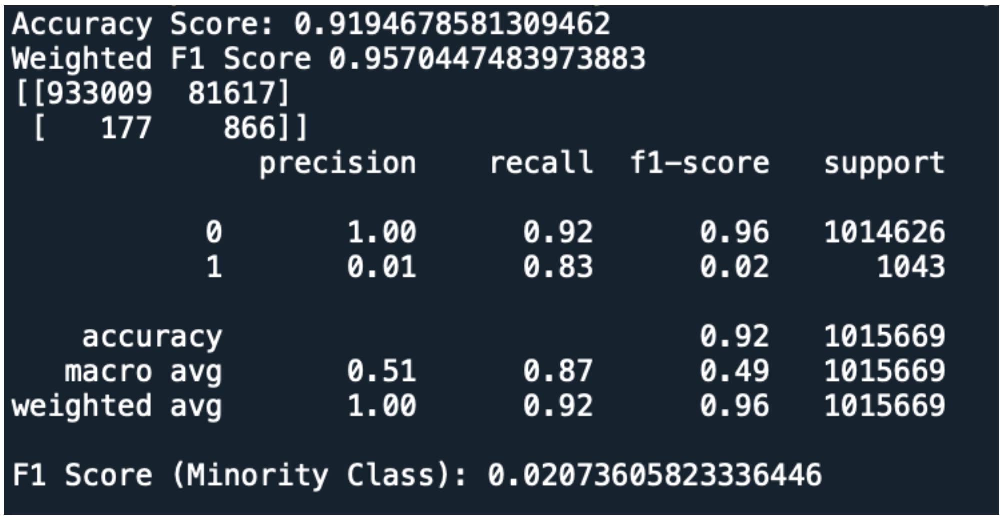
6.5 ConcreteXGBoost Tuning

A very similar method is used for tuning ConcreteXGBoost, the FHE equivalent of XGBoostClassifier. ShuffleSplit is used and defined with same variables. GridSearchCV is used with same scoring, but we added “n_bits” to parameter grid since the concrete ML model requires additional parameter for quantization. So, we need to specify the number of bits over which inputs, outputs and weights will be quantized. This value can influence the predictive performance of the model as well as its inference running time, and therefore can lead the grid-search cross-validation to find a different set of parameters. In previous case examples provided by Zama team, setting this value to 2 bits outputs an acceptable accuracy score while running faster. Even though Frery et al. (2023) observed significant performance improvement with higher bitwidths, we set n_bits to 3 in our dissertation to control inference time. The rest of the parameter grid is same with the XGBoost tuning. Best performing parameters in concrete model are listed below:

```
Best hyper-parameters found in 1756.98s : {'learning_rate': 0.1,
'max_depth': 4, 'n_bits': 2, 'n_estimators': 4}
```

6.6 Results

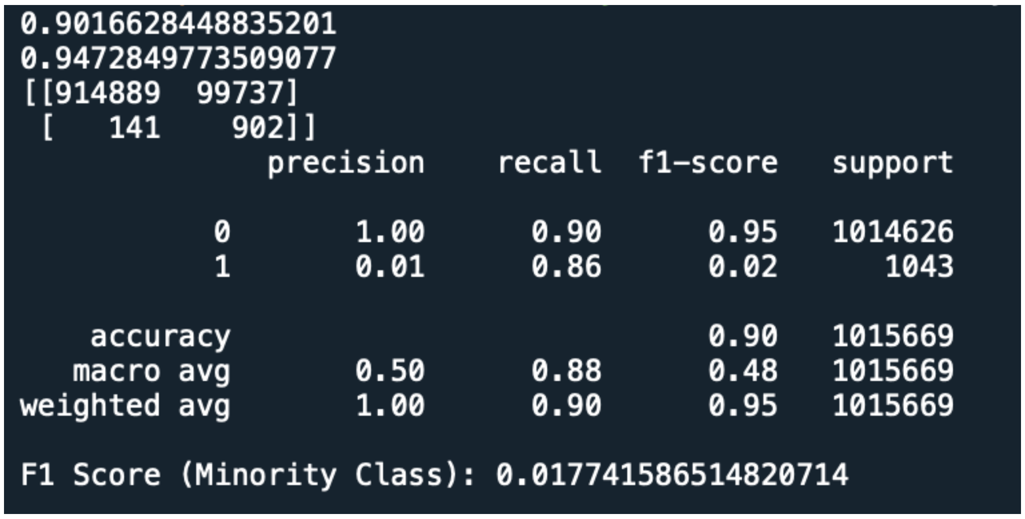
In this section we describe the experiments we performed to compare XGBoost and ConcreteXGBoost. We use confusion matrix, accuracy and f1 score to evaluate performances of both models. We first evaluate our XGBoost model, with re-defined parameters, on the test set. The results below are gathered using test set, where the model comparison results were obtained using validation set. So, the clear impact of tuning is unclear. It should be noted that we were not able to implement careful parameter tuning in order to control inference time. Even though weighted accuracy and F1 score is above 90%, we observe significantly low F1 score with 2% for minority class. This is mostly due to its formulation, where true positives which are in very low number are compared to false positives, which has 100 times more cases due to class imbalances. The prediction cases can be observed in Fig. 5. This is why, to improve minority F1 score, we need to decrease the number of false positives (or we can use different weighting in formula like in previous works). It should be noted that false positives are more tolerated in transaction monitoring, compared to false negatives that should be avoided.



```
Accuracy Score: 0.9194678581309462
Weighted F1 Score 0.9570447483973883
[[933009  81617]
 [   177    866]]
              precision    recall  f1-score   support

           0       1.00      0.92      0.96   1014626
           1       0.01      0.83      0.02      1043

    accuracy                           0.92   1015669
   macro avg       0.51      0.87      0.49   1015669
weighted avg       1.00      0.92      0.96   1015669

F1 Score (Minority Class): 0.02073605823336446
```

Fig 5. Confusion matrix and performance metrics for XGBoost

Then we evaluate the predictions of concrete model after quantization, using the same test set and defined hyper-parameters. The aim is to study the impact of quantization on model performance. However, as seen previously, the grid-search cross-validation was done separately between the XGBoost and ConcreteXGBoost. For this reason, the two models do not share the same set of hyper-parameters, making their decision boundaries different. In addition, we were not able to test optimal number of bits for quantization. This is why the performance drop after quantization is expected. Below we can observe 90% accuracy and 94% weighted f1-score, and 1.7% minority f1-score. There is a 15% performance drop in minority F1 score, but considering the range of parameters we defined, it is expected. This can be further improved with careful tuning, especially with larger bitwidhts.



```
0.9016628448835201
0.9472849773509077
[[914889  99737]
 [   141    902]]
              precision    recall  f1-score   support

           0       1.00      0.90      0.95   1014626
           1       0.01      0.86      0.02      1043

    accuracy                           0.90   1015669
   macro avg       0.50      0.88      0.48   1015669
weighted avg       1.00      0.90      0.95   1015669

F1 Score (Minority Class): 0.017741586514820714
```

Fig 6. Confusion matrix and performance metrics for ConcreteXGBoost

Finally, we compute the predictions in FHE using ConcreteXGBoost. FHE computations shows 100% similarity as expected. This means that the model executed with the encrypted data results in the same predictions as the Concrete ML one, which is executed in clear and only considers quantization. This proves that XGBoost can be used on encrypted data without compromising from model's accuracy, disregarding the quantization phase which is not successfully implemented in this research. However, inference time was close to 1 week on Mac M1, which points a scalability issue for real life applications.

```
Prediction similarity between both Concrete ML models (quantiz
ed clear and FHE): 100.00%
Accuracy of prediction in FHE on the test set 90.16%
```

7. Conclusion

In this dissertation, we proposed a new transaction monitoring method by combining and implementing recent efficient PPML frameworks on a recent dataset. With FHE, PEML can work on encrypted data, which solves the privacy and security issues for the financial institutions. The aim was to improve AML programs for more accurate money laundering detections. This can be achieved by enabling data sharing among financial institutions without revealing sensitive information. We presented Privacy-Preserving Tree-Based Inference with TFHE framework (Frery et al., 2023). It provides FHE equivalents of Tree-Based classification algorithms, including XGBoost, so that they can work on encrypted data in a secure way. We were able to test advanced classification algorithm XGBoost, well-suited for such tasks, on an IBM-generated synthetic dataset that provides a robust and realistic simulation of financial transactions (Altman, 2023). This dataset includes complete transactions in an entire financial system, which was a great opportunity to demonstrate how current AML programs and transaction monitoring could be improved with data sharing.

Like in real world, IBM data has very high-class imbalance for launderings. A sophisticated oversampling method SMOTE is used for balancing the training set by generating synthetic laundering classes. We conducted analysis and comparison of five ML algorithms; LR, NB, RF, ADA, XGBC. While all of them, except LR, yielded high accuracy rates, our evaluations resulted in selection of XGBoost as the most efficient model for detecting complex laundering patterns. Then we introduced ConcreteXGBoost, which is the FHE equivalent of XGBoost. The experimental results demonstrated the feasibility of using FHE for machine learning algorithms in transaction monitoring. We achieved exact predictions after implementing FHE. However, we've observed performance drop after quantization with ConcreteXGBoost. This is mainly caused by setting quantization parameter 'n_bits' low due to the time and resource constraints. We've set bitwidths to 2 in our model, while Frery et al. (2023) observed significant performance increase between 2 and 5, metrics almost reaching to 90% from %60. This should be addressed further in future works.

This study does not aim to achieve best performing model due the time and resource constraints, but provided a simple demonstration of the proposed framework. However, to improve the performance of Concrete model, we can start with increasing bitwidhts in quantization. In addition, hyper-parameter can be very important for XGBoost's performance. We've only changed 3 parameters with using low ranges. More advanced and careful tuning can significantly improve this model. On the other hand, one of the main challenges for such tasks is dealing with class imbalances. Even though we used SMOTE, there can be further improvements to deal with this. For example, Altman et al. (2023) address this by incorporating the measure of weighting the predictions of the minority class higher in the loss function. Working on the same dataset, they achieved 63% minority f1 score by combining XGBoost with Graph Feature Preprocessor (GFP) (IBM, 2022), which creates additional features for the datasets. As their results are much more promising than our model, the proposed TFHE scheme should be implemented with their model in the future works. We further suggest using GNN models from Egressy et al. (2023) with this TFHE scheme and compare their results with the combination of XGBoost and GFP.

Finally, we've kept the scope of this work under secure training. To achieve a goal with data sharing, we need to implement secure training as well. This can be done using techniques Differential Privacy (Dwork, 2006) or Federated Learning (Bonawitz et al., 2019).

8. References

Ali A.A., Khedr A.M., El-Bannany M., & Kanakkayil S. (2023). A Powerful Predicting Model for Financial Statement Fraud Based on Optimized XGBoost Ensemble Learning Technique. Applied Sciences 13, no. 4: 2272. https://doi.org/10.3390/app13042272

Altman E., Blanuša J., von Niederhäusern L., Egressy B., Anghel A., & Atasu K. (2023). Realistic Synthetic Financial Transactions for Anti-Money Laundering Models. Retrieved from https://arxiv.org/pdf/2306.16424.pdf

Altman, E. (2023). IBM Transactions for Anti Money Laundering (AML). Retrieved from https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml?select=HI-Small_Trans.csv

Bonawitz K., Eichner H., Grieskamp W., Huba D., Ingerman A., Ivanov V., Kiddon C., Koneˇcn`y J., Mazzocchi S., McMahan B., et al. (2019). Towards federated learning at scale: System design. Proceedings of machine learning and systems, 1:374–388, 2019.

Bourse F., Minelli M., Minihold M., & Paillier P. (2018). Fast homomorphic evaluation of deep discretized neural networks. In Advances in Cryptology – CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III, page 483–512, Berlin, Heidelberg, 2018. Springer-Verlag

Brakerski, Z. (2014). Fully homomorphic encryption without modulus switching from classical gapsvp. SIAM Journal on Computing, 43(5):1541–1563, 2014.

Brakerski Z., Vaikuntanathan V., & Gentry C. (2011). Fully homomorphic encryption without bootstrapping. IACR Cryptology ePrint Archive, 2011(277):1–16, 2011.

Cheon J.H., Kim A., Kim M., & Song Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In International conference on the theory and application of cryptology and information security, pages 409–437. Springer, 2017.

Chillotti I., Gama N., Georgieva M., & Izabachène M. (2020). Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology, 33(1):34–91, 2020.

Cramer J.S. (2002). The origins of logistic regression. SSRN 2002, 119, 1–16.

Dwork C. (2006). Differential privacy. In Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33, pages 1–12. Springer, 2006.

Egressy B., von Niederhäusern L., Blanuša J., Altman E., Wattenhofer R., & Atasu K. (2023). Provably Powerful Graph Neural Networks for Directed Multigraphs. Retrieved from https://arxiv.org/pdf/2306.11586.pdf

FCA. (2021). NatWest fined £264.8 million for anti-money laundering failures. Retrieved from https://www.fca.org.uk/news/press-releases/natwest-fined-264.8million-anti-money-laundering-failures/printable/print

Frery J., Stoian A., Bredehoft R., Montero L., Kherfallah C., Chevallier-Mames B., & Meyre A. (2023). Privacy-Preserving Tree-Based Inference with TFHE. Retrieved from https://arxiv.org/pdf/2303.01254.pdf

Freund Y., Schapire R.E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. 1997, 55, 119–139. [CrossRef]

Gentry, C. (2009). A fully homomorphic encryption scheme. Stanford university, 2009.

Gentry C., Sahai A., & Waters B. (2013). Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I, pages 75–92. Springer, 2013.

Ho T.K. (1995). Random decision forests. In Proceedings of the 3rd International Conference on Document Analysis and Recognition, Lausanne, Switzerland, 5–10 September 1995; IEEE: New York, NY, USA, 1995; Volume 1, pp. 278–282.

Hunt T., Song C., Shokri R., Shmatikov V., & Withcel E. (2018). Chiron: Privacy-preserving Machine Learning as a Service. Retreived from https://arxiv.org/pdf/1803.05961.pdf

IBM (2023). IBM Transactions for Anti Money Laundering (AML). Retreived from https://ibm.ent.box.com/v/AML-Anti-Money-Laundering-Data

IBM Research. (2022). Graph Feature Preprocessor PyPI Documentation. https://snapml. readthedocs.io/en/latest/graph_preprocessor.html Accessed: 2023-01-10. 13

IBM Research. (2022). Snap ML PyPI package. https://pypi.org/project/snapml/ Accessed: 2023-01-10.

Jaume G., Nguyen A.-p., Martínez, M. R., Thiran, J.-P., and Gabrani M. (2019). edGNN: a Simple and Powerful GNN for Directed Labeled Graphs. arXiv preprint arXiv:1904.08745.

LexisNexis. (2023). True cost of compliance. Retrieved from https://risk.lexisnexis.co.uk/insights-resources/white-paper/true-costs-of-compliance?trmid=BSUKFC23.FCC.Orch.PHGO-860910&utm_source=google&utm_medium=paid-search&utm_campaign=BSUKFC23.FCC.Orch&gclid=Cj0KCQjwrMKmBhCJARIsAHuEAPThbN68IFmVzlGOOQ_I4oGQUOhzc6d4OsdRVdlwDgycD9Wo2NbXOQUaAp9zEALw_wcB

Lyubashevsky V., Peikert C., & Regev O. (2010). On ideal lattices and learning with errors over rings. SIAM Journal on Computing, 39(6):2181–2213, 2010.

McKinney, T. (n.d.). Fraud Detection in Python. Retrieved 18.08.2023 from https://trenton3983.github.io/files/projects/2019-07-19_fraud_detection_python/2019-07-19_fraud_detection_python.html

McKinsey & Company. (2022). The fight against money laundering: Machine learning is a game changer. Retrieved from https://www.mckinsey.com/capabilities/risk-and-resilience/our-insights/the-fight-against-money-laundering-machine-learning-is-a-game-changer

Pappalardo, A.: Xilinx/brevitas (2021). https://doi.org/10.5281/zenodo.3333552

Rach, S. (2021). FCA fines increase nearly four-fold. Retrieved from https://www.ftadviser.com/fca/2021/12/20/fca-fines-increase-nearly-four-fold/

Ramalho, B. (2023). Secure machine learning via homomorphic encryption. Retrieved from https://repositorio-aberto.up.pt/bitstream/10216/151947/2/636672.pdf

Regev, O. (2005). The learning with errors problem. Journal of Cryptology, 18(4):267–302, 2005.

Roshan, B. (2021). Transaction Fraud Detection. Retrieved from https://www.kaggle.com/code/benroshan/transaction-fraud-detection/notebook

Sato R., Yamada, M., & Kashima H., (2019). Approximation ratios of graph neural networks for combinatorial problems. Advances in Neural Information Processing Systems, 32.

Sreedharan, M., Khedr A.M., El Bannany, M. (2020) A comparative analysis of machine learning classifiers and ensemble techniques in   financial distress prediction. In Proceedings of the 2020 17th International Multi-Conference on Systems, Signals & Devices (SSD), Monastir, Tunisia, 20–23 July 2020; IEEE: New York, NY, USA, 2020; pp. 653–657.

Stoian, A., Frery, J., Bredehoft, R., Montero, L., Kherfallah, C., Chevallier-Mames, B. (2023). Deep Neural Networks for Encrypted Inference with TFHE. In: Dolev, S., Gudes, E., Paillier, P. (eds) Cyber

Security, Cryptology, and Machine Learning. CSCML 2023. Lecture Notes in Computer Science, vol 13914. Springer, Cham. https://doi.org/10.1007/978-3-031-34671-2_34

Sun X., Zhang P., Liu J. K., Yu J., & Xie W. (2020). Private Machine Learning Classification Based on Fully Homomorphic Encryption. in IEEE Transactions on Emerging Topics in Computing, vol. 8, no. 2, pp. 352-364, 1 April-June 2020, doi: 10.1109/TETC.2018.2794611.

US DOJ (2022). Danske Bank Pleads Guilty to Fraud on U.S. Banks in Multi-Billion Dollar Scheme to Access the U.S. Financial System. Retrieved from https://www.justice.gov/opa/pr/danske-bank-pleads-guilty-fraud-us-banks-multi-billion-dollar-scheme-access-us-financial

UNODC (2022). Money Laundering. Retrieved from https://www.unodc.org/unodc/en/money-laundering/ overview.html

You J., Gomes-Selman J. M., Ying, R., & Leskovec, J. (2021). Identity-aware graph neural networks. In Proceedings of the AAAI conference on artificial intelligence, volume 35, 10737–10745.

Zama. (n.d.). What is Concrete ML?. Retrieved 11.08.2023 from https://docs.zama.ai/concrete-ml/

9. Appendix

9.1 Python Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 10 01:47:35 2023

@author: berhankarabay
"""

#%% Load libraries

import requests
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import time

from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE



from sklearn.model_selection import GridSearchCV, ShuffleSplit
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier
from concrete.ml.sklearn import XGBClassifier as ConcreteXGBClassifier



pd.set_option('display.max_columns', 700)
pd.set_option('display.max_rows', 400)
pd.set_option('display.min_rows', 10)
pd.set_option('display.expand_frame_repr', True)
pd.options.display.float_format = "{:,.2f}".format


#%% load dataset

filename = 'HI-Small_Trans.csv'
df = pd.read_csv(filename)
print(df.columns)

#%% 3. Data Description & Visualizations

df.info()
df.head()
print(df.describe())

#Counting the occurrences of fraud and no fraud

occ = df['Is Laundering'].value_counts()
occ
# Print the ratio of fraud cases
ratio_cases = occ/len(df.index)
print(f'Ratio of fraudulent cases: {ratio_cases[1]}\nRatio of non-
fraudulent cases: {ratio_cases[0]}')



#%% Feature Engineering | Timestamp

#Due to the patterns given in our data, we choose to use only dates.

# Convert the 'Time Stamp' column to datetime
df['Timestamp'] = pd.to_datetime(df['Timestamp'], format='%Y/%m/%d
%H:%M')

# Extract the day and create a new column 'Day'
df['Day'] = df['Timestamp'].dt.day
df = df.drop(['Timestamp'], axis=1)
# Reorder the columns

print(df)

#%%Removing alphabetical values from Account IDs
```

```
import hashlib

unique_count = df['Account'].nunique()
unique_count1 = df['Account.1'].nunique()


print("Number of unique Account values before:", unique_count)
print("Number of unique Account.1 values before:", unique_count1)


# Create a hash function to generate unique integer IDs

def generate_unique_id(account_number):
    hash_str = hashlib.sha256(account_number.encode()).hexdigest()
    return int(hash_str, 16) % 10**8

# Create a dictionary to store mappings
account_id_mapping = {}

# Function to get or generate unique ID for an account number
def get_or_generate_id(account_number):
    if account_number not in account_id_mapping:
        account_id_mapping[account_number] =
generate_unique_id(account_number)
    return account_id_mapping[account_number]

# Apply the mappings to your columns
df['Account_ID'] = [get_or_generate_id(account) for account in
df['Account']]
df['Account.1_ID'] = [get_or_generate_id(account) for account in
df['Account.1']]

unique_count = df['Account_ID'].nunique()
unique_count1 = df['Account.1_ID'].nunique()


print("Number of unique Account_ID values after:", unique_count)
print("Number of unique Account.1_ID values after:", unique_count1)

df = df.drop(['Account', 'Account.1'], axis=1)

# Print the result
print(df)


#%% Feature Engineering



#Checking if the amount paid and received is equal. Looks like its not.

different_attributes = df[df['Amount Received'] != df['Amount Paid']]
print(different_attributes)


# To simplfy and reduce dimensions, we'll create a new column
"currencychange"(1 for yes, 0 for no) and remove currency columns"
```

```
# We'll convert all "Receiving Currency" to USD. To apply exchange
rates we'll use Open Exchange Rates API.

#First mapping of non-standard currency names to ISO currency codes
currency_mapping = {
    'US Dollar': 'USD',
    'Australian Dollar': 'AUD',
    'Bitcoin': 'BTC',
    'Brazil Real': 'BRL',
    'Canadian Dollar': 'CAD',
    'Euro': 'EUR',
    'Mexican Peso': 'MXN',
    'Ruble': 'RUB',
    'Rupee': 'INR',
    'Swiss Franc': 'CHF',
    'Shekel': 'ILS',
    'Saudi Riyal': 'SAR',
    'UK Pound': 'GBP',
    'Yen': 'JPY',
    'Yuan': 'CNY',

}

# Update 'currency' column using the mapping
df['Receiving Currency'] = df['Receiving
Currency'].map(currency_mapping)
df['Payment Currency'] = df['Payment Currency'].map(currency_mapping)



#%% Feature Engineering | Functions


# Define a function to check if the currencies are the same
def check_currencies(data):
    if data['Receiving Currency'] != data['Payment Currency']:
        return 1
    else:
        return 0
'''
def same_bank_indicator(data):
    if data['From Bank'] == data['To Bank']:
        return 1
    else:
        return 0
 '''
#Surge indicator
def surge_indicator(data):
    '''Creates a new column which has 1 if the transaction amount is
greater than the threshold
    else it will be 0'''
    data['surge']=[1 if n>10000 else 0 for n in data['Amount
Received']]

#Frequency indicator
def frequency_receiver(data):
    '''Creates a new column which has 1 if the receiver receives money
from many individuals
```

```
    else it will be 0'''

data['freq_Dest']=data['Account.1_ID'].map(data['Account.1_ID'].value_c
ounts())
    data['freq_dest']=[1 if n>30 else 0 for n in data['freq_Dest']]
    data.drop(['freq_Dest'],axis=1,inplace = True)



#%% Feature Engineering | Applying Functions

# Applying check_currencies function
df['currency_change'] = df.apply(check_currencies, axis=1)
df['currency_change'].value_counts()
'''
# Applying same_bank function
df['same_bank'] = df.apply(same_bank_indicator, axis=1)
df['same_bank'].value_counts()
'''
# Applying surge_indicator function
surge_indicator(df)
df['surge'].value_counts()

# Applying frequency_receiver function
frequency_receiver(df)
df['freq_dest'].value_counts()

#%% Feature Engineering | Fixing currencies to USD


# Get exchange rate data (example: using Open Exchange Rates API)
api_key = 'b15a7a606acf4ccc8c98330f30d5268b'
base_currency = 'USD'
exchange_url =
f'https://openexchangerates.org/api/latest.json?app_id={api_key}&base={
base_currency}'
response = requests.get(exchange_url)
exchange_rates = response.json()['rates']

# Merge exchange rates with the original DataFrame
df = df.merge(pd.DataFrame(exchange_rates.items(), columns=['Receiving
Currency', 'exchange_rate']), on='Receiving Currency')

# Convert to USD
df['amount_usd'] = (df['Amount Received'] /
df['exchange_rate']).astype(int)

print(df)




#%% Feature Engineering | Dropping unnecessary columns


columns_to_drop = ['Amount Paid','Amount Received','Receiving
Currency','Payment Currency','exchange_rate']
df = df.drop(columns_to_drop, axis=1)
```

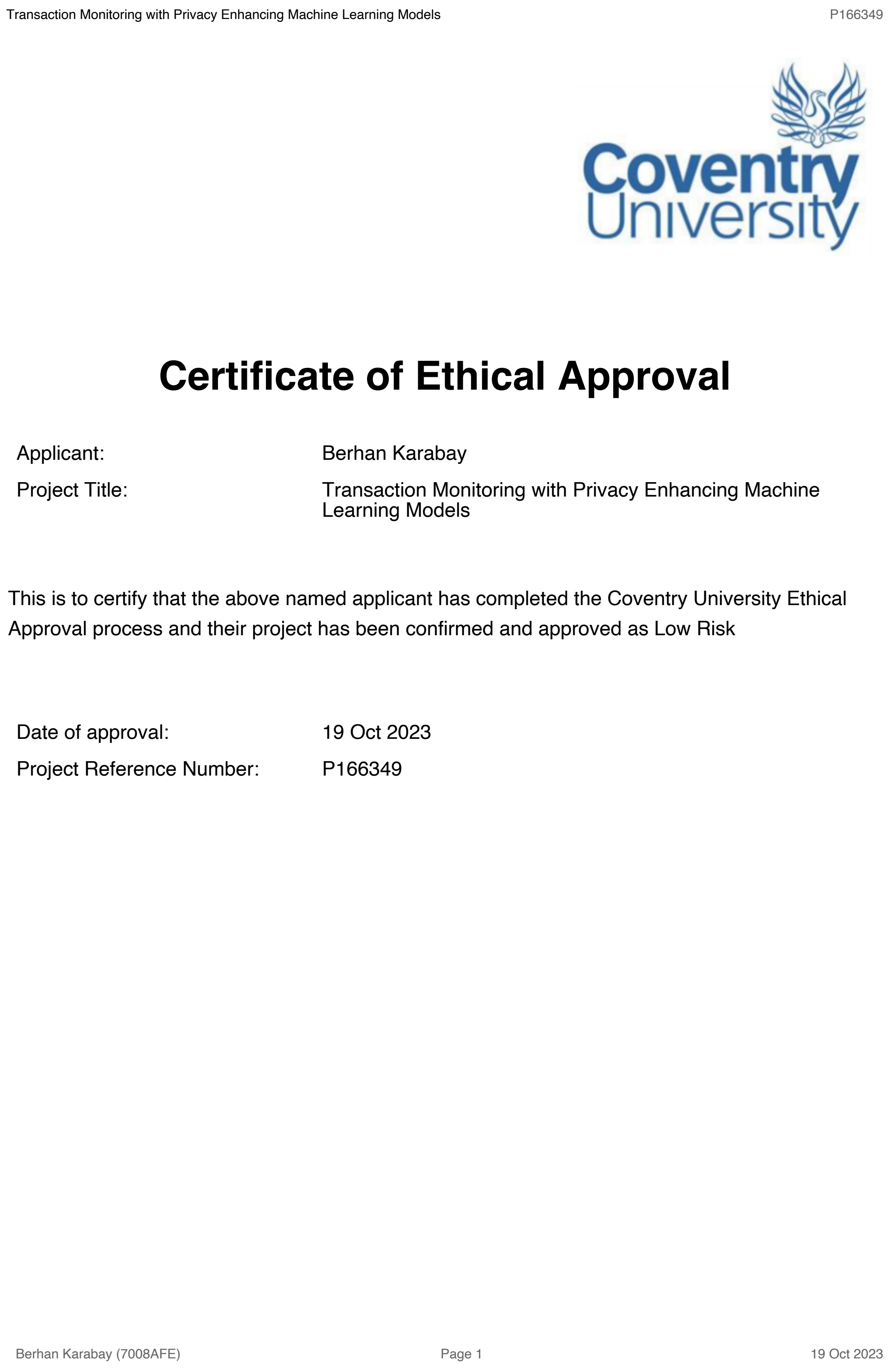
```
print(df.columns)
# Reorder the columns
new_column_order = ['Day'] + [col for col in df.columns if col != 'Day'
and col != 'Is Laundering'] + ['Is Laundering']
df = df[new_column_order]


print(df)



#%% One hot encoding??

df['Payment Format'] = pd.factorize(df['Payment Format'])[0] + 1

#correlation matrix
correlations = df.corr()
k = 10 #number of variables for heatmap
cols = correlations.nlargest(k, 'Is Laundering')['Is Laundering'].index
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f',
annot_kws={'size': 10}, yticklabels=cols.values,
xticklabels=cols.values)
plt.show()



#%% Splitting the dataset

array = df.values
X = df.drop('Is Laundering',axis=1)
Y = df['Is Laundering']

test_size = 0.20
seed = 7
X_model, X_test, Y_model, Y_test =
train_test_split(X,Y,test_size=test_size, random_state= seed)

validation_size = 0.25
X_train, X_val, Y_train, Y_val = train_test_split(X_model, Y_model,
test_size=validation_size, random_state=seed)

#%%Standardizing the numerical columns

method = SMOTE()

X_resampled, Y_resampled = method.fit_resample(X_train, Y_train)


#%% Evaluate algorithms: baseline

#num_folds = 10
#seed = 7
#scoring = 'accuracy'

# spot-check algorithms
```

```
models = []
models.append(('LR',LogisticRegression(solver='liblinear')))
models.append(('ADA',AdaBoostClassifier()))
models.append(('RF',RandomForestClassifier()))
models.append(('NB',GaussianNB()))
models.append(('XGBC',XGBClassifier()))

# Compare algorithms
results = []
names = []
for name, model in models:
    model.fit(X_resampled, Y_resampled)
    predictions = model.predict(X_val)
    accuracy = accuracy_score(Y_val, predictions)
    f1 = f1_score(Y_val, predictions, average='weighted')
    print("%s Validation Accuracy: %.2f%%" % (name, accuracy * 100))
    print(f"F1 Score: {f1:.4f}")
    names.append(name)
    results.append(accuracy)




#XGBC was best performing amonst classification models.No need to scale
our features when using XGBClassifier )

#%% Tuning XGBC


cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=seed)
param_grid = {

  "max_depth": list(range(1, 5)),

  "n_estimators": list(range(1, 5)),

  "learning_rate": [0.01, 0.1, 1],

}

model = GridSearchCV(XGBClassifier(), param_grid, cv=cv,
scoring="roc_auc")
model.fit(X_resampled, Y_resampled)

best_params = model.best_params_
print("Best Parameters:", best_params)

#%% Tuning Concrete XGBC

time_begin = time.time()

# The Concrete ML model needs an additional parameter used for
quantization
param_grid["n_bits"] = [3]

# Instantiate and fit the model through grid-search cross-validation
X_resampled = X_resampled.astype(np.float32)
```

```
concrete_model = GridSearchCV(ConcreteXGBClassifier(), param_grid,
cv=cv, scoring="roc_auc")
concrete_model.fit(X_resampled, Y_resampled)

cv_concrete_duration = time.time() - time_begin

print(f"Best hyper-parameters found in {cv_concrete_duration:.2f}s :",
concrete_model.best_params_)



#%%  Predicting Outcomes

# Compute the predictions in clear using XGBoost
clear_predictions = model.predict(X_test)

print("Accuracy Score:",accuracy_score(Y_test,clear_predictions))
print("Weighted F1 Score",f1_score(Y_test,clear_predictions,
average='weighted'))
print(confusion_matrix(Y_test,clear_predictions))
print(classification_report(Y_test,clear_predictions))

f1_minority = f1_score(Y_test, clear_predictions, pos_label=1)
print("F1 Score (Minority Class):", f1_minority)

# Compute the predictions in clear using Concrete ML
clear_quantized_predictions = concrete_model.predict(X_test)

print(accuracy_score(Y_test,clear_quantized_predictions))
print(f1_score(Y_test,clear_quantized_predictions, average='weighted'))
print(confusion_matrix(Y_test,clear_quantized_predictions))
print(classification_report(Y_test,clear_quantized_predictions))

f1_minority = f1_score(Y_test, clear_quantized_predictions,
pos_label=1)
print("F1 Score (Minority Class):", f1_minority)

# Compile the Concrete ML model on a subset
fhe_circuit =
concrete_model.best_estimator_.compile(X_resampled.head(100))



# Generate the keys
# This step is not absolutely necessary, as keygen() is called, when
necessary,
# within the predict method.
# However, it is useful to run it beforehand in order to be able to
# measure the prediction executing time separately from the key
generation one
time_begin = time.time()
fhe_circuit.keygen()
key_generation_duration = time.time() - time_begin


# Compute the predictions in FHE using Concrete ML
time_begin = time.time()
```

```
fhe_predictions = concrete_model.best_estimator_.predict(X_test,
fhe="execute")
prediction_duration = time.time() - time_begin

print(f"Key generation time: {key_generation_duration:.2f}s")
print(f"Total execution time for {len(clear_predictions)} inferences:
{prediction_duration:.2f}s")
print(f"Execution time per inference in FHE: {prediction_duration /
len(clear_predictions):.2f}s")


number_of_equal_preds = np.sum(fhe_predictions ==
clear_quantized_predictions)
pred_similarity = number_of_equal_preds / len(clear_predictions) * 100
print(

  "Prediction similarity between both Concrete-ML models"

  f"(quantized clear and FHE): {pred_similarity:.2f}%"
)

accuracy_fhe = np.mean(fhe_predictions == Y_test) * 100
print(
    "Accuracy of prediction in FHE on the test set "
f"{accuracy_fhe:.2f}%",
)
```

9.2 Ethical Approval



Certificate of Ethical Approval

Applicant:	Berhan Karabay
Project Title:	Transaction Monitoring with Privacy Enhancing Machine Learning Models

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:	19 Oct 2023
Project Reference Number:	P166349

Low Risk Research Ethics Approval

Project title

Transaction Monitoring with Privacy Enhancing Machine Learning Models

Record of Approval

Principal Investigator's Declaration

I request an ethics peer reviewI confirm that I have answered all relevant questions in this application honestly	X
I confirm that I will carry out the project in the ways described in this application. I will immediately suspend research and request an amendment or submit a new application if the project subsequently changes from the information I have given in this application.	X
I confirm that I, and all members of my research team (if any), have read and agree to abide by the code of research ethics issued by the relevant national learned society.	X
I confirm that I, and all members of my research team (if any), have read and agree to abide by the University's Research Ethics Policies and Processes.	X
I understand that I cannot begin my research until this application has been approved and I can download my ethics certificate.	X

Name: Berhan Karabay (7008AFE)

Date: 19/10/2023

Student's Supervisor (if applicable)

I have read this checklist and confirm that it covers all the ethical issues raised by this project fully and frankly.I also confirm that these issues have been discussed with the student and will continue to be reviewed in the course of supervision.

Name: Chi Tran

Date: 19/10/2023

Reviewer (if applicable)

Date of approval by anonymous reviewer: -