ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

MACHINE LEARNING TECHNIQUES FOR FRAUD DETECTION IN FINANCIAL TRANSACTIONS

MACHINE LEARNING AND CONTENT ANALYTICS (FT)

DATE: 29.09.2024

Full Name	Academic ID	Academic Mail
Vrouvaki Evrydiki	f2822302	evr.vrouvaki@aueb.gr
Konstantinos Giannakos	f2822303	kon.giannakos@aueb.gr

KEYWORDS

Anti-Money Laundering (AML), Money Laundering Detection, Financial Transaction Data, Graph Neural Networks (GNN), Graph Convolutional Network (GCN), Graph Attention Network (GAT), Graph Isomorphism Network (GIN), Class Imbalance, Synthetic Minority Over-sampling Technique (SMOTE), Focal Loss, Binary Cross Entropy (BCE), Model Evaluation, Fraud Detection

CONTENTS

LIST OF FIGURES

INTRODUCTION

In recent years, detecting money laundering activities has emerged as a priority for financial institutions and regulatory authorities worldwide. The challenge lies in distinguishing between legitimate financial transactions and those associated with illicit activities, such as money laundering. Traditional detection methods often suffer from high false positive and false negative rates, making it difficult to accurately identify fraudulent activities. This project leverages advanced machine learning techniques, including Graph Neural Networks (GNNs), to address this problem by improving the detection of money laundering within financial transaction data.

The primary goal of the project is to explore how basic Machine Learning methods and Graph Neural Network (GNN) methods can be applied to Anti-Money Laundering (AML) detection, focusing on the strengths and limitations of these models when faced with imbalanced data. Using a synthetic dataset provided by IBM, the project simulates a financial environment where both legitimate and illicit activities are present. The purpose of this work is trying to solve the AML detection problem entirely but to assess how well foundational GNN techniques can analyze transaction patterns and detect instances of money laundering, laying the groundwork for more advanced methods in future research.

The project follows a structured approach, beginning with data collection and preprocessing, followed by model selection, training, and evaluation. Various machine learning models, including GNNs such as Graph Convolutional Networks (GCN), Graph Attention Network (GAT), and Graph Isomorphism Network (GIN), are explored to assess their effectiveness in detecting laundering transactions. Additionally, techniques like Synthetic Minority Oversampling Technique (SMOTE) are implemented to address the class imbalance inherent in the dataset, further enhancing the model's ability to identify fraudulent cases.

The aim of this project is to contribute to the growing field of machine learning in financial crime detection by presenting a comprehensive analysis and evaluation of machine learning models applied to AML. The results of this work provide insights into the strengths and limitations of different techniques, offering a potential guide for future improvements in the fight against money laundering.

OUR PROJECT

The main idea of this project is to leverage advanced machine learning techniques and Graph Neural Networks to detect money laundering activities within financial transaction data. The project seeks to find the business challenge of reducing false positives and false negatives in AML detection systems, providing a more accurate solution for identifying illicit transactions.

MISSION AND GOALS

The vision of this project is to explore and develop machine learning approaches that address the challenges of class imbalance and poor precision, contributing to the broader domain of financial fraud detection. The goal is to improve the performance of AML detection systems by

integrating advanced techniques like GNNs and SMOTE to strike a balance between recall, which is the sensitivity in detecting fraudulent cases, and precision, which is for minimizing false positives.

Our approach highlights the importance of data processing, hyperparameter tuning, and the exploration of various model architectures. By applying these techniques, we aim to build a solution that not only identifies laundering activities but also minimizes wrong classifications, making AML systems more reliable.

DATA COLLECTION

The collection of datasets for this case study is a synthetic financial transaction one that is generated by IBM. It can be easily accessed by a site that is named as "Kaggle" in this specific link: "https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml/data". This is designed to simulate a financial ecosystem where both legitimate and illicit activities happen [9]. This collection is created to assist in the study and detection of money laundering, which is a global financial crime. This involves hiding the origins of illegally obtained funds through complex transaction layers. Due to the limitations in accessing real-world financial data, especially in accurately labeling transactions as either legitimate or money laundering, IBM's synthetic dataset provides an excellent alternative, by offering a fully labeled data for research and analysis.

This dataset models an environment where individuals, companies, and banks take part in various transactions. It captures the full money laundering cycle, which includes the placement, layering, and integration of illicit funds. Unlike real-world data, where a bank only sees its own transactions, this dataset provides a comprehensive view of an entire financial ecosystem. As a result, it offers the opportunity to develop AML models that can be trained from a wide variety of financial interactions.

This collection of datasets is divided into two groups based on the ratio of illicit activity, Group HI (higher illicit ratio) and Group LI (lower illicit ratio). Each group contains datasets of varying sizes, such as small, medium, and large, to accommodate different computational needs. The synthetic data has different date ranges, with the largest datasets containing nearly 200 million transactions, and laundering events are clearly marked for each transaction. This structure allows researchers to not only build machine learning models for AML but also to analyze transaction patterns associated with money laundering. Additionally, IBM offers larger datasets upon request for those looking to perform large-scale analyses.

DATASET OVERVIEW

The "HI-Small_Trans.csv" dataset is part of a larger collection and is the csv file that is utilized in this data analysis. This can be found in the link "https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml/data?select=HI-Small_Trans.csv". In addition to this transaction data, there is a companion file called "HI-Small_Patterns.txt", which contains laundering transactions categorized by one of eight known laundering patterns. For this case study, this file is not utilized.

This dataset focuses on a HI, meaning it contains a relatively larger proportion of laundering transactions compared to the corresponding LI datasets. This dataset covers a 10-day period of transactions, including interactions between banks, individuals, and companies. The dataset consists of approximately 5 million transactions and includes around 5100 labeled laundering transactions. These transactions can be used to develop and test models for detecting illicit behavior, particularly with GNNs.

More specifically, it contains important information about the exact date and time of each transaction, the bank accounts which send and receive money loads, both identified by unique account numbers, the amount that is paid and received in each transaction, the currency and payment format and the payment method. The dataset is labeled with a binary field, identifying whether each transaction is related to money laundering or not. The specific details that are provided in the dataset are briefly shown in Table 1.

Column Name	Column Description
Timestamp	Year/Month/Day Hour/Minute
From Bank	Numeric code for bank where transaction originates
Account	Hexadecimal code for account where transaction originates
To Bank	Numeric code for bank where transaction ends
Account.1	Hexadecimal code for account where transaction ends
Amount Received	Monetary amount received in "From" account (in currency units of the next column)
Receiving Currency	Currency of "To" account
Amount Paid	Monetary amount paid (in currency units of the next column)
Payment Currency	Currency of "From" account
Payment Format	How transaction was conducted, e.g. cheque, ACH, wire, credit cards, etc.
Is Laundering	0/1 value with 1 = Transaction is Laundering, 0 = Not

Table 1: The "HI-Small_Trans.csv" dataset overview.

DATA PROCESSING

Before proceeding with the main part of this analysis, several important values are checked. O begin with, the dataset consists of 5078345 entries and 11 columns. As far as the columns are concerned, the "Timestamp", Account", "Account.1", "Receiving Currency", "Payment Currency" and "Payment Format" column values are represented by objects or strings. The "From Bank", "To Bank" and "Is Laundering" columns are filled with integer values, stored as int64. Additionally, the "Amount Received" and "Amount Paid" contain floating-point values that represents the transaction amounts. This dataset uses approximately 426.2 MB of memory. This summary which identifies the data types and structure of the dataset, includes both numeric and categorical data. This summary of the dataset is conducted to comprehend the type of data for each column in the dataset. Finally, the total count of missing values in each column is

calculated, which is equal to zero for each one column. By acquiring an overview of the dataset's structure, it helps to understand its content, missing data, and resource consumption.

Another data transformation process includes some useful renames of columns. More specifically, the "Account.1" and "Account" columns are renamed as "To Account" and "From Account" respectively. This makes the column names more descriptive, reflecting the origin and destination accounts in a financial transaction dataset.

Several data preprocessing steps are after followed to ensure that the dataset is correctly formatted for analysis. First, the values in the "Timestamp" column are converted from a string format to a datetime one for easier manipulation of dates and times. Also, the "Amount Received" and "Amount Paid" columns are converted into numeric data types. If there are invalid values i.e., non-numeric, they will be replaced with a null value. Columns like "From Bank", "To Bank", "From Account", "To Account", "Receiving Currency", "Payment Currency", "Payment Format" and "Is Laundering" are converted into categorical data types, which is more appropriate for columns that represent distinct categories or groups rather than continuous numeric data. Finally, a brief check for missing values is again performed to ensure that no missing values are present after data transformations. Since the dataset lacks from null values, it means that the data conversions are performed successfully.

NUMERIC VARIABLE ANALYSIS

The next step involves the generation of a statistical summary of the numeric columns in the dataset. Several key descriptive statistics are provided for each of the numeric columns, including "Amount Received (USD)" and "Amount Paid (USD)". To begin with, for the "Amount Received (USD)", all records are unique, with an average amount received approximately 367470.4. Also, there is significant variation in the data, with a standard deviation of about 24381430 USD, indicating a wide range of amounts received. The highest amount received is 26993380000 USD. For the "Amount Paid (USD)", all records are again unique, with a total of 5078345 entries. The average amount paid in USD is approximately 370245.1. These data have a standard deviation of around 24664270 USD, suggesting a broad range of transaction amounts. The maximum amount paid is the same with the corresponding value for the "Amount Received (USD)". In summary, both columns show a high variability in transaction values, with a wide range from very small amounts to extremely large ones. The median values i.e., around 907 USD, indicate that half of the transactions are relatively small compared to the high maximum values. The data also shows that some very large transactions (in the billions) are present, as reflected by the large maximum values in both columns.

The two plots in Figure 1 illustrate the distribution of the log-transformed "Amount Received" for two subsets of the dataset, the left plot represents all data, while the right plot shows only transactions where laundering is flagged (i.e., "Is Laundering" = 1). The purpose of these plots is to visually compare the distributions across all transactions and laundering-specific transactions, rather than draw insights about specific monetary values, since the dataset contains multiple currencies in this case. It should be mentioned that the log transformation helps in normalizing the data, as financial transactions typically have a highly skewed distribution. This transformation alters the range and makes the data more interpretable.

The left plot shows the distribution of the log-transformed amounts received across all transactions in the dataset. The distribution is right-skewed, with most transactions centered between log values of 5 and 10, which corresponds to amounts in the range of approximately

100 to 10000 monetary units. There is a small tail extending towards higher log values, indicating the presence of much larger transactions.

On the right plot, the log-transformed "Amount Received" is depicted where "Is Laundering" = 1. This plot displays the distribution of log-transformed amounts received for transactions specifically flagged as money laundering. The distribution is more concentrated, with most laundering transactions falling within the log values of 9 to 12, corresponding to amounts between 1000 and 200000 monetary units. Compared to the general transaction distribution, the laundering transactions have a more centralized distribution with less variation, suggesting that money laundering activities tend to involve moderately large sums of money, avoiding both very small and extremely large transactions. The absence of a significant tail in the right plot implies that laundering transactions tend to avoid extremely high amounts, possibly to avoid detection by financial institutions.

Laundering Transactions are clustered in a relatively narrow range of amounts compared to the broader distribution in the overall dataset. This suggests that laundering often involves transactions within a "safe" monetary range that might minimize suspicion. The general distribution shows a wide range of transaction sizes, but laundering transactions focus on moderately high amounts, highlighting a strategic behavior by those laundering funds. Log transformation is useful here to reduce the effect of extreme values and better visualize the distribution, but the key insight is that laundering transactions are less diverse in value and typically involve substantial, but not excessively large, sums.

Figure 1: The distribution of the log-transformed "Amount Received" for the entire dataset (left)and for the case where laundering is flagged (right).

Figure 2 below shows the distribution of the log-transformed "Amount Paid" for the entire dataset as well as in the case when laundering is flagged. It is clear that the two plots in Figure 2 and the corresponding in Figure 1 are nearly identical. The two sets of plots i.e., the previous "Log-Transformed Amount Received" and the current "Log-Transformed Amount Paid", display similar comparisons, both focusing on the general transaction data on the left and laundering-specific transactions on the right.

If only the left plots of Figures 1 and 2 are considered, in both plots for all data, right-skewed distributions are observed. Most transactions are clustered between log values of 5 and 10, meaning they typically range from 100 to 10000 in their respective currencies. The tails of both distributions extend towards higher log values, indicating a smaller number of larger transactions. However, the "Amount Paid" plot has a slightly broader range in its tail, suggesting a few more high-value payments compared to the amount received. In general, however, the shape of both histograms is similar, reflecting that for most transactions, the amount received is typically close to the amount paid, with only a small discrepancy seen in larger payments.

For laundering-specific transactions, both the "Amount Received" and "Amount Paid" plots show concentrated distributions around log values of 9 to 12 or approximately 1000 to 200000 in their respective currencies. There is a clear similarity in the laundering transactions' concentration in both plots, though the "Amount Paid" distribution appears to have slightly more transactions closer to log value 10. The narrow and sharp peaks in both plots suggest that laundering activities are centered around moderately high values and involve more predictable, controlled amounts, avoiding extreme outliers or very small transactions.

The key insight from comparing these sets of plots is that laundering transactions tend to focus on moderate, controlled amounts, while the general dataset shows more variety, especially at the higher end of the transaction spectrum. The presence of multiple currencies in the data limits insights into the exact value ranges, but the distributional patterns remain informative for detecting potential laundering behavior.

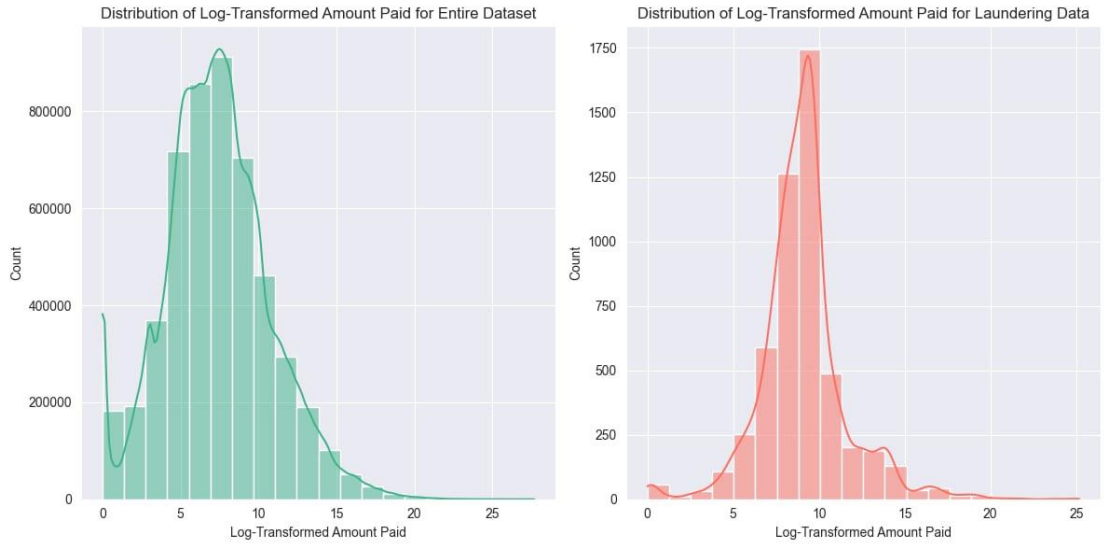

Figure 2: The distribution of the log-transformed "Amount Paid" for the entire dataset (left)and for the case where laundering is flagged (right).

Due to the common behavior of the "Amount Paid" and "Amount Received" variables, two checks are performed on the dataset to assess consistency between payment currency and receiving currency, as well as the amount paid, and amount received in transactions. The result shows that 98.58% of the transactions have matching currencies, indicating that the vast majority of transactions are conducted in the same currency for both payment and receipt. Similarly, it is checked if the "Amount Paid" matches the "Amount Received" for each transaction. The result also shows that 98.58% of the transactions have matching amounts,

meaning that in most cases, the amount paid by the sender matches the amount received by the recipient.

The results suggest a high level of consistency in the dataset, both in terms of currency and amounts. Most transactions are straightforward, with no discrepancies between the currency used for payment and receipt, nor between the amount paid and received. The small percentage of transactions where either the currency or the amount does not match warrant further investigation.

These mismatches between the “Payment Currency” and “Receiving Currency”, as well as between the “Amount Paid” and “Amount Received” are studied. There are 72170 transactions where the “Payment Currency” does not match the “Receiving Currency”. This indicates that a small portion of transactions, about 1.42%, involve some form of currency conversion, where the payer’s currency differs from that of the recipient. Transactions with currency mismatches shows records where the “US Dollar” is used for payment, but the receiving currency is another one, like “Euro” or “Australian Dollar”. This points to transactions involving cross-border payments, where currency conversion is required.

Also, there are 72158 transactions where the “Amount Paid” does not match the “Amount Received”. This discrepancy could be due to reasons such as fees, exchange rate fluctuations in cross-currency transactions, or other adjustments made during the transfer process. This means that currency conversion may play a significant role in these differences.

Some useful results are the fact that most amount mismatches are associated with currency mismatches, which suggests that the currency conversion process is a key factor leading to differences between the amounts sent and received. Overall, while most transactions are consistent in both currency and amounts, these mismatches provide valuable insight into cross-border financial activity and potential reasons for discrepancies in transaction data.

The transactions with the largest percentage discrepancies between the “Amount Paid” and the “Amount Received” are further investigated. The percentage difference is calculated in each case, indicating how much the “Amount Paid” exceeds or differs from the “Amount Received” in these transactions.

Some of these transactions have extremely large discrepancies between the “Amount Paid” and “Amount Received”, ranging from 92 to 125 million percent. These enormous differences suggest that very small amounts were received compared to the amounts paid. This suggests that discrepancies are linked to cross-currency transactions, specifically where large amounts in traditional currencies like “Yen” or “Ruble” are converted into “Bitcoin”.

For this reason, a scatter plot is generated that visualizes the relationship between the “Amount Paid” and “Amount Received”, both converted to USD in Figure 3. Each point represents a transaction, with the x-axis indicating the amount paid and the y-axis showing the amount received. Most of the points form a tight cluster along a line near the diagonal, indicating a close correlation between the “Amount Paid” and “Amount Received” for the majority of transactions. This suggests that for many transactions, the amount paid and received are nearly equal or proportional, with minimal discrepancies. A few points are located far from the main cluster, specifically at the higher end of the plot, for amounts above 1 billion USD. These outliers represent transactions where extremely large amounts were paid and received. The fact that these points are still somewhat close to the diagonal indicates that, despite the large amounts, the amounts paid and received are reasonably proportional. While most of the data points align closely with the diagonal, there are minor deviations where the amount received is slightly less than the amount paid. These discrepancies could result from currency conversion losses as highlighted above.

In summary, the plot shows that transaction amounts tend to be balanced for most of the dataset, with a few extreme values representing large transactions that still follow a similar pattern. These findings reinforce the idea that most financial exchanges in this dataset are handled smoothly and consistently.

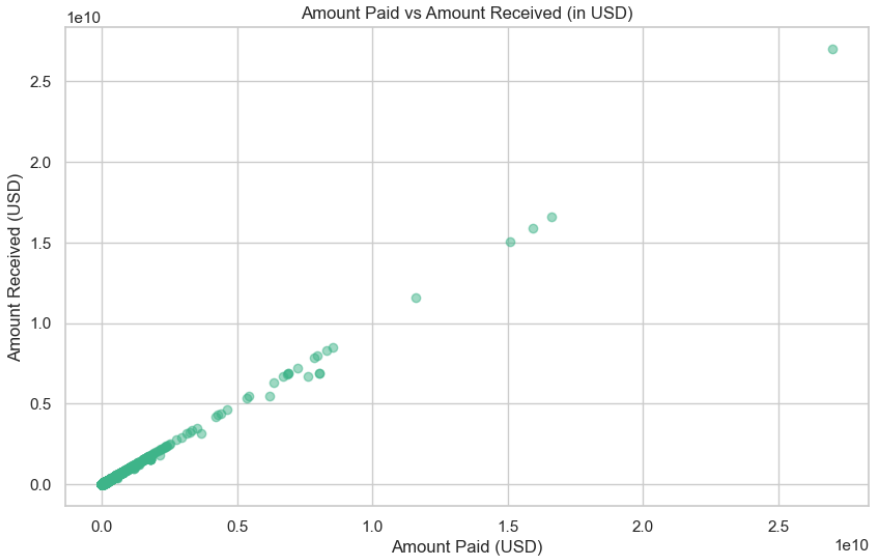

Figure 3: Line chart for the "Amount Paid" and "Amount Received" in USD.

TRANSFORMATIONS OF CURRENCY TO USD

The next step of this analysis includes some necessary transformations. These transformations include the task of converting monetary amounts in various currencies to US dollars (USD) for both the "Amount Received" and "Amount Paid" columns in a dataset, based on the "Receiving Currency" and "Payment Currency" correspondingly. This enables consistent comparisons across different currencies by converting all amounts into a single standard currency i.e., the USD.

If finding the unique values in the "Payment Currency" column of the dataset, currencies that are present in the dataset can be identified and converted into USD. In Table 2 below, the conversion rates from various currencies to USD are depicted. These conversion rates are used to convert amounts in various currencies into US dollars for both the "Amount Received" and "Amount Paid" columns.

Currency Type (1 Unit)	**Conversion to USD**
Bitcoin	62691.60
Euro	1.12
Australian Dollar (AUD)	0.68
Yuan	0.14
Rupee (INR)	0.012
Yen	0.0069

Mexican Peso (MXN)	0.052
UK Pound (GBP)	1.33
Ruble	0.011
Canadian Dollar (CAD)	0.74
Swiss Franc (CHF)	1.18
Brazil Real (BRL)	0.18
Saudi Riyal (SAR)	0.27
Shekel (ILS)	0.26

Table 2: Conversion rates from one currency unit to USD.

Based on these conversion rates, two new columns are created i.e., the "Amount Paid (USD)" and "Amount Received (USD)" which contains the converted amounts from "Amount Paid" and "Amount Received" columns correspondingly, from one currency type to USD. The output enables a better comparison of financial transactions across different currencies. This approach ensures consistency in comparing financial data by converting all currency values into a common standard. Similarly, the columns "Amount Received (USD)" and "Amount Paid (USD)" are converted into numeric, ensuring that all currency data in USD is numerical.

DATETIME VARIABLE ANALYSIS

The next group of variables for study contains the datetime variables i.e., the "Timestamp" one. The two lineplots in Figure 4 illustrate the trend of transactions over time, comparing the total number of transactions (left plot) with the number of laundering transactions (right plot) across several days in September 2022, which is the time range of the data. The goal of these plots is to show how the volume of transactions, both legitimate and laundering-related, changes during this time period.

The left plot shows a sharp decline in the total number of transactions after September 1, where the volume starts above 1 million transactions but quickly drops to below 200000 by September 3. A brief increase is observed between September 6 and September 8, followed by a significant drop to almost zero by September 11. After September 11, the transaction volume remains near zero, indicating that there was a substantial reduction in transaction activity during that period.

In contrast to the overall transaction trend, the right plot shows that the number of laundering transactions increases from September 1 to September 7, peaking at around 500 transactions. After September 7, laundering activity begins to decline, with a steep drop from September 9 onward, almost reaching zero by September 11. The pattern suggests a rise in laundering transactions as overall transaction volume decreases, followed by a synchronized decline in both overall and laundering transactions from September 9 onward.

It is obvious that there is a disconnect between the trends in total transactions and laundering-specific transactions, especially in the early days of September. As the overall transaction volume sharply drops, laundering activities initially rise, suggesting that illicit transactions might continue for a while even as legitimate transactions decrease. After September 7, both trends decline rapidly, indicating that laundering transactions tend to track the overall transaction volume with a slight delay. The significant decline in laundering activity after September 9 mirrors the drop in overall transactions, reinforcing the connection between transaction opportunities and laundering activity. This analysis indicates that laundering

activities are likely opportunistic, increasing in a period of higher overall transaction volume but tapering off once transaction volumes fall significantly.

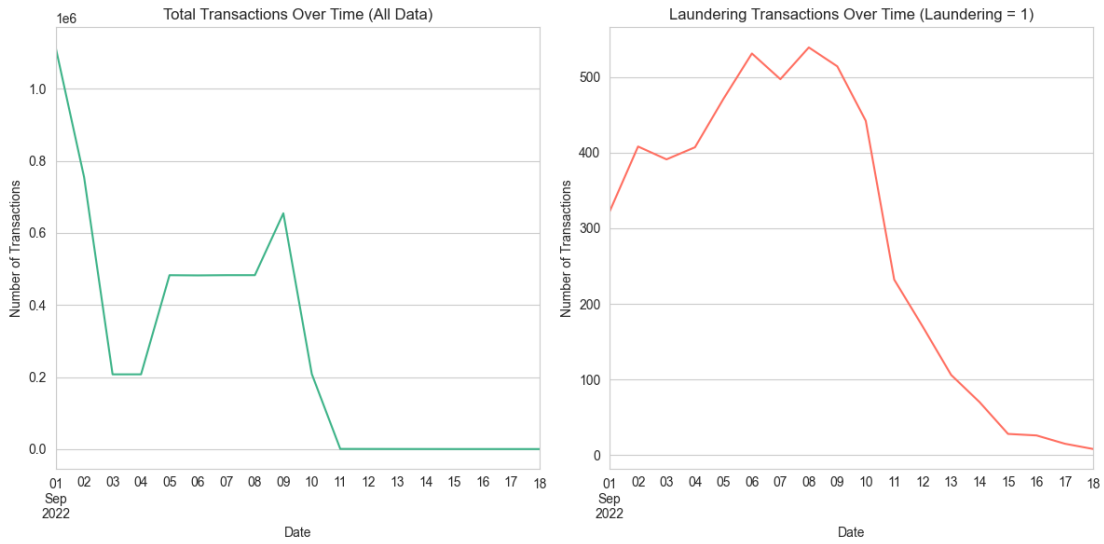

Figure 4: Total transactions over time (left) and laundering transactions over time (right).

CATEGORICAL VARIABLE ANALYSIS

The last group of variables for study contains the categorical variables. As before, the Univariate Analysis on categorical variables is conducted, by generating count plots. Some valuable plots are those concerning the three categorical columns in the dataset, the "Receiving Currency", "Payment Currency", and "Payment Format". The count plots visualize the frequency of each unique value or category within those columns, allowing to observe the distribution of different variables.

The first variable for extensive analysis is the "Payment Format". The two bar plots in Figure 5 that are displayed show the distribution of payment formats in a dataset, with the left plot visualizing all data, and the right plot specifically visualizing transactions that are flagged as laundering (i.e., "Is Laundering" = 1). As far as the left bar plot is concerned, this one displays the count of different payment formats used across all transactions in the dataset. The "Cheque" option is the most used payment format, with nearly 1.75 million transactions. "Credit Card" also has a high usage, followed by "ACH" (Automated Clearing House), "Cash", and "Reinvestment". Less commonly used formats include "Wire" and "Bitcoin", which have significantly lower counts compared to the other formats. As for the right plot, this one focusses specifically on the payment formats used in transactions flagged as money laundering (Laundering = 1). Unlike the general distribution in the left plot, "ACH" is overwhelmingly dominant in laundering transactions, with over 4000 instances. Other payment formats such as "Cheque", "Credit Card", "Cash", and "Bitcoin" have minimal usage in laundering, with much smaller counts.

As a result, "ACH" stands out as the primary payment method associated with laundering transactions, which has a significant deviation from the overall dataset where "Cheque" and "Credit Card" dominate. This could indicate that criminals may prefer the "ACH" payment format for illicit activities due to its electronic nature and the ability to move large sums

efficiently. On the other hand, the widespread use of “Cheque” and “Credit Card” in legitimate transactions suggests these are more common for everyday financial activities, while their use in laundering is minimal. “Wire” and “Bitcoin”, while not widely used in either set, still appear more in general transactions than in laundering cases, possibly because they are used less frequently overall or because “Bitcoin” could be perceived as traceable in this context. For this reason, certain payment methods like “ACH” could potentially be flagged for further investigation when trying to detect laundering activities.

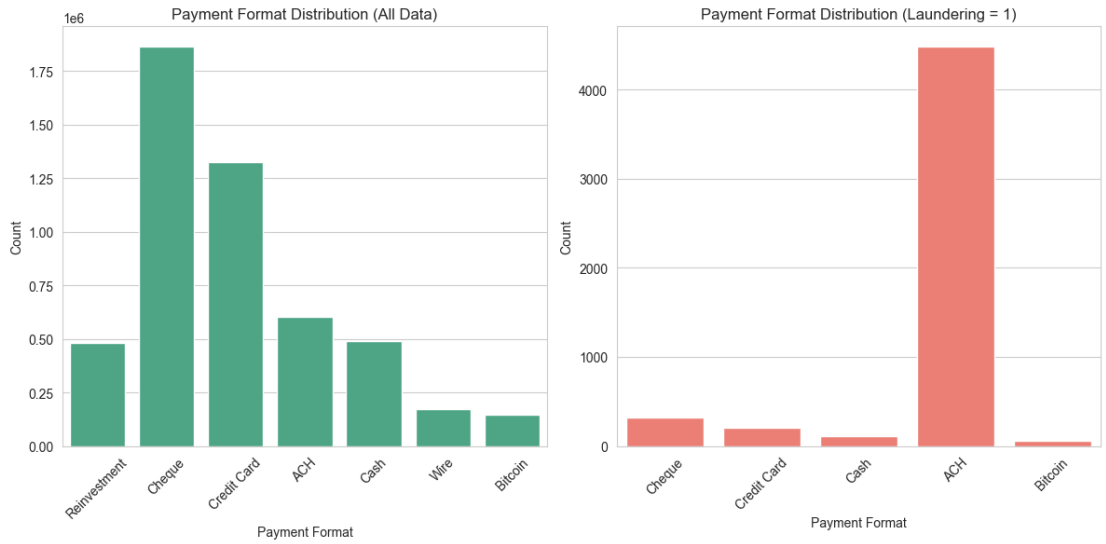

Figure 5: The distribution of the Payment Format for the entire dataset (left)and only for the case where laundering is flagged (right).

The next variable for analysis is the “Payment Currency”. The two bar plots in Figure 6 visualize the distribution of payment currencies in the dataset, with the left plot representing all data and the right plot focusing on transactions flagged as laundering. On the left plot concerning all data, the “US Dollar” is the most used currency in the dataset, with close to 1.75 million transactions. This indicates that the majority of transactions are conducted in USD. The “Euro” is the second most common currency, with around 700000 transactions. “Swiss Franc”, “Yuan”, “Shekel”, and “UK Pound” make up the next largest transaction volumes, though they are far less frequent than USD and Euro. Other currencies, such as “Rupee”, “Bitcoin”, “Canadian Dollar”, and “Australian Dollar”, are used less frequently, each contributing a smaller share to the overall transaction volume.

On the right plot, even within the laundering-specific transactions, the “US Dollar” remains the dominant currency, with nearly 2000 transactions. “Euro” and “Swiss Franc” also continue to play significant roles in laundering activities, though at a much lower scale compared to the total transaction volume. Interestingly, currencies such as “Saudi Riyal” and “Brazil Real”, which were barely noticeable in the general transaction data, show a notable presence in laundering transactions, particularly “Saudi Riyal”, which stands out as more commonly used in laundering than in general transactions.

A useful insight is that the “US Dollar” and “Euro” are the most used currencies across both general and laundering-specific transactions. This is likely due to the global dominance of these currencies in financial transactions. While “Swiss Franc” also appears frequently in both data sets, the right plot shows some currencies, such as “Saudi Riyal” and “Brazil Real”, becoming

more prominent in laundering transactions compared to their minor role in the overall dataset. This suggests that some lesser-used currencies in the broader dataset might play a disproportionately larger role in laundering activities.

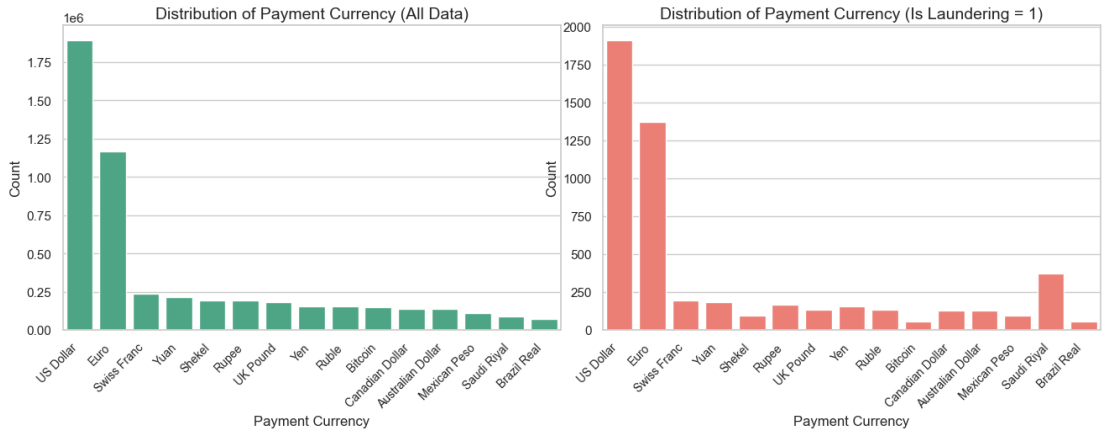

Figure 6: The distribution of the Payment Currency for the entire dataset (left)and only for the case where laundering is flagged (right).

This next bar plot in Figure 7 illustrates the proportion of laundering transactions relative to the total number of transactions for each payment currency. The y-axis represents the proportion of laundering transactions, while the x-axis shows the different currencies used in these transactions. The “Saudi Riyal” has the highest proportion of laundering transactions by a significant margin, with a rate of around 0.0040. This suggests that a much larger percentage of transactions conducted in Saudi Riyal are flagged as laundering, making it a notable outlier compared to other currencies. “Euro”, “Canadian Dollar”, “Brazil Real”, “Mexican Peso”, “Ruble”, “US Dollar”, “Yuan”, and “Yen” exhibit moderate proportions of laundering transactions, with rates hovering between 0.0010 and 0.0015. These currencies are more commonly used in laundering transactions compared to others but do not stand out as much as the Saudi Riyal.

The extremely high laundering proportion associated with the Saudi Riyal is a red flag and may indicate that transactions involving this currency require additional scrutiny. This suggests a potential pattern of higher laundering activities associated with this currency compared to others.

Figure 7: Laundering rate by payment currency.

Last but not least, the barplot concerning the distribution of various payment formats used in the dataset is visualized. The x-axis represents the count of transactions, while the y-axis lists the different payment formats. The plot indicates how frequently each payment format appears in the dataset. The "Cheque" is the most frequently used payment format, with the count approaching 1.75 million transactions. The second most common method is the "Credit Card", with slightly more than 1 million transactions. The "ACH" (Automated Clearing House) method ranks third, with a count just under 750000 transactions. Also, "Cash" transactions account for a smaller, yet significant number, slightly above 500000. The "Reinvestment" format has a moderate frequency, with slightly over 250000 transactions. Also, "Wire" transfers are used less frequently, with the count being relatively low compared to others, below 100000. Last but not least, the "Bitcoin" is the least common payment format in this dataset, with a count that is very minimal compared to the other payment methods.

The next bar chart in Figure 8 illustrates the top 10 Banks ranked by the number of transactions they originate, as represented by the "From Bank" column. Each bank is represented by its unique identifier, and the y-axis reflects the number of transactions each bank has initiated.

Bank 70 stands out significantly, originating over 400000 transactions, far exceeding the other top banks. This disproportionate number suggests that Bank 70 plays a central role in the financial network or has a much higher transaction volume compared to others. The substantial dominance of Bank 70 may indicate that it is a large institution, potentially handling transactions for a wide range of customers. This could also be a sign of specific operational focus or specialization in certain types of transactions. The uniformity of transaction volume among the rest of the banks in the top 10 suggests that outside of Bank 70, the financial activity is more evenly distributed across multiple institutions. This distribution highlights the importance of monitoring high-activity banks like Bank 70 for potential anomalies or risks, especially in anti-money laundering efforts, as such institutions are likely to be critical nodes in the financial ecosystem.

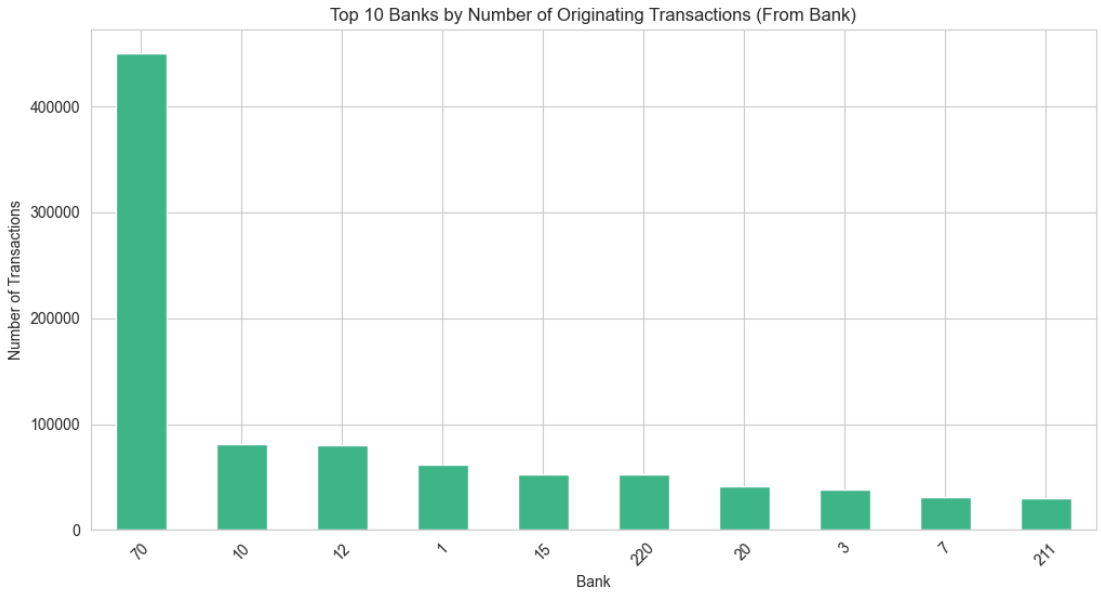

Figure 8: Top 10 banks by number of originating transactions.

Unlike the earlier plot for originating transactions, the top 10 banks by the number of destination transactions are checked. In this case, the destination banks show a more evenly distributed transaction count in the corresponding figure, so no further analysis on this plot is conducted.

BIVARIATE DESCRIPTIVE ANALYSIS

The next step of this process includes the study of the correlation between the variables of the dataset. The heatmap that is displayed in Figure 9 shows the correlation matrix for various numerical columns in the dataset, including “Amount Received”, “Amount Paid” and their corresponding log-transformed values. Each cell in the matrix represents the Pearson correlation coefficient between two variables, with values ranging from -1 i.e., perfect negative correlation, to 1 i.e., perfect positive correlation. The intensity of the color indicates the strength of the correlation, which means that redder cells indicate a stronger positive correlation, while bluer cells indicate weaker correlations.

There is a high correlation of 0.84 between “Amount Received” and “Amount Paid”, indicating that, in most transactions, the amounts received and paid are closely related. This suggests that transactions are likely balanced or involve similar sums, which is typical for many financial exchanges.

Accordingly, the log-transformed values of “Amount Received” and “Amount Paid” have a perfect correlation 1, as expected. Log transformations are applied to both columns in a similar manner, so their correlation is naturally high. These transformed values also show weak correlations with the original amount columns.

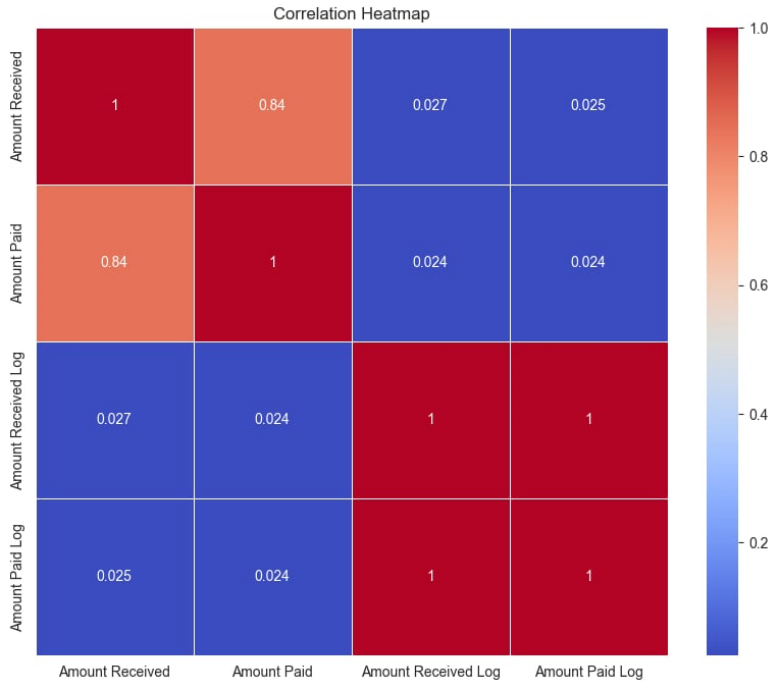

Figure 9: Correlation Matrix between the numerical columns of the dataset.

XGBOOST AND LIGHT GMB METHODS

This chapter deals with the XGBoost and Light GBM methods, which are two popular gradient boosting algorithms used for binary classification. These methods aim to predict whether a financial transaction is associated with money laundering, based on the dataset. The chapter begins by exploring the XGBoost method, where hyperparameters like learning rate, depth, and the number of estimators is tuned using "GridSearchCV" to optimize performance. Next, the Light GBM method is introduced, offering a more optimized approach in terms of speed and efficiency. Light GBM also undergoes hyperparameter tuning, and similar to XGBoost, it faces challenges with class imbalance. Both models demonstrate the importance of handling class imbalance, such as money laundering detection, where performance on minority classes is crucial despite high overall accuracy. This section emphasizes the need for further techniques like SMOTE to improve minority class detection.

DATA PREPROCESSING

The dataset ought to be prepared through cleaning and encoding categorical data, and then split into training, validation and test sets for further analysis. To begin with, the columns "Timestamp", "From Account", and "To Account" are dropped from the dataset as they are not needed for modeling. Categorical columns like "Receiving Currency", "Payment Currency", and "Payment Format" are converted into numeric values using One-Hot Encoding. Machine learning models typically cannot handle categorical data directly, so one-hot encoding transforms these columns into binary columns representing each unique category. This particular One Hot Encoder converts each categorical value into a new column, ensuring that

all categories are represented numerically. Following this, the original categorical columns are dropped from the dataset since their numerical representations are now included. This step ensures that all features in the dataset are numerical and suitable for modeling. These features consist of all the independent variables that will be used to predict whether a transaction is laundering or not.

Following these steps, a two-step data splitting process is used to divide the dataset into training, validation, and test sets. This strategy is designed to build a robust model by ensuring that it is tested on unseen data and fine-tuned using a separate validation set. The training set is used to train the model. The validation set is used to fine-tune and validate the model during training, whereas the test set is used to evaluate the model's performance on unseen data after training and validation.

Splitting the dataset is performed in a specific way. More specifically, the entire dataset is initially divided into two parts, 80% of it is used for training and 20% for testing. By setting aside a portion of the data for testing, it is ensured that the model's performance can be assessed on unseen data, which gives a more accurate measure of how well the model generalizes. After this, a second split is performed into the training and validation sets. 64% of the initial dataset is used for final model training and 16% for validation, which is used to tune the model's hyperparameters and monitor its performance during training. The validation set is important because it helps prevent overfitting. The model is trained on the training set, but its performance is validated during training using the validation set to ensure it generalizes well before testing it on the test set. This ensures that the model is not overfitted to the training data. By dividing the data into training, validation, and test sets, a good practice is followed in machine learning to ensure the model is properly trained, tuned, and evaluated.

XGBOOST METHOD

After these steps, the XGBoost classifier is initialized, which is designed for binary classification problems, as in this case. The objective parameter is set to binary, meaning the model will predict probabilities and classify based on a logistic regression function. The loss function is optimized by the model during training. Log loss is commonly used for binary classification as it penalizes wrong predictions based on the confidence of the prediction.

The next step includes the definition of the hyperparameters to be tuned. These hyperparameters include the maximum depth, the number of estimators, and the learning rate. The maximum depth controls the maximum depth of each tree in the model. Higher values can capture more complex patterns but also increase the risk of overfitting. The number of estimators is the number of trees or boosting rounds in the model. Finally, the learning rate controls the step size in each boosting step. Lower values ensure that the model learns more cautiously, while higher values allow faster learning. For this purpose, the "GridSearchCV" is used to find the best combination of hyperparameters. It evaluates the model using 3-fold cross-validation for each combination of hyperparameters in the grid. The metric used to evaluate performance during cross-validation is the F1 score. The F1 score is chosen because it balances precision and recall, which is especially useful when the dataset is imbalanced as in this case. The model is trained using the training data with each combination of hyperparameters. After tuning, the best set of hyperparameters is found along with the corresponding best cross-validation score. These hyperparameters are a learning rate equal to 0.2, a maximum depth of 7 and the number of estimators equal to 200. Although the model achieves an accuracy of 99.90%, this result is misleading due to the imbalanced nature of the dataset, where the vast majority of the labels

belong to the negative class. As shown in the confusion matrix, the model has excellent performance in predicting the majority class but struggles with the minority class. Specifically, while the precision for the non-laundering class is quite high at 75%, the recall is only 6%, indicating that the model is missing many of the positive cases. Consequently, the F1 score for class 1 is low and equal to 0.11, reflecting poor performance in handling the minority class.

LIGHT GBM METHOD

The next method to be used is the Light Gradient Boosting Machine (Light GBM). As before, the Light GBM classifier is initialized, which is a popular gradient boosting algorithm optimized for speed and performance. The task is again binary classification, where the target variable can take two values, and the loss function used is binary log loss. The hyperparameters are optimized using "GridSearchCV" as before, with the same hyperparameters. The model is trained using the training data with each combination of hyperparameters. The best set of hyperparameters is selected based on the F1 score.

The best parameters are a learning rate equal to 0.1, a maximum depth equal to 7 and the number of estimators equal to 200. The model achieves an accuracy of 99.89%, which is high, but due to the severe class imbalance, accuracy is not an ideal metric again. As shown in the confusion matrix, the model performs very well in predicting the majority class but struggles to predict the minority class. The precision for the laundering class is 40%, but the recall is only 3.4%, meaning that the model misses many positive cases. Consequently, the F1 score for the laundering class is 0.06, indicating poor performance in detecting the minority class. The ROC-AUC score of 0.52 suggests that the model is only slightly better than random guessing for classifying the minority class. This result, like with the previous XGBoost model leads to poor performance in identifying the minority class, suggesting that further techniques, such as SMOTE, are necessary to improve minority class detection.

SYNTHETIC MINORITY OVER-SAMPLING TECHNIQUE (SMOTE)

In order to address the issue of class imbalance in the training data, SMOTE is applied. The primary goal of SMOTE is to create a more balanced dataset by oversampling the minority class, in this case laundering transactions, which can help improve the performance of machine learning models when dealing with imbalanced datasets. In practice, SMOTE generates synthetic examples of the minority class to balance the dataset. The purpose of using this technique is that, in an imbalanced dataset, such as the one in this case where laundering transactions are far fewer than non-laundering transactions, machine learning models tend to perform poorly on the minority class. SMOTE generate these synthetic samples of the minority class based on the feature space of existing instances rather than by simply duplicating them.

The SMOTE is applied to the training data. After applying this technique, the number of laundering transactions in the training set is equal to the number of non-laundering transactions. This balanced dataset improves the ability of the model to identify laundering transactions and reduce the risk of the model being biased towards the majority class.

XGBOOST METHOD WITH SMOTE

The implementation of the XGBoost classifier with SMOTE is applied to handle class imbalance. The XGBoost classifier is again instantiated with the objective of binary classification. A grid search is set up using "GridSearchCV" to find the best combination of hyperparameters for the XGBoost model. The parameters being tuned are the maximum depth, the number of estimators and the learning rate. The grid search runs with 3-fold cross-validation to test different combinations of these parameters and uses the F1 score as the evaluation metric. The model is trained using the oversampled data, where SMOTE has been applied to balance the minority class. After the grid search completes, it prints the best hyperparameters found and the best F1 score achieved. The best XGBoost model is used to make predictions on the test data.

The best hyperparameters for XGBoost are a learning rate of 0.2, a maximum depth of 7 and number of estimators equal to 200. These hyperparameters lead to the best performance in terms of the F1 score during cross-validation. The accuracy is equal to 97.87%, which is very high, indicating that the model correctly predicted the majority of cases. However, accuracy alone is not a sufficient metric due to the class imbalance. The F1 score is quite low and equal to 0.0569, which indicates that the model is not performing well in balancing precision and recall, particularly for the minority class. The precision of 0.0298 is extremely low, which shows that out of all the fraud cases the model predicted, only about 3% were actually fraud. This suggests a high number of false positives. The recall of 0.5998 is surprisingly high for the minority class, indicating that the model is good at identifying actual fraud cases, but due to low precision, many of its positive predictions are incorrect. The ROC-AUC score of 0.7895 is decent and shows that the model has a reasonably good ability to distinguish between fraud and non-fraud cases.

In general, for the majority class, precision, recall, and F1 scores are very high, meaning the model performs exceptionally well at identifying non-fraud cases. For the minority class, precision is very low, meaning the model incorrectly predicts many non-fraud cases as fraud, while the recall is much higher, indicating it successfully identifies 60% of actual fraud cases.

As for the confusion matrix, it is revealed that 993387 non-laundering cases are correctly identified. 21195 of non-laundering cases are incorrectly classified as laundering, 435 laundering cases are missed by the model and 652 laundering cases are correctly identified. The number of false positives is notably high, which reflects the low precision score.

As a result, SMOTE helps the model become more sensitive to detecting the minority class, but at the cost of precision. To improve this further, other techniques could be applied to improve precision while maintaining recall. Also, although the accuracy is very high, this is largely because the dataset is dominated by non-fraudulent transactions. The accuracy metric here does not fully reflect the model's ability to detect laundering. Therefore, metrics like F1 score, precision, and recall are more insightful in this scenario.

LIGHT GBM METHOD WITH SMOTE

The use of the LightGBM classifier with SMOTE is now applied in order to handle class imbalance in this binary classification problem. To begin with, the LightGBM classifier is instantiated with a binary classification objective. A "GridSearchCV" is used to tune the hyperparameters for this model. The parameters being tuned include again the maximum depth,

the number of estimators and the learning rate. After that, 3-fold cross-validation is applied using the F1 score as the evaluation metric, optimizing the model for this score. The model is trained using the oversampled data where SMOTE has been applied to balance the minority class. Once the best parameters are found, the model is used to predict the outcomes on the test data.

The best hyperparameters are the learning rate of 0.2, the maximum depth of 7 and the number of estimators of 200. The overall accuracy is 97.56%, which is high. However, accuracy alone is not a reliable metric in the presence of class imbalance. The F1 score is 0.053, reflecting the balance between precision and recall for the minority class. Precision is 0.028, indicating that only 2.8% of the transactions the model flagged as fraudulent were actually fraud. The recall is 0.638, which is much higher than precision. This shows that the model correctly identified 63.8% of the actual fraudulent cases, indicating that it is relatively good at capturing fraudulent transactions, even though it generates many false alarms. The ROC-AUC score is 0.807, which is quite good, showing that the model can effectively distinguish between the majority and minority classes.

As for the confusion matrix, 990156 non-laundering transactions are correctly classified, 24426 non-laundering transactions are incorrectly flagged as laundering, contributing to the low precision. Also, 393 laundering transactions are missed by the model and 694 laundering transactions are correctly identified.

In general, the model has a strong ability to detect laundering transactions, but it does so at the cost of a very low precision, meaning many legitimate transactions are wrongly flagged as fraud. The application of SMOTE has helped improve recall, making the model more sensitive to detecting the minority class. However, the increase in recall has come with a trade-off in precision, resulting in many false positives. Although accuracy is high, it is not a reliable measure in this imbalanced dataset. The low F1 score reflects the model's poor performance in balancing precision and recall for fraud detection. A relatively high ROC-AUC score shows that the model is generally good at distinguishing between fraudulent and non-fraudulent transactions, even though the precision is low.

COMPARISON WITH THE XGBOOST METHOD WITHOUT SMOTE

As for the accuracy, accuracy without SMOTE is extremely high, but this is misleading due to the overwhelming number of non-laundering transactions dominating the dataset. After applying SMOTE, the accuracy drops slightly, but this drop is acceptable because the model is now better at identifying fraudulent transactions. As for the F1 score, without SMOTE, the F1 score is low, which indicates poor performance in detecting fraudulent transactions. After applying SMOTE, the F1 score drops further, showing that despite SMOTE improving recall, the precision remains too low to balance out the F1 score effectively. As for the precision, without SMOTE, the precision for laundering transactions is 0.753, meaning that when the model identifies a transaction as laundering, it is correct 75.3% of the time. However, this is at the cost of an extremely low recall. With SMOTE, precision drops dramatically, indicating that a large number of non-laundering transactions are being incorrectly flagged as laundering. This is a key trade-off with SMOTE, as the model is now overly sensitive to the minority class. As for the recall, without SMOTE, the recall is 0.062, highlighting a failure to capture the minority class. With SMOTE, recall improves significantly. This shows that SMOTE is effective in making the model more sensitive to the minority class, addressing the recall issue. As for the

ROC-AUC, without SMOTE, the ROC-AUC score is barely above random guessing, indicating poor discriminatory power. With SMOTE, the ROC-AUC score improves significantly, showing that the model is now better at distinguishing between laundering and non-laundering transactions. As for the confusion matrix, without SMOTE, the model correctly classifies most of the non-laundering transactions but misses a large number of laundering cases. With SMOTE, the number of true positives increases dramatically, but at the cost of introducing more false positives, showing the trade-off between precision and recall.

In conclusion, without SMOTE, the model prioritizes precision over recall, meaning it correctly identifies a small number of laundering cases with high precision, but misses the majority of them, which results in a low recall. With SMOTE, the model improves its ability to detect laundering, but the trade-off is a significant drop in precision, leading to many false positives. The F1 score also declines due to the imbalance between precision and recall, even though the model now catches a much higher percentage of fraudulent transactions.

COMPARISON WITH THE LIGHT GBM METHOD WITHOUT SMOTE

Without SMOTE, the accuracy is really high, almost perfect. With SMOTE, accuracy drops, which is still high. However, this drop is acceptable as the model improves its performance in detecting laundering transactions, as evidenced by the much higher recall. As for the F1 score, without SMOTE, the F1 score is low due to the poor recall. With SMOTE, the F1 score slightly decreases. As for the precision, without SMOTE, the precision is high which comes at the cost of extremely low recall. With SMOTE, precision drops, which reflects the model's tendency to produce a large number of false positives. As for the recall, without SMOTE, the model identifies only 3.4% of actual laundering transactions. With SMOTE, recall improves significantly, meaning the model correctly identifies 63.8% of laundering transactions. This increase shows that SMOTE helps the model become more sensitive to the minority class. As for the ROC-AUC, without SMOTE, the ROC-AUC is barely better than random guessing. With SMOTE, the ROC-AUC improves significantly, indicating that the model is much better at distinguishing between laundering and non-laundering transactions after applying SMOTE. As for the confusion matrix, without SMOTE, the model has very few false positives, but it misses most laundering transactions. With SMOTE, the model detects many more laundering transactions, but at the cost of a much higher number of false positives, which affects the overall precision.

As a result, with SMOTE, the model improves its ability to detect laundering, but generates more false positives, while without SMOTE, the model has high precision but struggles to identify most fraud cases. Finding a balance between these trade-offs is crucial, especially in real-world fraud detection scenarios. For this reason, the GNNs are utilized in order to find a better way of detecting laundering transactions and get closer to the main goal of this project.

GRAPH NEURAL NETWORK (GNN)

In this chapter of the project, the focus is on evaluating three specific types of GNNs [1, 2, 3, 4, 8], GAT, GCN, and GIN. These models are implemented to handle graph-structured data effectively, such as identifying suspicious transactions in financial networks. The chapter introduces the different aggregation mechanisms that each model uses to capture node and

graph features. GAT employs an attention-based mechanism to dynamically weigh the importance of neighboring nodes. GCN, on the other hand, aggregates node information through a weighted averaging process. Finally, GIN distinguishes itself by using sum aggregation followed by a multi-layer perceptron (MLP) to transform and enrich node representations.

Each model offers distinct advantages depending on the task's complexity. The chapter dives into the practical application of these methods in the context of fraud detection, highlighting their performance, strengths, and limitations in capturing the relationships within financial transaction networks.

DATA PROCESSING FOR GNN

The processing of the dataset begins with preparing it for easier analysis by going through a series of processing steps. First, encoding of the categorical columns "Payment Format", "Payment Currency", and "Receiving Currency" is applied, which replaces text-based categories with numeric labels. The "Timestamp" column, initially in string format, is converted into a numeric one and then normalized to a range between 0 and 1. The normalization ensures that timestamps are comparable on a common scale without large numerical values dominating, allowing for easier comparisons between transaction times. Next, unique account identifiers are created by combining the bank name with the account number, making it clear which bank is associated with each transaction. This step helps in differentiating accounts more easily in the dataset.

Furthermore, the dataset is split into two separate parts, one focusing on money that has been received by different accounts, and the other on money that has been paid out. This separation helps in analyzing where money is coming from and where it is going. A list of all the different currencies used in the transactions is also created, helping with currency-related analysis. After all these steps, the dataset is sorted and organized so that it is ready for further examination. the transaction data are further processed to gather information about accounts and their activity in terms of money paid and received in different currencies. More specifically, a list of all accounts is compiled, including both the ones sending and receiving money, and any suspicious accounts potentially involved in money laundering are labeled. The accounts are then cross-referenced, and duplicates are removed to ensure each account is unique and labeled accordingly. Next, the average amount of money is calculated for each account paid and received in each currency. This helps in identifying the typical behavior of each account across various currencies. These averages are added as new columns to the data, providing detailed attributes that describe the financial activity of each account. Finally, the processed data is structured into two parts, a dataset containing the financial attributes for each account i.e., how much they pay and receive on average in each currency, and labels indicating whether each account is suspected of money laundering. The result is a structured dataset that highlights financial behavior and flags potentially risky accounts.

The financial transaction data are then processed to create a graph structure, where each account is represented as a node, and each transaction between accounts is represented as an edge. The goal is to map the transaction flow between accounts, which is useful for analyzing the relationships and interactions in the data. First, a unique identifier is assigned to each account in the dataset, creating a mapping between account numbers and these identifiers. Then this mapping is used to convert the sender and receiver columns into numerical form, where the identifiers are used to represent the "From" and "To" relationships in the transaction graph.

Next, a tensor is built that captures the connections between accounts. This tensor contains two rows, the first row represents the “From” nodes or sender accounts, and the second row represents the “To” nodes or receiver accounts, effectively forming a graph of transactions. In addition to the structure of the graph, the attributes of each transaction are extracted, such as the timestamp, amount paid or received, currency, and payment format. These attributes are stored and are used as edge features in the graph, providing context for each transaction. The result is a graph-ready dataset that can be used for further analysis or modeling.

GRAPH FORMULATION

All these financial transaction data are ready to be transformed into a graph structure to be used for anti-money laundering detection. All raw transaction data are loaded and processed, and a graph representation is generated where each node represents an account, and each edge represents a transaction between two accounts.

The process starts by loading the raw CSV file that contains the transaction data. The data are then processed by encoding categorical variables, normalizing timestamps, and generating unique account identifiers. Account-level information is also aggregated, such as the average amount of money paid or received in each currency for every account, allowing for more detailed account attributes. An account is also identified whether it is involved in suspicious transactions or not. For the graph structure, two main components are created, the nodes, which represents the accounts with attributes such as average payment or receipt data, and edges representing the transactions between these accounts. Each edge is characterized by attributes like the transaction amount, currency, and timestamp. The final output is a graph dataset that includes node attributes or account information, edge attributes or transaction details, and labels indicating whether an account is involved in suspicious activity.

Having formatted and prepared the graph dataset, several graph methods are utilized, such as GAT (Graph Attention Network), GCN (Graph Convolutional Network), and GIN (Graph Isomorphism Network). All these three are popular types of GNNs, each designed to learn and make predictions from graph-structured data. While all three models’ aggregate information from neighboring nodes in a graph, they differ significantly in how they perform this aggregation and how expressive they are in capturing graph structures. More specifically, the first major difference is based on the aggregation mechanism. GAT uses attention-based aggregation, GCN uses weighted averaging, and GIN uses sum aggregation followed by an MLP. The second difference is the expressiveness. GIN is the most expressive in capturing graph structures, followed by GAT due to its attention mechanism, while GCN is the least expressive but more computationally efficient. Based on each method’s complexity, GAT is more complex due to the dynamic attention mechanism, GIN is also complex with its MLP-based transformations, while GCN is relatively simpler and faster to train. These differences make each model suitable for different types of tasks. GAT is ideal when node importance varies, GCN is effective for tasks where simplicity and computational efficiency are key, and GIN excels in scenarios requiring nuanced understanding of graph structure. All methods are briefly described and performed below.

GRAPH ATTENTION NETWORK (GAT)

The first method for extensive analysis is the GAT [3, 4, 5]. This method is an advanced neural network that is designed specifically for handling graph-structured data. Traditional neural networks work well with grid-like data, such as images or sequences, but graphs have a more complex structure where each node has an arbitrary number of connections or edges to other nodes. GAT addresses this by leveraging an attention mechanism that allows the model to focus on the most important neighbors for each node, assigning different weights or attention scores to each connection. The goal of a GAT is to learn meaningful representations of the nodes by considering both their own features and the features of their neighboring nodes. Its attention mechanism dynamically weighs the importance of neighboring nodes when updating the representation of a given node. Instead of treating all neighbors equally, GAT learns which nodes are more influential to the target node based on the task, such as identifying suspicious behavior in this case. The model assigns higher attention weights to important neighbors and lower weights to less relevant ones. It also uses multiple attention heads to allow the model to capture different types of relationships between nodes. Each head processes the graph independently, and their outputs are either concatenated or averaged to provide a richer representation for each node. The core of GAT is the process of aggregating information from neighboring nodes. For each node, the model collects information from its neighbors, weighs this information using the attention scores, and combines it with the node's own features to generate an updated representation. This process is repeated across multiple layers, allowing nodes to gather information from farther away in the graph such as second-degree or third-degree neighbors.

The model takes in several parameters, such as the number of input channels or features per node, hidden channels or the number of units in the hidden layer, output channels for the final predictions, and the number of attention heads used in the GAT layers. The model consists of two graph attention layers, which allow the model to focus on different parts of the graph by assigning attention weights to the edges. The attention mechanism helps the model learn which neighboring nodes are more important for the task. The first GAT layer takes the input features and computes new node representations by considering neighboring nodes and edge attributes. It uses multiple heads to capture different perspectives from neighbors. The second GAT layer further processes the node representations from the first layer, using a reduced number of heads and simplifies the representation for each node. The model also includes a fully connected linear layer at the end to reduce the node embeddings into the desired output size. A sigmoid activation function is applied at the end to output values between 0 and 1, which is particularly useful when the task is binary classification or in this case, determining whether an account is involved in money laundering.

The forward method, which is used then, defines how data flows through the network. The input features are passed through a dropout layer to prevent overfitting by randomly setting some of the features to zero during training. The first GAT layer processes the input features using the attention mechanism to compute new node embeddings based on the graph structure and neighboring node features. The output is passed through an ELU activation function to introduce non-linearity, followed by another dropout layer. The second GAT layer further processes the node embeddings to refine them, followed by another ELU activation. The final linear layer transforms the node representations into output predictions, and the sigmoid function ensures that the predictions are probabilities between 0 and 1.

The output of the model is a set of probability scores for each node in the graph. Each score indicates the likelihood of that node being classified into the target class, such as whether an account is involved in money laundering or not. In this case, the output typically corresponds to the number of target classes or a single class if it is a binary classification task. The final

output is a tensor where each value represents the model's confidence that a specific node or account is involved in suspicious behavior.

The attention mechanism also allows the model to dynamically focus on the most relevant neighbors for each node, making the GAT method particularly powerful for scenarios where relationships between entities are complex and not all connections are equally important. In the context of financial transactions, this means the model can weigh which transactions or neighboring accounts are most influential in determining whether an account is suspicious. This adaptability makes GAT highly effective for tasks like fraud detection, anomaly detection, and risk assessment in networks of interactions.

GRAPH CONVOLUTIONAL NETWORK (GCN)

The second method of extensive study is the GCN [4, 6]. This method is another approach designed to also handle graph-structured data. GCN is used for tasks like node classification, link prediction, or graph-level classification. Unlike traditional neural networks, which operate on grid-like data, GCNs work on graphs, where the relationships between nodes are represented as edges, and each node has features that describe it. The central idea behind GCN is extending the idea of convolution, which is commonly used in images, to graphs. Instead of a fixed grid of pixels, GCNs aggregate information from neighboring nodes. The convolution on graphs aggregates information from a node's neighbors and updates its features based on this aggregation. This allows GCNs to learn meaningful node representations by considering both the node's features and its neighborhood in the graph. GCNs aggregate the information from neighboring nodes to update the features of a target node. This means that for each node, the model looks at its immediate neighbors and combines their features with the node's own features. The first layer aggregates information from the first-degree neighbors, which are nodes directly connected to the target one, and subsequent layers can aggregate information from nodes further away in the graph, like second-degree neighbors. This process allows the model to capture the broader context of each node in the graph, not just the node's individual characteristics. As the model propagates through the graph, the features of the nodes are transformed using learned weight matrices, similar to how neural networks transform inputs in traditional models. Each node's updated feature representation is a combination of its own features and the features of its neighbors, weighted by their importance in the context of the task.

GCNs typically consist of multiple layers. Each layer further refines the node representations by looking deeper into the graph structure. The first layer aggregates and transforms information from immediate neighbors. In the second layer, the nodes are aware of their neighbors' neighbors, capturing a wider perspective of the graph. As the layers stack, the node's representation becomes richer, incorporating information from increasingly distant nodes in the graph. Like traditional neural networks, GCNs apply non-linear activation functions, such as ELU, ReLU, after each convolution layer. This introduces non-linearity into the model, allowing it to capture more complex patterns in the data. Additionally, dropout regularization is used to prevent overfitting. Dropout randomly ignores some node features during training, ensuring the model does not become too reliant on specific features and is forced to learn more generalized representations.

In this case, the GCN model starts with input features for each node. The input features for all nodes are stored in a matrix where each row corresponds to a node, and each column corresponds to a feature. The first graph convolutional layer takes the input features and

aggregates information from the neighboring nodes to compute a new set of features for each node. This allows each node's representation to be influenced by its neighbors. The learned weights in this layer determine how much influence each neighboring node should have on the target node's updated representation. The second convolutional layer continues this process by taking the output from the first layer and refining it further. At this point, each node's representation not only reflects its immediate neighbors but also the neighbors of those neighbors, capturing more distant information in the graph. Unlike GATs, the node features are uniformly weighted and averaged based on the graph structure. After the two convolutional layers, a fully connected linear layer is used to transform the node representations into the final output space. The output from the linear layer is passed through a sigmoid function, which maps the output values to a range between 0 and 1. In the case of binary classification, the output can be interpreted as the probability that a node belongs to a certain class.

The GCN model outputs a prediction for each node in the graph. In the case of a transaction graph, where the task is to detect suspicious accounts, the output for each node is a probability score between 0 and 1. A score close to 1 means the model predicts that the account is likely involved in money laundering, while a score close to 0 indicates that the account is not suspicious. By leveraging both the node's individual features and its relationships in the graph, the GCN model is able to make more informed predictions, taking into account the entire network of transactions. The GCN method is powerful because it learns from both the features of the nodes and the structure of the graph, making it especially effective for tasks where relationships between entities play a critical role.

GRAPH ISOMORPHIC NETWORK (GIN)

Last, the GIN method is a graph neural network designed to capture node-level representations with a focus on distinguishing between different graph structures [7]. In general, GINs are highly expressive. In the context of financial transactions, GIN can be used to learn deep node embeddings that reflect not only the node's features but also the structure of its surrounding neighborhood, making it ideal for tasks like fraud detection.

GINs are designed to address the limitations of other graph neural networks, like GCN, when it comes to distinguishing between different graph structures. GIN enhances the ability to differentiate between graph nodes by aggregating and updating node features in a more powerful way. This makes it suitable for tasks where the structural properties of nodes are important. Similar to other graph neural networks, GINs follow the principle of neighborhood aggregation. For each node in the graph, the GIN model aggregates information from neighboring nodes and updates the node's features. However, GIN uses a specific aggregation function that makes it more capable of distinguishing nodes based on their neighborhood structures. The core feature of GIN is its use of fully connected neural networks, the well-known multi-layer perceptron (MLP), within each layer. After aggregating the features from the neighbors, GIN applies a non-linear transformation using a neural network, allowing for more flexibility and expressiveness when learning node features.

Graph isomorphism refers to the concept of two graphs being structurally identical, even if their node labeling is different. GIN is designed to be able distinguish different graph structures even when the graphs have similar node distributions or relationships. This is crucial in tasks where the structure of interactions between entities may provide key insights.

GIN differs from methods like GCN is the way this aggregation is performed. GIN applies a sum aggregation, which aggregates the features of neighboring nodes by summing them This makes it more expressive in capturing the structure of the local neighborhood. This aggregation is followed by an MLP that transforms the aggregated features in a highly non-linear way. This makes the network more flexible and capable of learning complex patterns. In GIN, the key innovation lies in the use of MLPs to transform the node features after aggregation. Instead of directly averaging the features from neighbors as in GCN, GIN passes the aggregated features through a small fully connected neural network, allowing it to learn richer and more complex feature representations. This transformation makes GIN more powerful and expressive because the MLP can learn intricate dependencies between features, which is crucial for distinguishing differences between nodes. The use of sum aggregation instead of mean or max pooling is critical in GIN. The sum operation helps ensure that the model can differentiate between different sets of neighbors, even when they have similar mean values. For example, two accounts might have similar transaction volumes or average, but the sum of the transactions could reveal key differences in the behavior patterns of the accounts. GIN captures these differences more effectively than other GNNs, making it better suited for tasks where subtle variations in the graph structure are important.

In this case, the GIN architecture uses two layers, each equipped with its own MLP. These layers handle both the aggregation and transformation of node features. The first MLP takes the input node features and processes them through two fully connected layers with a ReLU activation function in between. This helps the model learn more complex, non-linear relationships between the features. The second MLP follows a similar structure but reduces the dimensionality of the hidden layer. This provides a compressed, abstracted feature representation of each node that captures the essential information after two rounds of aggregation and transformation. In the forward pass, the model takes the node features and the graph structure. The first layer aggregates the features of neighboring nodes for each node by summing them and then passes the result through the first MLP. This allows each node to update its feature representation based on the sum of its neighbors' features. After a step to prevent overfitting, the second layer performs another round of neighborhood aggregation and transformation using the second MLP, further refining the node embeddings. Finally, a linear layer transforms the node embeddings into the final output space, and a sigmoid function is applied to output values between 0 and 1. After each aggregation step, the model applies a ReLU activation function, introducing non-linearity into the learning process. Dropout is used after each layer to prevent overfitting by randomly ignoring some node features during training. This ensures that the model generalizes well to new data and does not become too reliant on specific features.

The output of the GIN model is a probability score for each node in the graph. This score represents the likelihood that a specific node belongs to a given class. For instance, in this transaction graph where the goal is to detect suspicious accounts, the output for each account is a score between 0 and 1. A score close to 1 indicates a high probability that the account is involved in money laundering or fraudulent activities, while a score closer to 0 suggests that the account is not suspicious.

DATA PREPARATION FOR GNN

Having defines the structure of each GNN method, the next part includes the process of training a graph neural network for detecting suspicious activity in this transaction dataset. The dataset

is loaded, and processed, and is converted into a graph format. The nodes are then split into three subsets, which are the training, validation, and test sets. The split is performed randomly, with 20% of the nodes being reserved for validation and another 20% for testing. The remaining 60% of the nodes are used for training the model. This ensures that the model can be evaluated on unseen data during training. Next, the data loaders are created that sample nodes and their neighbors from the graph for each batch. This sampling approach is important because it reduces the complexity of processing large graphs. This is done by only loading a subset of the nodes and their neighbors at a time. In this case, for each node, 30 neighbors are sampled across two layers, meaning it considers not only the node's direct neighbors but also the neighbors of those neighbors. Three loaders are created. For each loader, a batch size of 256 is used, meaning that during each iteration, the model processes 256 nodes along with their neighbors. This approach allows efficient training and evaluation of large graphs without needing to load the entire graph into memory at once.

The output of this code is three separate data loaders, which are responsible for providing batches of node data and their local neighborhoods during training, validation, and testing phases. These loaders enable the graph neural network model to be trained in small batches, improving computational efficiency and memory usage when handling large graphs. The overall goal is to allow the model to learn from the structure of the transaction network and predict whether accounts are likely involved in suspicious activities such as money laundering.

Having prepared and processed the dataset properly, several loss functions are checked in order to optimize the hyperparameter tuning process. The first loss function under study is the Binary Cross Entropy Loss Function (BCE), and the second one is the Focal Loss Function. Hyperparameter tuning is essential because it allows for optimization of the mode's performance by finding the best combination of hyperparameters for the task. Unlike model parameters which are learned during training, hyperparameters are external settings that directly affect how well the model learns and generalizes to new data. Poorly chosen hyperparameters can lead to suboptimal performance, overfitting, or underfitting, so tuning them is critical. Different datasets require different configurations. Some datasets may be noisy, others may be large and complex, or they may have an imbalance between classes. Hyperparameter tuning helps in adapting the model to the specific characteristics of the dataset. In general, hyperparameter tuning is a crucial step in the machine learning pipeline because it helps to create a well-performing, robust model that can be deployed with confidence.

BINARY CROSS ENTROPY LOSS FUNCTION (BCE)

The BCE loss function is frequently used during hyperparameter tuning when training models for binary classification tasks, as is this case. It plays a crucial role in evaluating the model's performance during the tuning process. It measures the difference between the predicted probability or output of the model and the actual label or ground truth. The BCE helps the model learn how close its predictions are to the true binary labels, and it is especially useful when the output of the model is a probability, typically between 0 and 1. The formula is applied to each individual data point, and then the average is taken across all data points in the batch.

One of its key components is the true label, that represents the actual outcome for a given data point. In binary classification, it is either 0 or 1. Another key component is the probability predicted by the model that the data point belongs to the positive class, which is typically obtained by applying a sigmoid activation function to the model's raw output. Lower BCE loss

indicates better model performance, meaning the predicted probabilities are closer to the true labels.

When tuning hyperparameters, the BCE loss on the validation set is often used as a primary metric to evaluate different hyperparameter combinations. After testing multiple hyperparameter configurations, the model that achieves the lowest BCE loss on the validation data is typically considered the best-performing configuration. BCE loss also helps monitor the stability of the training process. If the BCE loss is not decreasing as expected during training, it may indicate issues with the hyperparameter settings, such as a learning rate that is too high or low, poor weight initialization, or insufficient model complexity. Hyperparameter tuning adjusts these settings to minimize the loss.

During the training phase, BCE loss is computed on the training data to guide the optimization process. The model adjusts its parameters based on this loss to minimize errors and improve predictions. After each epoch or set of epochs, the BCE loss is computed on the validation set. This validation loss is critical during hyperparameter tuning because it indicates how well the model generalizes to unseen data. Hyperparameter tuning typically aims to minimize this validation BCE loss. In some cases, BCE loss can also be used with early stopping criteria. If the validation BCE loss does not improve after a certain number of epochs, the training process is stopped to avoid overfitting. Hyperparameter tuning then continues with the next set of hyperparameters.

In general, BCE Loss is central to hyperparameter tuning in binary classification tasks because it serves as the metric that helps you evaluate how well the model is performing across different hyperparameter combinations. The goal of hyperparameter tuning is often to minimize BCE loss on the validation set, ensuring that the chosen hyperparameters lead to the most accurate and well-generalized model.

MODEL TRAINING WITH BCE

By choosing the BCE criterion for hyperparameter tuning, the selected GNN model is trained on graph-structured data and performs hyperparameter tuning to optimize the model's performance. This is designed to work with the three GNN architectures i.e., the GIN, GCN, and GAT. The appropriate GNN model is initialized and is fine-tuned by adjusting the hyperparameters, such as the learning rate, hidden channels, and dropout rate, over a number of epochs. If the GAT model is chosen, additional parameters like the number of attention heads are considered during initialization. The model is trained using the Stochastic Gradient Descent (SGD) optimizer, with the learning rate specified by the user. Following this, the training process is run over a specified number of epochs. In each epoch, the model is set to training mode, and for each batch of training data, predictions are made by feeding the node features, edge connections, and edge attributes into the model. The predicted values are compared to the true labels, and the loss is computed. The optimizer updates the model's parameters by minimizing this loss. The function tracks the total training loss for each epoch. After each epoch, the model is evaluated on a validation set to calculate the validation loss and accuracy. Early stopping is implemented based on a patience parameter. If the validation loss does not improve over a few epochs, the training stops early to prevent overfitting. The model with the best validation performance is saved. During validation, the model is set to evaluation mode, where no gradients are calculated thus efficiency is increased. The validation loss and accuracy are computed to monitor the model's performance on unseen data. The output of this function

is a trained GNN model, along with the best validation loss and validation accuracy achieved during training. These metrics help determine how well the model generalizes to unseen data.

HYPERPARAMETER TUNING WITH BCE

Hyperparameter tuning is performed on the three GNN models to find the best combination of hyperparameters for each model that results in the highest validation accuracy. More specifically, three sets of hyperparameters are defined, the learning rates, which specifies how quickly the model adjusts its parameters during training, the hidden channels which control the number of neurons in the model's hidden layers, the dropout rates, which is a regularization technique used to reduce overfitting by randomly dropping some neurons during training, and hyperparameter combinations, which generates all possible combinations of the hyperparameters from the given lists. This ensures that each combination of learning rate, hidden channels, and dropout rate is tried. For each model, the function iterates through every hyperparameter combination. For each combination, the model is trained on the training data and evaluate its performance on the validation data. The validation accuracy and loss are calculated for each combination, and the function tracks the best-performing hyperparameters.

The hyperparameter tuning and evaluation of the three methods yield interesting results, particularly concerning the detection of fraudulent transactions. For the GIN model, the best hyperparameters are a learning rate of 0.001, 64 hidden channels, and a dropout rate of 0.3, leading to a validation accuracy of 0.97486 and a test accuracy of 0.97536. However, the model completely fails to detect any positive fraud cases, as indicated by its precision, recall, and F1 score of 0, which highlights issues related to class imbalance.

Similarly, the GCN model performs with a best validation accuracy of 0.97483, using 16 hidden channels, a learning rate of 0.001, and a dropout rate of 0.5. Although it achieves a test accuracy of 0.97547, it also fails to identify any fraudulent cases, as shown by the 0 precision, recall, and F1 score. This failure indicates that the class imbalance problem is causing both the GIN and GCN models to ignore the minority class during predictions.

The GAT model, with a learning rate of 0.01, 16 hidden channels, 8 attention heads, and a dropout rate of 0.3, performs slightly better in detecting laundering. It achieves a test accuracy of 0.89697 and a precision of 0.0304, recall of 0.1027, and F1 score of 0.0469. While GAT is more successful in identifying laundering cases compared to GIN and GCN, the low F1 score still indicates poor precision and recall, suggesting that significant improvements are needed.

EVALUATION OF THE THREE MODELS WITH BCE

Based on previous hyperparameter tuning results, the next step of the process involves the evaluation of the best-performing models, using their optimized hyperparameters. For each model, the best hyperparameters are applied, and the model is trained on the training set. After training, the models are evaluated on the test set, where various performance metrics are calculated, including test loss, test accuracy, precision, recall, and F1 score. These metrics provide insights into how well each model generalizes to unseen data, with a specific focus on detecting laundering transactions.

While the GIN model achieves a high-test accuracy of 97.54%, it fails to predict any positive laundering cases, as indicated by the precision, recall, and F1 score of 0. This suggests the model is heavily biased toward the majority class, which is the non-laundering transactions, indicating severe class imbalance issues.

Similar to the GIN model, the GCN model shows a high-test accuracy of 97.55%, but fails to detect any laundering cases. The much higher test loss of 2.06 compared to GIN, which is 0.23, suggests the model struggles more to fit the data, potentially overfitting on the majority class while ignoring the minority class.

The GAT model, while having a lower overall accuracy of 89.70%, compared to GIN and GCN, performs slightly better in detecting laundering cases, as seen in the non-zero precision of 0.304, recall of 0.1027, and F1 score of 0.0469. However, the performance is still poor, with many false positives and many false negatives. This indicates that although GAT is more effective in identifying laundering, it still struggles with imbalanced data.

In order to improve performance, advanced techniques such as SMOTE should be considered to address the class imbalance.

MODEL TRAINING WITH BCE AND SMOTE

In order to handle class imbalance in laundering detection, SMOTE is applied to the training data and the three different GNN models i.e., the GIN, GCN, and GAT, are trained on this resampled data. The features and labels of these resampled nodes are converted back into tensors, and the training portion of the dataset is updated with this synthetic data. Following this, a new data loader is created for the resampled training data. This loader is responsible for feeding the SMOTE-augmented data into the models during training, ensuring that both original and synthetic training data are used. The three models are trained with different combinations of hyperparameters. During each epoch, the models are trained on the SMOTE-augmented training data and evaluated on a validation set. The validation accuracy and loss are tracked, and early stopping is applied if the validation loss does not improve after a certain number of epochs. The models are then tuned using a grid search over specified hyperparameter ranges. The best combination of hyperparameters for each model is determined based on validation accuracy. For GAT, the number of attention heads is also tuned. The hyperparameters that yield the highest validation accuracy are saved for each model. After tuning, each model is evaluated on the test set to compute metrics such as test loss, test accuracy, precision, recall, and F1 score. These metrics provide insights into how well the models generalize and how effectively they identify the minority class.

HYPERPARAMETER TUNING WITH BCE AND SMOTE

Moving on to the results section, the best hyperparameters for the GIN model are a learning rate of 0.0001, 32 hidden channels, and 0.5 dropout rate. As for the test results, the test loss is equal to 14.0182, the test accuracy is of 0.81093, and the precision is equal to 0.0213. The recall is equal to 0.1486, and F1 score equals to 0.0373. Although SMOTE is applied, the GIN model struggles to detect laundering transactions, as reflected in the very low F1 score. This indicates that SMOTE alone did not perform well on handling the severe class imbalance.

In the GCN model case, the best hyperparameters are a learning rate of 0.0001, 32 hidden channels, and 0.6 dropout rate. As for the test results, the outcomes include a test loss of 44.3312, test accuracy of 0.52658, with improved recall of 0.5111 but very low precision of 0.0266, leading to an F1 score of 0.0505. The GCN model shows a significant increase in recall, meaning it detected more fraudulent cases, but at the cost of precision, indicating that many false positives are generated, which is reflected in the low F1 score and poor overall performance.

Last, for the GAT model, the best hyperparameters are a learning rate of 0.0001, 32 hidden channels, a 0.3 dropout rate, and 8 attention heads. As for the test results, the test loss is of 63.6731, the test accuracy equals to 0.30847, with the highest recall of 0.7009 but very low precision of 0.0246, resulting in an F1 score of 0.0475. The GAT model achieves the highest recall, meaning it is most successful at identifying laundering cases, but precision suffers drastically, leading to many false positives. This imbalance between precision and recall caused the F1 score to remain low.

In general, despite the application of SMOTE to balance the classes, all models struggle with laundering detection, as seen by the low F1 scores. SMOTE helps to improve recall in some cases such as for GCN and GAT, but the models still generate many false positives, leading to poor precision.

COMPARISON WITH THE HYPERPARAMETER TUNING WITH BCE WITHOUT SMOTE

The evaluation of laundering detection using BCE across the three different models provides insight into the performance of these models both with and without SMOTE. SMOTE is employed to handle class imbalance, where fraudulent transactions are underrepresented. In both cases, the models struggled to effectively predict the minority class, but the results varied in several key areas.

For the models that are trained without SMOTE, the performance, as measured by precision, recall, and F1 score, is uniformly poor. Both the GIN and GCN models fail to predict any laundering cases, with precision, recall, and F1 scores all at 0. Despite achieving high validation accuracy and strong test accuracy, these models are completely ineffective in detecting the minority class. The GAT model performs slightly better, achieving a better recall, but its precision is still extremely low, resulting in a weak F1 score.

After applying SMOTE to balance the training data, the models show some improvements, but the results are still far from ideal. The GIN model improves slightly in terms of recall, but its precision and F1 score remain low, and the test accuracy drops with a much higher test loss. Similarly, the GCN model, which achieves better recall, sees a significant reduction in test accuracy and an increase in test loss. The trade-off is that GCN has a modest F1 score but still suffers from very low precision. The GAT model, while achieving the highest recall, suffers from extremely low precision, leading to a marginal improvement in the F1 score compared to the non-SMOTE version. GAT's test accuracy also drops drastically.

As a result, models without SMOTE generally have much higher test accuracy and lower test loss compared to those trained with SMOTE. However, this high accuracy is misleading because the models are mainly classifying the majority class. SMOTE causes a substantial drop in test accuracy and an increase in test loss, but this reflects the models attempting to address the class imbalance. Without SMOTE, the GIN and GCN models completely fail to detect

laundering transactions, while GAT performs marginally better. With SMOTE, recall improves across all models, especially for GCN and GAT, but this comes at the expense of precision, leading to low F1 scores. SMOTE helps the models identify more laundering cases but also introduces many false positives, resulting in poor overall performance.

EVALUATION OF THE THREE MODELS WITH BCE AND SMOTE

The performance of three different GNN models that have been trained using SMOTE is evaluated and tuned to their best hyperparameters. The goal is to assess how well these models detect laundering transactions, particularly focusing on the minority class, which represents laundering cases. SMOTE is applied to the training data to address class imbalance by oversampling the minority class. After training, each model is evaluated on a test set, and several metrics are calculated to find the model's performance.

As far as the GIN model is concerned, it achieves moderate test accuracy of 0.8109, but it performs poorly in detecting laundering transactions. The very low precision of 0.0213 and F1 score of 0.0373 indicate that the model struggles to differentiate fraudulent cases from non-fraudulent ones. Despite the application of SMOTE to handle class imbalance, the GIN model produces a large number of false positives, reflected by the low precision.

As for the GCN model, it exhibits a significant improvement in recall with a value of 0.5111 compared to GIN, meaning it detects more laundering transactions. However, this came at the cost of precision, which remained very low at a value of 0.0266, indicating that the model produces a substantial number of false positives. The overall low accuracy of 0.5266 and high test loss of 44.3312 suggest the model struggle to generalize across the data, particularly when balancing between predicting laundering and non-laundering cases. While it identifies laundering better than the GIN model, its overall performance is still poor due to the high number of misclassifications.

Last but not least, the GAT model achieves the highest recall of 0.7009, indicating that it successfully identifies the majority of laundering cases. However, this high recall come at a severe cost to precision with a value of 0.0246, meaning the model generates a very high number of false positives, leading to a low F1 score of 0.0475. The GAT model's poor overall accuracy of 0.3085 and extremely high test loss of 63.6731 reflect its inability to generalize well, despite identifying laundering cases. The imbalance between precision and recall suggests that while GAT is sensitive to laundering detection, it lacks the specificity to effectively distinguish between the two different cases.

In summary, all models, despite the use of SMOTE to address class imbalance, continue to struggle with properly identifying fraudulent cases. The main issue observed across all models is a trade-off between precision and recall. While recall improves, particularly in GCN and GAT, it is at the expense of precision, leading to poor overall F1 scores. The GAT model performs the best in terms of recall, identifying most laundering cases, but it produces an excessive number of false positives, as reflected in its very low precision. The GCN model structs a better balance between recall and precision compared to GIN but still underperforms overall, showing difficulties in reducing false positives. The GIN model, despite having moderate test accuracy, struggles the most in correctly identifying fraudulent transactions, with very low recall and precision.

COMPARISON WITH THE EVALUATION OF THE THREE MODELS WITH BCE WITHOUT SMOTE

The evaluation of the GIN, GCN, and GAT models for laundering detection, using BCE, demonstrates stark differences between the models trained with and without SMOTE. The goal of SMOTE is to address the imbalance in the dataset, where fraudulent transactions are underrepresented, by generating synthetic examples for the minority class. However, the results show that while SMOTE helps improve recall, it introduces other challenges, notably in terms of precision and overall accuracy.

Without SMOTE, the GIN model achieves a low-test loss and a high test accuracy, but it completely fails to detect any laundering cases, as indicated by precision, recall, and F1 scores all at 0. This suggests the model heavily favors the majority class and ignores the minority class entirely. Similar to the GIN model, the GCN model has a high-test accuracy and a test loss, but again, it does not detect any laundering cases, with all metrics for fraud detection remaining at 0. This reflects an overemphasis on the majority class and an inability to generalize to the minority class. The GAT model, while showing a lower test accuracy, does marginally better in detecting laundering, achieving a precision of 0.0304, recall of 0.1027, and an F1 score of 0.0469. While still poor, these values suggest that GAT is slightly more sensitive to detecting the minority class compared to GIN and GCN, though its performance is far from satisfactory.

After applying SMOTE, the GIN model sees a significant drop in test accuracy and a sharp increase in test loss. However, it can detect some laundering cases, reflected in its precision, recall, and F1 score. While these metrics are still very low, SMOTE enables the model to identify more laundering cases compared to when it is trained without SMOTE. The GCN mode's performance also sees a drastic drop in test accuracy and a very high-test loss. However, it achieves a much higher recall, meaning it detects over half of the laundering cases, though at the cost of precision, leading to a modest F1 score. The model trades off precision for recall, detecting more fraud but generating many false positives. The GAT model exhibits the most significant changes with SMOTE, with its test accuracy and test loss. However, its recall is shot up, indicating that it detects most laundering cases, but at the expense of precision which leads to a moderate F1 score. This result demonstrates a severe trade-off, GAT becomes highly sensitive to laundering detection but generates an overwhelming number of false positives.

In general, the application of SMOTE improves the models' ability to detect laundering transactions but drastically reduce precision and overall performance, as seen in the higher test losses and lower accuracy. To improve these models, additional strategies need to be explored. Techniques like Focal Loss, which focuses more on hard-to-classify examples, could better address the class imbalance.

FOCAL LOSS (FL)

Focal Loss is a modified version of the BCE Loss that is designed to address the issue of class imbalance in classification tasks. In standard classification problems, especially when the majority class significantly outnumbers the minority one, models tend to focus on correctly predicting the majority class while neglecting the minority one. This can lead to poor performance in identifying rare or difficult-to-classify instances, such as fraudulent transactions or rare diseases.

Focal Loss introduces a focusing parameter, the gamma, that adjusts the loss contribution from easy and hard examples. For easy-to-classify examples, the loss is down-weighted, reducing their impact on the model's learning. For hard-to-classify examples, the loss is amplified, ensuring the model pays more attention to learning from these challenging cases. The goal is to focus the model on difficult examples and reduce the overwhelming influence of correctly classified, easy examples. This helps balance the learning process, particularly in cases where the model is biased toward the majority class.

The Focal Loss consists of several key components. The first one is the sigmoid activation, where the input values are passed through a sigmoid function to convert them into probabilities. Another important component is the Focal Loss term, which decreases the loss for correctly classified examples and increases the loss for misclassified ones, depending on the value of gamma. The higher the gamma, the more focus is placed on hard-to-classify examples. As for the class weights, if class weights are provided, the loss for each example is further scaled by the weight assigned to its true class. This ensures that minority classes can be emphasized even more if necessary. The loss can be aggregated using either mean or sum, depending on the reduction method specified. This procedure is particularly useful in cases of class imbalance, where one class dominates the dataset. Standard BCE loss would treat all errors equally, leading the model to focus more on the majority class. In contrast, Focal Loss allows the model to focus on difficult or minority examples, improving its ability to detect rare classes.

COMPARISON OF FOCAL LOSS WITH BCE LOSS FUNCTION

Both Focal Loss and BCE are widely used loss functions in classification tasks, particularly in binary classification problems. While they share some similarities, their differences lie in how they handle certain challenges, especially class imbalance and hard-to-classify examples.

One important advantage of BCE is that it is straightforward and works well for many standard classification tasks. It performs well when the dataset is balanced, meaning both classes are represented equally. However, BCE treats all errors equally, regardless of how frequently the classes appear in the dataset. In highly imbalanced datasets, BCE may focus too much on the majority class, leading the model to ignore the minority class. BCE does not distinguish between easy-to-classify examples and difficult ones. As a result, easy examples dominate the training process, reducing attention on the harder, minority examples.

On the other hand, Focal Loss is an enhancement of BCE designed to handle the limitations of imbalanced datasets by focusing more on hard-to-classify examples. Focal Loss is introduced to address class imbalance by down-weighting the loss contribution from easy-to-classify examples and increasing the focus on difficult, misclassified examples.

One advantages of Focal Loss is that by using the modulating factor, Focal Loss gives more importance to examples that the model finds hard to classify, like those from the minority class in imbalanced datasets. Furthermore, it automatically adjusts the loss contribution for each class, making it well-suited for scenarios with extreme class imbalance. It reduces the impact of the majority class while focusing more on learning the minority class. Moreover, the gamma parameter allows control over how much emphasis is placed on hard examples. A higher value of gamma increases this emphasis. However, Focal Loss introduces additional hyperparameters, such as gamma and optional class weights, which need careful tuning to achieve optimal performance. Also, due to the extra modulating term, Focal Loss may be

slightly more computationally expensive than BCE, although this is typically negligible for most applications.

Focal Loss is a more advanced version of BCE designed to handle class imbalance and difficult examples by down-weighting easy cases and focusing on hard examples. While BCE is simpler and works well in balanced datasets, Focal Loss is the better choice for imbalanced datasets or when the minority class is critical to detect, as it forces the model to pay more attention to these challenging cases.

MODEL TRAINING WITH FOCAL LOSS

Having comprehend the important components of Focal Loss, the next steps of the process include designing a useful code to train and fine-tune the GNN models using Focal Loss for this binary classification task. This trains the three different GNN architectures on the laundering dataset, with the aim of finding the optimal hyperparameters, such as learning rate, hidden channels, dropout rate, and, in the case of the GAT model, the number of attention heads.

For each model that is initially selected, it is initialized with specific hyperparameters, such as the number of hidden channels, learning rate, and dropout rate. For the GAT model, the number of attention heads is also specified. The key feature here is the use of Focal Loss instead of the BCE loss. During each epoch, the model is trained on batches of data. The predictions are made by the model, and the Focal Loss is computed based on the predicted values and true labels. The loss is then backpropagated, and the model's weights are updated using the SGD optimizer. After each training epoch, the total training loss is accumulated to track the model's progress. After each epoch of training, the model is evaluated on a validation set to compute the validation loss and validation accuracy. This helps monitor how well the model generalizes to unseen data. The model uses early stopping to prevent overfitting. If the validation loss does not improve for a certain number of consecutive epochs, the training process stops early. This ensures that the model does not continue training if it is no longer learning or if it is beginning to overfit to the training data. After training, the model's performance is assessed based on the best validation loss and validation accuracy. These metrics help determine which hyperparameter combination performs best.

HYPERPARAMETER TUNING WITH FOCAL LOSS

The next part includes some processes that perform hyperparameter tuning for the three different GNN models, using Focal Loss as the loss function. The primary goal is to identify the best combination of hyperparameters for each model to maximize performance on this classification task, particularly in handling class imbalance. This tuning process systematically explores different combinations of learning rates, hidden channels, dropout rates, and attention heads for the GAT model to determine which configuration leads to the highest validation accuracy.

To begin with, several key hyperparameters, such as learning rate, hidden channels, and dropout rate, are define and iterates through all possible combinations of these parameters for each model. For the GAT model, the number of attention heads is also tuned. By varying these hyperparameters, the process seeks to find the optimal configuration that will enable each model to generalize well and achieve the best validation performance. Each hyperparameter

combination is used to train the model with Focal Loss, which is specifically designed to address class imbalance by focusing more on minority class and reducing the contribution of majority class. Focal Loss ensures that the models pay more attention to the minority class during training. The models are trained for a specified number of epochs, and their performance is evaluated based on validation accuracy after each training iteration. For each model, the hyperparameter combination that yields the highest validation accuracy is recorded. The performance of each model is compared across different hyperparameter settings and stores the best-performing parameters. Once all combinations have been evaluated for a specific model, the script stores the best hyperparameters for that model, ensuring that future training can use this optimal configuration.

The results of training the three graph neural network models, using Focal Loss to handle class imbalance, are presented with the goal of improving the detection of laundering cases. The key metrics that are evaluated include validation accuracy, test loss, test accuracy, precision, recall, and F1 score. While all models perform reasonably well in terms of validation accuracy, the results show a consistent challenge with precision, which limited the overall F1 scores.

The GIN model achieves high validation accuracy of 97.50% and reasonable test accuracy of 97.49%, showing that the model generalizes well on the overall dataset. However, the model struggles with precision which is equal to 0.0157 and recall which is 0.2035 when detecting laundering cases, indicating that it still heavily favors the non-laundering transactions. The F1 score of 0.0294 reflects this imbalance, as the model is able to detect some laundering cases but generates a significant number of false positives. This suggests that, despite using Focal Loss, the model is still not focusing enough on the minority class.

As for the GCN model. it achieves the highest recall of 0.7445 among the three models, meaning it successfully detects the majority of laundering transactions. However, the precision of 0.0245 is very low, indicating that many of these detections are false positives. The model's test accuracy is also low at 0.25598, likely due to its over-sensitivity to the minority class, resulting in poor overall classification performance. The F1 score of 0.0474 reflects this imbalance between precision and recall. While the GCN model effectively identifies laundering cases, the high false positive rate severely impacts the model's usefulness.

Last but not least, the GAT model exhibits very high recall equal to 0.9047, meaning it detects almost all laundering transactions in the test data. However, like the GCN model, its precision is low and equal to 0.0276, indicating that a large portion of the predictions are false positives. This leads to a low F1 score of 0.0535 despite the model's strong performance in identifying laundering. The test accuracy of 0.20439 is also poor, demonstrating that the model, while effective at identifying fraud, is not well-balanced overall and still struggles with classifying non-laundering cases.

Across all three models, the F1 scores remains low despite using Focal Loss to address class imbalance. This is largely due to low precision scores, meaning that although the models improve in detecting laundering cases, they also generate many false positives. This suggests that while Focal Loss helps the models focus more on the minority class, it is not sufficient to fully resolve the precision-recall trade-off.

EVALUATION OF THE THREE MODELS WITH FOCAL LOSS

The performance of the three GNN models that are trained using Focal Loss is evaluated with their best hyperparameters, based on previous hyperparameter tuning. The objective is to assess

how well each model generalizes to the test data, particularly focusing on detecting laundering transactions. The evaluation metrics include test loss, test accuracy, precision, recall, and F1 score.

For each case, the model is initialized using the best hyperparameters found during the tuning process. For each model, the function computes predictions on the test set. These predictions are compared against the true labels. The predictions are thresholder to create binary class labels i.e., laundering if the prediction is greater than 0.5, and non-laundering otherwise. The Focal Loss is computed to assess how well the model minimizes errors, especially in detecting minority-class (fraud) cases.

The first model under study is the GIN one. This model achieves moderate test accuracy but struggles with detecting laundering cases, as shown by its low precision of 0.0221 and recall of 0.1325. This indicates that the model misclassifies many laundering cases as non-laundering and had many false positives when predicting laundering. The F1 score of 0.0379 reflects the model's inability to balance precision and recall effectively, making it less reliable for fraud detection.

Comment: The GCN model performs similarly to GIN in terms of test accuracy, with slightly higher at 0.8409. However, its precision of 0.0227 and recall of 0.1298 remain very low, showing that it also struggles with detecting laundering cases. The F1 score of 0.0387 reflects this imbalance between precision and recall. Like GIN, GCN struggles with many false positives, leading to a modest improvement in F1 but not enough to make the model highly effective for fraud detection.

Last but not least, the GAT model achieves the highest test accuracy of 0.9014, significantly higher than GIN and GCN. However, its recall of 0.0812 is lower than the other models, meaning it misses more laundering cases. Despite this, the precision of 0.0257 is slightly better than the other models, indicating that it produces fewer false positives. The overall F1 score of 0.0390 remained low, similar to the other models, meaning that despite better accuracy, the GAT model still struggles with classifying laundering cases effectively.

In general, all models struggle with detecting laundering cases, likely due to class imbalance in the data. Although Focal Loss is used to address this issue, it is not enough to produce strong performance in detecting fraud, as indicated by the low precision and recall scores. The GAT model shows the best overall accuracy and slightly better precision, but GIN and GCN have higher recall. This shows that GAT is better at avoiding false positives, while GIN and GCN are slightly better at detecting laundering but at the cost of more false positives. In order to improve performance, further steps like SMOTE could be explored to increase the models' precision and recall, especially in detecting rare fraudulent cases.

MODEL TRAINING WITH FOCAL LOSS AND SMOTE

SMOTE and Focal Loss are then combined to train and fine-tune the three different GNN models. The purpose of applying SMOTE is to handle the class imbalance issue by oversampling the minority class, while Focal Loss is used to focus the model's learning on difficult-to-classify examples such as laundering transactions. The process involves hyperparameter tuning for each model to determine the best combination of learning rate, hidden channels, and dropout rate for optimal performance.

SMOTE is applied to the training data to create synthetic examples from the minority class, ensuring that the model has more balanced data to learn from. This helps in addressing the class imbalance issue that typically causes models to favor the majority class. Focal Loss is used as the criterion for training the models. It includes a gamma parameter and class weights that help adjust the contribution of each class in the loss computation. The code trains the models using the SMOTE-applied training data. The models are trained for a number of epochs, and early stopping is used to halt training if the validation loss does not improve after a set number of epochs.

HYPERPARAMETER TUNING WITH FOCAL LOSS AND SMOTE

Multiple combinations of learning rates, hidden channels, and dropout rates are explored to determine which hyperparameter settings yield the best validation accuracy for each model. For GAT, the number of attention heads is also tuned.

The GIN model has the highest validation accuracy of 0.9750 and performs relatively well in terms of recall, which is equal to 0.8850, meaning it is effective at detecting a large portion of the laundering cases. However, its precision of 0.0248 is low, indicating that it generates a significant number of false positives. The F1 score of 0.0482 reflects the imbalance between precision and recall. Although the model is sensitive to detecting laundering, the low precision suggests that it struggles to differentiate between fraud and non-fraud cases effectively.

The GCN model achieves the highest test accuracy of 0.8537 but has very low recall 0.0734, meaning it misses most of the laundering cases. Its precision of 0.0145 is also the lowest among the three models. The F1 score of 0.0242 reflects this poor balance between precision and recall. The high-test accuracy is misleading here, as the model's performance in detecting the minority class is quite poor, likely because it is overly focusing on the majority class.

Last but not least, the GAT model has moderate performance across the board. It achieves a recall of 0.7851, meaning it detects the majority of fraudulent cases, but like the GIN model, it suffers from low precision of 0.0231. This indicates that it produces a high number of false positives, leading to a F1 score of 0.0449. Its test accuracy of 0.1765 is lower than GCN but higher than GIN, reflecting a slightly better balance between detecting fraud and avoiding false positives.

In general, both GIN and GAT excel at detecting fraudulent cases, as indicated by their high recall scores. However, the trade-off for this high recall is an increase in false positives, leading to low precision and modest F1 scores. All models struggle with precision, meaning they generate too many false positives. This problem persists despite the use of SMOTE and Focal Loss, which suggests that while the models become better at identifying fraud, they are not effective in filtering out false positives. Furthermore, the class imbalance continues to be a major challenge for all models, even with SMOTE and Focal Loss applied. While SMOTE improved the models' ability to detect the minority class, it also introduces more false positives, leading to poor precision and F1 scores.

COMPARISON WITH THE HYPERPARAMETER TUNING WITH FOCAL LOSS WITHOUT SMOTE

The application of SMOTE has a noticeable impact on the models, particularly in improving their recall scores. However, it also leads to trade-offs in other areas, especially precision and test accuracy. As for the GIN model, before SMOTE, it has a high-test accuracy but low recall, meaning it struggles to detect most fraudulent cases. After SMOTE, it improves the GIN model's recall dramatically, but at the cost of a significant drop in test accuracy and still low precision. While the F1 score improved slightly, the model still struggles to balance laundering detection with false positives. As for the GCN model, before SMOTE, the GCN model has the best recall but also has low precision and test accuracy, indicating a strong sensitivity to laundering detection but at the cost of false positives. With SMOTE, the GCN model sees a sharp decline in recall and precision, indicating that it misses most laundering cases and becomes overly focused on the majority class. While its test accuracy improves, the model's performance in detecting laundering worsened overall, as reflected in the low F1 score. Last but not least, before SMOTE, the GAT model has the highest recall and a relatively good F1 score. However, it still struggles with precision and has a low-test accuracy. SMOTE results in a drop in recall and precision for the GAT model, along with a slight reduction in F1 score. Test accuracy also declines, indicating that SMOTE does not offer significant improvement for this model in terms of overall performance.

While SMOTE helps to improve laundering detection, it does not necessarily improve the models' overall performance due to the continued issue of false positives. The F1 scores remains low across all models, indicating that further adjustments are needed to balance precision and recall. While it improves the ability of the GIN and GAT models to detect laundering, the models still struggle with low precision and generate many false positives. The GCN model performed worse after applying SMOTE, indicating that it may not benefit from oversampling.

EVALUATION OF THE THREE MODELS WITH FOCAL LOSS WITH SMOTE

After the hyperparameter tuning, the three GNN models are trained using Focal Loss and SMOTE. The goal is to assess how well these models, which have already been fine-tuned with the best hyperparameters, perform on the original imbalanced test set without applying SMOTE to the test data. The evaluation process computes key performance metrics like test loss, test accuracy, precision, recall, and F1 score to understand how well each model identifies fraudulent cases.

The process begins with the appropriate model selection. For the GAT model, additional hyperparameters like the number of attention heads are considered. The model is evaluated using Focal Loss on the original test data. Each model's predictions are compared against the true labels. Binary classification is performed, where predictions greater than 0.5 are classified as laundering. The test loss is computed using Focal Loss, and the number of correct predictions is tracked to calculate test accuracy.

Useful insights can be made based on the results. More specifically, the GIN model shows very high recall of 0.8850, meaning it detected a large portion of laundering cases. However, a precision of 0.0248 remains low, suggesting that many false positives are generated. The test accuracy is extremely low to 0.1381, reflecting that the model struggles to classify non-laundering cases effectively. The F1 score of 0.0482 indicates a significant imbalance between precision and recall, where the model is highly sensitive to detecting laundering but suffers from misclassifying non-fraud cases as fraudulent. Next, the GCN model achieves the highest test accuracy of 0.8537 among the models, meaning it performs well on non-laundering cases.

However, its recall is very low and equal to 0.0734, meaning it fails to detect most laundering transactions. This indicates that the model is biased toward the majority class, which is typical in imbalanced datasets. The precision of 0.0145 and F1 score of 0.0242 are also the lowest, showing that the model struggled to identify fraud effectively. Lasr, the GAT model has strong recall of 0.7851, meaning it detects most laundering cases, but at the cost of low precision which equals to 0.0231, indicating that many false positives are generated. The test accuracy of 0.1765 is low, reflecting poor performance on the majority class. The F1 score of 0.0449 is slightly better than GCN but still low, indicating that the model has not achieved a good balance between precision and recall.

In general, across all models, a trade-off between precision and recall is observed. Models like GIN and GAT excel in recall, detecting most laundering cases, but their precision remains low, meaning they produce many false positives. On the other hand, the GCN model has higher test accuracy but struggles with recall, meaning it misses most fraud cases. The F1 score for all models is relatively low, indicating a poor balance between precision and recall. The GIN and GAT models, while better at detecting laundering, suffer from low precision, causing their overall F1 scores to remain low. The GCN model, though more accurate in classifying non-fraud cases, is ineffective at detecting fraud, leading to the lowest F1 score.

COMPARISON WITH THE EVALUATION OF THE THREE MODELS WITH FOCAL LOSS WITHOUT SMOTE

The comparison between the evaluation results before and after applying SMOTE in combination with Focal Loss reveals significant differences in how each model performs, particularly in terms of balancing precision, recall, and test accuracy.

All three models show an improved recall after SMOTE, with the GIN model seeing the most significant increase, followed by GAT. This indicates that SMOTE helps the models become more sensitive to detecting the minority class. Despite improvements in recall, precision remains low across all models after SMOTE, meaning the models continues to generate many false positives. This highlights that while SMOTE improves detection of fraudulent cases, it introduces a higher number of false positives, reducing the overall precision. Also, for the GIN and GAT models, test accuracy drops significantly after applying SMOTE, which suggests that while they become better at detecting laundering cases, their ability to correctly classify non-fraudulent cases decreases. The GCN model, however, maintains high test accuracy, though it struggles with laundering detection. Furthermore, the F1 score improves slightly for both the GIN and GAT models after SMOTE, indicating a better balance between precision and recall. However, the GCN model experiences a drop in F1 score, suggesting that SMOTE negatively impacts its overall performance.

EXPERIMENTS-SETUP AND CONFIGURATION

The experiments for this project were conducted on a system equipped with an AMD Ryzen 5 5600G processor with 3.90 GHz, with Radeon Graphics, 16 GB of RAM, running a 64-bit version of Windows 11 Home with version 23H2. The project was implemented using Python, leveraging some key libraries, such as *pandas* for data manipulation, *numpy* for numerical

operations, *matplotlib* and *seaborn* for data visualization, *scikit-learn* for model evaluation and SMOTE, *xgboost* and *lightgbm* for boosting models, and *torch* for deep learning with GNNs.

RESULTS AND QUANTITATIVE ANALYSIS

In this project, three GNN models are evaluated, the GIN, GCN, and GAT. After hyperparameter tuning, the models are tested using Binary Cross Entropy and Focal Loss, both with and without SMOTE to handle the imbalanced dataset. The results indicate that despite achieving reasonable accuracy, the models struggle with recall and precision, especially in detecting the minority class or the money laundering transactions.

From the quantitative analysis, GIN consistently demonstrates the highest recall values but suffered from low precision, indicating that while it successfully flags many potential laundering transactions, it also produces a high number of false positives. GCN and GAT, on the other hand, have even more issues with false positives and negatives, leading to overall lower F1 scores.

As a result, GIN achieves the best validation accuracy and recall but low F1 scores indicate a struggle with precision. GCN has the highest test accuracy but performs poorly on recall. GAT manages a higher recall than GCN but still underperforms in terms of F1 score due to low precision. In summary, the basic GNN models do not perform well on the imbalanced dataset despite using techniques like SMOTE and Focal Loss. These models require further refinement or more advanced techniques like message-passing mechanisms to improve their ability to detect minority classes.

QUALITATIVE AND ERROR ANALYSIS

The core challenge in this project lies in the extreme imbalance between the classes, with laundering transactions making up only a small fraction of the overall dataset. While SMOTE helps to balance the training data, the models still struggle with detecting the minority class effectively. GIN, while more balanced in terms of recall, has too many false positives. This is evident from its poor precision. Both GCN and GAT exhibit high recall values at the cost of precision, indicating that while the models are sensitive to laundering transactions, they frequently misclassify legitimate transactions as suspicious.

DISCUSSION, COMMENTS AND FUTURE WORK

The experiments with simple GNNs demonstrate the limitations of these models in handling highly imbalanced datasets. Despite using techniques like SMOTE and Focal Loss, the models are unable to achieve high precision or balanced F1 scores. This underscores the need for more sophisticated methods that can better manage imbalanced datasets in graph-based scenarios.

As part of the next steps, exploring advanced methods like message-passing or reverse message-passing GNNs is proposed, as suggested by recent literature such as the paper

“Provably Powerful Graph Neural Networks for Directed Multigraphs”. These methods have been shown to improve performance in similar tasks and could help in building a more effective laundering detection system.

Furthermore, while the simpler GNN models are firstly chosen to be tried, as graph enthusiasts and students, the importance of progressively experimenting with more advanced techniques is recognized to deepen the understanding and potentially enhance the results in future iterations of this project.

MEMBERS AND ROLES

One of the key aspects of this project was the effective allocation of tasks necessary for its successful completion. Both team members collaboratively undertook the initial phase, which involved descriptive analysis and an in-depth study of the dataset. This approach was adopted to ensure a comprehensive understanding of the dataset, as meaningful analysis comes from a deep understanding of the matter.

The implementation of machine learning algorithms was divided into two segments. Evrydiki Vrouvaki was responsible for the application of the XGBoost and LightGBM methods, while Konstantinos Giannakos focused on the GNN algorithms. Despite this division, all computational results were reviewed and discussed by both team members. At each project milestone, extensive discussions were conducted to ensure accuracy and clarity of the results. Ultimately, the success of the project was achieved through consistent collaboration and teamwork.

TIMEPLAN

The project timeline was established at the beginning of this project. Throughout this period, the project team maintained regular communication via web calls to ensure alignment on processes, provide updates on task statuses, and finalize deadlines for each assigned task.

In the initial weeks, the team’s primary focus was conducting research on various websites and reviewing of scientific papers to identify a compelling and potentially business-relevant idea for the project. Once the topic was selected, the team shifted its attention to data-related tasks. Data collection and processing were prioritized, followed by a thorough investigation of suitable models for GNN. The objective was to identify models that would offer optimal performance and results.

During the latter half of the project period, the team’s efforts concentrated on the development and implementation of the code, evaluation of the models based on their performance, and the preparation of the final report. The project timeline is briefly described in Table 3.

Task	Start Date	End Date	Duration
Research for the main purpose of this project.	13/07/2024	21/07/2024	9 days

Finalization of the project idea and Setup of the timeline.	22/07/2024	28/07/2024	7 days
Data Collection, data processing, model investigation.	29/07/2024	31/07/2024	3 days
Code implementation, report writing.	01/08/2024	31/08/2024	31 days
Final Evaluation, report writing.	01//09/2024	22/09/2024	22 days
Report finalization.	23/09/2024	29/09/2024	7 days

Table 3: Project timeline.

BIBLIOGRAPHY

[1] Dwivedi, V. P., Joshi, C. K., Luu, A. T., Laurent, T., Bengio, Y., & Bresson, X. (2023). Benchmarking graph neural networks. Journal of Machine Learning Research, 24(43), 1-48.

[2] Waikhom, L., & Patgiri, R. (2021). Graph neural networks: Methods, applications, and opportunities. arXiv preprint arXiv:2108.10733.

[3] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems, 32(1), 4-24.

[4] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... & Sun, M. (2020). Graph neural networks: A review of methods and applications. AI open, 1, 57-81.

[5] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.

[6] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[7] Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How powerful are graph neural networks?. arXiv preprint arXiv:1810.00826.

[8] Egressy, B., Von Niederhäusern, L., Blanuša, J., Altman, E., Wattenhofer, R., & Atasu, K. (2024, March). Provably Powerful Graph Neural Networks for Directed Multigraphs. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 38, No. 10, pp. 11838-11846).

[9] Altman, E., Blanuša, J., Von Niederhäusern, L., Egressy, B., Anghel, A., & Atasu, K. (2024). Realistic synthetic financial transactions for anti-money laundering models. *Advances in Neural Information Processing Systems*, *36*.