Tilburg University

# MACHINE LEARNING AND DEEP LEARNING FOR ETHEREUM FRAUD DETECTION

## A COMPARATIVE ANALYSIS OF LGBM AND MLP

XIAOHUI GAO

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN DATA SCIENCE & SOCIETY
AT THE SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
OF TILBURG UNIVERSITY

STUDENT NUMBER

2110073

COMMITTEE

dr. Boris Čule
Julija Vaitonytė

LOCATION

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science &
Artificial Intelligence
Tilburg, The Netherlands

DATE

June 24th, 2024

WORD COUNT

8,571

## ACKNOWLEDGMENTS

## CONTENTS

# MACHINE LEARNING AND DEEP LEARNING FOR ETHEREUM FRAUD DETECTION

A COMPARATIVE ANALYSIS OF LGBM AND MLP

XIAOHUI GAO

**Abstract**

The increasing adoption of blockchain technology, while revolutionary, has also facilitated fraudulent activities due to anonymity, resulting in significant financial losses and heightened threats to the security of the Ethereum network. This study proposes a comparative analysis of Light Gradient Boosting Machine (LGBM) and Multi-Layer Perceptron (MLP) models to identify the optimal approach for addressing this issue. The initial experiment, aimed at tackling data imbalance, reveals that non-sampled data yields superior performance by maintaining neutrality and leveraging the models' inherent capabilities. Subsequent model optimization further highlights the LGBM model's superiority, boasting higher performance metrics: F1-score (0.969), accuracy (0.986), AUC-ROC score (0.9987), and precision (0.994). Moreover, LGBM demonstrates superior computational efficiency, with a training time of 0.6662 seconds and an inference time of 0.0142 seconds, positioning LGBM as a more robust and efficient solution, particularly in real-time scenarios requiring rapid and accurate predictions. Future research should explore novel data preprocessing techniques and incorporate diverse datasets to further enhance its effectiveness and applicability.

## 1 DATA SOURCE, ETHICS, CODE, AND TECHNOLOGY STATEMENT

The data was obtained from Vagif Aliyev via Kaggle, 2021. This research did not involve collecting data from human participants or animals, the original generator of the data used in this thesis retains the ownership during and after the completion of this study. All figures and tables in this thesis were created by the author. Figures 1, 8 and 31, however, are respectively sourced from SlowMist, Dinesh Sivakumar and Aziz et al., we acknowledge the original version of these visualizations and have provided proper citation. Additionally, ChatGPT-4 was employed to assist with debugging. We also utilized a thesaurus and incorporated paraphrasing suggestions from ChatGPT-4 to improve clarity and flow. A list of used Python libraries including version numbers can be found in Appendix A

## 2 INTRODUCTION

### 2.1 *Problem Statement*

Ethereum, the second most popular blockchain after Bitcoin, distinguishes itself through its unique functionalities. While Bitcoin has evolved primarily into a digital asset for long-term investment, Ethereum operates as a decentralized platform facilitating the execution and verification of smart contracts, enabling a diverse array of applications beyond financial transactions (Oliva et al. (2020); Aswin and Kuriakose (2020)). The extensive adoption of Ethereum, evidenced by its total market capitalization of $418,424,313,496 as reported by CoinGecko, and the proliferation of applications in various sectors such as Decentralized Finance (DeFi), Game Finance (GameFi), and Non-Fungible Tokens (NFTs), underline its significance. However, this exponential growth has been accompanied by a surge in fraudulent activities including money laundering, phishing scams, hacking, and other illicit practices (Johnson, 2020), posing significant challenges to the security and integrity of the Ethereum network.

SlowMist – the crypto security and audit company has released a report in terms of Blockchain Security and Anti-Money Laundering as Figure 1 suggested. The report has revealed that there were 280 DeFi security incidents, representing 60.77% of the total incidents. These incidents resulted in losses totaling $773 million. This marks a significant improvement compared to 2022, which saw 183 incidents with losses amounting to approximately $2.075 billion, reflecting a 62.73% reduction in losses year-over-year.

Ethereum experienced the largest losses on record, totaling $487 million, making it the top-ranked entity in terms of losses. More information about other blockchains was detailed in Appendix B

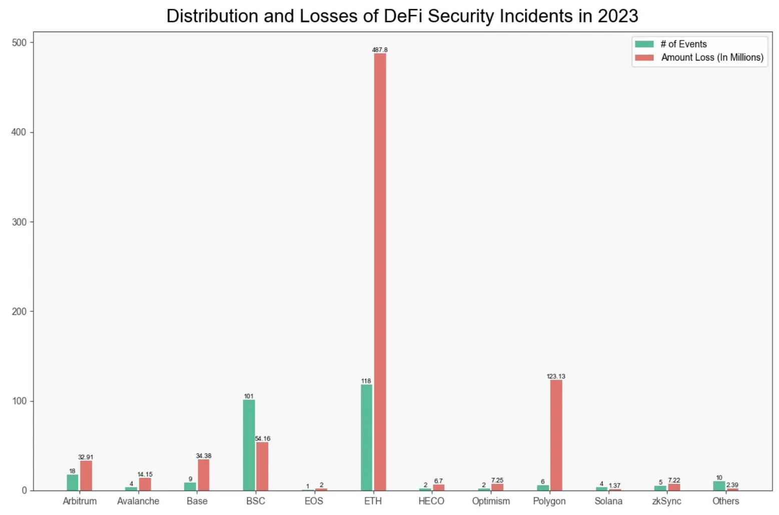

Figure 1: Distribution and losses on different blockchains from SlowMist (2024)

This issue, primarily concerning security within the DeFi space, is a subset of the broader challenge posed across the whole Ethereum network. The potential for diverse forms of illicit behavior requires a comprehensive analysis to develop effective solutions. While researchers like Farrugia et al. (2020) have explored fundamental machine learning models such as Extreme Gradient Boosting (XGBoost), K-Nearest Neighbors (KNN), and Decision Trees to enhance performance, Aziz et al. (2022) identified the LGBM as the optimal model in this context. Concurrently, studies like Chen et al. (2020) have investigated deep-learning neural networks, to address the same challenge. This divergence in methodological approaches highlights the need for further research to identify the most effective model for the Ethereum ecosystem.

This thesis conducted a comparative analysis of two models, the machine learning-based LGBM and the deep learning-based MLP, to ascertain their respective effectiveness in the binary classification task of fraud detection within the blockchain and crypto domain. The models were trained and evaluated independently to approximate the best-performing one.

## 2.2 *Societal and Scientific Relevance*

While previous work has predominantly focused on either machine learning or deep learning models for fraud detection, this thesis aims to bridge the gap by conducting a comparative analysis between the LGBM, identified as the optimal machine learning model by Aziz et al. (2022), and the

MLP, a well-established deep learning architecture for classification tasks. This novel approach seeks to illuminate the relative strengths and weaknesses of these two distinct models in the specific context of blockchain fraud detection.

From a societal perspective, the implications of this research are profound. As blockchain technology continues to gain widespread adoption across various industries, ensuring the security and integrity of blockchain transactions becomes increasingly vital. According to ychart, the cumulative unique addresses on the Ethereum blockchain have reached 271.73 million, reflecting a 16.34% increase from the previous year. This rapid growth implies the urgent need for effective fraud detection mechanisms that can help mitigate malign actions, prevent substantial financial losses, protect user assets, and maintain trust in decentralized systems. By enhancing the security of Ethereum and other blockchain platforms, this project supports the broader societal goal of fostering secure, transparent, and reliable digital ecosystems (Jin & Xiao, 2022), which are essential for the continued growth and adoption of blockchain technology in areas such as finance, supply chain management, and beyond.

## 2.3 *Research Strategies*

Previous research has been compartmentalized, focusing primarily on either machine learning domain (Ibrahim et al. (2021); Aziz et al. (2022); Farrugia et al. (2020)) or deep learning space (Chen et al. (2020); Hu et al. (2021)), often extolling the virtues of their respective approaches. Since Aziz et al. (2022) highlighted the exceptional performance of LGBM in Ethereum fraud detection and MLPs have gained a well-established reputation for handling classification tasks, addressing the divergence in previous research prompts a central question for this thesis:

RQ *How does MLP compare to LGBM in detecting fraud on Ethereum?*

To address this question, the following three sub-questions were formulated:

Given the prevalence of SMOTE (Aziz et al., 2022) and undersampling (Bartoletti et al., 2018) in prior research, without a thorough evaluation of their impact and the absence of sampling in other studies (Pham and Lee (2016); Monamo et al. (2016)), the following question is of particular interest:

SQ1 *What approaches, including non-sampling and sampling methods, are most effective for each model in handling class imbalance?*

After determining the best sampling method, this study leverages the previously employed LGBM (Aziz et al., 2022) and MLP as baseline models for optimization, the fine-tuned LGBM and MLP models are then compared against their respective baselines to assess the effect of optimization on model performance, eventually leading to the sub-question:

SQ2 *Which model demonstrates better performance?*

Finally, considering the importance of computational efficiency in practical applications, as stressed by Ibrahim et al. (2021) in their time comparison, another sub-question is posed:

SQ3 *How do training time and inference speed compare between MLP and LGBM?*

## 2.4 *Main Findings*

The LGBM model demonstrated superior performance compared to the MLP model, accomplishing higher F1-score (0.969 versus 0.935), accuracy (0.986 versus 0.972), AUC-ROC score (0.9987 versus 0.9880), and precision (0.994 versus 0.944), indicating greater robustness in identifying frauds. Both models performed optimally on non-sampled data, suggesting the synthetic sampling techniques did not offer any additional advantage.

Additionally, the LGBM model demonstrated significantly faster training (0.6662s versus 34.32s) and inference (0.0142s versus 34.32s) times compared to the MLP model, proving its computational advantages for real-time fraud detection applications.

# 3 RELATED WORK

The ever-present threat of fraud permeates the dominant Bitcoin network and the broader crypto and blockchain landscape. This section delved into existing research on fraud detection within this domain, focusing naturally on Bitcoin due to its market position. However, our attention shifted to a more specific exploration of research on Ethereum fraud detection. By identifying current gaps and outlining our motivations, we aim to make contributions and bolster Ethereum's ability to defend against fraudulent activities.

## 3.1 *Area of Research*

Crypto and blockchain fraud posed a persistent threat, prompting researchers to continually innovate anomaly detection methods, particularly for the dominant player, Bitcoin. Prior research by Pham and Lee (2016) explored unsupervised learning techniques (k-means, Mahalanobis distance, Unsupervised SVM) on user and transaction graphs within the Bitcoin network. Their dataset, limited to a 100,000 subset due to computational constraints, contained over 6 million users, and 37 million transactions with detailed 30 identified thieves. Interestingly, both Unsupervised SVM and Mahalanobis distance methods identified similar suspicious users, successfully detecting 2 thefts and 1 loss case. Sharing the same unmodified dataset, Monamo et al. (2016) investigated trimmed k-means clustering in comparison to standard k-means, which struggles with outliers. Trimmed k-means offered improved performance by identifying 5 out of the 30 known fraudulent users, demonstrating an improvement over the previous approach. Another study (Bartoletti et al., 2018) examined data mining methods (RIPPER, Bayes Net, Random Forest) to detect Ponzi schemes in Bitcoin transactions. They created a dataset with two categories: Ponzi schemes (P) and regular transactions (nP), reflecting real-world imbalance (32 Ponzi versus 6400 non-Ponzi). Classifiers initially struggled due to the imbalance. Techniques like undersampling helped but increased false positives. Cost-sensitive learning, which prioritizes catching Ponzi schemes, yielded the best results with Random Forest. Overall, Random Forest with cost-sensitive learning showed promise for effectively detecting Ponzi schemes in Bitcoin transactions.

## 3.2 *Past Work on Ethereum Fraud Detection*

### 3.2.1 *Using Machine Learning Models*

At the same time, various studies pioneered the use of advanced models and feature engineering for Ethereum fraud detection. Farrugia et al. (2020) utilized an XGBoost model with hyperparameter tuning via 10-fold cross-validation on a dataset sourced from the Ethereum Scam Database and a local Geth client connected to the Ethereum network to distinguish scam and abnormal accounts. This methodology achieved an average accuracy score of 0.963 (± 0.006) and an AUC score of 0.994 (± 0.0007).

A further interesting finding is examined by Ibrahim et al. (2021), which utilized a correlation coefficient to select 6 features and investigated the Decision Tree, Random Forest, and K Nearest Neighbors (KNN). Notably, each algorithm demonstrated a significant time reduction in processing: the Decision Tree exhibited a decrease from 0.24s to 0.05s, the Random Forest from 4.85s to 0.62s, and the KNN from 0.01s to 0s accompanied by better performance with the F-measure increasing from 0.974 to 0.976, particularly for the Random Forest algorithm.

Similarly, Aziz et al. (2022) compared the capacity of multiple models for Ethereum fraud detection such as Adaptive Boosting (ADABOOST), Support Vector Classifier (SVC), Extreme Gradient Boosting (XGBOOST), KNN, LGBM, MLPClassifier, Random Forest, Logistic Regression. The dataset used in their study consisted of 9,841 rows and 17 columns, derived from the Classic (2024) blockchain and subjected to sampling using the SMOTE. They analyzed the corresponding scores of each approach as Figure 2 explained. LGBM achieved the highest accuracy on both the training set and the test set described in Figure 2, scoring at 0.9865 and 0.986, respectively. Further optimization pushed the LGBM test accuracy to 0.9903. However, the inability to replicate the results due to undisclosed hyperparameter values limited the usefulness of the findings from them. It's important to note that the MLP Classifier employed by Aziz et al. (2022) was a pre-defined model from the scikit-learn (Pedregosa et al., 2007) library, evaluated without hyperparameter tuning. This approach limited the model's reproducibility and adaptability. In contrast, we built an MLP neural network with three hidden layers using the Keras (Chollet et al., 2015) library, allowing us greater flexibility and optimization through a rigorous hyperparameter tuning process.

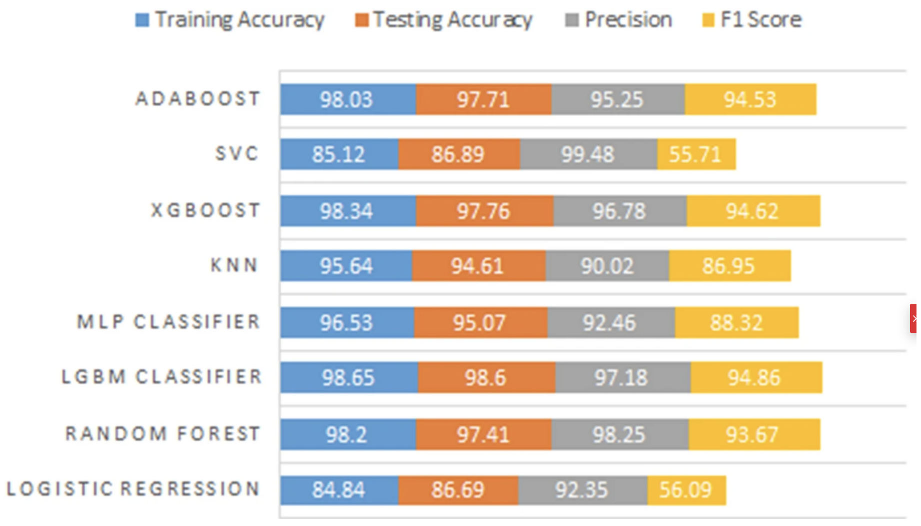

Figure 2: Model results from Aziz et al. (2022)

### 3.2.2 *Using Deep Learning Models*

Furthermore, additional research has applied deep learning models to detect fraud. Chen et al. (2020) proposed a solution to distinguish phishing accounts on the Ethereum using Graph Convolutional Networks (GCN) and autoencoders. Their dataset crawled from Ethereum, included transaction information with nearly 3 million nodes and over 1,000 labeled phishing nodes. They compared three feature engineering approaches: raw features, graph embeddings (DeepWalk/Node2Vec/LINE), and a GCN-based model. Evaluated on datasets of varying sizes (30,000, 40,000, and 50,000 nodes) and metrics (AUC, Recall, Precision), the GCN model consistently outperformed the others, as shown in Table 1.

Table 1: GCN Performance

| MODEL | AUC | Recall | Precision | F1 |
|---|---|---|---|---|
| **Graph size=30,000** | | | | |
| GCN | 0.5725 | 0.1453 | 0.7294 | 0.2357 |
| **Graph size=40,000** | | | | |
| GCN | 0.5725 | 0.1453 | 0.6648 | 0.2289 |
| **Graph size=50,000** | | | | |
| GCN | 0.5866 | 0.1735 | 0.6278 | 0.2636 |

Hu et al. (2021) categorized over 10,000 Ethereum smart contracts, identifying four distinct behavior patterns through manual analysis which laid the groundwork for developing 14 features that could serve to differentiate between contract types. Leveraging these features, they employed a Long

Short-Term Memory (LSTM) network for the classification, achieving good results across key metrics such as precision, recall, and the F1 score which respectively are 0.88, 0.7, 0.77.

## 3.3 *Research Gap*

A critical review of discussed existing research revealed three key shortcomings. First, there hasn't been any significant work dedicated to conducting a comparative analysis between machine learning and deep learning models, impeding our ability to pinpoint the optimal methodologies for this task. Second, the selection of sampling techniques was often subjective and inconsistent across studies without distinguishing their impact. Some studies, like those by Aziz et al. (2022) and Bartoletti et al. (2018), employed SMOTE and undersampling, while others neglected them entirely. Finally, model efficiency, measured by training time and computational cost, was largely overlooked. Ibrahim et al. (2021) were the only ones to mention training time, highlighting the need for a more comprehensive evaluation of model efficiency, especially for real-world deployment with large datasets.

## 3.4 *Motivation and Contributions*

Prior literature suggested LGBM achieved the highest accuracy (0.9903) among machine learning models (Aziz et al., 2022) , while LSTM, representing a deep learning approach, achieved superior precision (0.88), recall (0.70), and F1-score (0.77) (Hu et al., 2021). Our work aims to bridge this gap by providing a detailed analysis of how models from two different paradigms handle the evolving complexities of Ethereum fraud patterns.

Motivated by the lack of consensus on optimal sampling techniques in prior research, this study aims to thoroughly evaluate a broader range of methods, including SMOTE, ADASYN, undersampling, and a non-sampled baseline. This evaluation is particularly pertinent for scenarios with potentially mild class imbalance, as it seeks to discern whether complex sampling techniques offer significant advantages in such cases. By investigating these factors, we aim to provide valuable insights into appropriate sampling methodologies for future research, ultimately paving the way for more robust and efficient models to protect the Ethereum network.

Building upon the work of Ibrahim et al. (2021), who observed substantial reductions in training time, particularly for KNN (reported as very close to zero), we delved deeper into model efficiency by analyzing training time and inference speed. This in-depth analysis provides valuable insights into the computational resources required by the models. While training time reflects the learning phase, prediction time highlights the efficiency of making real-time predictions on new data. Both factors are crucial for real-world deployment regarding fraud detection on Ethereum.

## 4 METHODOLOGY AND EVALUATION

### 4.1 *Summarized Workflow*

Figures 3 and 4 detailed the research methodology and modeling pipeline. The process included EDA, data cleaning and preprocessing, and data splitting. Best sampling techniques were selected using the validation set. Hyperparameter tuning was performed on the combined training and validation sets with 5-fold cross-validation. The final tuned-model comparison was explored on the test set, with training time and inference speed measured and averaged over iterations.

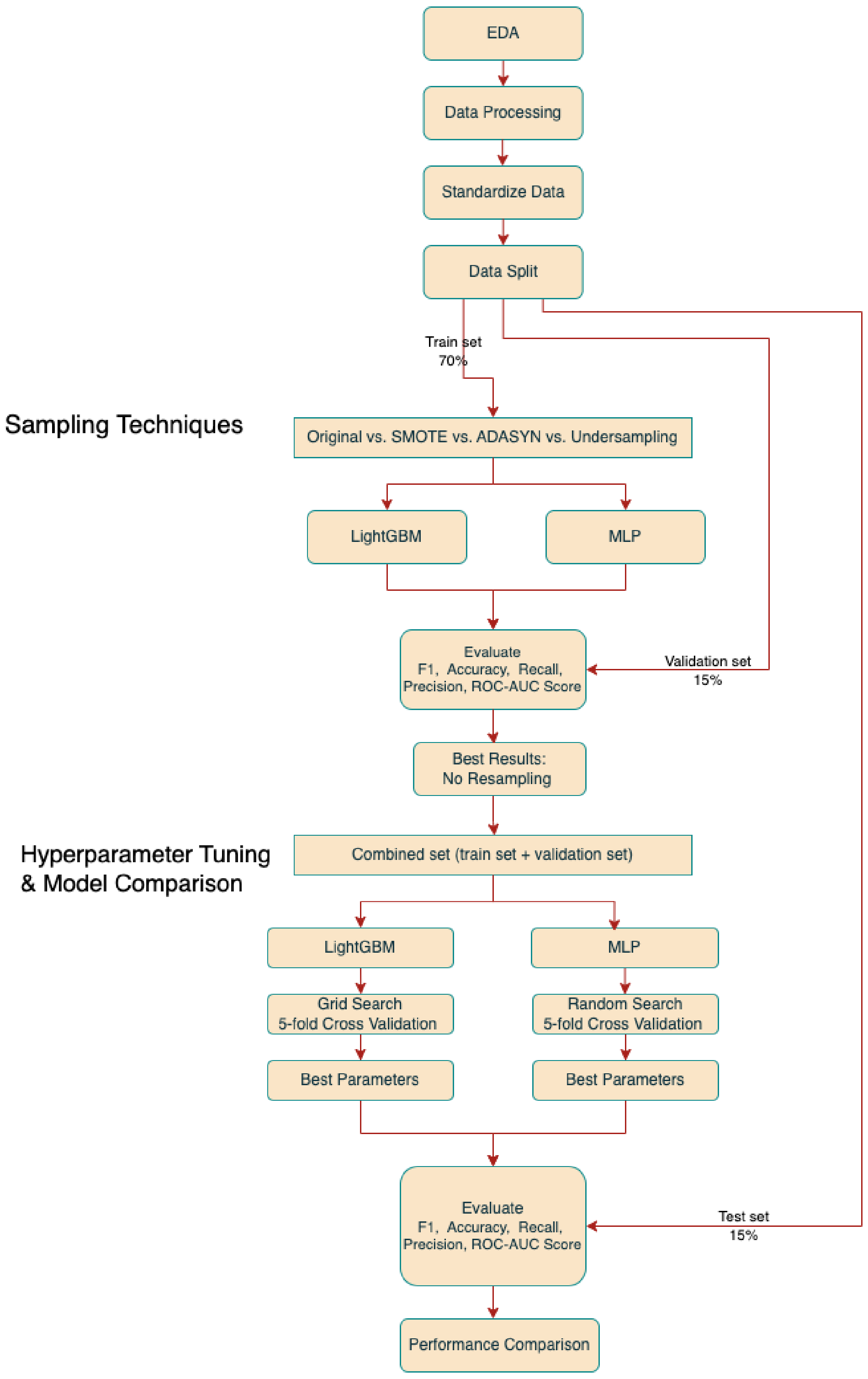
EDA
Data Processing
Standardize Data
Data Split
Train set 70%
Sampling Techniques
Original vs. SMOTE vs. ADASYN vs. Undersampling
LightGBM
MLP
Evaluate F1, Accuracy, Recall, Precision, ROC-AUC Score
Validation set 15%
Best Results: No Resampling
Hyperparameter Tuning & Model Comparison
Combined set (train set + validation set)
LightGBM
MLP
Grid Search 5-fold Cross Validation
Random Search 5-fold Cross Validation
Best Parameters
Best Parameters
Evaluate F1, Accuracy, Recall, Precision, ROC-AUC Score
Test set 15%
Performance Comparison

Figure 3: Sampling techniques comparison and final model comparison

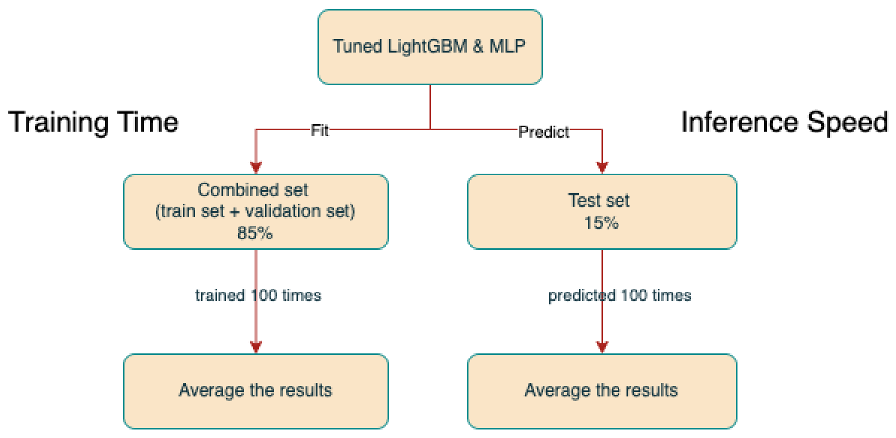

Figure 4: Training time and inference speed

## 4.2 *Dataset Description*

The dataset utilized in this thesis consists of transaction records from the Ethereum blockchain, comprising 9,841 entries and 50 features. It covers various transaction attributes detailed in Appendix C, including time dynamics, value transfers, and contract interactions, all of which are potential indicators of fraudulent activity. The data was collected by Vagif Aliyev through the Etherscan API, EtherscamDB API, and is publicly accessible on Kaggle.

Each data entry represents a unique Ethereum address and is characterized by a set of features that capture various aspects of its transactional behavior. These features are predominantly numerical with three textual attributes.

## 4.3 *Exploratory Data Analysis*

This section was performed to investigate the dataset in detail.

Upon loading the data from a CSV file into a DataFrame, we identified non-unique indices, suggesting potential duplicate entries. Further investigation revealed entries with identical features. The number of duplicates ranged from one to three occurrences per entry. As noted by Nauman and Herschel (2022) and Dozmorov et al. (2015), duplicates can introduce noise and negatively impact model performance. Therefore, addressing these duplicates is crucial for data cleaning, which will be covered in the following subsection.

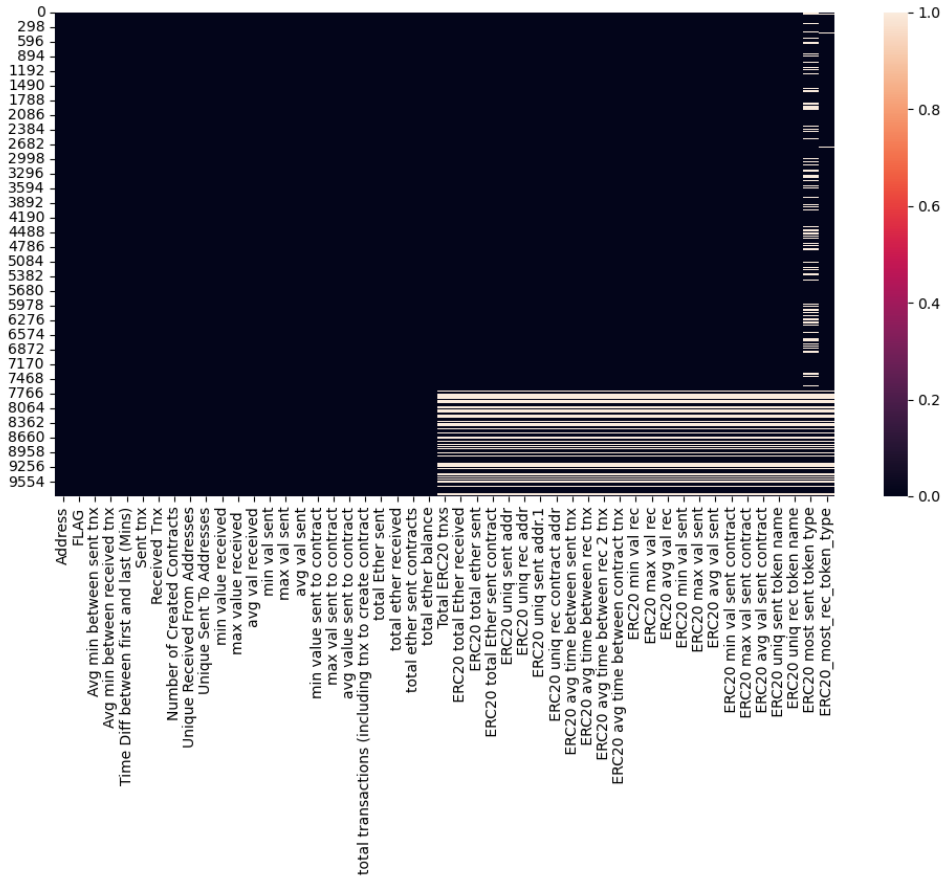

Figure 5: Missing values

Figure 5 presents a heatmap visualizing missing data across all features in the dataset. Each row represents an individual Ethereum account. Notably, a significant portion of observations have missing values, particularly in features following "Total ERC 20 tnxs." This suggests a potential pattern of missing data in these features. Additionally, the "ERC20 most sent token type" column exhibits a particularly high prevalence of missing values.

Appendix D details missing data proportions per feature. Features like "Total ERC20 transactions" and "FLAG" exhibit no missing values, while "ERC20 most sent token type" has the highest percentage(27.4%). Others show varying degrees, with many around 8.4%, suggesting potential data recording or retrieval inconsistencies. Extensive missing data can reduce the effective sample size, weakening statistical power and potentially leading to unreliable conclusions (Little & Rubin, 2019). Additionally, it may introduce bias and skew results, particularly if missingness is correlated with other variables (Xiong & Pelger, 2023). Therefore, a suitable imputation method is essential and will be discussed later.

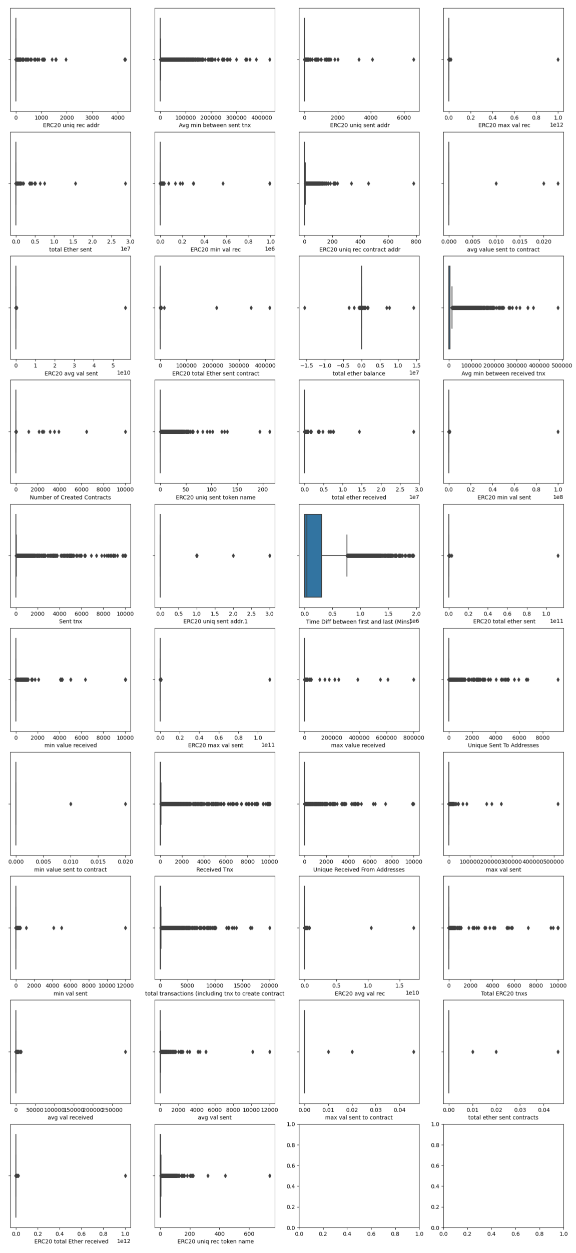

Figure 6: Feature distribution

Figure 6 presents boxplots for the numerical features, visualizing the median, interquartile range (IQR), and outliers. The median and IQR highlight central tendency and variability, while outliers indicate potential anomalies requiring further investigation. The figure reveals significant variations in feature distributions. Features such as "ERC20 avg val sent" "ERC20 min val sent" and "max val sent to contract" exhibit noticeably smaller distributions, while "Avg min between sent tnx", "total Ether sent" and others show upper outliers, suggesting values exceeding typical ranges. Notably, "total ether balance" displays a lower outlier. These outliers can negatively impact model training, leading to overfitting as the model overemphasizes extreme values (Zhao & Akoglu, 2023), which can distort the model's perception of the data distribution, resulting in biased predictions and reduced accuracy (Montgomery et al., 2023). In addition, variability in feature distributions can hinder the learning process. Features with smaller distributions might be under-represented, while those with larger ranges might dominate the learning process (Montgomery et al., 2023). This imbalance can affect model generalizability, limiting its ability to adapt to unseen data. To address these issues, robust preprocessing techniques such as outlier clipping are crucial for creating a balanced dataset that accurately reflects underlying patterns.

The "FLAG" column serves as the binary outcome variable, with a value of 1 indicating a flagged fraudulent account. The analysis identified a class imbalance: approximately 77.82% (7,644 entries) belong to the non-fraudulent class (0), while 22.18% (2,179 entries) represent fraudulent transactions. The dataset was constructed 3 years ago by Vagif Aliyev mentioned in 4.2, the representativeness of the class imbalance in real-world scenarios cannot be definitively ascertained. However, imbalanced datasets can lead to biased models that favor the majority class, potentially overlooking or misclassifying the minority class of fraudulent entries (Wongvorachan et al., 2023). While the imbalance is not severe, it warrants careful handling to ensure model accuracy and reliability. To mitigate this issue, several sampling techniques, including SMOTE, ADASYN, and undersampling, were assessed in the later section 4.6.1, allowing us to determine the optimal approach regarding this specific context.

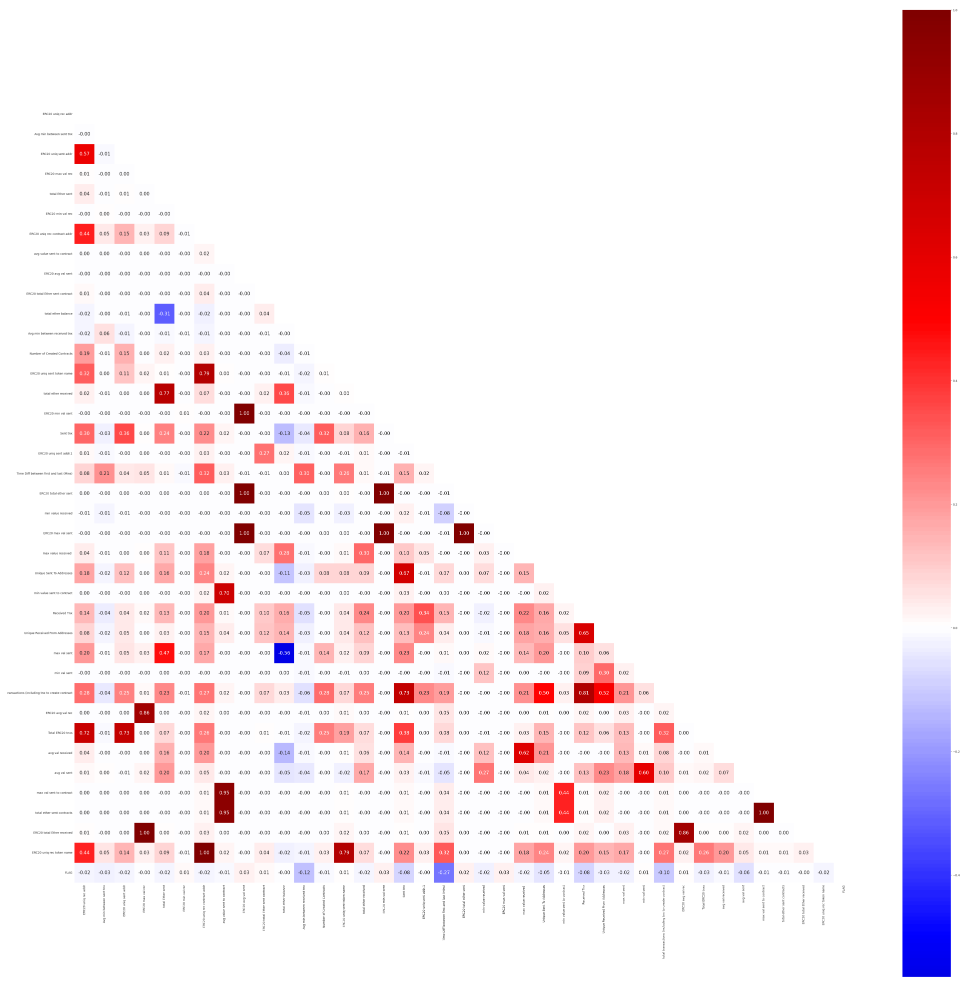

Figure 7: Correlation matrix

Analysis of the correlation matrix (Figure 7) reveals strong relationships between transaction metrics. Total ether sent to contracts correlates with both average and maximum values sent per contract, indicating larger transactions drive overall volume. Similarly, unique recipient addresses correlate with token type variety, reflecting diverse transaction activities. Minimum and maximum values sent to contracts also significantly impact the total ether sent, suggesting a consistent pattern with a limited value range. These highly correlated features can introduce multicollinearity, where redundant information impedes isolating individual feature effects on the outcome variable (Fried, 2020). They can also lead to inflated standard errors and potentially hamper model interpretability (Kyriazos & Poga, 2023). Additionally, overfitting can occur as models prioritize capturing noise in training data rather than underlying patterns, reducing generalizability. Feature selection techniques will be employed in subsequent sections to address these concerns.

## 4.4 *Data Cleaning and Preprocessing*

### 4.4.1 *Data cleaning: Removing Duplicates*

Examination of the dataset revealed a limited number of observations exhibiting identical values across all features. These duplicate entries, constituting a negligible fraction of the data (initial: 9,841 observations, final: 9,823 observations), were subsequently removed. As duplicates offer no additional information and can potentially introduce redundancy or bias into model training (Dozmorov et al., 2015), their elimination is crucial for ensuring the integrity and reliability of the subsequent model development process.

### 4.4.2 *Data cleaning: Deleting Features with Zero-variance*

This step involved identifying and removing zero-variance features. These features, characterized by identical values across all observations, lack discriminatory power in classification tasks (Wang et al., 2021). Essentially, they contribute no new information relevant to the target variable and can be safely removed without compromising the integrity of the data. Boeschoten et al. (2023) highlighted removing such features streamlines the modeling process by reducing computational complexity and potentially improves model interpretability by focusing on informative features.

### 4.4.3 *Data cleaning: Imputation of Missing Data*

Median imputation was chosen as the preferred approach for its robustness against outliers, prevalent in financial data (Zhu et al., 2018). Unlike the mean, the median remains unaffected by extreme transaction values, making it a more suitable measure of central tendency for skewed distributions (Brooks, 2019). This approach safeguards data integrity and preprocessing consistency, crucial for reliable analysis. Moreover, the median's efficiency and simplicity are well-suited for large datasets (Leys et al., 2019), aiding in maintaining both speed and accuracy within the fraud detection process. Median imputation minimizes bias risk, enhancing the validity of subsequent fraud identification analyses.

### 4.4.4 *Data cleaning: Clipping Outliers*

To mitigate the influence of outliers in numerical features, quantile clipping was applied. This technique restricts values to a specific interquartile range

(IQR), typically between the 5th and 95th percentiles. This range is chosen because, while robust to some outliers, MLP and LGBM models can still be sensitive to extremely skewed data. By constraining extreme values, quantile clipping promotes data consistency and reduces the potential for misleading analysis. Consequently, this approach improves the ability to identify underlying patterns and potential fraudulent activities within the data.

### 4.4.5 *Data Preprocessing*

HIGHLY-CORRELATED FEATURES Feature selection addressed multicollinearity concerns through a model-specific approach. LGBM, due to its tree-based architecture, tolerated moderate correlation. However, excessive correlation (>90%) could slow split efficiency and bias feature prioritization, affecting performance and interpretability. Conversely, MLPs retained correlated features. The complex architecture of neural networks allows each feature to influence multiple neurons in various ways, and the adaptive nature of MLPs facilitates the selection of informative features during training (Abdolrasol et al., 2021). Explicit removal based solely on correlation was unnecessary.

CATEGORICAL FEATURES The categorical column "Address" was excluded from the dataset as it directly identified users and lacked predictive power for fraudulent behavior. The remaining categorical features, "ERC20 most sent token type" and "ERC20_most_rec_token_type", were removed due to their vast number of unique categories as shown in Table 2. Encoding these features using classic methods like one-hot encoding would lead to dimensionality issues and computational burdens, impacting model efficiency. Besides, given that the homogeneity of the ERC-20 token standard causes the specific token type less informative than transaction value and volume, which are typically denominated in stable currencies like USDT and ETH (native token of the Ethereum network), their limited practical utility in fraud detection further supported the removal. Also, the dynamic nature of the ERC-20 token landscape necessitates frequent model updates, incurring increased complexity and maintenance costs.

| **Categorical Features** | **Unique Values** |
|---|---|
| ERC20 most sent token type | 304 |
| ERC20_most_rec_token_type | 466 |

Table 2: ERC20 token types and their unique values

DATA SPLITTING The dataset was partitioned into training (70%), validation (15%), and testing (15%) sets using stratified random sampling to maintain class distributions. The validation set was primarily used for selecting optimal sampling techniques. Subsequently, the training and validation sets were combined for hyperparameter tuning of LGBM and MLP models using 5-fold cross-validation. The final model performance was assessed on the held-out test set. This approach, implemented with sci-kit-learn's **'train_test_split'** function and a fixed random seed, ensures reproducibility and mitigates potential biases.

SCALING Feature scaling using StandardScaler, which normalizes features to zero mean and unit variance (Equation 1)), was applied to ensure equitable feature contributions and prevent any single feature from dominating the learning process (Géron, 2022). This is particularly beneficial for gradient-based models like MLPs, where scaling improves backpropagation efficiency and convergence speed (Géron, 2022). While tree-based models like LGBM are generally less sensitive to feature scaling, standardization remains a recommended preprocessing step in most machine learning pipelines to facilitate a fair comparison and prevent biases towards features with larger magnitudes. (Scikit-Learn Developers).

$$Z = \frac{(x-\mu)}{\sigma} \tag{1}$$

## 4.5 *Algorithms*

In this research, we specifically investigated two algorithms, which are LGBM and MLP.

**LGBM** has been widely used and highly regarded for classification tasks. Built on the gradient boosting framework, it minimizes a loss function by iteratively adding weak learners to form a strong learner, optimizing the following objective function:

$$\text{Objective} = \sum_{i=1}^{N} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{2}$$

LGBM grows trees leaf-wise, expanding the leaf with the largest loss reduction:

$$\Delta L = \frac{1}{2}\left(\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} + \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right) - \gamma \tag{3}$$

In this case, it allows the model to capture complex patterns. LGBM uses a histogram-based algorithm to speed up the training process by binning feature values:

$$H_j = \sum_{i \in I_i} g_i, B_j = \sum_{i \in I_i} h_i \tag{4}$$

This ensures fast training and prediction, crucial for real-time fraud detection.

For our specific scenario, LGBM provides significant advantages in speed, scalability, memory usage, and accuracy compared to other tree classifiers.

We initialized LGBM model using the **LGBMClassifier()** function with default parameters including **'learning_rate=0.1'**, **'max_depth=-1'**, **'n_estimators=100'**, and **'num_leaves=31'**. These parameters were chosen for their substantial impact on model performance and served as the primary focus for subsequent optimization.

**MLPs** are fundamental artificial neural networks widely used for classification tasks. The architecture of an MLP comprises an input layer, multiple hidden layers, and an output layer (Xiao et al., 2017). Each layer has nodes fully connected through adjustable weights.

During forward propagation, features are passed through the network. Each hidden layer node computes a weighted sum of its inputs and adds a bias term:

$$z_j^l = \sum_{i=1}^{n} w_{ij}^l x_i^{l-1} + b_j^l \tag{5}$$

The weighted sum $\mathrm{z}_\mathrm{j}^\mathrm{l}$ is then passed through an activation function $\phi$:

$$a_j^l = \phi(z_j^l) \tag{6}$$

This introduces non-linearity, allowing the network to learn complex patterns in the data. The output layer generates the final prediction, using activation functions like softmax for multi-class classification or sigmoid for binary classification.

During backpropagation, the error between the predicted output and the true label is calculated using a loss function, typically the cross-entropy loss for classification tasks:

$$L = -\sum_{i=1}^{N} y_i \log(O_i) \tag{7}$$

For binary classification, the cross-entropy loss can be simplified as:

$$L = -y \log(p) - (1 - y) \log(1 - p) \tag{8}$$

The gradients of the loss function concerning each weight are computed, and the weights are updated using gradient descent (Haji & Abdulazeez, 2021).

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial L}{\partial w_{ij}} \tag{9}$$

where $\eta$ is the learning rate. This iterative process continues until the loss reaches a minimum or an acceptable threshold, indicating that the model has learned the optimal weights and biases.

The MLP model architecture in this study was designed to accommodate the 38-feature dataset, including highly correlated features discussed in 4.4.5. The initial hidden layer of 38 units facilitated one-to-one feature mapping and learning of intricate patterns. Subsequent hidden layers (15 and 5 units) progressively reduced dimensionality to focus on important features and mitigate overfitting. ReLU activation was used in all hidden layers to introduce non-linearity and mitigate the vanishing gradient problem. The final single-unit output layer with sigmoid activation mapped the output to a probability between 0 and 1, suitable for binary classification tasks.

Figure 8 visually represents the architecture and operational mechanism of the MLP model employed in this study.

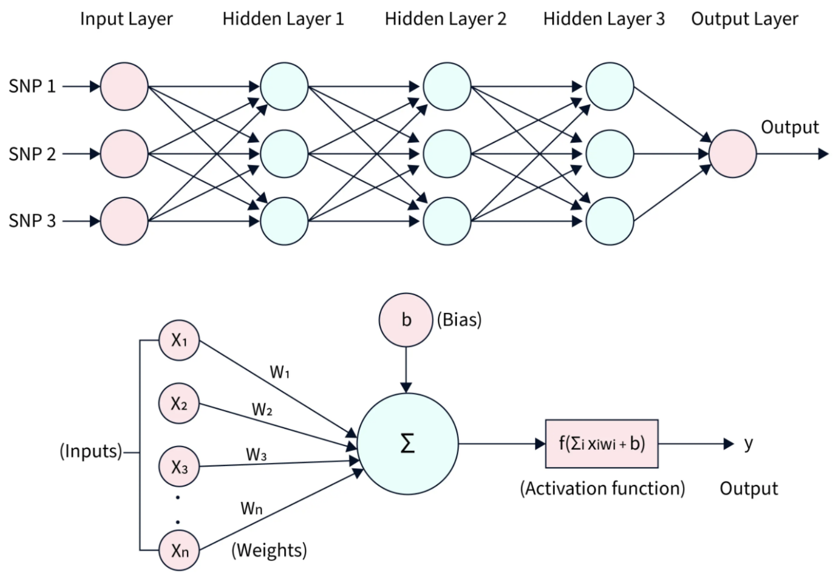

Figure 8: MLP architecture from Dinesh Sivakumar (2024)

The model was compiled with the Adam optimizer, binary cross-entropy loss (appropriate for binary classification), and accuracy metric. Table 3 details the model architecture and parameters.

Table 3: MLP Keras Model Summary

| **Layer (type)** | **Output Shape** | **Param #** |
|---|---|---|
| `dense_8 (Dense)` | (None, 38) | 1,482 |
| `dense_9 (Dense)` | (None, 15) | 585 |
| `dense_10 (Dense)` | (None, 5) | 80 |
| `dense_11 (Dense)` | (None, 1) | 6 |
| **Total params** | | **2,153 (8.41 KB)** |
| **Trainable params** | | **2,153 (8.41 KB)** |
| **Non-trainable params** | | **0 (0.00 Byte)** |

## 4.6 *Experimental Setup*

This section outlines the research plan to compare various sampling techniques: SMOTE, ADASYN, undersampling, and non-sampled baseline. All performance was assessed on the validation set to define the optimal sampling method. Hyperparameter tuning, operating 5-fold cross-validation, was then applied to the combined training and validation datasets with the selected sampling technique. Following optimization, a final performance comparison was conducted on the held-out test set with an assessment of computational efficiency, quantified by reference speed and training time for each optimized model.

### 4.6.1 *Comparing Sampling Techniques (RQ1)*

NON-SAMPLING Non-sampling represents the original, unaltered training dataset without balancing the class distribution. This approach serves as a baseline to understand model performance on imbalanced data. Training LGBM and MLP on non-sampled data allows us to compare how well the models handle class imbalance naturally, providing a reference for other sampling methods.

SMOTE SMOTE is a widely adopted method for addressing class imbalance by generating synthetic samples of the minority class. In this study, the SMOTE algorithm from the imbalanced-learn (Lemaître et al., 2017) library was employed to resample the training dataset. Synthetic samples were created by interpolating between existing minority class instances (Mansourifar & Shi, 2020), thereby increasing the representation of the minority class in a manner that preserves the underlying data structure.

ADASYN ADASYN, an alternative synthetic sampling technique to generate synthetic data points specifically for minority class instances that are more challenging to learn (Abdullahi et al., 2023). This method dynamically adjusts the number of samples based on the estimated difficulty level, thereby improving the classifier's ability to differentiate between classes in the presence of imbalance.

UNDERSAMPLING Undersampling decreases the number of majority class to balance the distribution (Devi et al., 2020). This technique involves randomly removing samples from the majority class, which can mitigate classifier bias towards the dominant class. However, a potential drawback is the loss of potentially valuable information inherent in the discarded samples.

Table 4 supplies a detailed overview of how each sampling technique modified the dataset, elucidating the resultant changes in class distribution and sample size.

| Sampling Techiniques | Non-fraud | Fraud | Total |
|---|---|---|---|
| Non-samping | 5,351 | 1,525 | 6,876 |
| SMOTE | 5,351 | 5,351 | 10,702 |
| ADASYN | 5,351 | 5,515 | 10,866 |
| Undersampling | 1,525 | 1,525 | 3,050 |

Table 4: Sampled data

### 4.6.2 *Model Comparison (RQ2)*

After deciding the optimal sampling method for each model (non-sampling for both LGBM and MLP here), hyperparameter tuning proceeded on the combined training and validation data.

For the LGBM model, a grid search was conducted to optimize key hyperparameters, including **n_estimators**, **learning_rate**, **num_leaves**, and **max_depth** as outlined in Appendix E. This optimization aimed to balance model complexity and learning dynamics. Using 5-fold cross-validation for robust model selection, the optimal hyperparameter configuration was identified. The model was then retrained on the combined training and validation data and assessed on the test set.

The MLP model underwent hyperparameter tuning using the Keras-Tuner to identify the optimal configuration. The training and validation datasets were concatenated and standardized, later split into five folds through K-fold cross-validation.

The tuning process focused on several key parameters, including the number of units in each hidden layer, activation functions, and learning rate defined by Appendix F. These chosen values were guided by common practices and a desire to explore a broad spectrum of configurations for each hyperparameter. The model architecture was defined as a sequential model with three hidden layers, each using hyperparameters selected from the predefined ranges and options. Conventionally, the first layer has a higher number of units to extract complex patterns and relationships within the input data, the succeeding hidden layers progressively decrease in size to learn higher-level representations from previous layers by discarding irrelevant or noisy details and are less prone to overfitting,

During each iteration, the **RandomSearch** tuner searched for the hyperparameter space, evaluating model accuracy on the validation set with the objective of maximizing this metric. The search process was constrained to a maximum of 10 trials per search and one execution per trial. The optimal hyperparameters identified in this process were then utilized to construct the final model, which was thereon trained on the combined dataset and evaluated on the held-out test set. The training process consisted of 20 epochs to ensure sufficient model learning.

### 4.6.3 *Calculating Training Time and Reference Speed (RQ3)*

To estimate the computational efficiency of both the LGBM and MLP models, average training time and inference speed were measured. For both models, training time was determined by repeatedly fitting the model with the optimal hyperparameters obtained from grid search (LGBM) and random search (MLP) over 100 iterations. Inference speed was calculated by averaging the time taken to predict the test dataset over 100 repetitions.

This analysis of training time and inference speed delivers a valuable understanding of the computational efficiency of each model, informing the practical implications of deploying these models in real-world scenarios where both speed and performance are critical considerations.

## 4.7 *Evaluation Methods*

This thesis aims to address a binary classification problem characterized by moderate class imbalance (77.82% versus 22.18%). Model evaluation for both the LGBM classifier and the MLP neural network primarily emphasized metrics relevant to the minority class, namely the F1 score, recall, and ROC-AUC score. However, accuracy, precision, and the confusion matrix were also considered to provide a thorough review of model performance.

The emphasis on class 1 (fraudulent transactions) $F_1$ score, a metric harmonizing precision, and recall, stems from the critical need in fraud detection to balance the accurate identification of fraudulent activity with minimizing false positives that disrupt legitimate users. A high $F_1$ score signifies a model's effectiveness in achieving this balance, which is crucial for the practical deployment and efficacy of fraud detection systems.

While both precision and recall are essential metrics in fraud detection, this study prioritized recall over precision. This is grounded in the understanding that recall quantifies a model's ability to specify the maximum number of fraudulent transactions, thereby minimizing the risk of undetected fraud. Although high precision is desirable to minimize false positives and their associated consequences for legitimate users, ensuring high recall is paramount for effectively detecting and preventing fraudulent activities, thus maintaining the integrity and trust of the system.

The ROC-AUC score served as a critical metric considering the discriminative power of the classification models across all thresholds. A high ROC-AUC score indicates the model's ability to distinguish between fraud and legitimate users or transactions, a crucial factor for robust fraud detection systems.

Moreover, analysis of the confusion matrix provided a nuanced understanding of model performance, especially concerning the minority (fraudulent) class, a critical aspect given the imbalanced nature of the dataset. This detailed evaluation is essential for identifying areas for model refinement and improving its accuracy in real-world fraud detection applications.

Accuracy, the ratio of correctly predicted instances to total predictions, served as the primary metric for assessing overall model performance. High accuracy is particularly desirable in fraud detection systems as it minimizes the operational burden of false alarms, thereby improving the system's efficiency and effectiveness.

Lastly, we measured the training time and inference speed to consider the efficiency of each model. These metrics are crucial, as they directly impact a model's practicality. Reduced training time facilitates more frequent model updates to adapt to growing fraud practices, while rapid inference speeds promote the timely detection and mitigation of fraudulent dealings.

# 5 RESULTS

This section reports the results of the sampling technique impact and details the selected technique based on performance metrics. It also delineated the optimal hyperparameter configurations identified per model and provided a relative analysis of their respective training times and inference speeds.

## 5.1 *Results Experiment: Sampling Techniques*

The impact of different sampling techniques on the results of both LGBM and MLP was discussed in this section and the final selection of the best ones was concluded at the end.

### 5.1.1 *Sampling Techniques for LGBM*

The effect of different sampling methods for the LGBM model included comparing overall performance metrics and their corresponding confusion matrices based on the validation set.

PERFORMANCE COMPARISON For LGBM, the performance metrics for different sampling methods on the validation set described in Table 5 summarized that the model trained on the original (non-sampled) training dataset outperformed all the other sampled datasets, achieving the highest F1-score (0.974), ROC-AUC score (0.9988), accuracy (0.988), precision (0.970) and slightly lower recall (0.976). Among the sampling techniques, the model with an undersampled dataset had a high recall (0.991), but significantly lower in precision (0.635) and F1-score (0.772), indicating that while it captured more fraudulent transactions, it also misclassified a higher number of legitimate transactions. Models with SMOTE and ADASYN datasets, while improving recall to 1 and 0.997 respectively, saw a substantial drop in precision (0.578 and 0.530) and F1-score (0.732 and 0.692), reflecting a trade-off between detecting fraud and avoiding false positives.

| LGBM | Accuracy | Precision | Recall | F1 | ROC-AUC |
|---|---|---|---|---|---|
| Non-sampling | 0.988 | 0.970 | 0.979 | 0.974 | 0.9988 |
| SMOTE | 0.838 | 0.578 | 1 | 0.732 | 0.9865 |
| ADASYN | 0.803 | 0.530 | 0.997 | 0.692 | 0.9870 |
| Undersampling | 0.872 | 0.635 | 0.991 | 0.774 | 0.9879 |

Table 5: LGBM performance comparison on the validation set with different sampling methods

CONFUSION MATRICES Analyzing the confusion matrices from Figure 9 - 12 revealed LGBM without sampling as the superior approach for this task. It boasted the highest correctly predicted diagonal values, suggesting an accurate classification of a greater proportion of data points across both classes. While there are a few false negatives, which may suggest occasional misclassification of minority class instances, the overall low number of false positives and negatives suggested a balanced performance, signifying its ability to minimize errors for both majority and minority classes without resorting to sampling techniques.

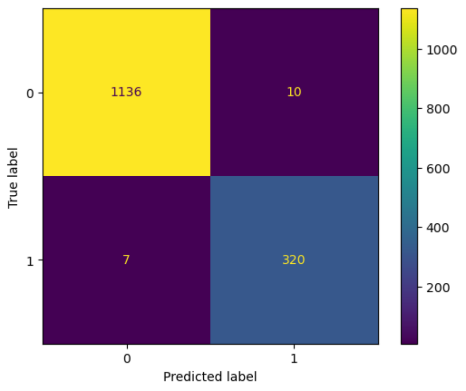

Figure 9: LGBM validation confusion matrix without sampling

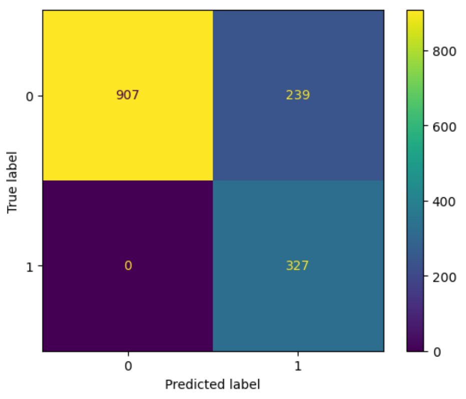

Figure 10: LGBM validation confusion matrix with SMOTE

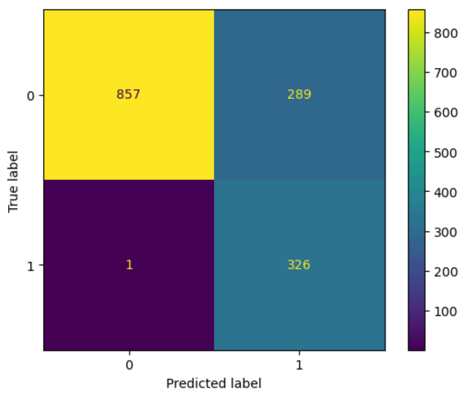

Figure 11: LGBM validation confusion matrix with ADASYN

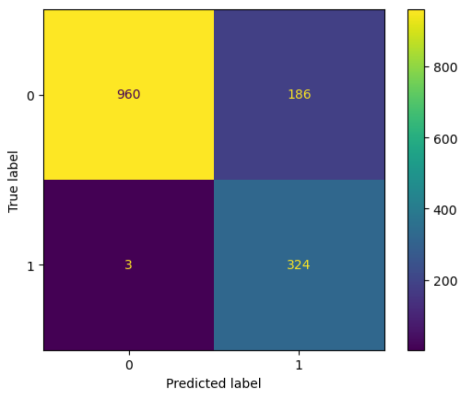

Figure 12: LGBM validation confusion matrix with Undersampling

### 5.1.2 *Sampling Technique for MLP*

A similar analysis of various sampling methods was executed regarding MLP. This included comparing overall performance, assessing loss and accuracy for each approach, and comparing their corresponding confusion matrices.

PERFORMANCE COMPARISON Similar to the LGBM model, the MLP model trained on the non-sampled dataset exhibited the best metrics on the validation set, as indicated in Table 6. It achieved the highest F1-score (0.925), accuracy (0.965), ROC-AUC score (0.9949), and precision (0.890), with a slightly lower recall (0.963). SMOTE improved recall to 0.988 and held a higher F1-score (0.887), meaning better identification of the minority fraudulent class; however, it also led to a decrease in precision, signaling an increase in false positives. Regardless, utilizing any of sampling techniques resulted in a drop in precision, indicating more false positives. ADASYN and undersampling displayed improvements in recall (0.997 and 0.985, respectively) but at the cost of lower precision (0.743 and 0.576) and F1-scores (0.851 and 0.727).

| MLP | Accuracy | Precision | Recall | F1 | ROC-AUC |
|---|---|---|---|---|---|
| Non-sampling | 0.965 | 0.890 | 0.963 | 0.925 | 0.9949 |
| SMOTE | 0.944 | 0.805 | 0.988 | 0.887 | 0.9936 |
| ADASYN | 0.923 | 0.743 | 0.997 | 0.851 | 0.9915 |
| Undersampling | 0.836 | 0.576 | 0.985 | 0.727 | 0.9876 |

Table 6: MLP performance comparison on the validation set with different sampling methods

CONFUSION MATRICES According to the results from Figures 13 - 16, models trained on the dataset with SMOTE and ADASYN were particularly adept at identifying the minority class, which came at the expense of increased false positives. Undersampling, although effective in reducing false negatives, suffered from a significant rise in false positives. Notably, MLP without sampling displayed a balanced performance between false positives and negatives without introducing artificial data manipulation, making it the preferred strategy.

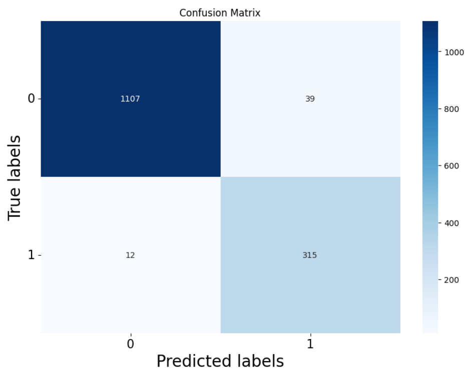

Figure 13: MLP validation confusion matrix without sampling

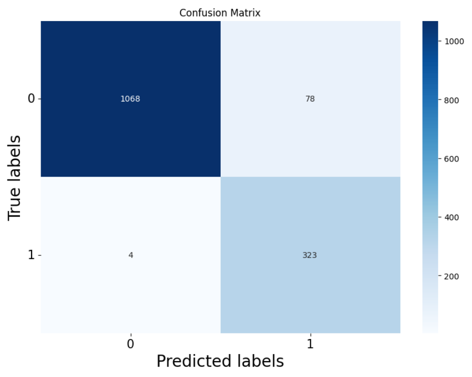

Figure 14: MLP validation confusion matrix with SMOTE

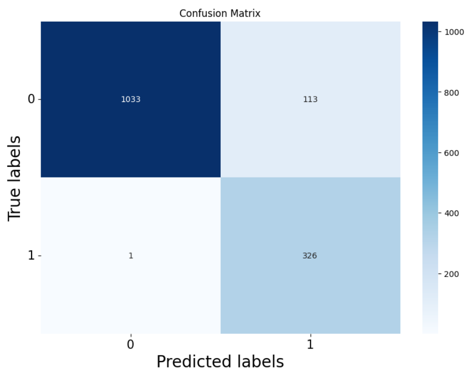

Figure 15: MLP validation confusion matrix with ADASYN

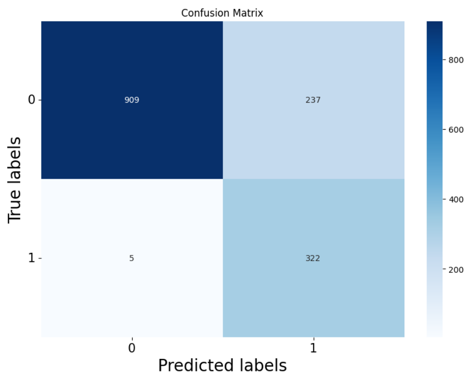

Figure 16: MLP validation confusion matrix with Undersampling

LOSS AND ACCURACY CURVES For the MLP model trained on a non-sampled training dataset, Figure 17 explained that the training loss steadily decreased, reaching a low value as epochs progressed, which indicates that the model was effectively learning from the data. The validation loss also showed a decreasing trend, although with tiny fluctuations. In contrast, Figure 18 indicated the training accuracy was high, suggesting that the model performed well on the training data. Similarly, the validation accuracy was high but slightly lower than the training accuracy during the last 10 epochs, suggesting a bit overfitting.

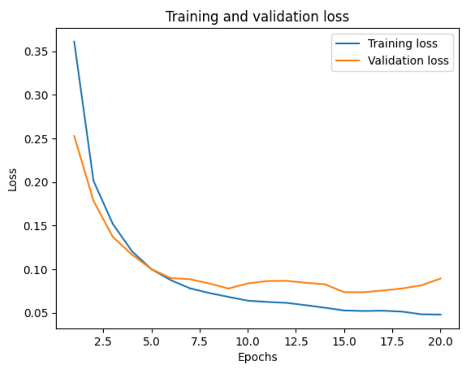

Figure 17: MLP train and validation loss Without sampling

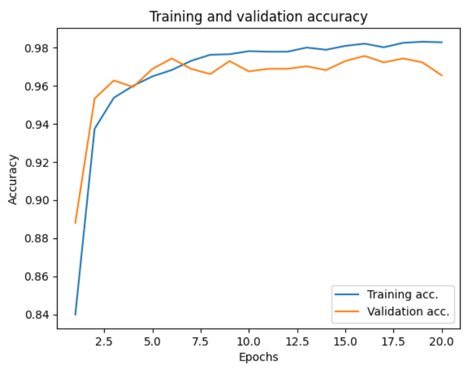

Figure 18: MLP train and validation accuracy without sampling

After resampling the dataset by involving SMOTE, Figures 19 - 20 demonstrated that the training loss decreased significantly, indicating that the model learned effectively from the augmented data. However, the validation loss fluctuated more, proposing that the model struggled to generalize on unseen data. The training accuracy remained high, leading to good performance on the training set. Nonetheless, the validation accuracy was lower than the training accuracy, indicating overfitting and poor generalization.

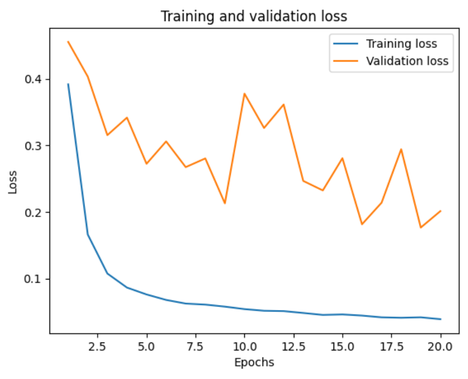

Figure 19: MLP train and validation loss with SMOT

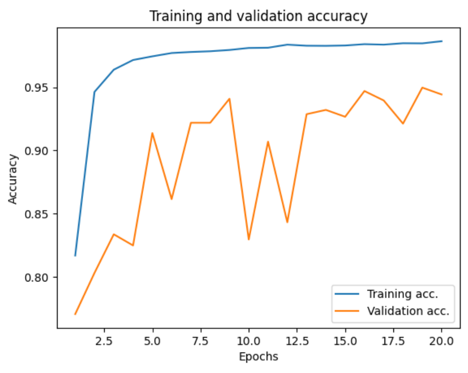

Figure 20: MLP train and validation accuracy with SMOTE

According to Figures 21 - 22, the model trained after applying ADASYN revealed a decrease in training loss, though the decline was less smooth compared to the model trained with SMOTE. The validation loss exhibited significant fluctuations, indicating instability in the learning process. Despite achieving high training accuracy, the validation accuracy was significantly lower, suggesting severe overfitting. This illustrated that while the model performed well on the training data, it struggled to maintain the same level of performance on the validation set.

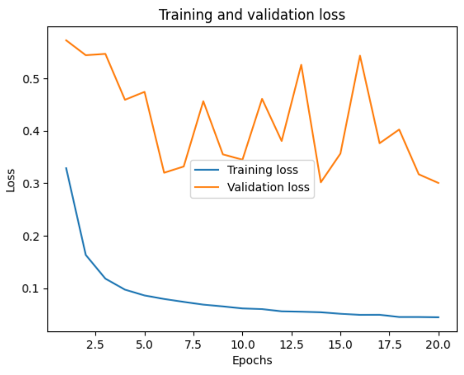

Figure 21: MLP train and validation loss with ADASYN

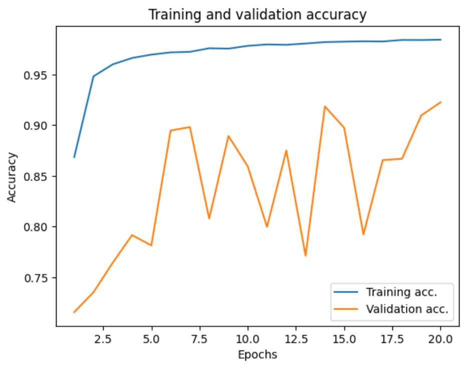

Figure 22: MLP train and validation accuracy with ADASYN

After undersampling the majority class, Figures 23 - 24 suggested the training loss decreased steadily, similar to the baseline model, indicating effective learning from the reduced dataset. However, the validation loss remained relatively high and fluctuated more, suggesting the difficulty of generalizing to unseen data. The model achieved high training accuracy, but the validation accuracy was much lower, indicating overfitting and poor performance on the validation set. This confirmed that the model performed well on the training data but was less effective in generalizing on new data.

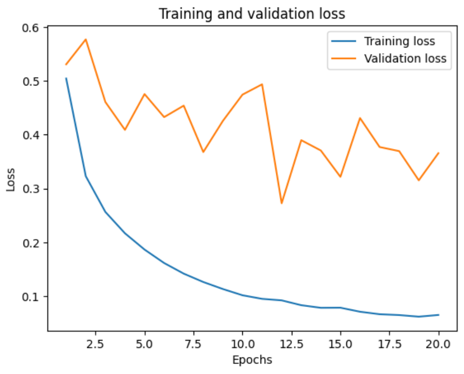

Figure 23: MLP train and validation loss with Undersampling

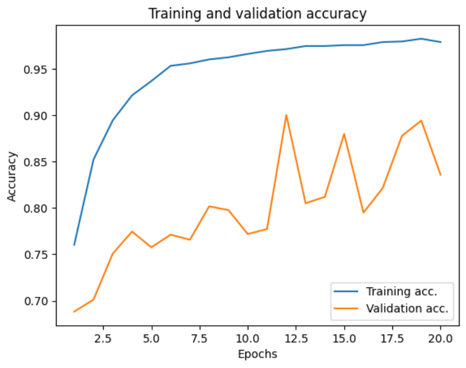

Figure 24: MLP train and validation accuracy with Undersampling

### 5.1.3 *Conclusion for Sampling Techniques*

Both LGBM and MLP models achieved optimal performance when trained on the non-sampled dataset, indicating a superior balance between precision and recall compared to models trained with sampling methods. The application of SMOTE and ADASYN resulted in the introduction of noise through the generation of synthetic samples, potentially leading to overfitting. Undersampling, while addressing class imbalance, led to the loss of information from the majority class, and potentially failed to capture the full complexity of the data (Fujiwara et al., 2020).

## 5.2 *Results Experiment: Model Comparison*

### 5.2.1 *Optimization of the Models*

Following the decision of the optimal sampling strategy for each model, hyperparameter optimization was ushered to improve model performance. A grid search approach was employed for the LGBM model, while a random search was utilized for the MLP model. This optimization process was facilitated by combining the original training and validation datasets, securing a thorough evaluation during 5-fold cross-validation.

HYPERPARAMETER VALUES Following the decision of the optimal sampling strategy for each model, hyperparameter optimization was ushered to improve model performance. A grid search approach was employed for the LGBM model, while a random search was utilized for the MLP model. This optimization process was facilitated by combining the original training and validation datasets, securing a thorough evaluation during 5-fold cross-validation. The optimal hyperparameter configurations for each model are presented in Appendix G, respectively.

ERROR ANALYSIS AND TRAINING/TEST COMPARISON To evaluate model implementation and diagnose potential overfitting or underfitting, a comparative analysis of training and test performance was executed for both the MLP and LGBM models. This evaluation utilized a combined training dataset incorporating the original training and validation sets.

**LGBM** achieved perfect training accuracy (0 false positives/negatives) according to Figure 25, suggesting potential overfitting. A slight performance drop on the test set (2 false positives, 18 false negatives) reinforced this concern as Figure 32 portrayed. This discrepancy indicated the model may be capturing data noise or specific patterns that don't generalize well.

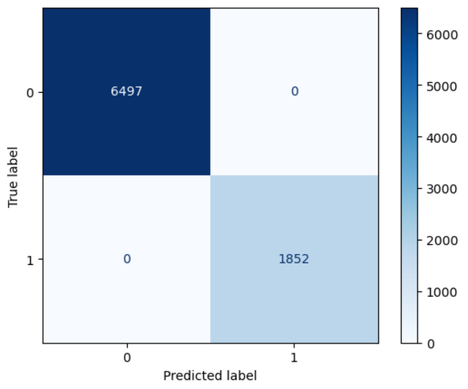

Figure 25: LGBM train confusion matrix

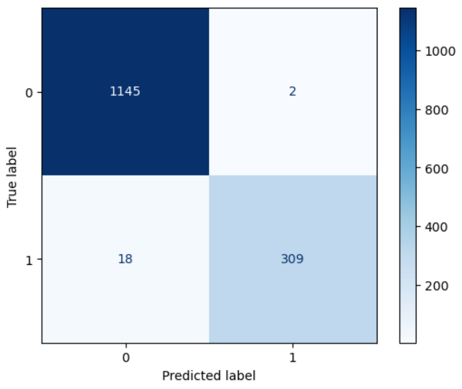

Figure 26: LGBM test confusion matrix

**MLP** model attained high accuracy with 6,429 true negatives and 1,807 true positives, along with 68 false positives and 45 false negatives on the training set as Figure 27 presented. Figure 28 illustrated it retained strong performance with 1,129 true negatives, 303 true positives, 18 false positives, and 24 false negatives on the test set. Though the low counts of misclassifications on both datasets indicated good generalization, a slight increase in errors from training to test data suggested potential overfitting.

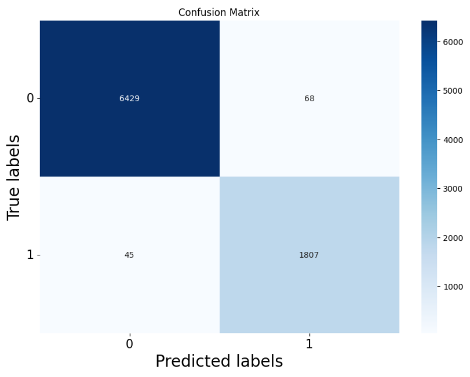

Figure 27: MLP train confusion matrix

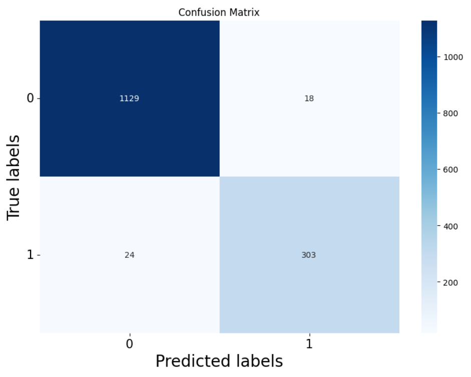

Figure 28: MLP test confusion matrix

Figures 29 and 30 depicted the MLP model's training process across 20 epochs. While training loss steadily decreased and accuracy reached a high and stable level, test loss and accuracy painted a different picture. Test loss initially plateaued but then exhibited a slight upward trend and test accuracy showed minor fluctuations with a downward trajectory. These trends suggested the model might be overfitting to the training data.

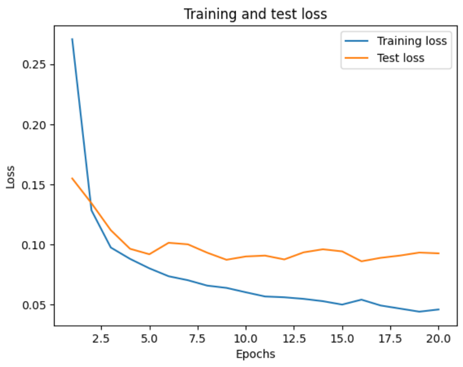

Figure 29: MLP train and test loss

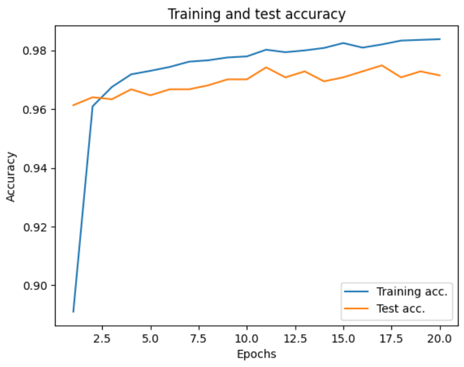

Figure 30: MLP train and test accuracy

### 5.2.2 *Final Model Comparison*

Evaluation of the held-out test set informed the superior performance of the LGBM model in the fraud detection task. The LGBM model consistently outperformed the MLP model across all key metrics, achieving higher accuracy, precision, F1-score, and ROC-AUC score, as detailed in Table 7. This statistically significant difference suggests that the LGBM model displays a superior ability to accurately identify fraudulent transactions while minimizing false positives, a critical factor in real-world fraud detection applications. The results provide compelling evidence for the suitability of the LGBM model in addressing the challenges posed by fraud detection within the context of this study.

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1** | **ROC-AUC** |
|---|---|---|---|---|---|
| LGBM | 0.986 | 0.994 | 0.945 | 0.969 | 0.9987 |
| MLP | 0.972 | 0.944 | 0.927 | 0.935 | 0.9880 |

Table 7: Performance comparison for fine-tuned models

## 5.3 *Results Experiment: Training Time and Inference Speed*

To fairly assess how quickly each model can train and make predictions in real-world scenarios, we averaged the time over 100 iterations, it's been proven that the LGBM model exhibited markedly superior efficiency, with a training time of 0.6662 seconds and an inference speed of 0.0142 seconds. In contrast, the MLP model required a considerably longer training time of 34.32 seconds and a substantially slower inference speed of 34.32 seconds. These findings underscore the computational advantages of the LGBM model for real-time fraud detection applications.

# 6 DISCUSSION

The primary goal of this study was to assess the effectiveness of LGBM and MLP models in detecting fraudulent activities within the realm of Ethereum (ETH) transactions. Several challenges were encountered, including determining which sampling techniques to use or whether to employ any sampling techniques at all to tackle class imbalance issues. Additionally, both LGBM and MLP models can automatically select features to learn from, which posed a difficulty in ensuring that the selected features were sufficient for learning while avoiding redundancy.

## 6.1 *Results Discussion and Comparison with Prior Work*

### 6.1.1 *What techniques are most effective for each model in handling class imbalance?*

The result of the first experiment indicated that the resampled datasets did not surpass the non-sampled data, suggesting that the LGBM and MLP models were already performing well enough without sampling. Several factors may contribute to the superior performance of non-sampling. Both LGBM and MLP models, in their native configurations, possess inherent mechanisms for handling imbalanced data, such as specialized loss functions and algorithms designed to mitigate the impact of class disparities. Also, the original data distribution helps avoid potential biases introduced by synthetic sampling techniques like SMOTE or ADASYN (Alex et al., 2024), which may not accurately reflect the true underlying distribution of the minority class. Thus, the model with non-sampled data provides a more reliable and generalizable evaluation in real-world scenarios.

In contrast to prior studies that applied specific sampling techniques without a comprehensive evaluation (SMOTE by Aziz et al. (2022); undersampling by Bartoletti et al. (2018)), this study systematically examined the impact of various sampling methods. The rigorous assessment, particularly on the validation set, provides nuanced insights into the effectiveness of each technique in addressing the class imbalance, highlighting the critical need for a deliberate and informed approach to sampling technique selection, a previously overlooked aspect.

### 6.1.2 *Which model demonstrates better performance?*

The second experiment revealed the LGBM model to be superior to the MLP model in accurately classifying fraudulent accounts, as evidenced by its higher overall performance metrics following optimization. This eminent performance is likely due to several characteristics inherent to the

LGBM algorithm. Notably, its ability to effectively handle class imbalance, its efficient feature selection mechanisms, and its reduced reliance on extensive hyperparameter tuning contributes to its greater practicality in fraud detection compared to the MLP model. This outcome aligns with the research conducted by Aziz et al. (2022), which compared various machine learning models and concluded that LGBM is incredibly practical.

Aziz et al. (2022) achieved the highest accuracy among all studies mentioned in 3. Their performance metrics demonstrate superior results compared to those of this study as illustrated in Table 8 and Figures 31 - 32. However, this exceptional performance was based on the blind choice of the SMOTE sampling method without revealing its impact. Moreover, the lack of transparency regarding their hyperparameter tuning process limits the reference value of their findings. In contrast, this study transparently documented the hyperparameter tuning process, ensuring the model optimization steps were clear and reproducible.

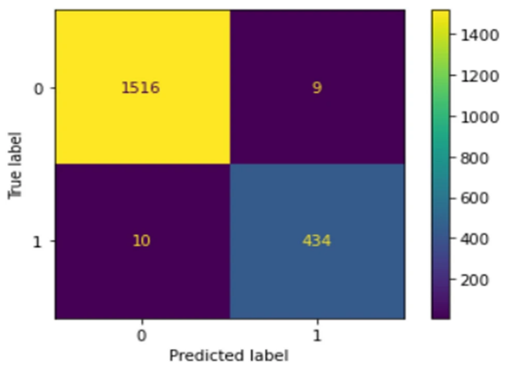

Figure 31: LGBM confusion matrix from Aziz et al. (2022)

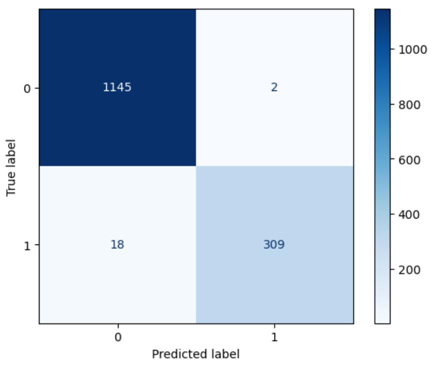

Figure 32: LGBM confusion matrix from this study

| | **Accuracy** | **Precision** | **Recall** | **F1** |
|---|---|---|---|---|
| Aziz et al. (2022) | 0.9903 | 0.9797 | 0.9775 | 0.9786 |
| This study | 0.986 | 0.994 | 0.945 | 0.969 |

Table 8: LGBM comparison between Aziz et al. (2022) and this study

### 6.1.3 *How do training time and inference speed compare between MLP and LGBM?*

The final experiment demonstrated that the LGBM model exhibited significantly superior computational efficiency compared to the MLP model, as evidenced by its significantly faster training time and inference speed. This efficiency positions the LGBM model as a particularly advantageous choice for applications where frequent model retraining or constrained computational resources are a concern. This distinction may result from fundamental differences in the algorithmic design and implementation. The histogram-based approach and leaf-wise growth strategy of LGBM, coupled with its ability for parallel processing and efficient memory management, enable faster training and inference in contrast to MLP's reliance on computationally intensive backpropagation. Furthermore, LGBM's inherent regularization strategies contribute to faster convergence and less susceptibility to overfitting. Conversely, MLPs, particularly with deeper architectures, exhibit higher computational and memory demands due to their greater complexity and iterative training process. These findings underscore the importance of selecting appropriate models based on available computational resources and desired speed, especially in real-time applications.

While Ibrahim et al. (2021) gained impressive reductions in processing times for traditional machine learning algorithms, notably with KNN reaching near-instantaneous (0 seconds) processing on a limited feature set of 6, focusing solely on training time, our research stresses the broader computational efficiency and practical applicability of the LGBM model by considering both training and inference speed on a larger dataset (31 features). The LGBM model's combination of fast training and inference speeds, coupled with its strong predictive performance and ability to handle a larger number of features, makes it a more compelling choice for applications where both speed and accuracy are critical, especially those with higher dimensional data. This is particularly relevant in scenarios requiring frequent model updates or facing resource constraints, solidifying LGBM's position as an effective and efficient solution for complex real-world problems such as fraud detection.

## 6.2 *Broader Societal Relevance*

By demonstrating the effectiveness of non-sampled data in certain scenarios, the investigation promotes model transparency and reliability, thereby enhancing trust in fraud detection systems. Furthermore, the findings emphasize the importance of an intentional and context-specific procedure for sampling technique selection, which can be generalized to other domains involving transaction pattern analysis, including various blockchains and the broader crypto industry. This rigorous approach not only mitigates potential biases but also contributes to the development of ethical AI systems in the financial sector.

The model comparison analysis presented in this study carries important societal implications. By identifying effective models and highlighting potential pitfalls, this study paves the way for future research aimed at developing more practical, scalable, and adaptable algorithms to mitigate financial losses and protect users within blockchain and crypto ecosystems. This contributes to establishing a foundation for responsible innovation and informed policy-making in the application of AI technologies.

Finally, the assessment of training time and inference speed is critical for real-time fraud detection systems given the high volume of daily Ethereum transactions (approximately 1.022 million) and new unique addresses (approximately 147,758), as reported by ychart and supported by Foundation. This stresses the importance of selecting a model, such as LGBM, capable of rapid fraud detection to effectively keep pace with the dynamic and high-volume nature of the Ethereum network.

## 6.3 *Limitations*

A limitation of this study lies in the utilization of a pre-cleaned dataset provided by a single source on Kaggle. Upon examination, 829 instances were found to share the same 25 missing feature values, raising concerns about the dataset's representativeness of real-world scenarios. Such a skewed or incomplete representation of the underlying data distribution could potentially lead to misleading conclusions regarding the sampling strategies, bias model performance, undermine the evaluation of model efficiency, and limit the generalizability of the findings.

Another notable restriction is that the superior performance of the LGBM model over the MLP model in this study, while evident, is difficult to definitively explain due to the inherent opacity of MLPs. Their "black box" nature inhibits interpretability, making it challenging to specify the exact factors contributing to their suboptimal performance. This lack of transparency obscures how individual features influence predictions and the model's overall decision-making process, potentially masking issues such as overfitting or biases. This underscores the importance of considering both accuracy and explainability when developing fraud detection systems, particularly in high-stakes domains where understanding the rationale behind model decisions is crucial.

## 6.4 *Future Work*

Firstly, future research endeavors should prioritize the incorporation of more comprehensive and diverse datasets, encompassing data from various sources such as third-party crypto data aggregator platforms (Dune, Nansen and Arkham) and Ethereum's native data website (Etherscan), where all transactions are recorded and addresses are marked as fraudulent if they have been associated with any fraudulent activity. This approach, coupled with manual inspection for missing values, is essential for developing more robust and resilient models.

Secondly, while the LGBM and MLP models with default hyperparameters exhibited the best performance on the validation set regarding the selection of sample methods, these parameter values were thereafter adjusted through hyperparameter tuning on the combined training and validation data. However, a direct comparison of model performance before and after hyperparameter tuning was not conducted on this combined dataset. Consequently, the potential performance gains achieved through hyperparameter tuning, relative to the initial models trained on the training set alone, remain unquantified. Future work should include a rigorous assessment of the impact of hyperparameter tuning on model performance to ascertain its efficacy in this context.

Thirdly, recognizing the capacity of LGBM and MLP models to process extensive datasets with intricate features, the significance of advanced feature engineering is underscored. Future work should prioritize the expansion of novel features that can effectively capture nuanced patterns and trends indicative of fraudulent activity. This may involve leveraging automated feature selection techniques to identify relevant features, as well as comprising domain-specific expertise to generate features that are particularly salient for fraud detection.

Lastly, it's essential to investigate advanced methods for mitigating the potential overfitting of sampling techniques, especially in the context of MLP model training. Promising approaches comprise the implementation of early stopping, regularization techniques, and cross-validation. Besides, a more extensive exploration of alternative model architectures and hyperparameter configurations could aid in identifying optimal solutions that effectively balance bias and variance, thereby enhancing model generalizability and overall performance.

# 7 CONCLUSION

This study compared machine learning (LGBM) and deep learning (MLP) models for fraud detection on the Ethereum blockchain. Results indicated that the LGBM model outperformed the MLP model across key performance metrics (F1-score: 0.969, accuracy: 0.986, precision: 0.994) with faster training (0.6662 seconds) and inference (0.0142 seconds) times. While these results were lower than those (F1-score: 0.9786, accuracy: 0.9903, precision: 0.9797) reported by Aziz et al. (2022), our study prioritized methodological rigor regarding sampling technique selection and transparency in hyperparameter tuning. Additionally, the approach carried out in this research favored a wider feature set over the reduced processing time (nearly 0 seconds) reported by Ibrahim et al. (2021) with limited features. Future research should explore the generalizability of the LGBM model to other blockchain platforms, examine alternative data preprocessing techniques to potentially improve performance, and incorporate diverse datasets to enhance the model's robustness and applicability.

## REFERENCES

Abdolrasol, M. G., Hussain, S. S., Ustun, T. S., Sarker, M. R., Hannan, M. A., Mohamed, R., Ali, J. A., Mekhilef, S., & Milad, A. (2021). Artificial neural networks based optimization techniques: A review. *Electronics*, *10*(21), 2689.

Abdullahi, H., Bashir, S. A., & Aminu, E. F. (2023). An improved adaptive synthetic sampling technique and machine learning model for enhanced imbalance medical data classification.

Alex, S. A., Nayahi, J. J. V., & Kaddoura, S. (2024). Deep convolutional neural networks with genetic algorithm-based synthetic minority over-sampling technique for improved imbalanced data classification. *Applied Soft Computing*, *156*, 111491.

Arkham. (2024). Powerful tools for linking cryptocurrency activity to real world individuals and institutions [Accessed: 118-06-2024].

Aswin, A., & Kuriakose, B. (2020). An analogical study of hyperledger fabric and ethereum. *Intelligent Communication Technologies and Virtual Mobile Networks: ICICV 2019*, 412–420.

Aziz, R. M., Baluch, M. F., Patel, S., & Ganie, A. H. (2022). Lgbm: A machine learning approach for ethereum fraud detection. *International Journal of Information Technology*, *14*(7), 3321–3331.

Bartoletti, M., Pes, B., & Serusi, S. (2018). Data mining for detecting bitcoin ponzi schemes. *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 75–84. https://doi.org/10.1109/CVCBT.2018.00014

Boeschoten, S., Catal, C., Tekinerdogan, B., Lommen, A., & Blokland, M. (2023). The automation of the development of classification models and improvement of model quality using feature engineering techniques. *Expert Systems with Applications*, *213*, 118912.

Brooks, C. (2019). *Introductory econometrics for finance*. Cambridge university press.

Chen, L., Peng, J., Liu, Y., Li, J., Xie, F., & Zheng, Z. (2020). Phishing scams detection in ethereum transaction network. *ACM Transactions on Internet Technology (TOIT)*, *21*(1), 1–16.

Chollet, F., et al. (2015). *Keras*. https://github.com/fchollet/keras

Classic, E. (2024). A blockchain-based distributed computing platform [Accessed: 27-05-2024].

CoinGecko. (2024). Cryptocurrency market cap platform [Accessed: 26-02-2024].

Devi, D., Biswas, S. K., & Purkayastha, B. (2020). A review on solution to class imbalance problem: Undersampling approaches. *2020 international conference on computational performance evaluation (ComPE)*, 626–631.

Dinesh Sivakumar. (2024). Introduction to neural networks and deep learning [Accessed: 11-06-2024].

Dozmorov, M. G., Adrianto, I., Giles, C. B., Glass, E., Glenn, S. B., Montgomery, C., Sivils, K. L., Olson, L. E., Iwayama, T., Freeman, W. M., et al. (2015). Detrimental effects of duplicate reads and low complexity regions on rna-and chip-seq data. *BMC bioinformatics*, *16*, 1–11.

Dune. (2024). Crypto data platform [Accessed: 17-06-2024].

EtherscamDB. (2024). Etherscam data base.

Etherscan. (2024). Etherscan [https://etherscan.io/apis [Accessed: (26-02-2024)]].

Farrugia, S., Ellul, J., & Azzopardi, G. (2020). Detection of illicit accounts over the ethereum blockchain. *Expert Systems with Applications*, *150*, 113318.

Foundation, E. (2023). Ethereum company [Accessed: 17-05-2024].

Fried, E. I. (2020). Lack of theory building and testing impedes progress in the factor and network literature. *Psychological Inquiry*, *31*(4), 271–288.

Fujiwara, K., Huang, Y., Hori, K., Nishioji, K., Kobayashi, M., Kamaguchi, M., & Kano, M. (2020). Over-and under-sampling approach for extremely imbalanced and small minority data problem in health record analysis. *Frontiers in public health*, *8*, 178.

Géron, A. (2022). *Hands-on machine learning with scikit-learn, keras, and tensorflow*. " O'Reilly Media, Inc.".

Haji, S. H., & Abdulazeez, A. M. (2021). Comparison of optimization techniques based on gradient descent algorithm: A review. *PalArch's Journal of Archaeology of Egypt/Egyptology*, *18*(4), 2715–2743.

Hu, T., Liu, X., Chen, T., Zhang, X., Huang, X., Niu, W., Lu, J., Zhou, K., & Liu, Y. (2021). Transaction-based classification and detection approach for ethereum smart contract. *Information Processing & Management*, *58*(2), 102462.

Ibrahim, R. F., Elian, A. M., & Ababneh, M. (2021). Illicit account detection in the ethereum blockchain using machine learning. *2021 international conference on information technology (ICIT)*, 488–493.

Jin, H., & Xiao, J. (2022). Towards trustworthy blockchain systems in the era of “internet of value”: Development, challenges, and future trends. *Science China Information Sciences*, *65*, 1–11.

Johnson, K. N. (2020). Regulating cryptocurrency secondary market trading platforms. *U. Chi. L. Rev. Online*, 26.

Kaggle. (2021).

Kyriazos, T., & Poga, M. (2023). Dealing with multicollinearity in factor analysis: The problem, detections, and solutions. *Open Journal of Statistics*, *13*(3), 404–424.

Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning [Accessed: 26-05-2024]. https://imbalanced-learn.org/

Leys, C., Delacre, M., Mora, Y. L., Lakens, D., & Ley, C. (2019). How to classify, detect, and manage univariate and multivariate outliers, with emphasis on pre-registration. *International Review of Social Psychology*, *32*(1).

Little, R. J., & Rubin, D. B. (2019). *Statistical analysis with missing data* (Vol. 793). John Wiley & Sons.

Mansourifar, H., & Shi, W. (2020). Deep synthetic minority over-sampling technique. *arXiv preprint arXiv:2003.09788*.

Monamo, P., Marivate, V., & Twala, B. (2016). Unsupervised learning for robust bitcoin fraud detection. *2016 Information Security for South Africa (ISSA)*, 129–134. https://doi.org/10.1109/ISSA.2016.7802939

Montgomery, T. M., Lehmann, K. D., Gregg, S., Keyser, K., McTigue, L. E., Beehner, J. C., & Holekamp, K. E. (2023). Determinants of hyena participation in risky collective action. *Proceedings of the Royal Society B*, *290*(2011), 20231390.

Nansen. (2024). Onchain data platform trusted by the best [Accessed: 12-06-2024].

Nauman, F., & Herschel, M. (2022). *An introduction to duplicate detection*. Springer Nature.

Oliva, G. A., Hassan, A. E., & Jiang, Z. M. (2020). An exploratory study of smart contracts in the ethereum blockchain platform. *Empirical Software Engineering*, *25*, 1864–1904.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2007). Scikit-learn: Machine learning in python [Accessed: 27-05-2024]. https://scikit-learn.org/

Pham, T., & Lee, S. (2016). Anomaly detection in bitcoin network using unsupervised learning methods. *arXiv preprint arXiv:1611.03941*.

SlowMist. (2024). Cryptocurrency audit company [Accessed: 01-06-2024].

Wang, J., Liu, Y., & Levy, C. (2021). Fair classification with group-dependent label noise. *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, 526–536.

Wongvorachan, T., He, S., & Bulut, O. (2023). A comparison of undersampling, oversampling, and smote methods for dealing with imbalanced classification in educational data mining. *Information*, *14*(1), 54.

Xiao, D., Li, B., Mao, Y., et al. (2017). A multiple hidden layers extreme learning machine method and its application. *Mathematical Problems in Engineering*, *2017*.

Xiong, R., & Pelger, M. (2023). Large dimensional latent factor modeling with missing observations and applications to causal inference. *Journal of Econometrics*, *233*(1), 271–301.

ychart. (2024). Ethereum transactions and addresses per day [Accessed: 17-05-2024].

Zhao, L., & Akoglu, L. (2023). On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights. *Big Data*, *11*(3), 151–180.

Zhu, J., Ge, Z., Song, Z., & Gao, F. (2018). Review and big data perspectives on robust data mining approaches for industrial process modeling with outliers and missing data. *Annual Reviews in Control*, *46*, 107–133.

## A APPENDIX

| Library | Version |
|---|---|
| pandas | 2.0.3 |
| numpy | 1.24.3 |
| seaborn | 0.12.2 |
| scikit-learn | 1.4.1.post1 |
| imbalanced-learn | 0.12.0 |
| lightgbm | 4.3.0 |
| matplotlib | 3.7.2 |
| tensorflow | 2.15.0 |
| keras | 2.15.0 |
| keras-tuner | 1.4.7 |

Table A9: Libraries and their versions

## B APPENDIX

Here are more details about the loss and the number of security events that happened on other blockchains:

- **Arbitrum**: Experienced 18 security events resulting in a total loss of $32.91 million.
- **Avalanche**: Had 4 security events with losses amounting to $14.15 million.
- **Base**: Reported 9 security events with $34.38 million in losses.
- **Binance Smart Chain (BSC)**: Suffered 101 security incidents leading to $54.16 million in losses.
- **Ethereum (ETH)**: Recorded the highest losses at $487.8 million from 118 security events, making it the most affected blockchain.
- **Polygon**: Faced 6 security incidents resulting in $123.13 million in losses.
- **Other Blockchains**: Included in the category of "Others" with various smaller incidents and losses totaling $2.39 million.

## C APPENDIX

This section offers a detailed explanation of each feature (variable) in dataset and its corresponding meaning or interpretation in the context of Ethereum transactions.

- **Index**: The index number of a row
- **Address**: The address of the Ethereum account
- **FLAG**: Whether the transaction is fraud or not
- **Avg min between sent tnx**: Average time between sent transactions for account in minutes
- **Avg min between received tnx**: Average time between received transactions for account in minutes
- **Time Diff between first and last (Mins)**: Time difference between the first and last transaction
- **Sent tnx**: Total number of sent normal transactions
- **Received tnx**: Total number of received normal transactions
- **Number of Created Contracts**: Total number of created contract transactions
- **Unique Received From Addresses**: Total unique addresses from which account received transaction
- **Unique Sent To Addresses**: Total unique addresses from which account sent transactions
- **min value received**: Minimum value in Ether ever received
- **max value received**: Maximum value in Ether ever received
- **avg val received**: Average value in Ether ever received
- **min val sent**: Minimum value of Ether ever sent
- **max val sent**: Maximum value of Ether ever sent
- **avg val sent**: Average value of Ether ever sent
- **min value sent to contract**: Minimum value of Ether sent to a contract
- **max val sent to contract**: Maximum value of Ether sent to a contract

- **avg value sent to contract**: Average value of Ether sent to contracts
- **total transactions (including tnx to create contract)**: Total number of transactions
- **total Ether sent**: Total Ether sent for account address
- **total ether received**: Total Ether received for account address
- **total ether sent contracts**: Total Ether sent to contract addresses
- **total ether balance**: Total Ether balance following enacted transactions
- **Total ERC20 tnxs**: Total number of ERC20 token transfer transactions
- **ERC20 total Ether received**: Total ERC20 token received transactions in Ether
- **ERC20 total ether sent**: Total ERC20 token sent transactions in Ether
- **ERC20 total Ether sent contract**: Total ERC20 token transfer to other contracts in Ether
- **ERC20 uniq sent addr**: Number of ERC20 token transactions sent to unique account addresses
- **ERC20 uniq rec addr**: Number of ERC20 token transactions received from unique addresses
- **ERC20 uniq sent addr.1**: Number of ERC20 token transactions sent to unique addresses (alternative count)
- **ERC20 uniq rec contract addr**: Number of ERC20 token transactions received from unique contract addresses
- **ERC20 avg time between sent tnx**: Average time between ERC20 token sent transactions in minutes
- **ERC20 avg time between rec tnx**: Average time between ERC20 token received transactions in minutes
- **ERC20 avg time between rec 2 tnx**: Average time between ERC20 token received transactions (alternative count)
- **ERC20 avg time between contract tnx**: Average time between ERC20 token transactions to contracts
- **ERC20 min val rec**: Minimum value in Ether received from ERC20 token transactions for account

- **ERC20 max val rec**: Maximum value in Ether received from ERC20 token transactions for account
- **ERC20 avg val rec**: Average value in Ether received from ERC20 token transactions for account
- **ERC20 min val sent**: Minimum value in Ether sent from ERC20 token transactions for account
- **ERC20 max val sent**: Maximum value in Ether sent from ERC20 token transactions for account
- **ERC20 avg val sent**: Average value in Ether sent from ERC20 token transactions for account
- **ERC20 min val sent contract**: Minimum value in Ether sent to contracts from ERC20 token transactions
- **ERC20 max val sent contract**: Maximum value in Ether sent to contracts from ERC20 token transactions
- **ERC20 avg val sent contract**: Average value in Ether sent to contracts from ERC20 token transactions
- **ERC20 uniq sent token name**: Number of unique ERC20 tokens transferred
- **ERC20 uniq rec token name**: Number of unique ERC20 tokens received
- **ERC20 most sent token type**: Most sent token for account via ERC20 transaction
- **ERC20_most_rec_token_type**: Most received token for account via ERC20 transactions

## D APPENDIX

| **Feature** | **Missing Data (%)** |
|---|---:|
| ERC20 most sent token type | 27.406 |
| ERC20_most_rec_token_type | 8.851 |
| ERC20 min val rec | 8.424 |
| ERC20 total Ether sent contract | 8.424 |
| ERC20 uniq sent addr | 8.424 |
| ERC20 uniq rec addr | 8.424 |
| ERC20 uniq sent addr.1 | 8.424 |
| ERC20 uniq rec contract addr | 8.424 |
| ERC20 avg time between sent tnx | 8.424 |
| ERC20 avg time between rec tnx | 8.424 |
| ERC20 avg time between rec 2 tnx | 8.424 |
| ERC20 avg time between contract tnx | 8.424 |
| ERC20 max val rec | 8.424 |
| ERC20 total Ether received | 8.424 |
| ERC20 avg val rec | 8.424 |
| ERC20 min val sent | 8.424 |
| ERC20 max val sent | 8.424 |
| ERC20 avg val sent | 8.424 |
| ERC20 min val sent contract | 8.424 |
| ERC20 max val sent contract | 8.424 |
| ERC20 avg val sent contract | 8.424 |
| ERC20 uniq sent token name | 8.424 |
| ERC20 uniq rec token name | 8.424 |
| ERC20 total ether sent | 8.424 |
| Total ERC20 tnxs | 8.424 |
| Address | 0.000 |
| max value received | 0.000 |
| FLAG | 0.000 |
| Others | 0.000 |

Table D10: Proportion of missing data for each feature

## E APPENDIX

| Parameter | Value 1 | Value 2 | Value3 |
|---|---|---|---|
| n_estimators | 100 | 1000 | 1500 |
| learning_rate | 0.1 | 0.05 | 1 |
| num_leaves | 31 | 101 | 151 |
| max_depth | -1 | 35 | 55 |

Table E11: LGBM hyperparameter values

## F APPENDIX

| Parameters | Range | Step Size | Default |
|---|---|---|---|
| 1st_unit | 32 - 512 | 32 | 38 |
| act_func | relu, tanh, sigmoid | | relu |
| 2nd_unit | 16 - 256 | 16 | 15 |
| act_func | relu, tanh, sigmoid | | relu |
| 3rd_unit | 8 - 128 | 8 | 5 |
| act_func | relu, tanh, sigmoid | | relu |

Table F12: MLP hyperparameters for three hidden layers

## G APPENDIX

| Parameter | Value |
|---|---|
| learning_rate | 0.1 |
| max_depth | -1 |
| n_estimators | 1500 |
| num_leaves | 31 |

Table G13: LGBM hyperparameter values

| Parameter | Value |
|---|---|
| 1st_unit | 416 |
| act_func | relu |
| 2nd_unit | 80 |
| act_func | sigmoid |
| 3rd_unit | 112 |
| act_func | sigmoid |
| learning_rate | 0.000912 |

Table G14: MLP hyperparameter values