



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Self-Supervised Learning with Graphical Context to Effectively Capture Complex Money Launder- ing Activities

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING - INGEGNERIA IN-
FORMATICA

Author: **Alen Kaja**

Student ID: 222795

Advisor: Prof. Davide Martinenghi

Co-advisors: Prof. Marwan Hassani (TU/e), Haseeb Tariq (TMNL)

Academic Year: 2023-24

Abstract

Money laundering poses a persistent challenge for financial institutions worldwide, as criminals continually adapt to avoid detection. With banks processing massive amounts of transactions daily, identifying subtle laundering activities remains difficult. Many existing systems address this problem, mainly relying on predefined rules and specific behavioral patterns. However, such methods often lead to costly investigations of false positives and missed suspicious activities. Anti-money laundering (AML) regulations are driving the development of advanced technologies to support and improve current systems. With abundant unlabeled financial data and limited labeled datasets, the focus is shifting toward unsupervised anomaly detection. Additionally, graph analysis approaches are creating new research opportunities and challenging existing paradigms. This thesis explores a self-supervised anomaly detection framework for detecting money laundering transactions in financial transaction datasets. Our two-stage approach begins with an Isolation Forest model to identify a set of normal transactions, forming a baseline representation of typical transaction behavior. In the second stage, we build a temporal graph of sequential transactions and train a Graph Neural Network (GNN) model on this graph, focusing solely on the normal transaction subgraph. The trained model is then applied to the full graph, generating prediction scores that highlight outliers. These scores, combined with transaction amounts, create a comprehensive risk indicator. The model's effectiveness is evaluated by ranking transactions based on this risk score, with detection accuracy measured at the transaction level. We primarily experiment with synthetic financial transaction datasets and assess the usability of a real-world aggregated dataset. We focus on assessing the accuracy of our framework in detecting typical money laundering patterns and measuring its effectiveness and limitations. The results demonstrate that the self-supervised paradigm, combined with innovative graph structures, holds promise for advancing AML capabilities.

Keywords: Anti-Money Laundering, Self-Supervised Learning, Temporal Graphs, Anomaly Detection

Abstract in lingua italiana

Il riciclaggio di denaro rappresenta una sfida continua per gli enti finanziari mondiali, in quanto i criminali si adattano per evitare l'individuazione. L'identificazione di tali attività risulta arduo, data l'enorme quantità di transazioni che le banche elaborano ogni giorno. Molti sistemi esistenti affrontano il problema affidandosi principalmente a regole predefinite e modelli comportamentali. Tuttavia, tali metodi portano spesso a costose indagini su falsi positivi e al mancato rilevamento di attività sospette. Pertanto, le normative antiriciclaggio stanno guidando lo sviluppo di tecnologie avanzate per migliorare i sistemi attuali. Vista la disponibilità di dati finanziari non etichettati e la scarsità di dataset etichettati, l'attenzione si sta spostando sul rilevamento di anomalie in modo non supervisionato. Sviluppi nella ricerca e nell'analisi dei grafi stanno offrendo nuove soluzioni avanzate. Questa ricerca esplora un approccio di rilevamento di anomalie auto-supervisionato per individuare transazioni di riciclaggio di denaro in dataset di transazioni finanziarie. Il nostro approccio in due fasi inizia con un modello Isolation Forest (IF) per identificare un insieme di transazioni normali, o lecite. Nella seconda fase, costruiamo un grafo temporale di transazioni sequenziali e addestriamo un modello di Graph Neural Network (GNN) esclusivamente sul grafo ottenuto delle transazioni normali. Il modello addestrato viene quindi applicato all'intero grafo, generando punteggi di anomalia per ogni transazione. Questi punteggi, insieme agli importi delle transazioni, creano un indicatore di rischio. L'efficacia del modello è valutata classificando le transazioni in base al punteggio di rischio e l'accuratezza del rilevamento è misurata a livello di transazione. Gli esperimenti si basano su dataset sintetici di transazioni e su un dataset reale aggregato. Valutiamo l'accuratezza dell'approccio nel rilevare tipici schemi di riciclaggio e ne misuriamo efficacia e limitazioni. I risultati dimostrano che il paradigma auto-supervisionato, combinato con strutture di grafo innovative, offre potenziale per avanzare le capacità del sistema antiriciclaggio.

Parole chiave: Antiriciclaggio, Apprendimento Auto-Supervisionato, Grafi Temporali, Rilevamento di Anomalie

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
1 Introduction	1
1.1 Outline	4
2 Background	5
2.1 Preliminaries	5
2.1.1 Money Laundering	5
2.1.2 Anti Money Laundering	7
2.1.3 Anomaly Detection	8
2.1.4 Financial Transaction Data	10
2.1.5 Money Laundering Patterns	11
2.2 Definitions	12
2.3 Problem Formulation	15
2.3.1 Goals and Challenges	15
2.3.2 Problem Definition	17
3 Related Work	19
3.1 General Approaches	19
3.2 Machine Learning in AML	21
3.2.1 Supervised Anomaly Detection	21
3.2.2 Unsupervised Anomaly Detection	23
3.3 Graph-based Anomaly Detection	26
3.4 Summary	29
4 The Method	31

4.1	Framework Architecture	31
4.2	Data Preprocessing	32
4.3	Feature Engineering	34
4.3.1	Temporal, Aggregated and Network Features	34
4.3.2	Graph Feature Preprocessor	37
4.3.3	Feature Selection	38
4.4	Unsupervised Filtering of Transactions	39
4.4.1	Isolation Forest	39
4.4.2	Filtering Threshold Selection	42
4.5	Graph Representation Learning	42
4.5.1	Temporal Graph of Sequential Transactions	43
4.5.2	GNN-based Outlier Detection Model	45
4.6	Anomaly Detection Step	47
4.7	Summary	49
5	Experimental Setup	51
5.1	Datasets	51
5.1.1	Synthetic Data	52
5.1.2	Real-world Data	54
5.2	Parameters	56
5.3	Hyperparameters	56
5.4	Setup	57
6	Experimental Evaluation and Results	59
6.1	Goals	59
6.2	Metrics	60
6.3	Results	61
6.3.1	Dataset Statistics	61
6.3.2	Stage 1: Isolation Forest Model	62
6.3.3	Stage 2: GNN-based Models	64
6.3.4	Scalability	66
6.3.5	Real-World Data Experiment	70
6.4	Summary	73
7	Conclusions and future developments	75
7.1	Conclusions	75
7.2	Limitations	76
7.3	Future Directions	76

Bibliography	79
List of Figures	87
List of Tables	89
Acknowledgements	91

1 | Introduction

Money can be considered the lifeblood of the global economy, flowing through the hands of individuals and businesses on a daily basis. In today's world, the rise of digitalization and the advent of new technologies are shaping a new financial infrastructure, which allows for faster, more interconnected, and increasingly digital payments. While cash usage is declining rapidly, instant payments and digital wallets are on the rise [5]. Every bank, depending on its size, processes from millions to billions of transactions every year. Although the majority are legitimate, only a small proportion of these transactions are criminal transactions, and estimates show that between 2-5% of global GDP, approximately €715 billion to €1.87 trillion, is laundered annually [60]. This is the well-known problem of money laundering, which affects society as a whole, by exploiting the financial institutions. Money laundering involves executing financial transactions to disguise the origin of illegally obtained proceeds, to make them appear legitimate [21]. Typically these proceeds can be traced to crimes or illegal activities such as drug trafficking, illegal arms sales, and tax evasion. Usually, a complex series of transactions and money flow schemes, crossing multiple banks and countries are employed to obscure the origin of such funds. Money laundering has been addressed and identified for years, and it is not a new problem, rather it is continuously evolving, highlighting the adaptability and resilience of financial criminals. Anti-Money Laundering (AML) is the set of laws, regulations, and systems that aim to combat the abuse of the financial system by preventing, detecting, and stopping the process of money laundering. Driven by increasingly stringent regulations and the need for financial institutions to comply with global standards, the AML market is expected to grow at a CAGR of 16.0% from 2024 to 2030 [26]. The requirements for new advanced systems are fueling the research in the field and the adoption of technological data-driven solutions.

Currently, financial institutions contrast money laundering by implementing measures at different stages according to a risk-based approach, meaning that the extent and depth of the monitoring are assessed depending on the customer or entity risk profiles. Know Your Customer (KYC) and Customer Due Diligence (CDD) are ongoing practices that involve continuous monitoring of behaviors and risk assessment. In particular, transaction

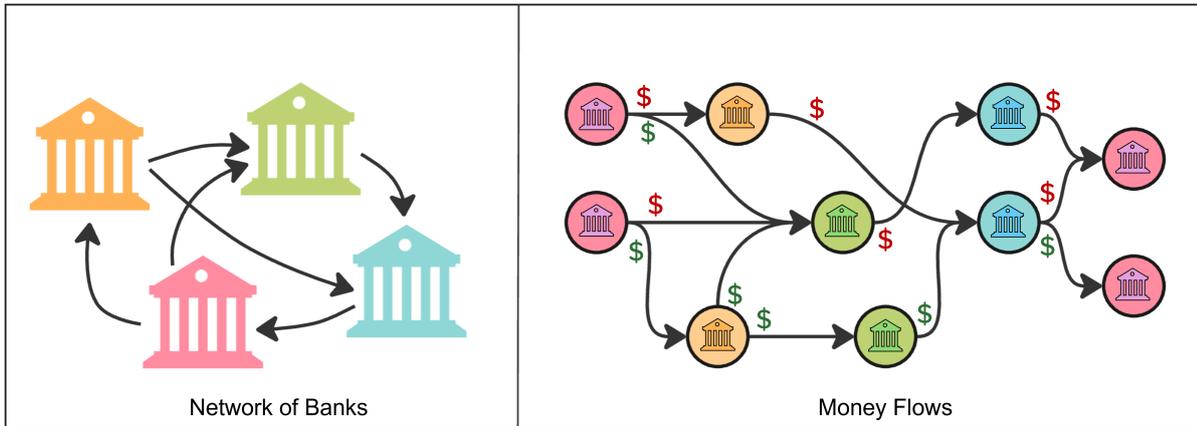


Figure 1.1: Network of banks and money flows. On the left panel, money transfers happen between banks. On the right, the money flow is detailed, illustrating how accounts within each bank (represented by circles) interact. The result is a complex money transfer chain. The exchanges may involve dirty money which criminals will try to clean. In this example, some funds eventually return to the originating bank after passing through intermediary accounts, mixing legitimate with illicit transactions. Exchanges often follow specific patterns, such as structuring or funneling.

monitoring (TM) is a critical process, predominantly based on filtering rules and dynamic thresholds. TM is carried out by analysts and automated systems to detect suspicious activities, which can be a starting point for further client investigation.

Despite the deployment of these systems, on average 95% of the generated alerts are false positives [53], and most of the anomalous activities remain undetected. This leads to unsustainable compliance costs, raising the need for tools able to prioritize anomalous transactions and reduce false positives. There are several factors which limit the effectiveness of current approaches. In general, banks have access exclusively to data regarding their own customers, with a limited context on the external relationships. Additionally, data quality is a concern. Investigations are time-consuming and banks rely on the feedback of financial authorities to record the outcome of an alert. As a result, labels for suspicious transactions are often incomplete and are only established over time.

Nowadays, domain experts are suggesting incorporating machine learning as an add-on tool, on top of the current rule-based methods [49]. On a positive note, Transaction Monitoring Netherlands (TMNL) [48] emerged as a unique initiative of the five major Dutch banks to tackle financial crime collaboratively. They employ smart machine-learning models to identify suspicious activities from bank transactions.

Broadly, machine-learning techniques can be classified as supervised or unsupervised.

Since money laundering often involves new and evolving patterns and data is usually unlabeled, it can be framed as an unsupervised problem. Specifically, it can be viewed as an anomaly or outlier detection problem because it involves identifying rare, suspicious transactions hidden within a vast volume of legitimate ones. The interest in anomaly detection has increased in recent years, pushed by the growing volume of data, information, and the associated threats [59]. With graph data becoming ubiquitous, the research community has proposed novel technologies focusing on graph structures and their applications in anomaly detection [2]. Given the relational nature of financial networks, graph-based methods can offer significant advantages in detecting hidden patterns of illicit behavior that traditional approaches may miss. However, despite the potential benefits, graph-based anomaly detection methods for anti-money laundering (AML) are still in an early phase of development and their application to AML remains limited. To be integrated into real-world AML systems, these methods require further exploration and validation [36].

To address these limitations, in this research thesis, we explore a self-supervised learning framework for identifying anomalous transactions in transaction data. In particular, our framework targets money transfer patterns that commonly relate to money laundering, such as cycles, fan-in, fan-out, and suspicious flow typologies. Without resorting to labels, we process transaction datasets to identify normal behavior and detect deviating anomalous behavior. The objectives of our approach are twofold. First, we aim to minimize false negatives by maximizing the recall of anomalous transactions. Second, we focus on reducing false positives by refining the detection process to identify the most relevant subset of suspicious transactions, thus enhancing the precision of the system. To achieve this, we investigate a first model to identify normal transactions, by producing an anomaly score for each transaction. This simple model aims to find a reasonable number of normal transactions to learn the normal representation. Next, we explore a graph-based model that leverages this knowledge to learn normal transaction behavior, thereby scoring and ranking anomalous transactions. This model uses a temporal graph of sequential transactions and outputs an anomaly score for each transaction. Finally, we perform anomaly detection on the scores of an isolated set of potentially suspicious transactions. These scores are combined with the transaction amount to rank potentially suspicious transactions more effectively. The final output is an optimized queue of potentially suspicious transactions, prioritized for further investigation.

This study aimed to address the following research questions:

- **RQ1** *How can the integration of unsupervised transaction filtering and anomaly detection on transaction graphs be optimized to effectively capture complex money*

laundering behaviors?

- **RQ2** *How can we leverage a temporal graph of sequential transactions for unsupervised anomaly detection?*
 - **RQ2.1** *How does the temporal graph perform with different scale of data, and what are its key requirements?*
- **RQ3** *How effective and efficient are the different anomaly detection models in our proposed pipeline for self-supervised money laundering detection?*
 - **RQ3.1** *Is the proposed approach scalable for detecting anomalous transactions in large-scale financial transaction graphs?*

While *RQ1* and *RQ2* are primarily explored in the methodology chapter, *RQ3* is addressed through the experimental evaluation, where the results are assessed. We experiment with a set of synthetic datasets, where labels are available to evaluate the pipeline. Additionally, we test a real-world dataset, which offers limited and aggregated information, lacking the ground truth, to give additional insights.

1.1. Outline

The remainder of this thesis is organized as follows: Chapter 2 provides essential background information related to money laundering, anomaly detection, useful definitions, and problem formulation. In Chapter 3, we review related work and state-of-the-art methods for anomaly detection in anti-money laundering, positioning our approach within the existing literature. Chapter 4 details the methodology used in our approach, outlining the pipeline implemented. Chapter 5 describes the experimental setup and datasets, describing the sources and steps applied. Chapter 6 evaluates the performance of the proposed method, presenting results and analyzing its effectiveness. Finally, in Chapter 7 we summarize our findings, discuss their implications, and suggest directions for future research.

2 | Background

This chapter serves as a platform to introduce and define the business topic. We expand and deepen the understanding of money laundering, anti-money laundering, and anomaly detection. We begin by offering an overview of the money laundering problem, and the preventive measures in place. Then we introduce the concept of anomaly detection within the context of anti-money laundering. Additionally, we describe the type of data of our interest and common money laundering patterns, followed by formal definitions of the main structures and graph notations that will be used throughout the analysis. Finally, we state the problem definition, along with the associated goals and challenges.

2.1. Preliminaries

This section outlines the key concepts required to understand the business context upon which our methodology is built.

2.1.1. Money Laundering

Money laundering is defined as the process of disguising the proceeds of crime to conceal their illicit origin and legitimize their future use. The primary objective of these activities is to conceal the true ownership and origin of these assets, which typically represent economic benefits derived directly or indirectly from predicate offenses. These offenses are the foundational crimes that precede money laundering activities. The intergovernmental organization Financial Action Task Force (FATF), a global authority combating money laundering and terrorist financing, identifies a wide range of predicate offenses that include crimes such as drug trafficking, fraud, corruption, and tax evasion [22]. The necessity to conceal illicit gains drives the need for criminals to maintain control over these proceeds or to alter their form.

Nowadays, such activities are predominantly carried out by Professional Money Launderers (PML). These individuals or entities possess specialized expertise that enables them to identify loopholes, exploit opportunities, and assist criminals in legitimizing illicit gains.

PMLs operate by providing services to criminals or Organized Criminal Groups (OCGs) in exchange for a fee or commission. PMLs can belong to three distinct categories: (i) individual PMLs, (ii) Professional Money Laundering Organisations (PMLOs), and (iii) Professional Money Laundering Networks (PMLNs). PMLs are mostly concerned with the destination of illicit funds and the mechanisms by which these funds are moved. Usually, their operations are on a large scale, involving transnational schemes that extend across borders.

According to FATF [23], the money laundering process typically unfolds in three distinct stages, as outlined in Figure 2.1. Each of these stages involves specific techniques aimed at obfuscating the source of the funds, by complicating the trail of transfers. Below is an explanation of each, highlighting the key processes and techniques commonly used to conceal the origins of illicit funds.

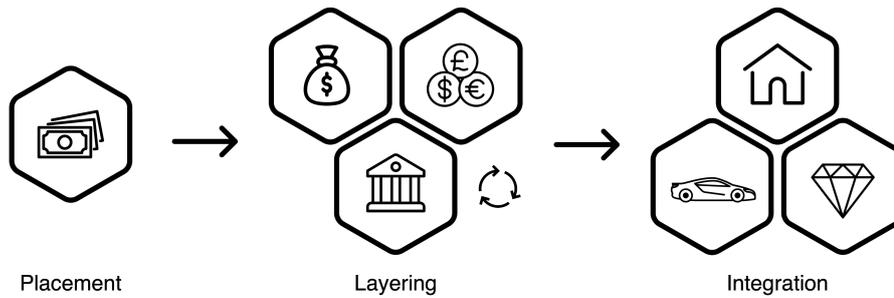


Figure 2.1: Illustration of the money laundering stages.

1. Placement Stage: criminal proceeds are transferred to, or collected by, PMLs

In the first stage, funds are transferred by physical or electronic means to PMLs or entities operating on their behalf. The introduction of funds into the scheme depends on the type of predicate offense and the way criminal proceeds are generated.

When illicit proceeds are introduced as **cash**, they are usually handed to a cash collector who may deposit the cash into bank accounts. The collector introduces the cash into the financial system through cash-intensive businesses or physically moves the cash to another region. Other criminal activities generate illicit proceeds held in **bank accounts**, such as fraud, embezzlement, and tax crimes. These proceeds rarely start as cash, but may end up as cash after being laundered. Usually, legal entities are established under whose names bank accounts may be opened for the purpose of laundering funds. Ultimately, criminals who obtain funds as **virtual currency** must have e-wallets or an address registered on a distributed ledger platform, which can be taken over by PMLs.

2. Layering Stage: funds moved by individuals and/or networks

In this stage, PMLs exploit account settlement mechanisms to make it hard to trace the funds. The layering stage is managed by individuals responsible for the coordination of financial transactions.

ML mechanisms for layering illicit proceeds earned in **cash** commonly include Trade-based money laundering (TBML) and fictitious trade, account settlements, and underground banking. Funds that were transferred to **bank accounts** are in most cases moved through complex layering schemes or proxy structures. Proxy structures involve an elaborate network of shell company accounts set up both domestically and internationally. By pooling money from various clients into the same accounts, it becomes challenging to trace the origin of funds associated with a specific client. Perpetrators of cybercrime or computer-related fraud, along with those selling illegal items online, frequently use money mule networks. The proceeds of their activities are typically maintained as **virtual currency**. These funds undergo a series of intricate transfers to obscure their origins and movements.

3. Integration Stage: laundered funds are returned to clients for investment or asset acquisition

In the last stage, funds are transferred to accounts controlled by the clients of the PML, their close associates, or third parties acting on their behalf. The PML may invest the proceeds on behalf of the clients in real estate, luxury goods, and businesses abroad. The funds can also be spent on goods deliveries to a country where the funds originated to or to a third country.

2.1.2. Anti Money Laundering

Anti-money laundering (AML) refers to a set of policies, laws, and procedures designed to prevent and detect money laundering activities. Financial institutions, serving as gatekeepers, are responsible for implementing these measures to safeguard the financial system. Among the key AML mechanisms, transaction monitoring plays a pivotal role, as it is one of the main measures adopted for detecting unusual behavior in customers. This monitoring is critical in the layering stage money of laundering, where illicit funds travel the most. However, a significant challenge arises when critical transactions are missing from the flow, creating gaps that make it harder to trace the complete laundering trail. Money laundering presents a unique complex challenge for anomaly detection systems. Its complexity stems from a variety of laundering techniques, where it is not realistic to

define a universal framework for detection. The lack of standardized regulations across countries further complicates the identification of fraudulent activities. Additionally, there is a blending effect of illicit transactions with legitimate ones. This blending makes it challenging, even for advanced automated detection systems, to differentiate between a true hit and a false positive.

2.1.3. Anomaly Detection

Anomaly detection (AD), also called outlier detection, is the branch of data mining that addresses the problem of identification of observations, events, or data points, called outliers, that deviate from the expected behavior of the dataset they belong to. Anomaly is a broad term that includes both outliers and novelties. It is used to define any abnormal instance in any context. In literature, we can categorize anomalies into three types, which are illustrated in Figure 2.2.

a) A **point anomaly** refers to an instance when a single data entry exhibits characteristics that are different from the majority of the dataset. For example, if a customer who typically deposits amounts of around €500, suddenly performs a deposit of €10K in a single transaction, this would be considered an outlier or an unusual occurrence.

b) A **contextual anomaly** arises when an observation behaves unusually only within a specific context or timeframe. This type of anomaly is often linked to data that varies over time. For instance, if a business reports monthly revenues between €50K and €70K, but records a revenue of €120K during a period with no justifiable increase in activity, this discrepancy would be classified as a contextual anomaly, potentially indicative of money laundering.

c) A **collective anomaly** occurs when a group of data points or transactions, individually unremarkable, collectively exhibit an abnormal pattern in comparison to the broader dataset. For example, a series of small, structured deposits below the reporting threshold across multiple accounts would be deemed anomalous, suggesting *smurfing*, a common technique in money laundering.

Generally, AD is a data-driven method capable of detecting unknown and undetected behavior. It does not rely on the expert description but relies purely on the data. Each data is assessed based on feature information or broader context. The complexity of AD models varies, ranging from simple algorithms to those capable of handling high-dimensional datasets in different input formats. AD has a range of applications and use cases across numerous industries, showcasing versatility and critical importance. For instance, it is used in manufacturing to monitor equipment and to identify malfunctions or defects. AD

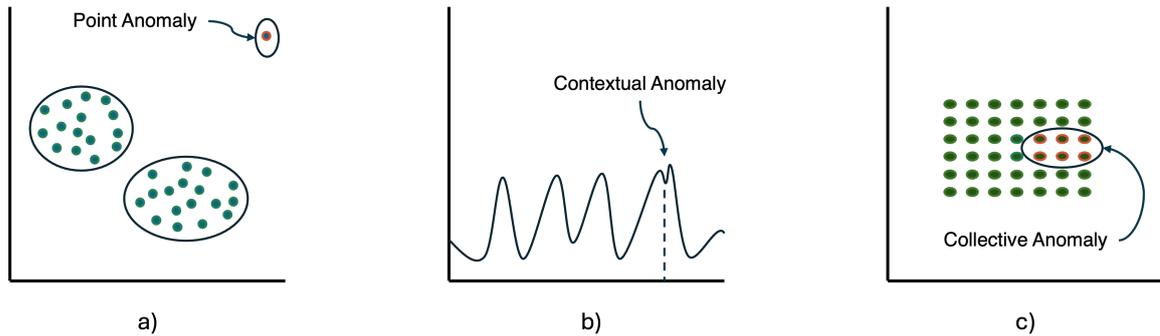


Figure 2.2: Illustration of different types of anomalies in data.

algorithms analyze sensor data and production metrics to prevent costly downtime due to unexpected equipment failure. In the realm of cybersecurity, intrusion detection systems use AD to identify unusual or suspicious user activities and network traffic patterns. Retail and e-commerce also benefit from AD, which is employed to detect unusual customer behavior patterns. In the banking, insurance, and stock trading industries, AD models are extensively used to identify fraudulent activities. Fraud detection focuses on protecting financial institutions and their customers from financial losses. The detection is usually more straightforward in case a ground truth is available. For instance, fraud can often be directly identified from the victim’s transaction records. In contrast, the detection of money laundering is far more complex and concealed. Money laundering rarely benefits from ground truth clarity. Compliance with legal and ethical standards is needed, and identifying suspicious transactions goes beyond spotting irregular transaction discrepancies. In AML efforts, there are many scenarios in which a transaction might appear anomalous, but there is no universal guideline or threshold to classify them. This underlying complexity makes AML detection a more intricate process.

Unknown Patterns. In the AML world, it is common to encounter the concepts of *known knowns*, *known unknowns*, and *unknown unknowns*. These terms are often used to categorize modus operandi or specific cases of suspicious activities. **Known knowns** refer to scenarios where established patterns are identified using traditional machine learning and domain expertise, translated into business rules. For example, financial institutions may already be aware of a specific money laundering modus operandi, allowing them to create models that effectively detect similar cases based on well-understood typologies. **Known unknowns** represent areas where uncertainty remains despite having some prior information. While red flags can suggest the presence of illicit networks, the extent and their evolution remain unclear. In this scenario, probabilistic models are employed to quantify the uncertainty, allowing us to anticipate and mitigate risks. For instance,

a model might discover suspicious networks, where some align with known cases, while others are entirely novel. However, the biggest challenge lies in the **unknown unknowns**. These refer to the hidden, unforeseen patterns that traditional rule-based systems are ill-equipped to detect. Money launderers exploit these gaps by deviating their operations from typical money laundering activities. Traditional systems, often miss these atypical transactions.

To combat *unknown unknowns*, financial institutions are shifting to the adoption of new tools, able to analyze vast amounts of data and context. By uncovering subtle irregularities, such tools can unearth well-hidden financial maneuvers that fuel launderers. AD with the integration of state-of-the-art models, offers significant potential in identifying unknown patterns of suspicious behavior. Finding unknown unknowns is essential not only for protecting banks from reputational damage but also for avoiding legal repercussions. Recalling false negatives is a crucial concern for financial institutions, which is sometimes more valuable than reducing false positives and investigating fewer cases.

2.1.4. Financial Transaction Data

Financial institutions consolidate information from multiple data sources into a centralized repository or data warehouse, to extract knowledge at various levels. This data repository is crucial for risk-based transactional monitoring and is not limited to transactional data alone. It also includes customer details, account information, and entity specifics. This comprehensive information is essential for business intelligence, serving as a valuable resource for analytics and the generation of actionable insights.

In this thesis, the primary datasets consist of financial transaction records, which are utilized both as tabular data and in a network representation. As illustrated in Figure 2.3, the input data contains key features including a unique transaction identifier, the account identifiers of both the sender and the receiver, and the amount being exchanged. Additionally, time-related data, typically in the form of a timestamp, is integral to the dataset. This level of granularity in the timestamp can vary across datasets. In the most detailed cases, timestamps include year, month, day, and at least hour and minute precision. In contrast, in aggregated datasets, time information may be presented as a broader interaction period between two accounts. This core structure forms the foundation of our analysis, enabling the creation and extraction of meaningful features. Furthermore, based on the available metadata of each specific transaction dataset, we adapt our research framework to exploit the available information from each dataset to its best potential.

Transaction ID	Source	Target	Amount	Timestamp
T1	A	B	3250	2024/08/09 14:31
T2	B	C	950	2024/08/09 15:10
T3	B	D	2200	2024/08/10 09:45
T4	A	B	1950	2024/08/12 02:47
T5	A	K	1800	2024/08/13 08:41
T6	B	D	3100	2024/08/17 15:57

Figure 2.3: Sample of financial transaction data.

2.1.5. Money Laundering Patterns

Money laundering does not always adhere to specific and recognizable patterns or structures. However, previous research [57] has attempted to model the three stages of money laundering - placement, layering, and integration - into a set of known patterns or suspicious typologies. These typologies simulate realistic suspicious behaviors and reflect common techniques used by criminals to disguise illicit funds. Common patterns include layering, round-tripping, smurfing or structuring, and cycle or funnel transactions. In this research we will focus on detecting instances of transactions potentially belonging to these specific patterns.

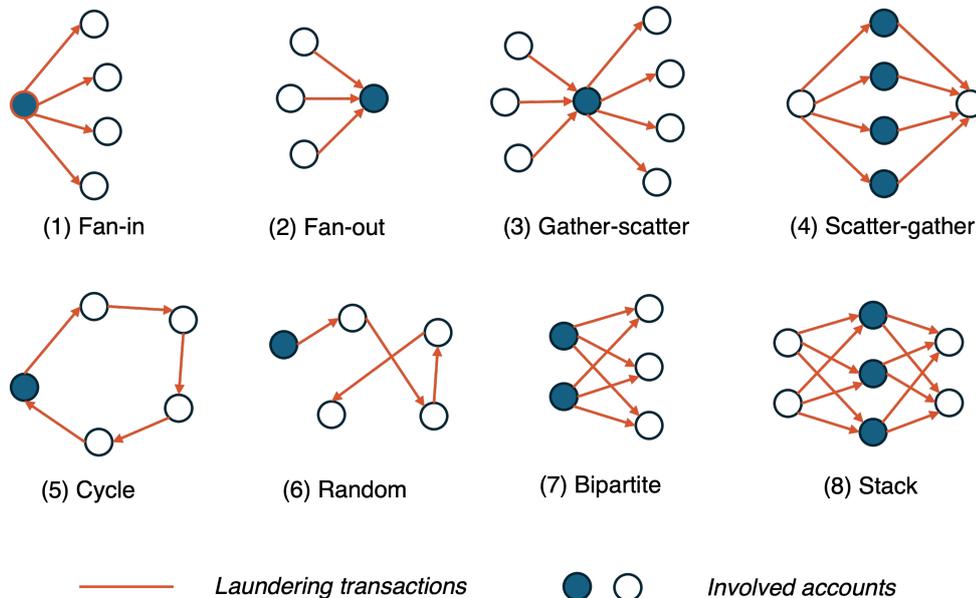


Figure 2.4: Common money laundering patterns injected in synthetic datasets.

2.2. Definitions

This section introduces the key definitions and notations necessary for understanding the proposed framework. These concepts act as foundational building blocks for the methodology outlined in chapter 4. Following the explanation of financial transaction data, we begin by identifying the transaction dataset, or tabular data. Then, from the transaction dataset, we construct a transaction graph.

Definition 2.2.1 (Transaction Dataset). *A transaction dataset is a two-dimensional matrix D where each row represents a unique financial transaction, and each column represents an attribute of that transaction. Formally, $D \in \mathbb{R}^{n \times k}$, where n is the number of transactions (or the dataset size) and k denotes the number of attributes.*

Definition 2.2.2 (Transaction Graph). *A transaction graph is represented as a static, directed graph $G = (V, E, X)$. V represents the set of nodes, with each node $v_i \in V$ corresponding to an individual financial account. The total number of nodes is denoted by m , such that $V = \{v_1, v_2, \dots, v_m\}$. E represents the set of directed edges, where each edge $e_{s,d} = (v_s, v_d)$ signifies a transaction from the sender account v_s to the destination account v_d . X is the edge attribute matrix, where each edge has an associated attribute vector $x_{s,d}$. The attribute vector $x_{s,d}$ contains details such as the timestamp t_e and the transaction amount a_e , for each $e \in E$.*

We make use of transaction graphs in two primary forms, namely *directed multigraphs* and *directed aggregated graphs*. A **directed multigraph** allows multiple edges between the same pair of source and target nodes. Each edge represents a single transaction, retaining its specific attributes. This representation preserves the granularity of the transaction dataset by distinguishing between unique transactions that occur over time. In contrast, a **directed aggregated graph** provides a simplified perspective. In this form, multiple edges between the same source and target nodes are consolidated into a single edge. The aggregation typically summarizes the transactional relationship between the accounts by combining specific attributes in an aggregated manner. While the number of nodes remains unchanged, the number of edges is reduced. This simplification is particularly useful for high-level analysis, such as examining overall money flow, where the detailed specifics of each transaction may be less relevant. Figure 2.5 below illustrates both representations.

In this study, we introduce an interesting graph structure inspired by previous work, which is capable of capturing the flow and trails of money through sequential transactions. This structure is referred to as *temporal graph of sequential transactions* [58]. We start from

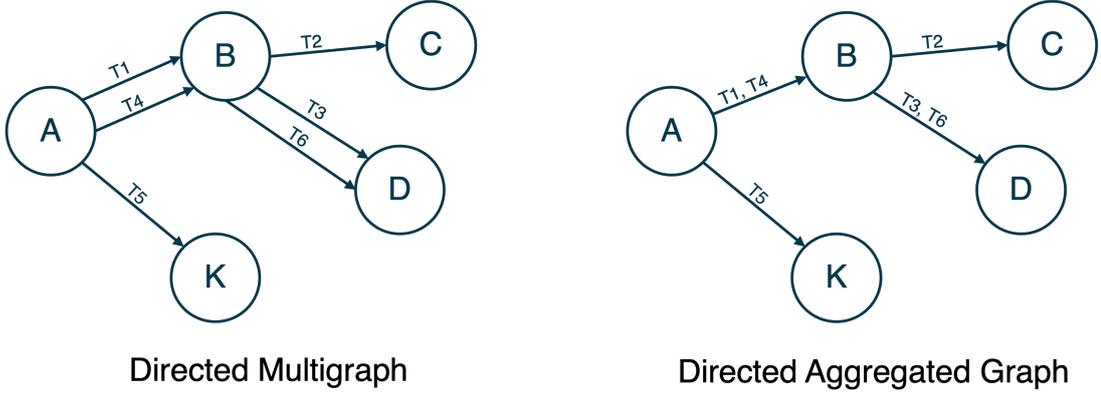


Figure 2.5: Comparison between a directed multigraph and a directed aggregated graph, illustrating how multiple transactions are represented individually in the multigraph, while they are consolidated into a single edge in the aggregated graph. This example follows from the sample transaction data presented in Figure 2.3.

this graphical structure and extend it with new types of edges to capture more nuanced patterns. Therefore, in addition to the original edge types which we call *flow*, we introduce *fan-out* and *fan-in* edges.

Definition 2.2.3 (Temporal Graph of Sequential Transactions). A temporal graph of sequential transactions is defined as a static, attributed, and directed graph $T = (V, E, X)$. V denotes the set of nodes, with each node $v_i \in V$ uniquely identified by a transaction identifier i ranging from 1 to n , and n representing the total number of transactions. E represents the set of directed edges, where each edge $e_{s,d} = (v_s, v_d)$ signifies a transaction from the source node v_s to the destination node v_d . X is the node attribute matrix, where each attribute vector $x_i = [f_i, b_i, t_i, a_i]$ corresponds to a node v_i and contains the following elements:

- f_i : the identifier of the from account associated with the transaction,
- b_i : the identifier of the beneficiary account receiving the transaction,
- t_i : the timestamp at which the transaction occurred,
- a_i : the amount involved in the transaction.

An edge $e_{s,d}$ exists between nodes v_s and v_d if one of the following conditions is satisfied:

1. **edge_type = flow**: the beneficiary account b_s of the source node v_s matches the from account f_d of the destination node v_d ;
2. **edge_type = fan-out**: the from account f_s of the source node v_s matches the from

account f_d of the destination node v_d ;

3. `edge_type = fan-in`: the *beneficiary* account b_s of the source node v_s matches the *beneficiary* account b_d of the destination node v_d ;

and the following conditions are also satisfied:

4. the *timestamp* t_s of the source node transaction occurs before the *timestamp* t_d of the destination node transaction, i.e., $t_s < t_d$;
5. the *timestamp* t_d of the destination node transaction occurs within a specified time window following the source transaction, such that $t_s < t_d \leq t_s + \Delta w$, where Δw is the time window parameter defining the "look-ahead" period.

Figure 2.6 illustrates in detail how edge types are built. Furthermore, Figure 2.7 illustrates how such a graph would be structured, building upon the examples depicted in Figure 2.3 and Figure 2.5. In this representation, transactions become nodes, with directed edges capturing the dependency relationship between them. In order to consider the dependencies existing within the sequence of transactions, a weighting procedure is employed, assigned to every $e \in E$ of T a weight w_e . This weight definition will be described in detail in Chapter 4. Throughout this study, we refer to the weighted T graph as 2nd order graph, in contrast to the simple 1st order graph previously introduced in the form of multigraph and directed aggregated graph.

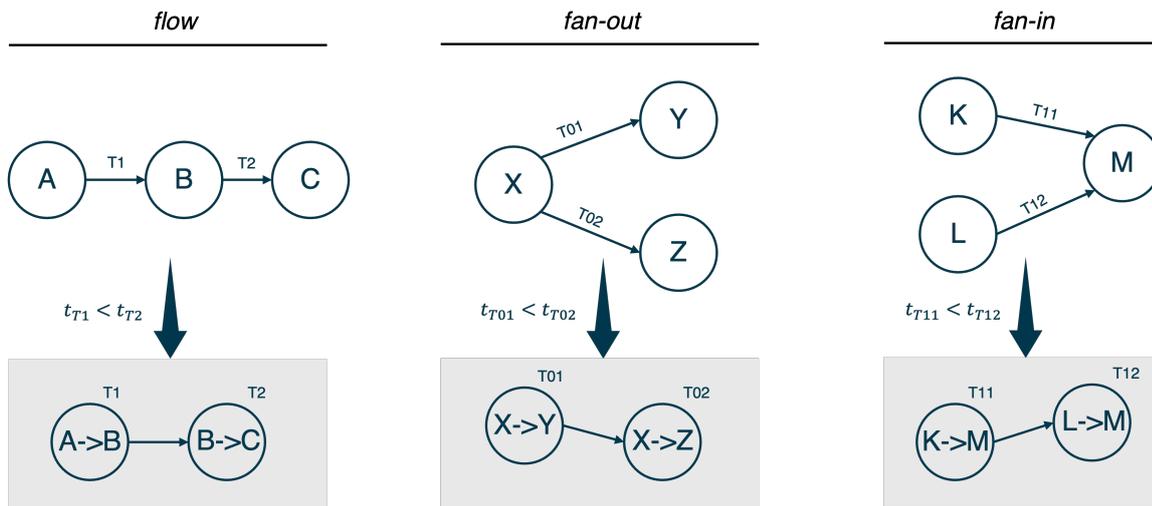


Figure 2.6: Edge transformation from the 1st to the 2nd order for flow, fan-out, and fan-in edge types.

We utilize the introduced graph representations independently for specific tasks in our pipeline, such as feature engineering or as input for graph-based models. In particular,

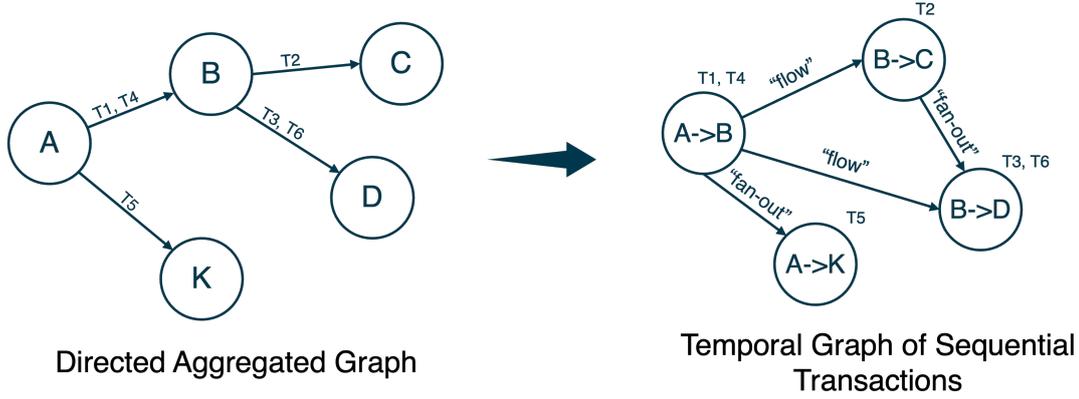


Figure 2.7: Building a temporal graph of sequential transactions. Each node represents a transaction, and the resulting graph may become disconnected if no dependencies exist within the specified time-frame parameter.

since transaction data is naturally suited for graph representations, this opens the way for the application of graph-based models. As this field is attracting increasing attention, graph representation learning has emerged as a powerful tool to leverage the structural properties of graphs to derive high-quality embeddings [27]. The embeddings are usually used for various downstream tasks such as anomaly detection. It is therefore essential to first define this concept, as it plays a central role in the state-of-the-art research and the framework’s design.

Definition 2.2.4 (Graph Representation Learning). *Graph representation learning is the process of learning low-dimensional vector representations of nodes, edges, or entire subgraphs within a graph, preserving the graph’s structural and relational information. Formally, given a generic graph $G = (V, E)$, the objective is to learn a mapping function $f : V \cup E \rightarrow \mathbb{R}^d$, where $f(v_i) \in \mathbb{R}^d$ for nodes $v_i \in V$ and $f(e) \in \mathbb{R}^d$ for edges $e \in E$, and d is the dimensionality of the embedding space.*

Table 2.1 provides a summary of the notations introduced so far for clarity and reference.

2.3. Problem Formulation

2.3.1. Goals and Challenges

Our primary objective for AML development is to effectively detect and mitigate illicit financial activities. This involves constructing a robust anomaly detection framework capable of distinguishing real anomalies from noise and from normal transaction data. An end-to-end anomaly detection methodology should be able to identify transactions

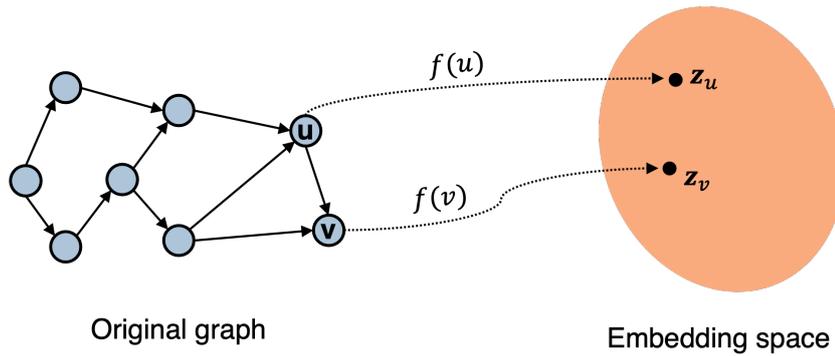


Figure 2.8: Illustration of graph representation learning through node embedding. The function f encodes nodes u and v into low dimensional vectors z_u and z_v , which preserve the structural and relational properties of the original graph.

that deviate from expected normal behavior in transaction datasets. The model must leverage known risk indicators and generate a score for each transaction, reflecting the likelihood of it being suspicious. Higher scores should correspond to a greater probability of anomalous records. Furthermore, the model must be sufficiently interpretable to explain the causes of an anomaly.

Considering these goals, the challenges in AML are multifaceted and involve:

- *severe class imbalance*: legitimate transactions significantly outnumber illicit ones, only appearing at a ratio as low as 1%. This imbalance can lead models to develop a bias toward predicting the majority class, missing rare fraudulent activities;
- *severe class overlap*: the features distinguishing between legitimate and illicit transactions are often minimal and subtle, resulting in a high degree of overlap between the normal and laundering classes. This overlap diminishes the model's potential to identify laundering instances since ML transactions closely mimic legitimate ones;
- *concept drift*: the nature of ML fraudulent schemes evolves constantly to evade thresholds and detection. This dynamic change is referred to as concept drift, meaning that the statistical properties of the target variable that we are interested in predicting change over time. As a consequence, models trained over historical data require continuous updates and tuning to maintain their detection ability;
- *uncertainty around data model*: the quality and structure of data in AML pose challenges. Issues such as missing values, erroneous entries, and unstructured information can lead to uncertainties in the modeling.

Given the typically unsupervised nature of AML problems, we focus on techniques that

Notation	Description
D	transaction dataset
n	number of transactions
k	number of transaction attributes
G	transaction graph
(V, E, X)	nodes, edges and attribute matrix
m	number of accounts
T	temporal graph of sequential transactions
f_i	source account of the transaction with index i
b_i	target account of the transaction with index i
t_i	timestamp of the transaction with index i
a_i	amount of the transaction with index i
Δw	time window parameter for sequential transactions
d	embedding size
$f : V \cup E \rightarrow \mathbb{R}^d$	embedding function

Table 2.1: Summary of the notations used in the definitions.

do not rely on predefined class labels. In real-world scenarios, it is rarely known whether a transaction is fraudulent, making it a requirement to develop detection models that can operate without explicit supervision.

2.3.2. Problem Definition

In this research, we aim to develop and evaluate a self-supervised anomaly detection framework designed to identify transactions belonging to money laundering patterns within financial transaction datasets.

The Problem. *Given a financial transaction dataset (D) characterized by a low occurrence of money-laundering instances and multiple transactional attributes (k), this research explores the feasibility and effectiveness of a self-supervised approach leveraging a graphical structure (T) for anomaly detection. The goal is to construct a framework that can identify suspicious laundering pattern transactions by learning normal transaction behavior. This learned knowledge is then used to score transactions belonging to a set of mixed normal and abnormal transactions, minimizing the false positives and maximizing the recall of true anomalies, all in the absence of labeled data.*

The first challenge in this research is to establish an unsupervised method able to effectively filter transactions and learn representations of *normal* behavior. This involves selecting a model that can process enriched transactional data and separate between normal and anomalous behavior without the need for labeled data. Once the normal be-

havior has been learned, the next step is to identify a suitable approach for graph learning, leveraging only the normal transactions to learn the structural relationship inherent to the normal part of the dataset. Following this, the model must be ultimately capable of detecting and predicting anomalous behavior by focusing on the embeddings of the anomalous transactions.

This problem formulation addresses the key challenges associated with transaction data, particularly the scarcity of labeled data in real-world scenarios. The primary value of this approach lies in its independence from labeled data, making it well-suited for practical applications.

In the following chapter, we will explore these key considerations by examining and comparing various approaches and frameworks, followed by a comprehensive overview of unsupervised and self-supervised methods tailored to address the identified goals and challenges.

3 | Related Work

In this section, we review the current research on the detection of money laundering behaviors. Approaches for AML can be broadly categorized into two main groups in the literature: supervised and unsupervised techniques. While supervised techniques refer to algorithms that learn from a set of labeled examples, unsupervised techniques aim to separate data points into different groups without labeling information. The label is typically a binary variable, representing either a *normal* or a *suspicious* transaction. Given the practical challenges in obtaining such labels, we focus on unsupervised and self-supervised learning for detecting money laundering behavior. This behavior can manifest at the transaction level as fraudulent transactions, at the account level as fraudulent account activity, and the flow or group level as money laundering networks. Within this chapter, we aim to delineate the landscape of existing literature addressing the detection of money laundering behaviors. We provide a holistic overview, suggesting how a new framework could align with or diverge from other approaches.

3.1. General Approaches

Rule-Based approaches are conventional methods used for AML, predominantly relying on decision tree-based models and rule-based classification engines. These systems implement thresholds defined by domain experts to address specific fraud problems. Transactions are monitored through dynamically tuned thresholds based on the account's history over time, and employ a layered escalation procedure, culminating with a Suspicious Activity Report (SAR)[22, 51]. Rajput et al. [50] proposed an ontology-based expert system to detect suspicious transactions, capable of suggesting transactions for further analysis and labeling. Although rule-based systems are effective in capturing certain fraud attempts, they fall short in detecting complex and unknown money laundering patterns. Additionally, these algorithms are often bypassed by fraudsters. Despite these shortcomings, transaction filtering by rules is indispensable in real-world applications. Nevertheless, more sophisticated techniques have been developed and integrated in order to provide a more robust and comprehensive approach to AML.

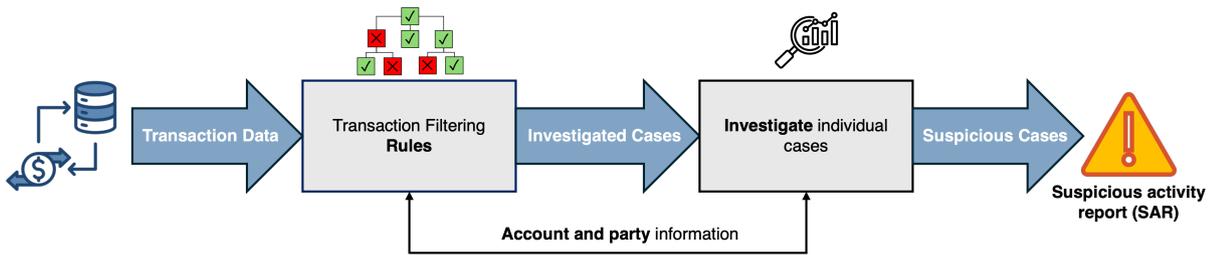


Figure 3.1: General workflow of rule-based suspicious activity reporting. Transaction data is first filtered using rules. These filtered cases are then investigated individually, by checking contextual information. Finally, suspicious cases are forwarded for further scrutiny.

Network and Graph Analysis techniques have garnered increasing attention for anomaly detection within transaction networks in recent years [15, 54]. In the financial context, network analysis allows for the study of relationships between individuals, banks, corporations, and specific transactions. By transforming transaction data into graph structures, network analysis exposes underlying patterns, clusters, and community connections. Coladon and Remondi [12] emphasize the role of social network analysis (SNA) in identifying nodes with a higher risk of fraud. They propose four distinct types of relational graphs that encapsulate various risk factors, including economic sectors of activities, geographic areas, transaction volumes, and connections among companies sharing common ownership or representatives. Their findings indicate that clients with higher risk profiles typically occupy less central roles in the transaction network and engage in larger transactions. In contrast, nodes associated with legitimate activities tend to cluster more densely, whereas criminals occupy more peripheral positions. This positioning could be due to new established companies with criminal purposes, that enter and exit the network rapidly, avoiding central roles. The authors suggest focusing attention on clients operating in more risky sectors and those who spread their connections across several regions and high-risk jurisdictions.

In their comprehensive literature review, Deprez et al. [14] scrutinized network analytics applied to AML by examining the top 10% of the most frequently cited papers within a set of 97 studies. They observed that many of the approaches are specifically tailored to tackle only a small part of the money laundering puzzle. This highlights a consensus that multiple and complementary models are needed in parallel, each detecting different money laundering typologies. Furthermore, the complexity of these methods often compromises their interpretability, necessitating the adoption of simpler techniques that rely on centrality metrics and other manually engineered features.

3.2. Machine Learning in AML

Machine learning has the potential to handle sizeable unstructured information, self-update, and recognize complicated patterns. Another qualitative analysis incorporating industry perspectives [49] highlights that the shortcomings of existing transaction monitoring systems are catalyzing the adoption of more sophisticated machine learning technologies. Chen et al. [10] presents a comprehensive review of the various machine learning and data-driven approaches employed in AML efforts. A systematic review of the supervised, semi-supervised, and unsupervised anomaly detection techniques applied to the detection of financial fraud can be found in [29].

3.2.1. Supervised Anomaly Detection

These methods operate on the premise that the dataset consists of labeled instances. Typically, these models classify new unseen data by comparing it with historical patterns. Support Vector Machines (SVM) have shown use for classification tasks, mainly for binary classification problems such as fraud detection or flagging suspicious financial transactions [13, 32]. SVMs operate by projecting the data into a higher-dimensional space where a separating hyperplane can be identified. They achieve this without higher complexity. It is dependent on a good selection of input features in discriminating fraudulent and legitimate transactions [52]. However, there is a tendency for SVMs to overfit the training data, since they need to be exposed to enough samples, and the classes are highly imbalanced. Moreover, the success of SVMs heavily relies on the careful selection of input features that can distinguish between fraudulent and legitimate transactions [52].

Random Forest (RF) is a supervised ensemble method that utilizes multiple Decision Trees (DT) combined with bagging to reduce variance in datasets. It is renowned for its efficacy in detecting credit card fraud, often outperforming Logistic Regression and basic Decision Tree models in terms of accuracy [1]. Multi-Layer Perceptron (MLP) networks, the simplest form of neural networks, are trained using backpropagation on labeled datasets and have been utilized effectively in fraud classification tasks [47]. Extreme gradient boosting (XGBoost)[9] is another system designed for speed and performance by sequentially building trees from an initial Decision Tree, with each subsequent tree aimed at correcting the errors of its predecessor.

In AML it is essential to consider the underlying contextual information to identify suspicious behavior. Recent advancements leverage graph-based features [19] and node embeddings [42] for improved performance. Graph-Feature Preprocessor (GFP) [6] is a software

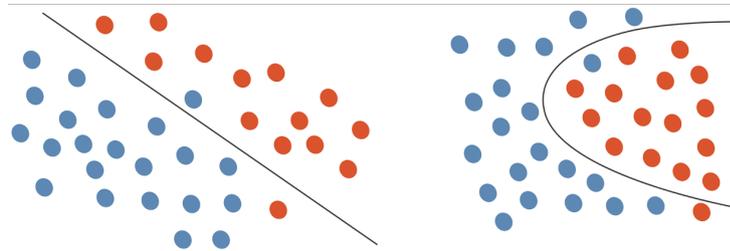


Figure 3.2: Illustration of Support Vector Machine classification. Instances are labeled (red and blue), and the black line represents the separator, linear (left) and non-linear (right).

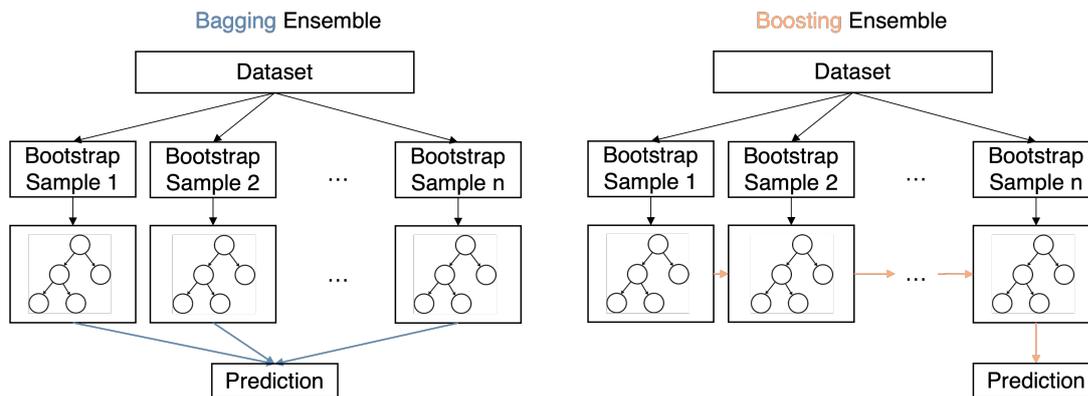


Figure 3.3: Ensemble techniques in Machine Learning. On the left, Random Forest uses bagging with multiple independent decision trees, averaging results for predictions. On the right, XGBoost employs boosting to sequentially correct previous trees' errors, to output a final prediction.

library that extracts features from transaction graphs indicative of money laundering, such as fan-in/out patterns, a degree in/out, scatter-gather, and cycle patterns. The incorporation of these features has notably enhanced the performance of supervised models, particularly those based on gradient boosting.

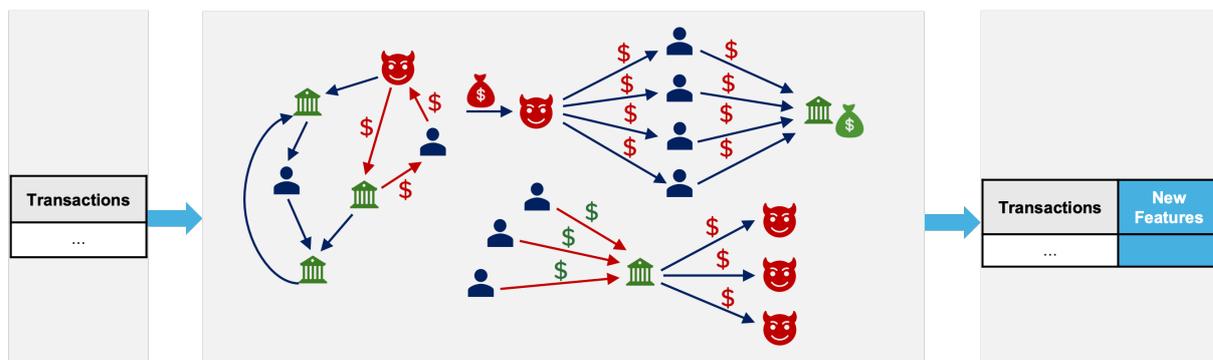


Figure 3.4: Overview of the Graph-Feature Preprocessor (GFP) workflow. GFP extracts features from transaction graphs indicative of common money laundering patterns.

While these techniques are successful, their primary application has been in credit card fraud detection, which has different dynamics than money laundering. Common pitfalls include overfitting and a bias towards the majority class. Additionally, the scarcity of labeled data in real-world settings restricts the effectiveness of supervised models. To address the data scarcity issue, several studies have generated synthetic datasets [3, 30, 57], offering promising avenues to advance research in the AML field. However, these limitations have shifted our focus towards exploring unsupervised methods which are proving to be increasingly valuable.

3.2.2. Unsupervised Anomaly Detection

Money laundering is inherently an unsupervised problem. Unsupervised techniques typically rely on anomaly detection algorithms that compare events against the expected behavior through deviation metrics. Expected behavior encompasses customer transaction profiling and clustering [25, 38].

A novel unsupervised approach, called Isolation Forest (IF) was introduced to isolate anomalous instances even from large, high-dimensional problems. It requires low memory and operates with low linear time complexity [41]. IF builds an ensemble of isolation trees that partition data recursively by selecting a random feature and a split value between its extremes. The principle behind IF is that anomalies, being simpler to isolate, require fewer partitions compared to normal observations. Recent efforts have focused on incorporating Isolation Forest (IF) into AML systems. Research by Yang [66] shows IF’s effectiveness

as part of an ensemble of unsupervised anomaly detectors, for capturing implicit money laundering transactions. Additionally, another study [56] highlights IF’s high detection accuracy and practicality, noting its success in uncovering undetected cases of worker’s compensation fraud.

Autoencoders (AE) are neural networks designed for unsupervised learning, focusing on input reconstruction [4, 11, 29]. An AE typically features a symmetric structure with an encoder and a decoder. The encoder compresses the input into a condensed form known as the latent space or encoding, using a bottleneck-shaped architecture. The decoder then reconstructs the output from this condensed representation. During training, AEs are exposed only to normal transactions, which they learn to replicate accurately. At testing time, if a transaction deviates significantly from the norm, the AE struggles to reconstruct it, resulting in a high reconstruction loss. Then, transactions with a reconstruction error exceeding a predetermined threshold are flagged as anomalous. AEs are also utilized as a dimensionality reduction technique to extract embeddings. These embeddings can be used for downstream tasks, such as classification [46]. Despite the recognition of AEs as preferred models for AML applications, there is limited evidence of their widespread use in the industry. This gap is largely due to the lack of access to relevant data by researchers, as much of it is confidential [36].

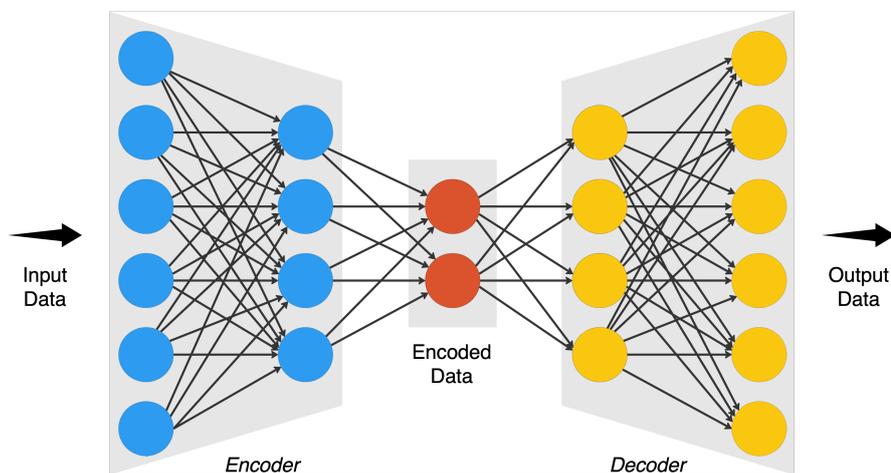


Figure 3.5: Illustration of a typical Autoencoder architecture.

Suspicious Money-Flows. Money laundering is usually associated with high-volume fund transfers executed rapidly through a chain of bank accounts, which obfuscate the trail of movements. There is a consistent family of approaches for unsupervised anomaly detection in AML that focuses on the movement of funds, aiming to capture suspicious money flows or paths.

Flowscope [40] searches dense flow of transfers in big tripartite or multipartite transaction graphs. The key intuition behind Flowscope is that large amounts of money need to traverse intermediate accounts before reaching the final destinations or beneficiaries (see Figure 3.6). The focus is on detecting dense multi-partite subgraphs, assuming that intermediate accounts have low balance, and are used mostly as bridges, to transfer out the received amount. Flowscope is deterministic in its approach and maximizes a custom metric designed to identify and quantify these dense flows effectively. This approach, while robust, depends on the initial assumptions and the targeted behavior.

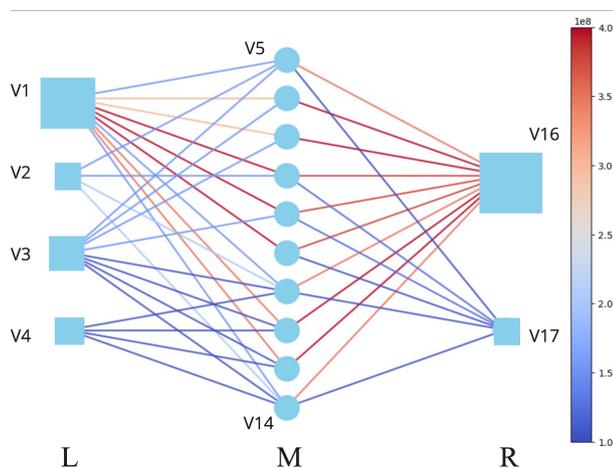


Figure 3.6: Visualization of dense money flows in a tripartite transaction graph. The *left* (L), *middle* (M), and *right* (R) nodes are different partitions of accounts. The thickness of nodes represents the magnitude of money they move. Similarly, the edge colors indicate different flows.

Following this work, Starnini et al. [55] propose a practical pipeline that focuses on the detection of smurf-like subgraphs. Their framework uses standard database joins to link sequential transactions and uses a time window and filtering thresholds. They target two types of structures, as illustrated in Figure 3.7. Although practical and simple, this method targets a specific motif and is only complementary to other techniques.

[39] presents a comprehensive framework designed to identify suspicious money laundering groups within large temporal networks. The starting point is modeling the transactions as a graph of transfers. Then maximally connected subgraphs (MCS) are identified; these are clusters of connected edges that cannot be expanded without losing connectivity. Leveraging domain knowledge, only relevant components from the graph are retained. The components are further decomposed using a community detection algorithm, optimized to handle a temporal directed graph. For each community, a ML score is calculated, to rank and report the highest-risk communities.

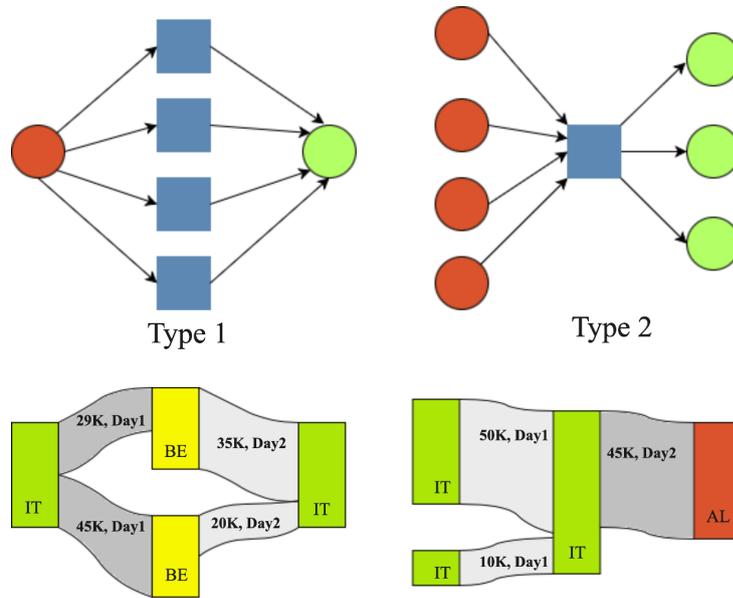


Figure 3.7: Visualization of smurf-like subgraphs proposed in [55]. The framework categorizes money-laundering behaviors in *Type 1* and *Type 2*. The lower part of the figure illustrates how sequential transactions are linked, showing amounts moved and transfer timings.

Fastman (Follow All Suspicious Trails of Money for All Nodes) [58] offers an innovative approach to extracting relevant flows of money in billion-scale transaction data. Utilizing a higher-order representation, this framework maps financial transactions as nodes in a directed graph, linking two nodes when the destination account of one transaction serves as the source for the next. These transaction chains are constructed within a specific time window, connecting each transaction to subsequent ones within this period. The sequential graph undergoes a decomposition into communities of flows, through a community detection algorithm. These communities are then analyzed for anomalous behavior, ranked, and flagged based on a set of undisclosed AML risk criteria. Fastman exemplifies the necessity of domain knowledge in AML and underscores the practical value of unsupervised learning, which relies solely on deterministic steps to derive insights from complex data patterns. This approach highlights the efficacy of unsupervised learning techniques in a domain heavily reliant on expert knowledge for identifying risk.

3.3. Graph-based Anomaly Detection

Graphs are inherently suited to modeling interactions and data flows in many areas. Anomaly detection in transaction graphs, referred to as GAD - Graph Anomaly Detection, can thus help in detecting fraud or money laundering unique instances. Graph anomalies

can be of different types, such as node, edge, sub-graph, or whole graph anomalies [37]. In recent years, graph analysis has benefited from neural networks, as new trends in machine learning and deep learning suggest using Graph Neural Networks (GNNs) [67]. A comprehensive classification of methods in graph anomaly detection utilizing neural networks is detailed in [33].

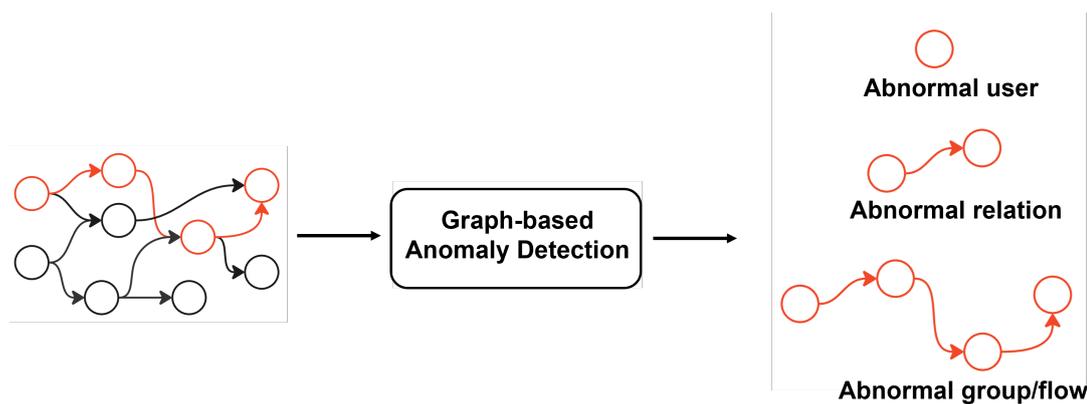


Figure 3.8: Types of graph-based anomaly detection: at the node level, abnormal users; at the link level, abnormal relations; and at the group or subgraph level, abnormal structures.

GNN framework. The main idea behind the GNN framework is message-passing, where nodes aggregate individual features from their neighborhood in the graph, update their representation, and forward it to the next layer. Successful GNN architectures have been actively explored, and include, but not limited to, graph convolutional network (GCN) [35], GraphSAGE [28], and graph attention network (GAT) [61].

One of their primary uses is in encoding the graph structure to a lower dimensionality for improving subsequent downstream tasks. In [64] the authors experiment with GCNs, showing how it can enhance the representation of a transaction. They concatenate embeddings obtained by GCN with original features to increase the accuracy of a Random Forest model. In another study, Altman et al. [3] demonstrate that GNN models can recognize laundering patterns in extremely imbalanced datasets. Egressy et al. [20] further adapt GNNs for directed multigraphs by introducing techniques such as multigraph port numbering, ego IDs, and reverse message passing, enabling the model to capture the directionality and complexity of transactions. They demonstrate how these transformations enable the detection of specific fraud patterns with high performance.

While supervised techniques perform well for node classification tasks, they are less applicable to anomaly detection due to their reliance on fully labeled datasets. Semi-supervised learning techniques try to fill this gap by leveraging a limited amount of data for the learning process. [31] employs a semi-supervised graph learning technique on transac-

tions graph to identify nodes involved in potential money laundering. A graph embedding model generates node embeddings, and with limited annotations from alerts, these embeddings are used to predict suspicious nodes based on their connections to known suspicious nodes.

However, since vast amounts of data are freely available from many information systems, there is increasing focus on unsupervised approaches. For instance, DOMINANT [16] operates in an unsupervised fashion, combining a GCN and a deep autoencoder framework to reconstruct the original attributed network with learned node embeddings from GCN. The reconstruction errors of nodes are then employed to flag anomalies.

Among unsupervised approaches, self-supervised learning (SSL) stands out. In SSL the model leverages data's inherent structure to infer a ground truth, from unlabeled data [65]. The training process involves presenting the model with input data and, based on the objective function, reconstructing the input as accurately as possible. Graph Autoencoder (GAE) is an increasingly popular approach for self-supervised learning and anomaly detection. [17] performs a GAE-based outlier detection, by training a model on a dataset where the majority of the data points are normal, and ranking outliers by the outlier factor, a measure of outlierness, defined as the average of reconstruction errors over all the input features.

Similarly, One-Class GNNs (OCGNN) [63] were introduced for detecting anomalies within attributed networks. OCGNN is a GNN-based GAD framework, which can use various backbone GNN layers, such as GCN, GAT or GraphSAGE. The approach assumes that all training nodes belong to a single class (normal), while the remaining testing nodes belong to a mixed class (normal and abnormal). The model learns a hypersphere around the normal data and points that fall outside this boundary are considered anomalous. This work underscores the potential of leveraging clean, normal training data to effectively identify abnormal instances.

LaundroGraph [7] is one of the first fully self-supervised works with graphs for AML detection. Financial interactions are represented as an attributed customer-transaction bipartite graph. The architecture is comprised of a graph neural network encoder trained on a user profile and a feed-forward anomaly predictor decoder. The task is a link prediction, where each incoming transaction gets an anomaly score. They show how leveraging underlying information present in the graph improves the detection performance. [45] proposes a GNN framework based on an enhanced self-supervised Deep Graph Infomax (DGI) [62] and a supervised Random Forest-based classifier to detect AML in Bitcoin. Their experimental results demonstrate the potential of self-supervised GNNs since the

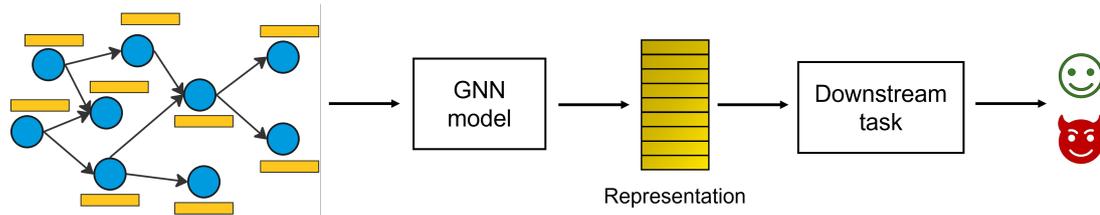


Figure 3.9: Workflow of GNN-based anomaly detection algorithms.

obtained node embeddings are used for feature augmentation to improve detection performance. Moreover, they suggest integrating this with unsupervised anomaly detection algorithms to detect illegal transactions in an unsupervised manner.

3.4. Summary

Supervised techniques require a substantial amount of annotated data to learn relations in the data. In the context of money laundering detection, obtaining such labels is prohibitive and there is a lack of shared real-world datasets. This presents several challenges: i) AML data is highly imbalanced, where legitimate ones outnumber the number of illicit transactions: ii) the demand for unsupervised machine learning models to be successfully implemented and validated in the industry is growing. As a consequence, supervised techniques are less practical for real-world applications compared to unsupervised methods

As evidenced by the literature reviewed, graph-based methods are a promising direction for the future of AML, particularly in tackling graph-level anomaly detection, where the objective is to identify suspicious nodes or edges that indicate potential fraudulent activities. Self-supervised learning, though not yet extensively explored, offers significant potential for advancement in this area. We aim to provide a new perspective, combining a unique graph structure with a self-supervised anomaly detection pipeline. Specifically, we explore a graph-based innovative approach that does not rely on labels.

Table 3.1: Relevant state-of-the-art anomaly detection approaches in AML.

Paper	Year	Type	Focus	Graph structure	Real-world evaluation
Altman et al. [3]	2024	Supervised	Binary classification	✓	✗
Egressy et al. [20]	2024	Supervised	Binary classification	✓	✗
Li et al. [40]	2020	Unsupervised	Dense subgraphs	✓	✓
Starnini et al. [55]	2021	Unsupervised	Transaction flows	✗	✓
Tariq and Hassani [58]	2023	Unsupervised	Communities of flows	✓	✓
Cardoso et al. [7]	2022	Self-supervised	Link Prediction	✓	✓
Lo et al. [45]	2023	Self-supervised	Graph embedding	✓	✓

4 | The Method

This chapter details the research framework implemented for our anomaly detection pipeline. The framework is divided into two primary stages, each contributing to identifying suspicious abnormal pattern transactions from financial transaction data. Specifically, we generalize the process into four main modules: (i) feature engineering, (ii) transaction filtering, (iii) graph representation learning, and (iv) anomaly detection. Figure 4.1 provides a high-level overview of the pipeline, which is thoroughly described in this chapter.

4.1. Framework Architecture

In the first stage, we apply an unsupervised technique to tabular data, focusing on transaction features that capture aggregated information of accounts and transactions and exploit the underlying structure of the transaction network. This stage is designed to provide an initial classification signal, distinguishing between *normal* and *potentially suspicious* transactions. The goal here is to develop a foundational knowledge of what constitutes normal behavior in the dataset, as well as to identify anomalies based mainly on transaction characteristics. The first stage can be summarized in the following steps:

1. Import a financial transactions dataset.
2. Process and enrich the dataset with relevant features.
3. Apply an unsupervised outlier detection model to score each transaction.
4. Classify transactions as *normal* or *potentially suspicious* applying a threshold to the scores.

Once the classification signal is defined, we proceed with the second stage of graph representation learning, followed by anomaly detection. In this phase, we construct a temporal graph out of transactions. We use the sequential graph of temporal transactions introduced earlier. This graph is transaction-centric and is designed to capture the relationships and dependencies of transactions over time. In this graph, transactions (nodes) are connected by three types of edges - fan-in, fan-out, and flow - each representative of common

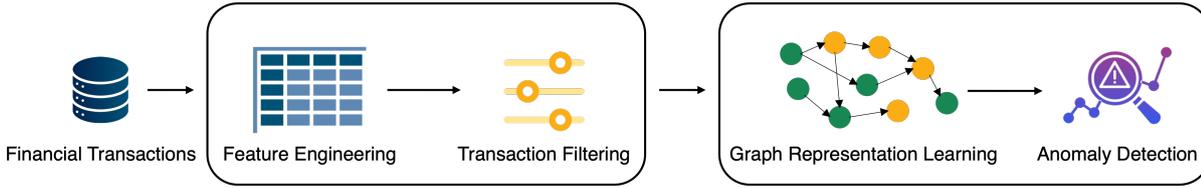


Figure 4.1: High-level overview of the framework.

money-laundering-related structures. In particular, we use this graph as input for a GNN model for outlier detection. Different models are trained on the identified *normal* transactions, to score the set of isolated transactions considered *abnormal*. Finally, anomaly detection is performed starting from the model output score for each node. These scores are combined with the transaction amounts to assign a combined final score to every transaction. The second stage follows these steps to flag a transaction as anomalous:

1. Create a temporal graph of sequential transactions.
2. Enrich nodes with features based on edge types.
3. Train a GNN model using as input nodes only *normal* transactions.
4. Test the GNN model over the entire data.
5. Perform anomaly detection on the prediction scores of the *abnormal* data.

In Figure 4.2 we provide a detailed visual representation of the pipeline. The subsequent sections delve into the main modules, breaking them into their smallest components and explaining the rationale behind each.

4.2. Data Preprocessing

Data preprocessing is the initial step of our pipeline, aimed at preparing a standardized input for the analysis and modeling phases. Our approach begins with transaction-level data. We use transaction records as the primary building blocks, where each data point represents an exchange of money between two accounts. As discussed in 2.1.4 when introducing financial transaction data, each transaction is characterized at least by the following set of features: transaction identifier *id*, *source* and *target* accounts, *amount* exchanged, and *timestamp* of the transaction.

Currency Conversion. To maintain consistency across datasets featuring transactions in multiple currencies, we standardize all transaction amounts to the same currency for a fair comparison, opting for the most present currency in the dataset. Therefore, we apply

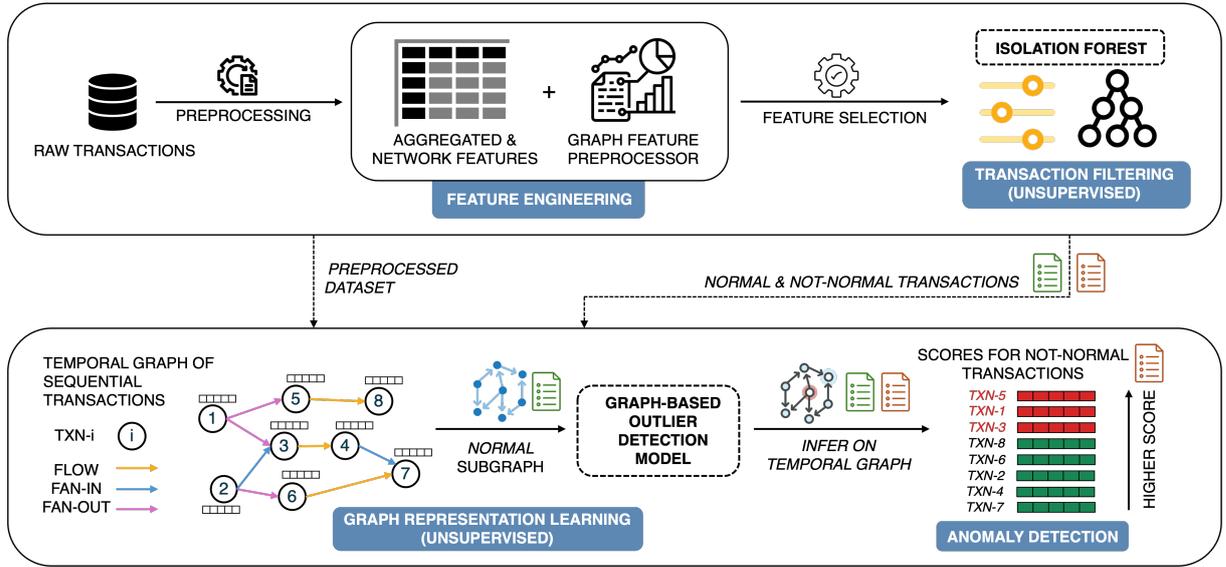


Figure 4.2: Detailed overview of the framework. The procedure is divided into 2 stages and the 4 main modules are highlighted. The input dataset is used for feature engineering and unsupervised scoring by Isolation Forest. Then, the temporal graph of sequential transactions is created from the entire dataset. The induced subgraph built on the *normal* nodes is used to train a graph-based outlier detection model. Finally, the model trained on *normal* instances scores each transaction, and anomaly detection is performed on the set of *abnormal* nodes for evaluation.

the currency conversion by inferring the exchange rates from the dataset for each currency pair, based on daily values derived from the dataset. In instances where daily exchange rates are missing, a forward-filling approach is employed, propagating the last available exchange rate value. This step ensures a uniform analysis without the noise introduced by currency fluctuations.

Categories Standardization. Categorical fields such as transaction type or currency are mapped into a unified lowercase format, to guarantee a naming convention. After this stage, we ensure data is in a suitable format for the subsequent modeling steps.

Date and Time Formats. We addressed the standardization of date and time formats across datasets by converting timestamps into *UNIX* time format. This conversion facilitates a consistent representation of time, which aims at streamlining the subsequent operations of feature extraction.

Data Cleaning. To ensure consistency, we aggregate transactions with the same source, target, and timestamp (or concurrent timestamps). High in- or out-degree accounts are filtered out by analyzing transaction counts, allowing us to focus on representative ac-

counts. A threshold on transaction counts is set to retain accounts with a reasonable number of transactions, guided by exploratory data analysis.

Injected Anomalies/Patterns. We focus on specific money laundering patterns and ensure that the dataset contains relevant anomalies. Anomalous patterns with only one transaction or that are disconnected - not part of a single weakly connected component - are removed. Thus, the injected anomalies consist exclusively of connected patterns, referred to as cases or money laundering patterns.

Following the initial data standardization, in the next section we describe the procedure of extracting informative features from the existing data, incorporating domain knowledge, to enhance the predictive accuracy of our models. This stage involves a careful consideration of the input attributes available in each dataset. Due to variability in these attributes, the features we engineer are inherently constrained and dependent on the specific dataset.

4.3. Feature Engineering

Here we focus on how we created and refined features to prepare our dataset for modeling, aiming to build both simple and aggregated features that differentiate laundering transactions from normal ones. The main challenge in detecting fraudulent or laundering activities is that such behaviors are usually only revealed when viewed in context. Therefore, we prioritize developing features that can capture this necessary contextual information. Leveraging business and domain knowledge is crucial in designing features that are both statistically relevant and practically significant to explain a laundering instance. This added expertise guides the feature selection process, ensuring that we use the most relevant and effective features for our analytical task.

Our feature engineering involves extensive use of aggregations and grouping operations on selected attributes to extract trends and patterns that raw transactional data may not readily exhibit. Additionally, to enhance the robustness and support a more comprehensive analysis, we integrate an external library tailored for feature generation.

4.3.1. Temporal, Aggregated and Network Features

We focus on extracting temporal, aggregated, and network features to characterize each transaction data point. These features provide a holistic view of the interaction dynamics over time. One of the most used techniques in feature engineering lies in the creation of aggregate counts based on rolling time frames. This helps us in synthesizing transaction data for immediate and extended periods. We use range day, number of days, or week. For

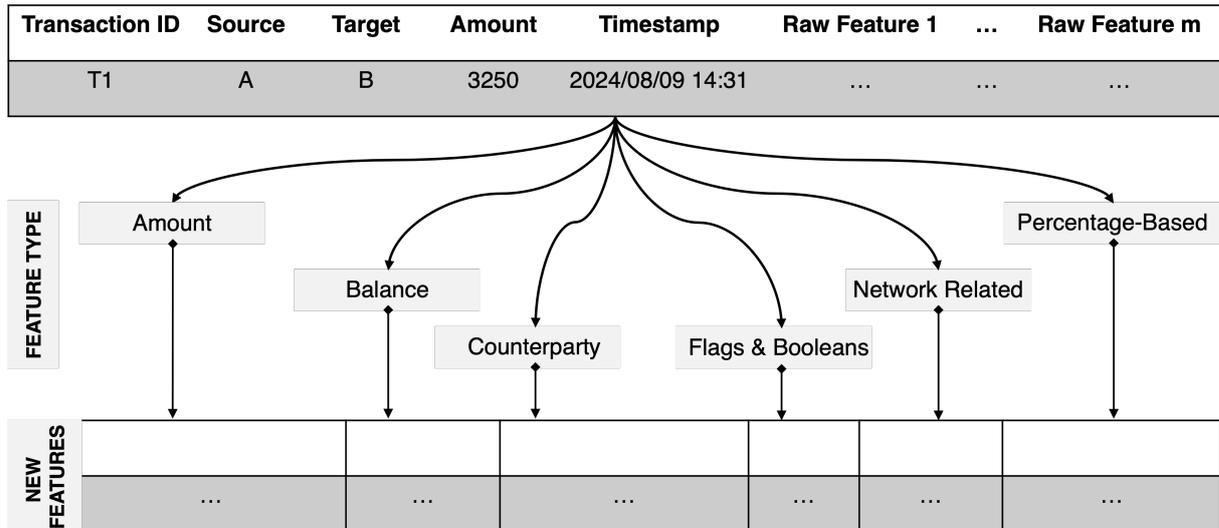


Figure 4.3: Feature engineering process for transaction data. This figure shows the various types of manually engineered features applied to each transaction vector, illustrating the new attributes derived.

instance, we compute averages, standard deviations, and total amounts to get precious insights.

1. **Amount Related:** The amount is one of the most relevant indicators for highlighting suspicious behavior. Therefore, we use this information extensively and calculate various features to assess the scale of transactions. Features such as the totals, average, and standard deviation aggregate transaction amounts across the entire dataset, providing statistical summaries for each source and target account. To capture temporal dynamics, we compute daily and weekly transaction totals, while also analyzing transaction data from both past and future periods defined by a specific number of days. as well as looking at the past and forward of a parameter number of days.
2. **Balance Dynamics:** We track changes in account balances before and after transactions to identify slacks and fluctuations. This involves setting an initial null balance and adjusting it as every transaction occurs, to capture the trajectory for each account.
3. **Counterparty Related:** These features assess the relationship between source and target accounts by counting distinct occurrences of the same counterparty over varying historical windows. These kinds of features help identify key or weak relationships, signaling either typical or atypical counterparties. For instance, we enumerate the number of unique counterparties each account interacts with. We

count reciprocal interactions between two parties on the same day, within the same week, and across the entire dataset. Another feature type aggregates the dataset by party and temporal range to compute average and total transaction amounts exchanged by counterparties.

4. **Flags and Booleans:** These binary features are designed to highlight the presence of factors or characteristics such as round amounts or types of transactions. In particular, we developed a feature that can detect transactions involving round or nearly round amounts, which are often observed in suspicious transactions. The logic is parametric and evaluates if the amount meets a predefined minimum threshold and then, through a minimum round amount multiplier with some tolerance, determines its roundness. Other indicators flag whether a transaction is conducted in a probably suspicious format as *cash*, crossed international borders, or involved multiple currencies, and if it occurred between different banks, respectively.
5. **Network Related:** We make use of network features to gain insights into the relationships and interactions of accounts within a broader network context. These are calculated over a directed graph constructed from transaction data. The graph allows us to quantify various metrics that reflect the connectivity and centrality of accounts within the network. Page rank for instance helps identify influential accounts within the network based on the number and quality of their other connections. In and out-degree measure the incoming and outgoing connections for each account, indicating potential hub activity. In addition, degree centrality quantifies how central an account is within the network. Finally, the clustering coefficient suggests nodes involved in coordinated activities.
6. **Percentages-Based Features:** These features offer normalized measures that facilitate comparative assessments across diverse transaction scales. These features are derived by calculating ratios, such as how an individual transaction's amount compares to the total amount sent or received by an account. Additionally, we employ a temporal windowing technique to evaluate transactions within specific time frames. For instance, we calculate what proportion of the total transactions over the past k days an individual transaction represents, where k is a parameter, or what significance a transaction holds within the next week's total transaction volume. This approach provides insight into the relative importance and timing of transactions, highlighting those that are unusually large or small within their temporal context. Such metrics are especially useful for detecting transactions that could have been strategically timed or structured to evade typical detection mechanisms, and for understanding the transaction's impact relative to ongoing account activities.

4.3.2. Graph Feature Preprocessor

To provide additional features for the unsupervised machine learning model, we decided to employ Graph Feature Preprocessor (GFP)[6], a recent library that proved useful in enhancing supervised models accuracy in detecting money laundering instances. The idea behind GFP is to enrich transaction tabular data with additional features that capture the underlying graph structure of the interactions. The GFP module processes transactions represented as temporal edges in a streaming fashion. The preprocessor inserts into its in-memory graph a batch of temporal edges and extracts various features from this graph. It allows to produce two types of features: *graph-pattern-based* and *vertex-statistics-based*.

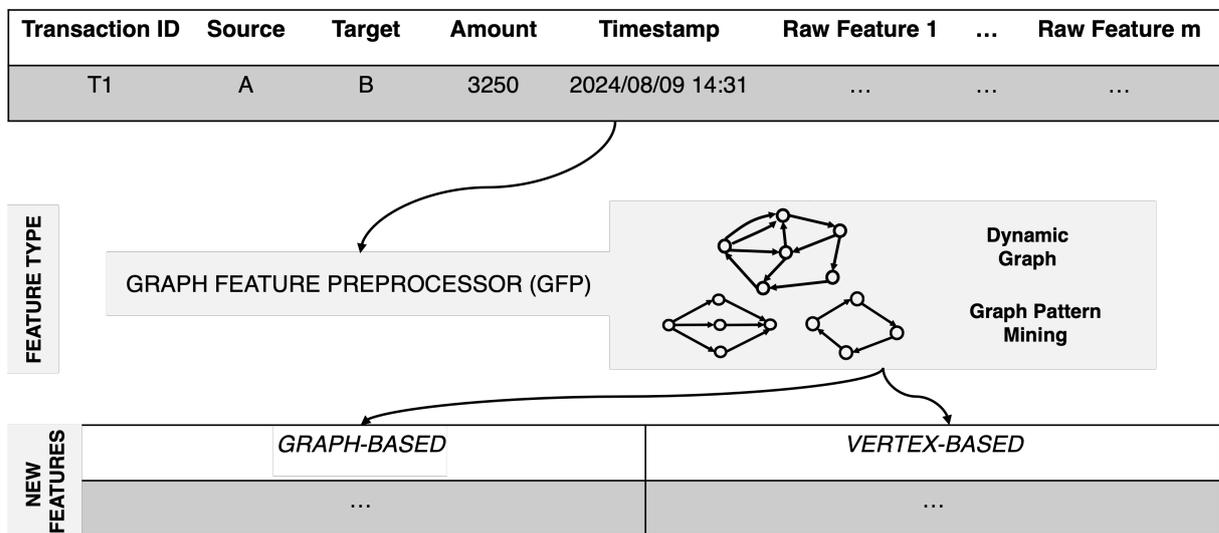


Figure 4.4: Graph Feature Preprocessor feature enrichment process.

- Graph-Pattern-Based Features:** these features capture recurring structures within the transaction network that may indicate typical or anomalous behaviors associated with money laundering. The features include **fan-in/fan-out**, which measures the number of incoming and outgoing edges for a node, **scatter-gather/gather-scatter**, to identify nodes where funds are scattered from or collected to, **simple cycle**, where no vertices are repeated except for the first and last, and **temporal cycle**, with edges ordered in time. These features are computed based on the source and target pairs, with a time-window parameter that constrains the batch of transactions under consideration. The window parameter can be specified for each pattern separately. The array-like parameter of bins defines the pattern histogram of bins for each pattern. Each bin in the histogram corresponds to the number of patterns of a certain size. The binning process allows for creating detailed attributes that describe the distribution of pattern sizes.

- **Vertex-Statistic-Based Features:** these features are computed based on the statistical properties of vertices (accounts) and edges (transactions). These features include sum, mean, minimum, maximum, median, variance, skew, kurtosis. Similarly to the graph-based features, it is possible to set a time-window parameter to compute the features in batches. However, the time window applies to every feature in the list. Additionally, it is possible to specify the columns for which to apply such features, which the authors in the paper suggest as the amount and transaction timestamp.

For each dataset, GFP processes transactions in the increasing order of their timestamps, so that graph-based features are extracted using the past data. This prevents data leakage in case training, validation, and testing graphs need to be created, without using future information. Details of the parameter settings for the module can be found under *Experimental Setup*, in chapter 5. Table 4.1 provides a schematic overview of the features generated by GFP.

Table 4.1: Overview of GFP Features.

Type	Feature	Description
<i>Graph-Based</i>	fan	Incoming/outgoing connections of a node
	degree	Incomin/outgoing connections of a node
	scatter-gather	Count of intermediary nodes
	temp-cycle	Count of cycles with ordered edges
	lc-cycle	Count of cycles with a length constraint
<i>Vertex-Based</i>	fan	Neighboring vertices connected to a node
	deg	Incident edges for a node
	ratio	The ratio of incoming to outgoing edges
	avg, sum, min, max, median	Statistical summaries of selected features
	var, skew, kurtosis	Higher-order statistical metrics

4.3.3. Feature Selection

After extracting a comprehensive set of features, feature selection is a necessary step in modeling. This step is performed to filter out irrelevant features which could bring noise. Since we have a rich set of features and a high number of data points, it is mandatory to reduce the complexity and avoid the curse of dimensionality. We aim to select relevant features only, but, due to the absence of label information to guide the search for discriminative features, unsupervised feature selection remains a challenging problem [18].

Since our models do not inherently provide feature importance scores, we explored various

alternatives and experimented with different subsets of features. Initially, we examined the correlation between features to avoid redundancy and ensure independence. Moreover, we tested unsupervised feature selection methods, such as DIFFI [8]. However, after empirical testing and experimentation, we selected a subset of features that primarily capture transaction amounts, transaction velocity, and the influence of source-target interactions. These are further combined with enriched features that describe cyclic patterns, fan structures, and temporal relationships.

4.4. Unsupervised Filtering of Transactions

Once all the necessary features are created and the most relevant subset of features is selected, the prepared dataset is fed into an unsupervised anomaly detection model. This model assigns scores to each data point, enabling the separation of normal and anomalous transactions by selecting the appropriate cutoff point. In particular, we employ Isolation Forest as an unsupervised model.

4.4.1. Isolation Forest

Isolation Forest (IF) [41] is a state-of-the-art unsupervised algorithm for anomaly detection, particularly effective for high-dimensional data. IF infers the anomaly score, a measure of outlierness for each data point. It is a decision-tree-based algorithm that consists of splitting sub-samples of the data according to a feature at random. The isolation procedure is based on recursive partitioning to define a region in the data domain where each point lies. The idea is that rarer observations will require fewer splits on some features to be isolated. The hypothesis is that anomalies, which differ significantly from normal instances, are more likely to be isolated earlier and end up in smaller partitions. As shown in Figure 4.5, transactions are represented in two dimensions for simplicity. Normal transactions are more challenging to isolate due to their proximity to another one and overlap in the feature space. In contrast, transactions with an anomalous combination of features are easier to isolate, since they tend to lie further from the dense cluster of normal transactions.

IF algorithm overview. First, a feature is randomly selected from the dataset and a threshold value within the range of that feature is chosen as splitting criterion. Based on the threshold value, the data is then partitioned into two subsets. This process is repeated recursively until all data points have been isolated into individual partitions, or until a predefined maximum depth is reached. The isolation path of a data point consists of the number of splits or partitions needed to isolate that point within a tree. IF

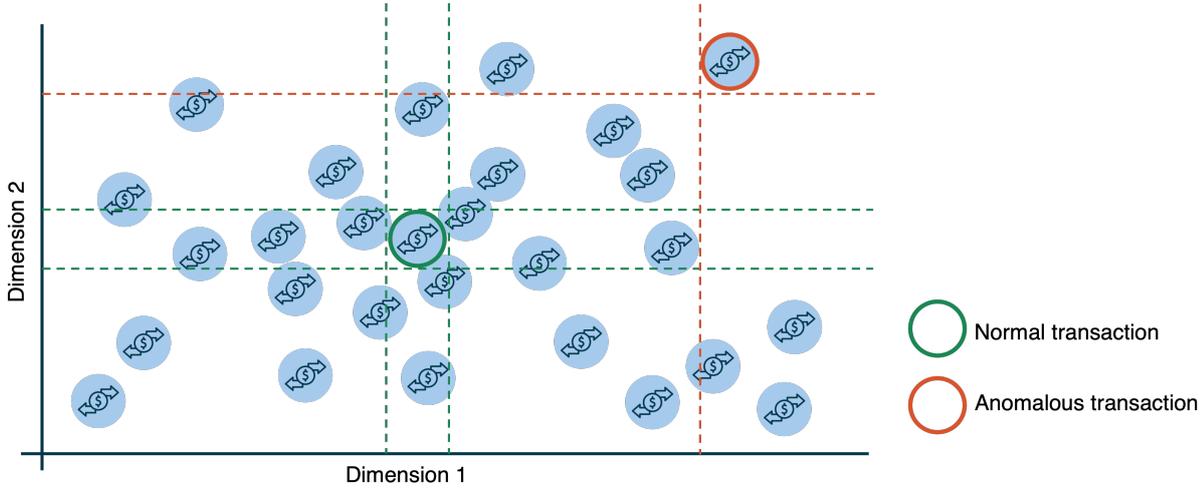


Figure 4.5: Principle of Isolation Forest. The green transaction, situated in a dense region, requires more splits to be isolated compared to the red transaction. The red anomalous transaction necessitates fewer splits (indicated by lines) to be separated from the rest of the data.

builds an ensemble of isolation trees. Each tree partitions the data randomly, generating different isolation paths for each data point across the ensemble. Anomalies are identified by evaluating these isolation paths across all trees. For each data point, the algorithm computes an anomaly score based on the average path length or separation distance across all trees in the ensemble.

IF algorithm details. The algorithm starts by creating a set of *isolation trees (ITs)* $\{t_1, \dots, t_n\}$ base anomaly detectors. ITs are data induced and each internal node v is associated with a randomly chosen splitting feature $f(v)$ and a randomly chosen splitting threshold $\tau(v)$. Each data point is split based on the value of the feature. Points for which the value of $f(v)$ is less than $\tau(v)$ are directed to the left child of node v , while the others to the right.

Given a dataset $D = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ of p -dimensional data points, each IT t is assigned a bootstrap sample $D_t \subset D$ and performs the splits based on the tests associated to internal nodes. Each bootstrap sample has a predetermined size $|D_t| = \psi$ for $t = \{1, \dots, T\}$. Data points in every D_t are recursively partitioned until all the points are isolated into individual partitions or the IT reaches a maximum depth defined as $h_{max} = \lceil \log_2(\psi) \rceil$, depending on the bootstrap sample size ψ . Each data point \mathbf{x}_i will end up in a leaf node $l_t(\mathbf{x}_i)$. The number of edges \mathbf{x}_i passes in its path from the root to the leaf node is equivalent to the depth of the leaf node $l_t(\mathbf{x}_i)$. This procedure is iterated over all the isolation trees, each assigned a different bootstrap sample. The anomaly score for a generic data point \mathbf{x}_i is

computed as

$$s(\mathbf{x}_i) = 2^{-\frac{\bar{h}(\mathbf{x}_i)}{c(\psi)}}$$

where $c(\psi)$ is an estimate of the average path length of a tree built with ψ instances, representing the average path length of an unsuccessful search in a Binary Search Tree, calculated as

$$c(\psi) = \begin{cases} 2H(\psi - 1) - \frac{2(\psi-1)}{\psi} & \text{if } \psi > 2, \\ 1 & \text{if } \psi = 2, \\ 0 & \text{otherwise,} \end{cases}$$

and $H(k)$ is the harmonic number estimated as $H(k) = \ln(k) + 0.5772156649$. $\bar{h}(\mathbf{x}_i)$ is the average path length associated with \mathbf{x}_i and is computed as

$$\bar{h}(\mathbf{x}_i) = \frac{1}{T} \sum_{t=1}^T h_t(\mathbf{x}_i)$$

representing the average path length of \mathbf{x}_i in all the trees of the forest. Figure 4.6 visualizes the ensemble mechanism of the IF model.

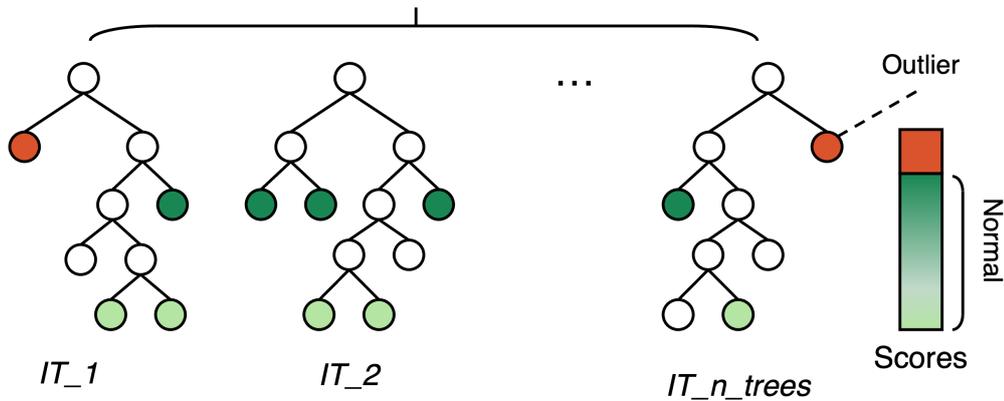


Figure 4.6: Ensemble of Isolation Forest Trees illustrating the isolation process with varying depths, where shorter paths indicate higher anomaly scores.

As a final step, anomalous data points are flagged selecting a threshold on the anomaly scores. This partitions the original dataset D into the subset of predicted inliers $P_I = \{\mathbf{x}_i \in D \mid \hat{y}_i = 0\}$ and outliers $P_O = \{\mathbf{x}_i \in D \mid \hat{y}_i = 1\}$, where $\hat{y}_i \in \{0, 1\}$ is the binary label produced by the thresholding operation indicating where the corresponding \mathbf{x}_i is anomalous ($\hat{y}_i = 1$) or not ($\hat{y}_i = 0$).

Advantages. IF is robust to the presence of outliers in training data since it focuses on isolating anomalies. IF can handle multi-modal distributions since it does not assume any

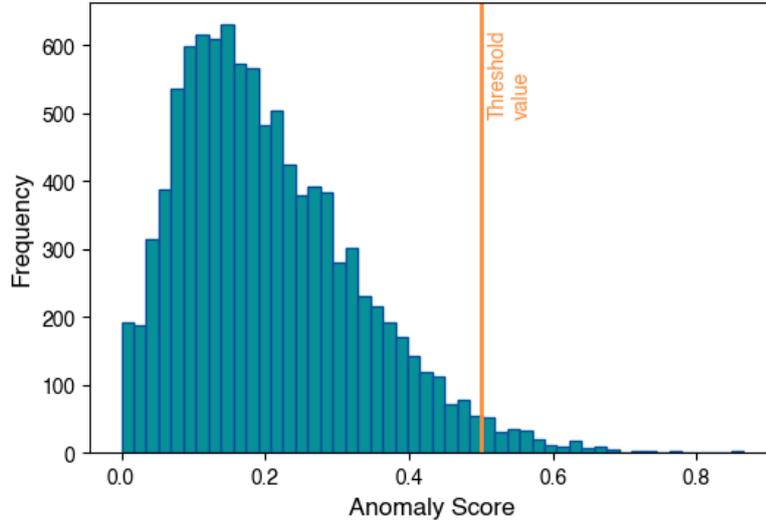


Figure 4.7: Illustration of the thresholding operation on anomaly scores. The plot shows the distribution of scores between 0 and 1, with the threshold (orange line) classifying data into normal (left) and potentially anomalous (right) instances.

specific data distribution. It is also much faster to fit, thanks to the efficient tree-based structure. Finally, IF does not rely on a distance metric to identify anomalies.

4.4.2. Filtering Threshold Selection

The Isolation Forest algorithm assigns a score to each data point based on the contamination parameter, which indicates the portion of anomalies expected in the dataset. Additionally, the model provides a decision function that returns an anomaly score for each data point. To determine a cutoff point that distinguishes between normal and anomalous data points, we employ a statistical approach by selecting a specific percentile of these scores as the threshold. Assuming that 70% of the data is normal, we choose a relatively relaxed percentile to balance the detection of true anomalies and minimize false positives. This approach helps reduce the risk of misclassifying anomalies as normal transactions.

4.5. Graph Representation Learning

Using the IF simple model, we can effectively identify a substantial portion of normal transactions. Building on this foundation, the objective of the graph representation learning phase is to leverage the understanding of *normal* behavior to learn a *normal* representation of the data. This phase incorporates insights from the IF model, which

separates normal and abnormal transactions, to guide the learning process of normal graph embeddings.

In this phase, we leverage a dedicated graph structure, the temporal graph of sequential transactions. This directed graph uses three types of edges to capture transaction dependencies, sequential order, and typical money laundering structures. Each node is associated with specific features, and each edge is assigned a weight. Once the graph is constructed, we extract the subgraph induced by normal nodes and their connections to train a GNN-based model on normal behavior. We then focus our analysis on the scores from the mixed dataset, which includes both normal and abnormal transactions, specifically examining the most suspicious 30% of transactions within the IF score distribution.

4.5.1. Temporal Graph of Sequential Transactions

This graph representation is designed to capture the flow of transactions over time, where each node represents a transaction and each edge denotes a sequential relationship between transactions. As described in Section 2.2, this graph is constructed to represent chains of transactions over time. We utilize a slightly modified version than the one proposed in [58]. We include all the transactions or nodes in the representation, without dropping relationships based on minimum edge weight. This inclusion guarantees the graph remains complete of all the input transactions, enabling a classification of all the input data points. Moreover, we add the fan-out and fan-in edge types. These additions enable the representation of more complex dependencies between nodes.

A pivotal parameter in constructing this graph is the selection of the time *window*, the temporal threshold that defines when two transactions are considered sequentially connected. The size of this window critically affects the graph’s density: larger windows increase the number of edges, capturing more extensive flows but also adding to the graph’s complexity. When setting this parameter, we aim to reach an optimal balance between capturing enough transaction dependency and containing the complexity and density of the graph structure.

Graph Construction. We construct the temporal graph of sequential transactions T from the preprocessed and cleaned transaction dataset D and we specify the time window parameter Δw for linking transactions. The minimum attributes needed for processing transactions are the source account f_i , the target account b_i , the timestamp t_i , and the amount a_i , for each transaction i in D . To link nodes we partition transactions by transaction date and, for each date we perform three types of iterative joins, bounded by the starting transaction date and the window parameter. The final set of edges E includes

three types of edges:

1. **edge_type = flow:** $E_{\text{flow}} = \{(i, j) \in D \times D \mid b_i = f_j \text{ and } 0 < t_j - t_i \leq \Delta w\}$, linking two transactions when the target b_i of transaction i matches the source f_j of transaction j , and the time difference $t_j - t_i$ is within the window Δw .
2. **edge_type = fan-out:** $E_{\text{fan-out}} = \{(i, j) \in D \times D \mid f_i = f_j \text{ and } 0 < t_j - t_i \leq \Delta w\}$, linking two transactions when the source f_i of transaction i matches the source f_j of transaction j , and the time difference $t_j - t_i$ is within the window Δw .
3. **edge_type = fan-in:** $E_{\text{fan-in}} = \{(i, j) \in D \times D \mid b_i = b_j \text{ and } 0 < t_i - t_j \leq \Delta w\}$, linking two transactions when the target f_i of transaction i matches the target b_j of transaction j , and the time difference $t_i - t_j$ is within the window Δw .

For each linked pair of transactions (i, j) in E , we calculate the time difference $\delta = |t_j - t_i|$, which is stored as an attribute in the nodes-attribute matrix X .

Node attributes. To enrich the node representation, we calculate specific node attributes based on transaction amounts and time differences, capturing the characteristics of each transaction within the context of its connected edges. These attributes are calculated for each type of edge from the perspective of both source and target transactions and are then assigned to each node (transaction) accordingly. The attributes are the following:

- **att_1** = $\min\left(1, \frac{\text{transaction amount of the source } f_i}{\sum \text{amounts of all target transactions } b_i}\right)$, representing the proportion of the source transaction's amount relative to the total of all connected target transactions, capped at 1.
- **att_2** = $\text{median}(\delta(f_i, b_i))$, where $\delta(f_i, b_i)$ is the time difference from the source transaction f_i to each connected target transaction b_i , capturing the typical timing of transactions flowing from the source.
- **att_3** = $\frac{|\text{transaction amount of the source } f_i - \text{median amount of all targets } b_i|}{\max(\text{transaction amount of the source } f_i, \text{median amount of all targets } b_i)}$, representing the relative difference in transaction amounts between the source transaction and the median amount of all target transactions connected to that source.

These three attributes are calculated from **flow**, **fan_in**, **fan_out** edges. In particular **att_1** is calculated for **flow** only, **att_2** is calculated for **flow**, **fan_out**, and **fan_in** types, and **att_3** is calculated for **fan_out** and **fan_in**. Then these 6 obtained attributes are re-calculated from the target perspective, leading to 12 total node attributes. These node attributes, derived from both the transaction amounts and the timing information, capture the transaction dynamics in different directions (inflow, outflow, and transactional

flow) and provide a comprehensive view of each transaction’s relational and temporal context within the graph.

Edge weights. The edges within the temporal graph T connect transactions chronologically. To quantitatively reflect this dynamic, the weight of each edge is calculated based on the monetary amounts transferred in the transactions. Given two consecutive transactions i, j connected by an edge $e_{i \rightarrow j}$, we define the edge weight as:

$$w_{i \rightarrow j} = \begin{cases} \frac{\text{amount}_j}{\text{amount}_i} & \text{if } \text{amount}_i > \text{amount}_j \text{ and } \text{edge_type} \text{ not in } [\text{fan_out}, \text{fan_in}], \\ \frac{\text{amount}_i}{\text{amount}_j} & \text{if } \text{amount}_i \leq \text{amount}_j \text{ and } \text{edge_type} \text{ not in } [\text{fan_out}, \text{fan_in}], \\ 1 & \text{if } \text{edge_type} \text{ in } [\text{fan_out}, \text{fan_in}]. \end{cases}$$

This formulation ensures that the edge weight reflects the proportion of the amount being transferred between sequential transactions, offering insights into the relative importance of each transaction within the flow. We decide to set to 1 the edge weight for `fan_out` and `fan_in` to enforce an equal connection and influence of these relationships.

4.5.2. GNN-based Outlier Detection Model

The temporal graph is used as input for a GNN-based outlier detection model. In our framework, our model utilizes a Graph Convolutional Network as the primary backbone. GCNs introduced in [35], are designed to handle graph-structured data $G = (V, E)$ with node features x_v for each $v \in V$ and edge weights. GCN is based on the graph convolutional layers as building blocks, which operate through a message-passing framework. Each node aggregates information from its neighbors to update its own representation, capturing local and global structures. In each layer the GCN message-passing framework propagates features across edges and updates node representations (embeddings) by aggregating features from neighboring nodes. Specifically, the GCN message-passing function for a node i at layer $l + 1$ is given by:

$$h_i^{(l+1)} = \sigma \left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{e_{ji}}{\sqrt{\hat{d}_j \hat{d}_i}} \cdot W^{(l)} h_j^{(l)} \right),$$

where:

- $h_i^{(l)}$ is the embedding of node i at layer l ,
- $\mathcal{N}(i)$ denotes the set of neighboring nodes of i ,

- e_{ji} represents the edge weight between node j and node i (defaulting to 1 if no weight is specified),
- $\hat{d}_i = 1 + \sum_{j \in \mathcal{N}(i)} e_{ji}$ is the adjusted degree of node i , which includes the self-loop,
- $W^{(l)}$ is a learnable weight matrix at layer l ,
- σ is a non-linear activation function, such as ReLU.

Each node i updates its representation by aggregating a weighted sum of the features of its neighbors, where edge weights e_{ji} modulate the contribution of each neighbor j . The normalization factor $\frac{1}{\sqrt{\hat{d}_j \hat{d}_i}}$ provides stability to the feature aggregation ensuring that nodes with varied connectivity (degrees) contribute proportionally.

Graph Data Splitting. We create a full graph data object from nodes and edges, then extract an induced subgraph containing only nodes (transactions) identified as *normal* by the IF model. To simulate an ideal scenario, we exclude any injected anomalous transactions incorrectly included in this set by the IF model, reinserting them into the remaining 30% of the mixed set. These misclassified data points are only a few, but we want to guarantee a clean normal set for the subsequent GNN-based model. In real-world settings, it is challenging to obtain a high-quality class of purely normal transactions. Our normal subgraph accounts for a large part of the transactions, set at 70% of the initial data points. For inference, we reserve the full graph to preserve the node connectivity and the original graphical structure. All node attributes and edge weights are normalized using min-max scaling and bounded to the range [0,1] to ensure balanced contributions during training.

Selected Models. For performing the core outlier detection we train three main models, under similar conditions.

1. **GAE - Graph Autoencoder** [34]: this model uses GCN layers to encode each node’s features into a lower dimensional embedding. The embeddings are then decoded to reconstruct the graph, focusing on adjacency matrix reconstruction. Poorly reconstructed nodes, which show a higher reconstruction error, can be identified as potential outliers.
2. **OCGNN - One Class Graph Neural Network** [63]: this model leverages the one-class principles in the graph setting. OCGNN applies GCN layers to obtain node embeddings and then calculates each node’s distance from a learned centroid in the embedding space. Nodes that are far from the centroid, or more distant, are signals of potential anomalies, as they differ from the normal distribution of the

data.

3. **CoLA - Contrastive Self-Supervised Learning for Anomaly Detection** [44]: this model consists of instance pair sampling, a GNN-based contrastive learning model, and anomaly score computation. First, CoLA samples node-local subgraph pairs. The GNN model then generates embeddings for each target node and its neighborhood, with a discriminator assigning scores that indicate the similarity between a node and its neighbors. Anomaly scores for each node are computed by aggregating scores over multiple sampling rounds, ranking nodes based on their likelihood of being anomalies.

Model Training. For model training, we use a *data loader* to efficiently sample neighbors for each node. Data is loaded with a consistent *batch size*, usually 4096, enabling the model to process a large number of nodes simultaneously. At each layer, the loader samples a limited number of *neighbors*, usually 5, balancing computational efficiency with neighborhood information. The model’s *hidden dimensions* define the size of the hidden layers, influencing the capacity and expressiveness of the extracted embeddings. Training is conducted on a *GPU*, which accelerates processing and is advantageous for large graphs. We select a model depth of 2 *layers*, to strike a balance between complexity and information aggregation from increasingly distant neighbors. The model is trained for 100 *epochs*, allowing it to iterate through the dataset multiple times for improved accuracy.

Inference Step. In the inference step, we pass the entire dataset through the trained model to generate predictions for each data point. We retain the prediction scores as output, with each score indicating the model’s likelihood or the extent to which a data point deviates from the expected behavior. These scores are then used for the final anomaly detection.

4.6. Anomaly Detection Step

The final step of our pipeline involves applying an anomaly detection technique to the scores obtained from the graph outlier detection model. Our testing set comprises the entire temporal graph of sequential transactions, including normal data and a small fraction of abnormal transactions representing money laundering patterns. We focus our evaluation on the 30% of data that complements the normal set. To compute the final anomaly score for each transaction, we follow these steps:

1. **Normalization of Raw Model Scores.** Let s_i denote the raw anomaly score for

transaction i obtained from the graph outlier detection model. We normalize these raw scores to the range $[0, 1]$ using min-max normalization:

$$s_i^{\text{norm}} = \frac{s_i - s_{\min}}{s_{\max} - s_{\min}} \quad (4.1)$$

where s_{\min} and s_{\max} are the minimum and maximum raw scores across evaluation transactions.

2. **Normalization of Transaction Amounts.** Let a_i denote the amount of transaction i . We normalize the transaction amounts to the range $[0, 1]$ using min-max normalization:

$$a_i^{\text{norm}} = \frac{a_i - a_{\min}}{a_{\max} - a_{\min}} \quad (4.2)$$

where a_{\min} and a_{\max} are the minimum and maximum transaction amounts across evaluation transactions.

3. **Initial Combined Score.** We compute an initial combined score by multiplying the normalized model score and the normalized transaction amount:

$$c_i = s_i^{\text{norm}} \times a_i^{\text{norm}} \quad (4.3)$$

4. **Weight Function to Target Specific Amounts.** To focus the detection on transactions with amounts under a specified threshold, we apply a weight function that adjusts the importance of each transaction based on its amount. The weight function is defined as:

$$w(a_i) = \frac{1}{1 + e^{k(a_i - \text{threshold})}} \quad (4.4)$$

where:

- a_i is the amount of transaction i ;
- k is a scaling parameter controlling the steepness of the weight decay (e.g., $k = 0.0001$);
- threshold is the amount threshold (e.g., €35,000).

This function assigns higher weights to transactions with amounts below the threshold and lower weights to those above it, as we target money laundering patterns and believe higher amounts require different investigative approaches.

5. **Final Anomaly Score.** We compute the final anomaly score for transaction i by multiplying the normalized model score, the normalized transaction amount, and the weight function:

$$\text{AnomalyScore}_i = c_i \times w(a_i) \tag{4.5}$$

By incorporating the weight function, we enhance the detection of anomalies in transactions with amounts under the specified threshold, aligning our focus with the amounts where money laundering is suspected to occur. This approach allows us to adjust the model's sensitivity to different transaction amounts, acknowledging that higher amounts may require separate analysis.

4.7. Summary

In this chapter we have outlined our research framework, beginning with the general pipeline structure in Section 4.1. We then detailed the crucial steps of data preprocessing and feature engineering in Sections 4.2 and 4.3, where we demonstrated how to prepare and enhance transaction data using advanced techniques. This was followed by unsupervised filtering of transactions in Section 4.4, a key stage aimed at autonomously identifying normal and abnormal transactions. The framework culminates in 4.5 and 4.6, where we have described the graph representation learning part and the subsequent anomaly detection. This approach allowed us to flag potential anomalies, providing a novel mechanism for detecting suspicious activities.

In the chapters that follow we will explore the datasets used in this study and detail the settings and configuration hyper-parameters for the models presented here, setting the stage for the evaluation and validation of our proposed methodology.

5 | Experimental Setup

This chapter presents the experimental setup used to evaluate our end-to-end framework. In Section 5.1, we introduce the datasets utilized in our experiments, primarily focusing on synthetic data, alongside an undisclosed real aggregated dataset. We explore the raw features of each dataset, discussing their limitations and the types of analysis they enable, given the information available. In Section 5.2, we detail the thresholds, window values, and other fixed parameters encountered during our experiments. We also include a motivation of the criteria used for setting these. Section 5.3 provides a detailed account of all the hyperparameters and settings applied to our models, to ensure a fair reproducibility of the experiments. Finally, in Section 5.4 we address the hardware and software requirements necessary to execute the experiments. It highlights the specific resources required for the successful implementation of the framework.

5.1. Datasets

One of the major limitations in the AML domain is the scarcity of real datasets from financial institutions due to privacy concerns. Moreover, obtaining a labeled dataset by human analysts presents an even greater difficulty. This constitutes a common barrier to experimenting and evaluating a new technique. Therefore we have relied mainly on synthetically generated data for development and evaluation purposes. These datasets typically involve the controlled injection of anomalous behavior or are based on population statistics. In our research, we have made use of the following synthetic datasets:

- **IBM Transactions for AML¹**: consists of six datasets divided into two groups, high-illicit and low-illicit ratio, where each group has small, medium and large size. All datasets are independent, and each of them comes with an additional file listing transactions involved in specific laundering patterns. All transactions are labeled, but not all laundering transactions are part of a specific pattern. Given the high quality of these datasets, we use them as the main source of experimentation and benchmarking.

¹IBM Transactions for AML dataset [3]

- **Real-World Aggregated Data:** is a comprehensive dataset spanning 2010–2020, containing approximately 1.6 million bank accounts and 4.1 million transactions. It includes unique identifiers for accounts, transaction counts, and total exchanged amounts, providing an aggregate view of transaction activity, from a bank’s perspective.

Further characteristics of each dataset are summarized in Table 5.1.

Dataset	Transactions	Accounts	Laundering Rate	Label
IBM Transactions for AML	5M-180M	500K-2100K	<0.12%	✓
Real-World Aggr. Data	4.1M	1.6M	N/A	✗

Table 5.1: Summary of descriptive statistics for the available datasets.

5.1.1. Synthetic Data

Following the overview of our datasets, we now focus on the *IBM Transactions for AML* set, which serves as our primary dataset due to its comprehensive simulation of real-world dynamics. This dataset models diverse scenarios typical of financial environments, thereby providing a robust platform for testing our models. These synthetic datasets consist of six datasets divided into two groups, high-illicit and low-illicit ratios, that simulate real financial transactions. The generator behind these datasets models a multi-agent virtual world composed of banks, companies, and individuals. The interactions between entities are associated with features such as account number, bank number, timestamp, currency type, amount, and payment format. A small fraction of agents within this ecosystem engage in illicit activities. The statistics for these datasets are detailed in Table 5.2.

Statistic	Small		Medium		Large	
	HI	LI	HI	LI	HI	LI
# Accounts	515K	705K	2077K	2028K	2116K	2064K
# Transactions	5M	7M	32M	31M	180M	176M
# Laundering Trans.	5.1K	4.0K	35K	16K	223K	100K
Laundering Rate (1 per N)	981	1942	905	1948	807	1750

Table 5.2: Statistics of synthetic transaction datasets.

Figure 5.1 illustrates a sample of these transactions, giving a visual example of how the features appear in a single data point.

- **Timestamp:** recorded in the format year/month/day hour/minute. Depending on the analysis, we extract the date component for aggregation purposes. We use it to

Timestamp	From Bank	Account	To Bank	Account.1	Amount Received	Receiving Currency	Amount Paid	Payment Currency	Payment Format	Is Laundering
2022/09/01 00:25	138395	80E5C5420	112	80EB2EFA0	9370.51	Swiss Franc	9370.51	Swiss Franc	Cash	0
2022/09/05 16:44	908	8008F4340	10642	8075F6CE0	107.24	US Dollar	107.24	US Dollar	Cheque	0
2022/09/07 00:51	16606	808D15020	113779	809BA2F80	4499.94	US Dollar	4499.94	US Dollar	Cheque	0
2022/09/09 02:16	11	800244AE0	47886	812508A10	22064.49	Euro	22064.49	Euro	Credit Card	0
2022/09/14 14:36	21611	8051B3BF0	11296	800C08200	17902.99	US Dollar	17902.99	US Dollar	ACH	1

Figure 5.1: Sample of realistic synthetic transactions.

order transactions in time. However, it is important to note that the order of transactions can differ, as the minute-level granularity allows for multiple transactions to occur simultaneously;

- **From Bank/To Bank:** represent the numeric codes for the originating bank and the receiving bank in a transaction, respectively. The presence of numerous unique values indicates that there is a wide network of banks behind. These bank codes can be combined with the account identifiers to generate more unique account references. Additionally, grouping by these fields enables the analysis of bank network flows, providing insights into how transactions are distributed across different financial institutions;
- **Account/Account.1:** contain the hexadecimal codes representing the accounts where the transaction originates and where it is received. There is a high number of unique accounts in the datasets, with some accounts conducting only a few transactions, while others stand out by a significant volume, almost 1-2% of total transactions. These high-volume accounts function similarly to central banks, with high outbound transfers;
- **Payment Currency/Receiving Currency:** represent the currencies used by the sender and the receiver accounts, encompassing up to 15 distinct values among the transactions. Various combinations of these currencies appear. To create unique identifiers, we merge the account IDs with the currencies. Additionally, to standardize the transactions, we infer exchange rates and attempt to convert all the transactions to a common currency;
- **Amount Paid/Amount Received:** represent the monetary amount transferred measured in the sender’s currency and the amount received in the receiver’s currency. These fields are equal when the sender’s currency matches the receiver’s currency. We utilize this information, along with the associated currency, to infer daily ex-

change rates. Additionally, these fields can reveal round amounts, near-round numbers, or amounts close to known thresholds, potential indicators of suspicious activities;

- **Payment Format:** indicates the type of transaction of each record and can assume several distinct values, including *Reinvestment*, *Cheque*, *Credit Card*, *ACH*, *Cash*, *Wire*, and *Bitcoin*. In particular, *Reinvestment* refers to the practice of using earnings from an investment to purchase additional units or shares. This type of transaction is usually a self-transaction, where the source equals the target. Then *ACH* (Automated Clearing House) transactions are electronic payments used in the US for direct transfer of funds. This type appears the most used for performing money laundering transactions in the datasets. It is interesting to note that these datasets also include *Bitcoin*, which sometimes is handled separately from other transaction types.
- **Is Laundering:** a binary class label where 1 indicates a transaction involved in money laundering, and 0 indicates it is not. The label applies to both individual laundering transactions and those that are part of broader laundering patterns. The laundering tag is transitive, meaning that it propagates through all subsequent transactions involving illicit funds, ensuring comprehensive tracking of laundering activities.

These synthetic datasets are specifically engineered to replicate complex financial flows and patterns that span multiple banks and currencies. These include eight distinct laundering techniques designed to mimic real-world money laundering activities, such as smurfing, layering, and round-tripping. These patterns represent common laundering schemes, categorized under *known* suspicious patterns, but might also appear in legitimate transactions. Specifically, this set of datasets includes a supplementary reference that identifies which transactions are linked to one of eight simulated laundering patterns. Each pattern, referred to as a *laundering attempt*, is detailed by typology and description within Table 5.3. While our framework primarily targets these predefined patterns, it aims to identify more broadly any transaction that exhibits suspicious characteristics.

5.1.2. Real-world Data

As part of our experimental evaluation, we utilize a real-world banking transaction dataset. Due to privacy concerns, detailed individual information cannot be disclosed. The dataset aggregates transactions spanning from 2010 to 2020. This dataset encompasses an extensive network of transactions involving approximately 1.6 million bank accounts and 4,1

Typology	Description
<i>fan-in</i>	one account receives money from multiple accounts
<i>fan-out</i>	one account sends money to multiple accounts
<i>scatter-gather</i>	<i>fan-out</i> followed by <i>fan-in</i>
<i>gather-scatter</i>	<i>fan-in</i> followed by <i>fan-out</i>
<i>cycle</i>	A sends money to B, B forwards to C, C transfers to A
<i>bipartite</i>	two accounts having an unobvious relationship with each other
<i>stacked bipartite</i>	extension of bipartite
<i>random</i>	a chain of similar amount transactions

Table 5.3: The patterns modeled in the IBM Transactions for AML datasets.

million transactions. The dataset is characterized by several key features:

- **start_id**, **end_id**: unique identifiers for the source and target accounts involved in the transactions;
- **year_from**, **year_to**: delineate the beginning and end of the period during which the transactions were recorded;
- **total**: the aggregated amount exchanged between accounts during the specified period;
- **count**: the total number of transactions conducted between the accounts within the timeframe.

start_id	total	count	year_from	year_to	end_id
hash_id_01	318	2	2018	2020	hash_id_02
hash_id_03	249	1	2020	2020	hash_id_04
hash_id_08	439	7	2018	2020	hash_id_07
hash_id_05	1000000	1	2020	2020	hash_id_11
hash_id_12	3920951	143	2015	2020	hash_id_06

Table 5.4: Sample of real aggregated transactions.

As expected, this dataset presents some challenges that complicate its integration into our pipeline. Primarily, the dataset only includes aggregated periods (in years) rather than single transaction timestamps, which hampers our ability to quantify temporal dependencies among transactions. While exploring layered networks constrained by time ranges is a potential workaround, it deviates from the focus of our current work. Additionally, the dataset is limited in terms of available attributes, which restricts the scope of our feature engineering efforts. This dataset seems better suited for network analysis and community detection.

Due to these challenges, we only test this dataset up to the IF model. It is important to note that the findings about this dataset are limited in scope, given its inherent constraints. This experiment helps to assess the broader applicability of our framework under limited data information.

5.2. Parameters

These are the variables that we fix before executing pipeline steps. For feature engineering, one key parameter is the window size for calculating percentage-based features, which compare a transaction to prior or future activity, called `rolling_days`. After testing, we selected a 5-day window, a trade-off given the transaction activity of the accounts in the datasets. For calculating the round amount boolean, we configure a `round_amount_multiplier`, `slack`, and `minimum_transaction_volume`, to determine the *roundness* of an amount. In the Graph Feature Preprocessor (GFP), we use the settings suggested by [6]: a six-hour `window` for scatter-gather patterns, a one-day `window` for other graph-based features, and a `cycle-length` constraint of 10, using `amount` and `timestamp` fields for vertex-statistics-based features. The `threshold` for the Isolation Forest is defined to set a reasonable separation between normal and anomalous transactions, aiming to reduce misclassification, though this is challenging. For this, we set the threshold assuming 70% of the data as normal and 30% of the data as potentially suspicious. For the temporal graph analysis, we set the `window` to 7 days after testing with different windows. This achieves a balance between capturing sufficient transaction connections and controlling graph density, which increases substantially with larger windows. Further considerations on these choices are provided in section 6.3, where we present results.

5.3. Hyperparameters

The selection and tuning of hyperparameters are crucial in optimizing the performance of the models used in our anomaly detection framework. This section details the final set of hyperparameters used for the IF model and GNN models.

IF Model. We tuned the IF model through experimentation and testing, to maximize the recall of anomalous instances, balancing the false positives. Our goal was to maximize the Area Under the Curve (AUC) metric, defined through the receiver-operator characteristic (ROC) curve, a curve of the true positive rate over the false positive rate for various thresholds (see section 6.2). Our approach focused on isolating normal anomalous data points from normal data points, maintaining a low false positive rate. Initially, we relied on

the well-established default parameters from the literature, which apply to a wide variety of tasks and datasets. However, through a series of experiments, we achieved the best results changing the following: (i) `n_estimators`, set to 100 to provide sufficient depth minimizing the risk of overfitting; (ii) `max_samples`, set to 0.01, ensuring a representative sample size of 1% of the data for each isolation tree; (iii) `contamination`, initially set to the known fraction of anomalies in the dataset but disregarded as we set the threshold assuming 70% of the data to be normal. Additionally, we set `random_state` to 42 for consistent runs, and `n_jobs` to `-1`, to use all the CPU cores.

GNN Model. The model is implemented using GNN, AE, and SSL frameworks. The hidden layers are set with the `hid_dim` parameter usually as 16. The `num_layers` parameter is set to 2 to capture and aggregate neighborhood information up to two hops from each node. Training is designed via mini-batch and sampling, where `batch_size` is set to values that fit in memory, as 4096, and `num_neighbors` is the number of nodes sampled at each iteration, set to 5. The data loader used is `NeighborLoader`. The models are trained over 100 epochs and the `learning_rate` is fixed to 0.004. All the tested models are implemented in a unified Python library for graph outlier detection[43].

5.4. Setup

Our experiments were conducted on a high-performance machine equipped with an Intel(R) Xeon(R) Platinum 8360Y CPU, with 512GB of RAM, and Linux 5.14 on an x86_64 architecture. For the implementation we used Python v3.11² for analysis and model development, taking advantage of its rich ecosystem for data processing, visualization, and machine learning. To improve efficiency in key sections of the pipeline, we incorporated `pyspark`³. GNN models used in the project were implemented with `pygod`⁴, which offers a convenient API for a range of graph-based outlier detection models. Our implementation is documented in the GitHub repository⁵. Core libraries included `numpy` and `pandas` for data manipulation, `scikit-learn` and `pytorch` for model building, `networkx` and `igraph` for graph processing, `snapml` for leveraging the Graph Feature Preprocessor (GFP), and `matplotlib` for creating visualizations and plots.

²Python version: <https://www.python.org/downloads/release/python-3110/>

³PySpark documentation: <https://pypi.org/project/pyspark/>

⁴PyGOD documentation: <https://docs.pygod.org/en/latest/>

⁵GitHub repository: https://github.com/alenkaja00/aml_transaction_ad.git

6 | Experimental Evaluation and Results

In this chapter, we present the evaluation of our proposed approach for self-supervised anomaly detection in money laundering. Our pipeline combines an Isolation Forest (IF) for transaction filtering and scoring, followed by a Graph Representation Learning phase. In this second stage, a GNN-based model is used to process a temporal graph of sequential transactions, leveraging the extracted normal behavior from the unsupervised model. The output score of this model is refined and corrected to produce a final anomaly score. This final score is used to rank transactions based on their *risk*. We evaluate and discuss the performance and runtime of these models individually, with a focus on the final results achieved from the GNN-based model. Additionally, we provide insights about core components, such as the feature enrichment module and the temporal graph construction. Our pipeline does not rely on labels, but we use the available labels only for evaluation purposes. In the following sections, we describe the goals of our evaluation, the metrics used, and the main results.

6.1. Goals

Through the experimental evaluation, we investigate **RQ3** and **RQ2** to get insights into the usefulness, effectiveness, and accuracy of our anomaly detection pipeline. This research question will help us understand how well the pipeline performs if exposed to different scales of data. We focus on two main questions: (i) assessing the scalability of the proposed approach in detecting anomalous transactions within large-scale financial transaction graphs, and (ii) examining the temporal graph of sequential transactions, including edge types and size across various data scales.

True Class	Predicted Class Positive	Predicted Class Negative
Positive	True Positive (TP)	False Negative (FN)
Negative	False Positive (FP)	True Negative (TN)

Table 6.1: Confusion matrix.

6.2. Metrics

Before introducing the metrics, it is important to define key terms. A *True Positive (TP)* represents a laundering transaction correctly identified as anomalous; a *False Positive (FP)* represents a normal transaction incorrectly classified as anomalous; a *True Negative (TN)* represents a normal transaction that is correctly identified as non-anomalous; a *False Negative (FN)* represents a laundering transaction that is incorrectly classified as normal. These values form the confusion matrix, shown in Table 6.1. Given the highly imbalanced nature of our datasets, we focus on metrics that are more meaningful for anomaly detection. This means excluding from the evaluation metrics such as *accuracy*, defined as $= \frac{TP+TN}{TP+TN+FP+FN}$, as a model can achieve high accuracy by simply predicting all instances as the majority class, thus failing to detect any anomalies. Based on these considerations, common metrics used to assess an anomaly detection system are the following:

- **Precision:** measures the proportion of correctly identified laundering transactions among all transactions classified as anomalous. Precision will be low if the model generates many false positives.

$$Precision = \frac{TP}{TP + FP} \quad (6.1)$$

- **Recall or True Positive Rate (TPR):** measures the proportion of true positive anomalies among all actual laundering transactions. Recall is high when the model effectively detects true positives, minimizing false negatives.

$$Recall = \frac{TP}{TP + FN} \quad (6.2)$$

- **F1-Score:** is the harmonic mean of precision and recall, which is useful for imbalanced datasets where class distribution is uneven.

$$F1_score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (6.3)$$

- **False Positive Rate (FPR):** measures the proportion of normal transactions incorrectly classified as anomalous.

$$FPR = \frac{FP}{FP + TN} \quad (6.4)$$

- **False Negative Rate (FNR):** measures the proportion of laundering transactions missed by the model.

$$FNR = \frac{FN}{FN + TP} \quad (6.5)$$

- **AUC-ROC:** measures the Area Under the Receiver-Operating Characteristic (ROC) Curve. The ROC curve plots the TPR and FPR at every threshold. The AUC score reflects the probability that the model will rank a randomly chosen positive example higher than a negative one. The score ranges from 0 to 1, where 0.5 indicates random guessing and 1 represents a perfect model.

$$AUC = \int_0^1 TPR(FPR^{-1}(x)) dx \quad (6.6)$$

- **Average Precision (AP):** measures the area under the precision-recall curve, indicating the model's ability to maintain high precision across recall levels. AP is especially useful for evaluating performance on imbalanced datasets.

$$AP = \sum_n (Recall_n - Recall_{n-1}) \cdot Precision_n \quad (6.7)$$

6.3. Results

In this section, we present and discuss the experimental results of our pipeline. As noted in [24], most evaluation protocols for anomaly detection methods rely on metrics such as F1-score or Average Precision (AP). However, these metrics are sensitive to the contamination rate of the dataset. In contrast, AUC is largely unaffected by arbitrary choices in the evaluation protocol. Since AUC does not depend on a specific threshold, it will offer better comparability across different models and datasets.

6.3.1. Dataset Statistics

The table below illustrates the impact of our pre-processing and filtering steps in refining the dataset. First, we eliminate injected anomalous transactions that don't contribute

to actual money laundering patterns. Next, we retain only those pattern transactions that have more than one linked transaction and are part of a weakly connected directed graph. This ensures that, when ignoring direction, there is a path between each pair of nodes. Lastly, we filter based on connectivity by setting thresholds of $t_{\text{src}} = 500$ for source accounts and $t_{\text{dst}} = 500$ for target accounts. This step preserves only transactions from accounts with fewer connections, helping to reduce noise by excluding highly connected accounts.

Dataset	HI-Small	LI-Small
Initial Transactions	5,078,345	6,924,049
<i>Filtered Transactions</i>	<i>4,619,279</i>	<i>6,300,203</i>
Reduction (%)	9.03	9.01
Initial Anomalies	3,209	1,023
<i>Filtered Anomalies</i>	<i>2,491</i>	<i>722</i>
Reduction (%)	22.37	29.42
Initial Patterns	370	117
<i>Filtered Patterns</i>	<i>270</i>	<i>84</i>
Reduction (%)	27.03	28.21
Anomalies/Transactions (%)	0.05	0.01

Table 6.2: Dataset Statistics Before and After Pre-processing and Filtering.

6.3.2. Stage 1: Isolation Forest Model

In this stage, we train an Isolation Forest (IF) model on the preprocessed transaction dataset and obtain anomaly scores for each data point using the model’s decision function. We set an anomaly detection threshold at the 70th percentile of the IF score distribution, flagging transactions above this threshold as potentially suspicious and those below as normal. Model performance is then evaluated using a confusion matrix with TP, FN, FP, and TN. Table 6.3 presents the metrics across tested datasets, Figure 6.1 displays the confusion matrices, and Figure 6.2 shows the AUC-ROC curves.

Anomaly Scores Distribution. In Figure 6.3 we present the distribution of the IF scores, for the *HI-Small* and the *LI-Small* datasets, normalized in the range $[0, 1]$ on the y-axis and transaction indices on the x-axis. The plot shows that the scores for anomalous

Dataset	Precision	Recall	FPR	FNR	AUC	AP
HI-Small	0.18	97.63	29.96	2.37	94.88	2.27
LI-Small	0.04	95.29	29.99	4.71	92.67	0.42

Table 6.3: Performance Metrics (%) for HI-Small and LI-Small Datasets.

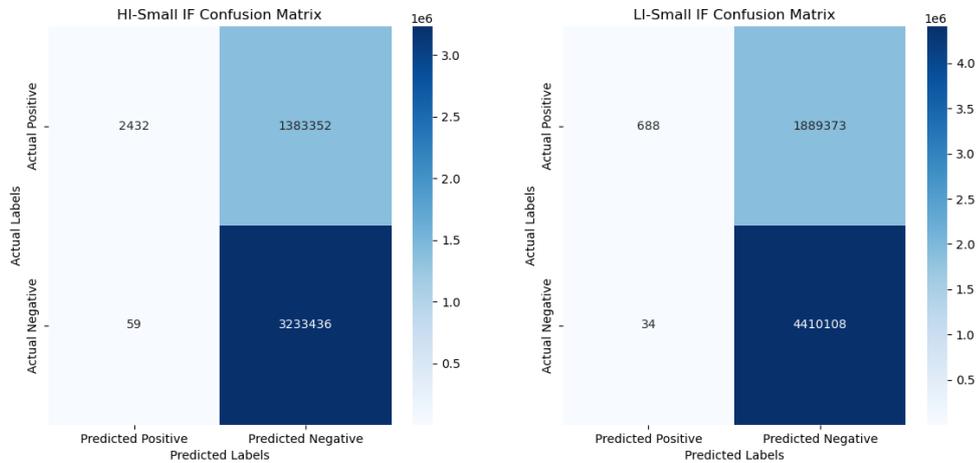


Figure 6.1: Confusion Matrices for the HI-Small and LI-Small Datasets.

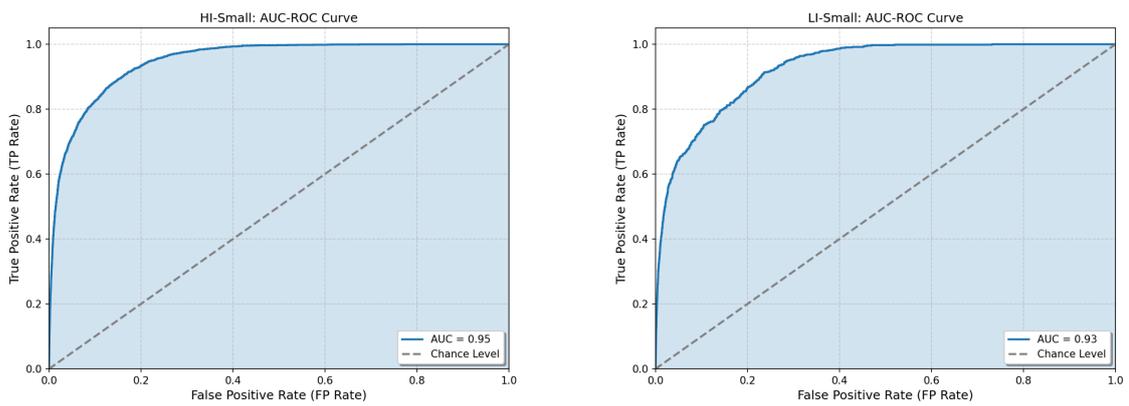


Figure 6.2: AUC-ROC Curves for HI-Small and LI-Small Datasets.

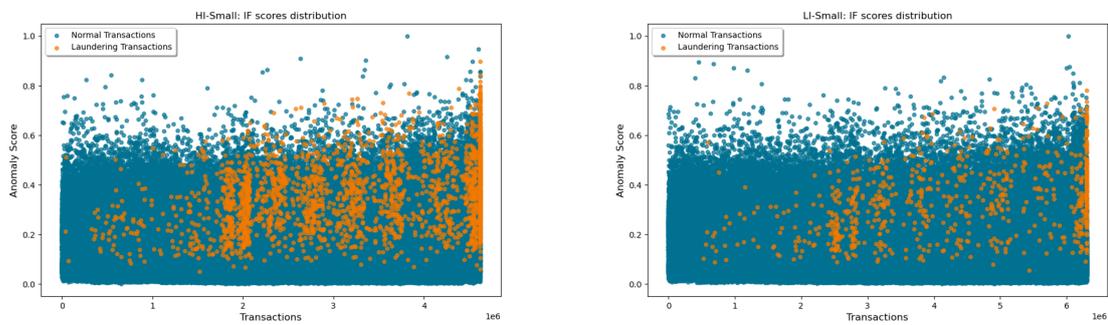


Figure 6.3: Scores for real *Normal* Transactions (blue) and *Anomalous* Transactions (orange).

money laundering transactions overlap with those of normal transactions, suggesting that these anomalous points are well-camouflaged within the dataset.

Model Interpretation. This is a simple model, primarily used as a filter. By adjusting the `threshold`, or `contamination` parameter in IF, we can balance *Recall* by allowing a variable number of *FP*. Experimental testing shows that the model efficiently handles extreme outliers, with a high AUC value indicating strong differentiation between normal and abnormal instances. However, in unsupervised anomaly detection, certain metrics may not fully reflect model performance as they do in supervised settings; here, *Precision* remains low, while *Recall* improves relaxing the cut point and allowing more FP, at the detriment of *Precision*. Overall, this model is well-suited for identifying outliers and separating them from the bulk of normal data, serving as an initial filter for isolating a large part of *normal* transactions from anomalous ones for further processing.

6.3.3. Stage 2: GNN-based Models

We test three GNN-based outlier detection models under the same conditions, with each model scoring input nodes (transactions). These models are trained on the graph data object derived from the induced subgraph of *normal* nodes identified by the IF model, with false negatives removed to ensure a clean training set. Inference is conducted on the entire graph to retain connectivity and structure. Our evaluation focuses specifically on the subset of nodes in the inference set, which includes both *normal* and *abnormal* nodes. Each model outputs an anomaly score for every node, which we refine by factoring in the transaction amount. The table below presents the final performance of the GNN-based models on the two synthetic datasets. The best results are highlighted in bold, with the second-best underlined. Overall, the models demonstrate similar performance across both datasets.

	HI-Small	LI-Small
GAE [34]	78.67	78.82
OCGNN [63]	70.70	73.43
CoLA [44]	<u>78.11</u>	<u>78.14</u>

Table 6.4: AUC-ROC Performance (%) of GNN-based Models on Synthetic Datasets.

In this evaluation, we assessed model performance by labeling the top 30% of highest-scored transactions as suspicious. Confusion matrix plots in Figure 6.6, Figure 6.7, and Figure 6.8 provide a detailed view of predictive accuracy. For each dataset and model, we calculated a combined anomaly score and set the 70th percentile as the threshold to

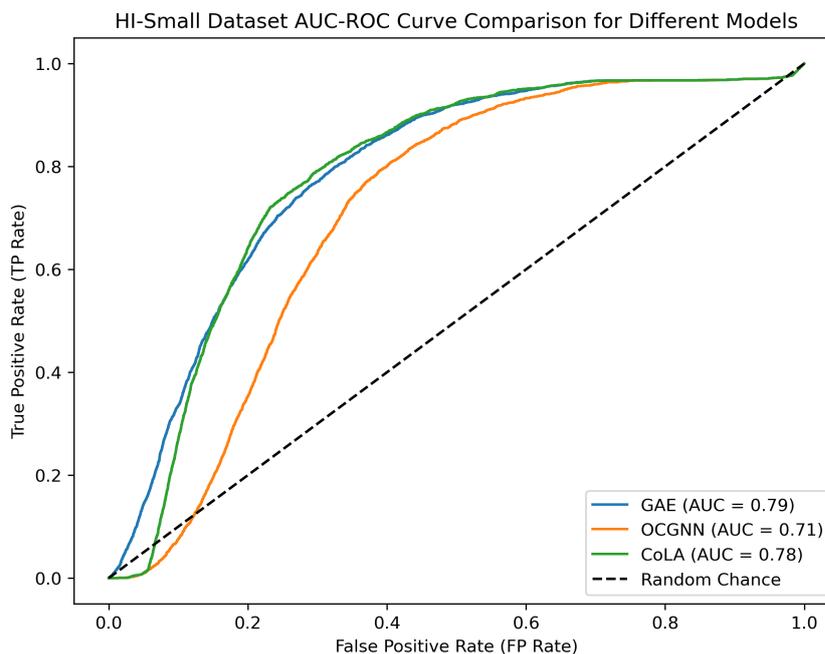


Figure 6.4: AUC-ROC Curve Comparison for the HI-Small Dataset.

identify the top 30% of transactions as high-risk or suspicious. Transactions above this threshold were labeled as *positive* (likely anomalies), while those below were labeled as *negative* (likely normal).

Anomaly Scores Distribution. The GNN-based models — GAE, OCGNN, and CoLA — demonstrate distinct scoring behaviors after applying the score refinement process. GAE predominantly assigns low anomaly scores, suggesting it identifies most data points as normal, with limited high-risk flags. OCGNN, with a broader score range, shows moderate sensitivity to anomalies. In contrast, CoLA produces scores concentrated near the high end, indicating a more aggressive anomaly detection approach. For better reference, Figure 6.9 and Figure 6.10 show the raw scores of the three models on each dataset before correction by transaction amount and weighting. The score refinement notably enhances AUC-ROC performance by up to 20%, achieving the results shown in Table 6.4 and underscoring the importance of post-processing for improved model sensitivity and interpretability.

Model Interpretation. This model offers an innovative approach to learning from large volumes of normal transaction data, capturing flow, fan-out, and fan-in relationships to target a wide range of money laundering patterns. It relies primarily on time-window connectivity to link transactions over time and performs best when trained on normal transaction instances. Additionally, the model depends on multiple hyperparameters,

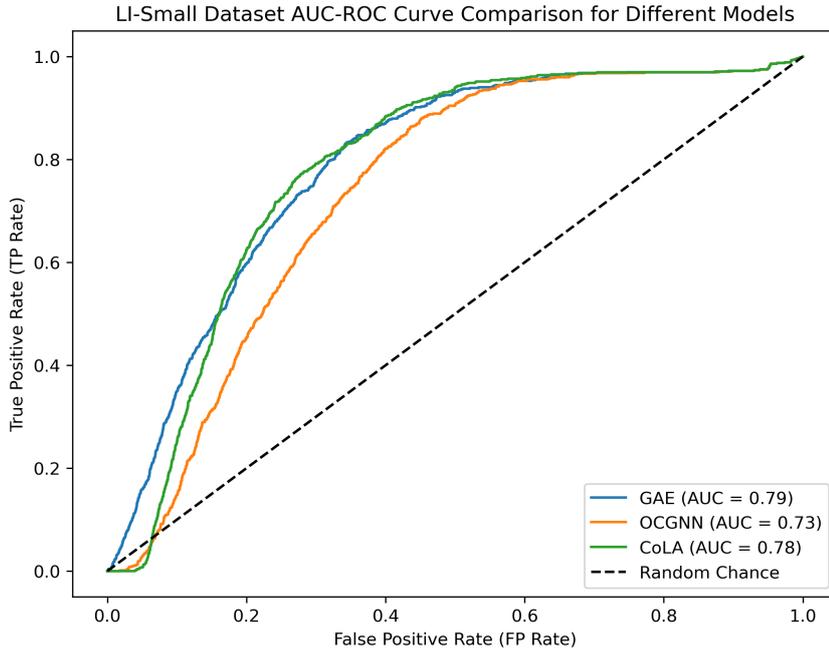


Figure 6.5: AUC-ROC Curve Comparison for the LI-Small Dataset.

which require careful tuning to optimize performance.

Best Performing Model. All tested models represent state-of-the-art approaches in graph outlier detection, each utilizing a distinct strategy for anomaly detection. Overall, there are no significant differences in performance among the three models. *GAE* appears to be the best-performing model across both datasets, offering specific advantages. The post-processing step, or score correction, seems to align the models' performance, possibly due to the datasets' similarity in size and attributes.

6.3.4. Scalability

In addition to evaluating model performance, we assess the scalability of key components in our pipeline. This analysis focuses on the temporal graph of sequential transactions and the training scalability of GNN-based models. We also provide insights into other components within the pipeline.

Preprocessing, Feature Engineering, and Graph Feature Preprocessor. Preprocessing is straightforward and minimally resource-intensive. In contrast, feature engineering is the most computationally demanding stage and requires careful handling for large transaction datasets. We experimentally assess the time and memory performance of the Graph Feature Preprocessor (GFP), a flexible module with parameters such as time windows, bin sizes, and cycle detection hops. In this research, we fixed GFP parameters

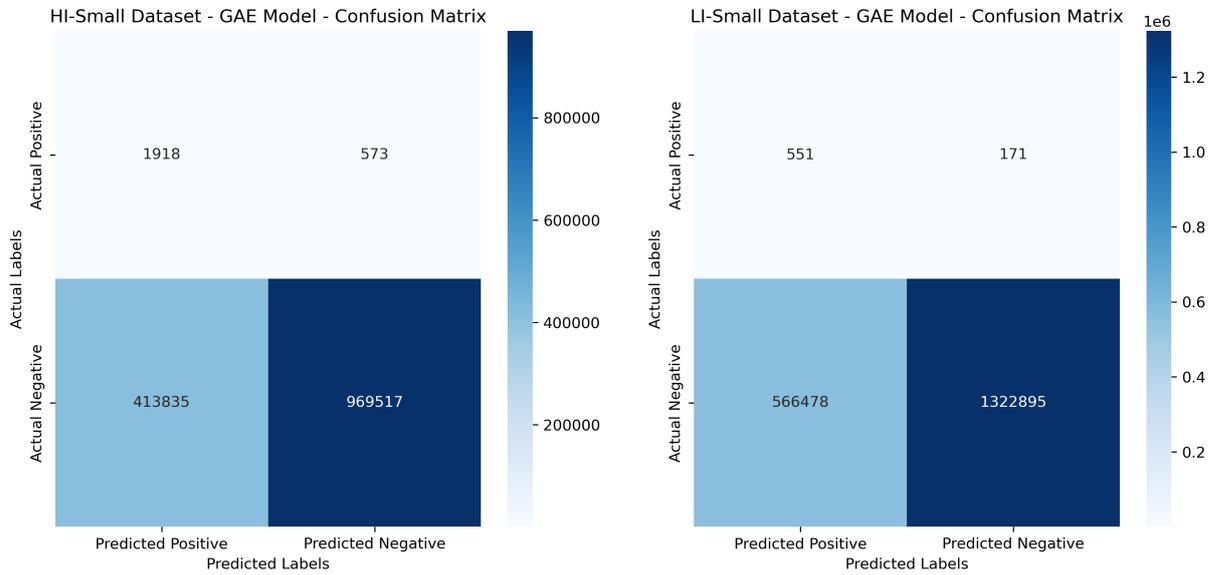


Figure 6.6: GAE Model - Confusion Matrix for HI-Small and LI-Small Datasets.

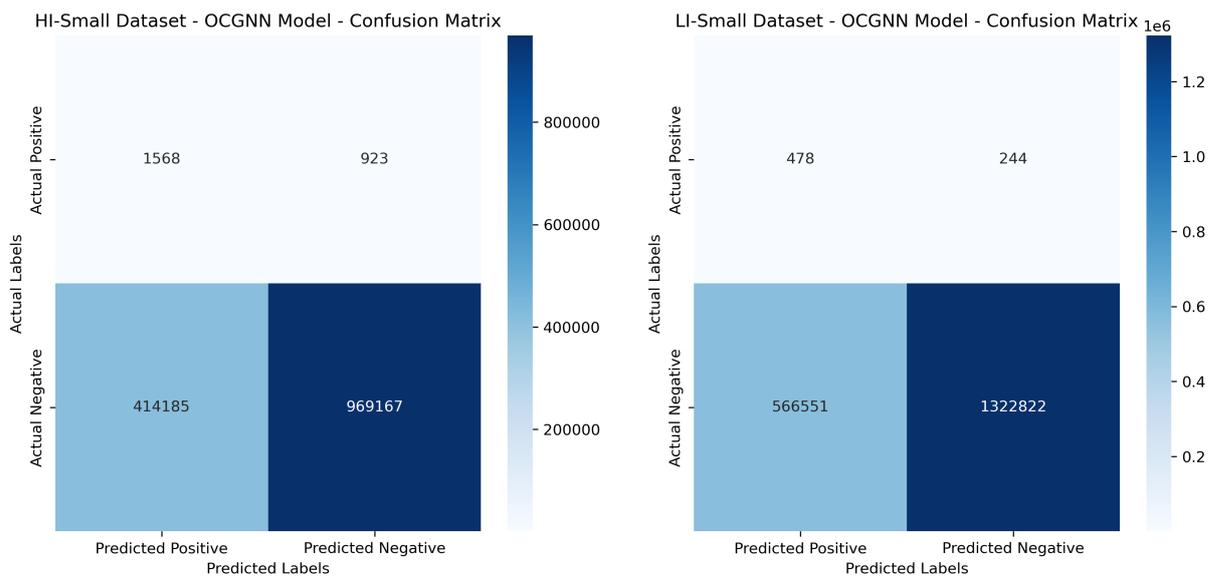


Figure 6.7: OCGNN Model - Confusion Matrix for HI-Small and LI-Small Datasets.

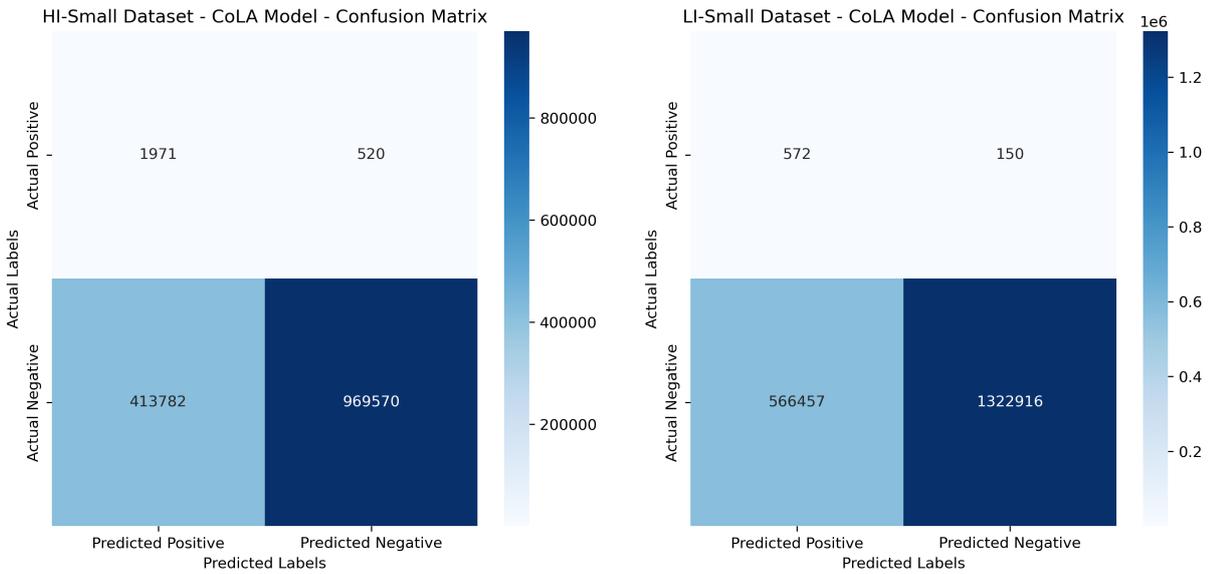


Figure 6.8: CoLA Model - Confusion Matrix for HI-Small and LI-Small Datasets.

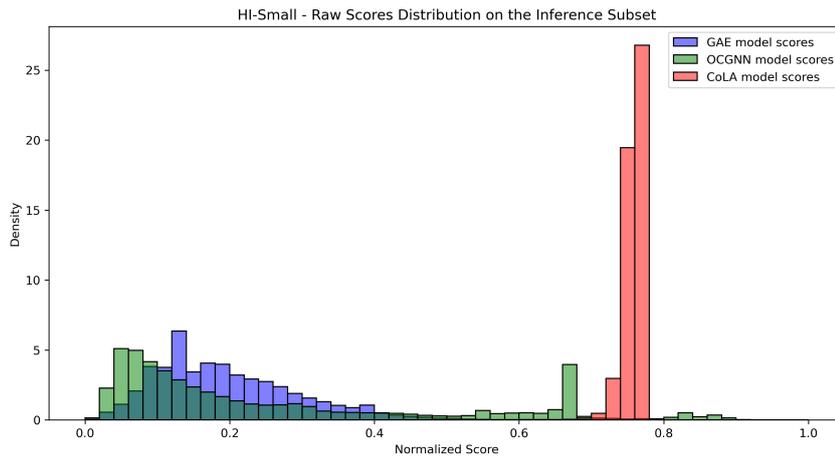


Figure 6.9: Model Raw Output Scores for HI-Small dataset.

to a standard configuration consistent with prior work [6].

Isolation Forest Model. Experimental testing indicates that training time scales linearly with dataset size, suggesting that computational demand is influenced more by hyperparameter settings than by data dimensionality. These findings highlight Isolation Forest’s suitability for high-dimensional data, supporting its effectiveness in anti-money laundering applications where handling large data volumes and timely decision-making are critical factors.

Temporal Graph Creation. Increasing the window parameter significantly raises the number of edges and the density of connections, requiring efficient handling. The joins,

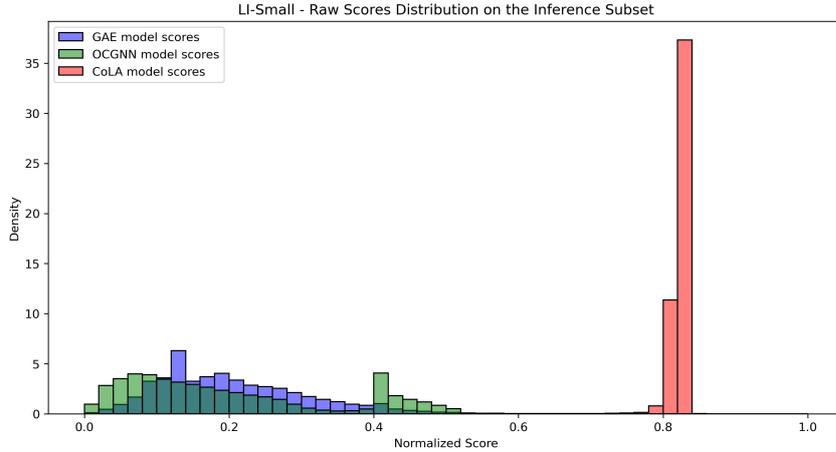


Figure 6.10: Model Raw Output Scores for LI-Small dataset.

organized by transaction date partitions, scale effectively. Removing high-degree nodes further streamlined iterative join operations, as high-degree accounts generate numerous connections for each dependency type, which can slow down processing. However, the overall processing remains highly efficient, as introduced in prior work [58].

	HI-Small Dataset	LI-Small Dataset
#nodes	4.6M	6.3M
#edges - $\Delta w = 3$	103.5M	140.7M
#edges - $\Delta w = 5$	144.1M	195.8M
#edges - $\Delta w = 7$	174.4M	237.1M

Table 6.5: Total number of temporal graph edges across different time-window parameters.

GNN-based Models. We report the training time per epoch (in seconds) for each model, along with total training times in Table 6.6 and Figure 6.13. Across datasets, runs were consistent, with the GAE model achieving the minimum epoch training time of 14.1 seconds on the HI-Small dataset and CoLA reaching a maximum of 24.4 seconds on the LI-Small dataset. Although CoLA is the most computationally expensive model, its training time is not significantly higher than the others. Total training time varied from approximately 23 to 40 minutes, depending on the model and dataset.

Model	HI-Small (s)	LI-Small (s)
GAE	14.1	19.9
OCGNN	14.4	20.5
CoLA	19.6	24.4

Table 6.6: Training time per epoch (in seconds) for different models on each dataset.

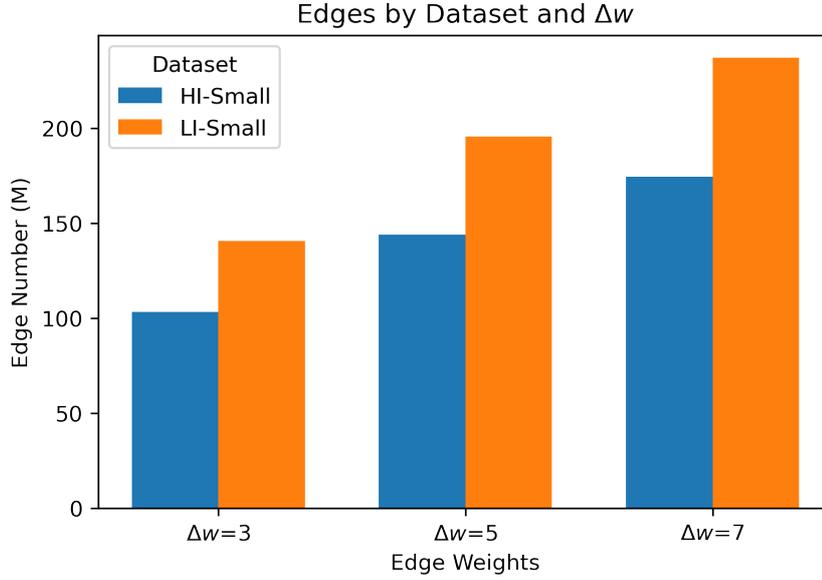


Figure 6.11: Increase of the total number of graph edges across different time-window parameters.

Scalability is crucial for real-world applications, where datasets consist of millions or even billions of transactions. Our results indicate that feature engineering is the most resource-intensive component, with manageable processing times on smaller datasets but increased demands on larger ones. With careful management of scale, however, this pipeline can be deployed in production environments where transaction monitoring occurs in batches or routine cycles. Further improvements are possible in feature engineering and model training by implementing techniques such as batching and dataset partitioning to optimize performance across large-scale datasets.

6.3.5. Real-World Data Experiment

This experiment uses a dataset with limitations deriving from its aggregated time format, which prevents the creation of the temporal graph of sequential transactions. Additionally, as the data is sourced from a single bank, it only reflects intra-bank transactions. Only a few risk indicators are assessable with this dataset given its attributes, presented in Table 5.4.

Network Analysis. A *directed graph* was constructed using source and target accounts for each transaction. Decomposing the graph into weakly connected components (WCC) revealed that nearly all accounts (99.8%) belong to a single large component. This predominant structure led us to focus exclusively on this main component, as transactions in

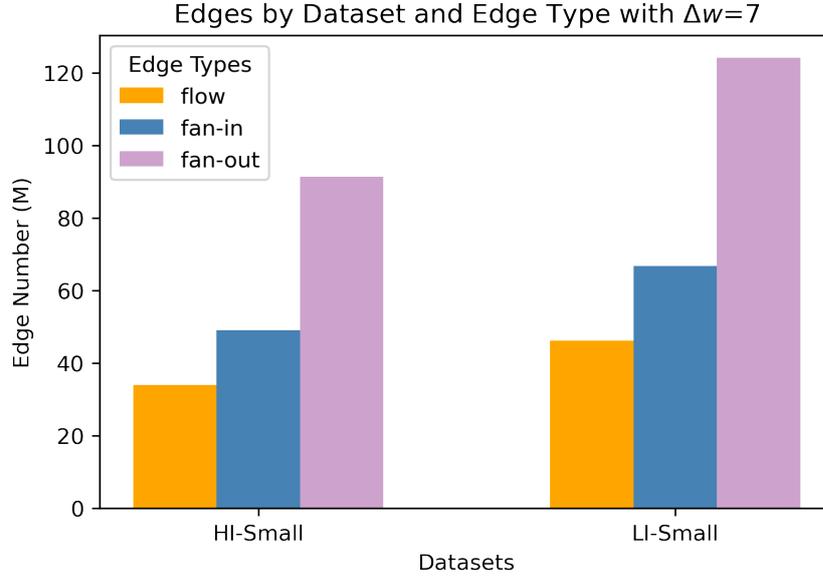


Figure 6.12: Edge count for every edge type and fixed time window.

smaller components represented less than 0.04% of the data. Table 6.7 details the top-3 WCCs.

WCC	# Nodes	% of Total Nodes	# Edges	% of Total Edges
1	1,622,173	99.89%	4,125,279	99.96%
2	27	<0.01%	27	<0.01%
3	15	<0.01%	16	<0.01%

Table 6.7: Statistics for the Top 3 Largest Connected Components in the Graph.

Feature Enrichment. We enhanced the dataset by engineering features that capture interactions between source and target accounts, transaction amounts over time, and the influence of accounts within the graph network. Following correlation analysis, we selected a subset of these features for input into the IF model.

Isolation Forest Model. Using the IF model with predefined hyperparameters (see section 5.3), we trained on the full dataset and generated anomaly scores for each transaction. Scores were normalized to a 0–1 range, where higher values indicate more suspicious transactions. The top-10% of transactions by score was flagged as suspicious.

Anomaly Scores Distribution. As we can see from Figure 6.14, the majority of anomaly scores are concentrated at lower values. After isolating suspicious transactions based on IF scores, we recalculated connected components and observed a similar structural pattern as in the full dataset, as can be seen in Table 6.8. In terms of transaction amounts, anomalous transactions exhibited a significantly higher median amount

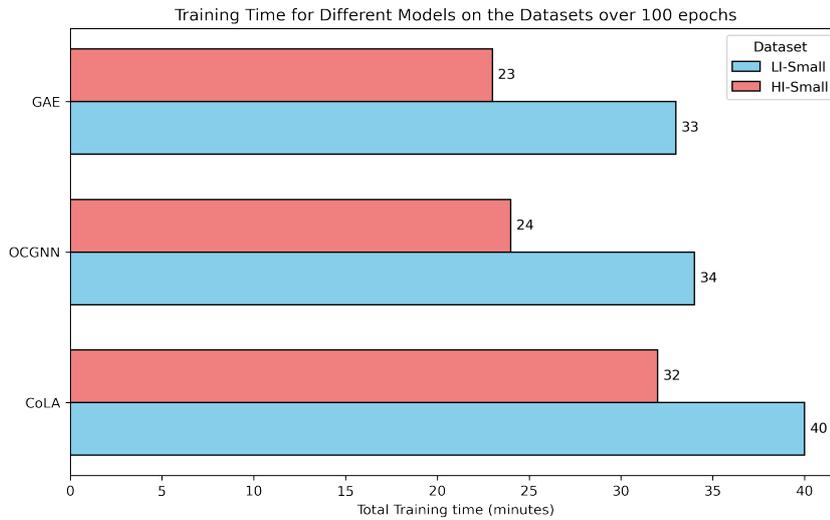


Figure 6.13: Total training time for each model over 100 epochs.

of 44,192.5, compared to 1,443.0 for normal transactions. The total amount of funds marked as anomalous reached 223,834,780,014 (10% of total transactions), while *only* is 66,954,460,615 associated with normal transactions (90% of total transactions). This stark contrast highlights the model's reliance on transaction amount as a primary risk indicator, given the limited set of features available. In Figure 6.15 transaction score is plotted against the amount for a sample of transactions.

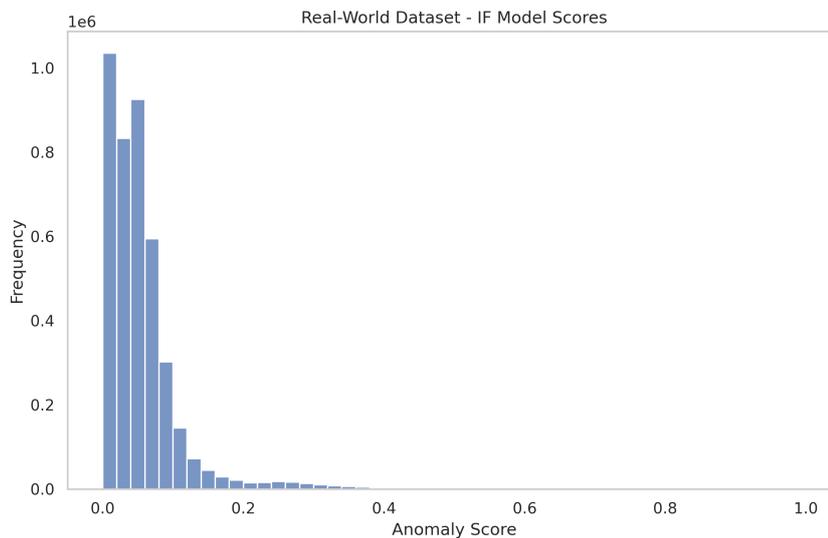


Figure 6.14: Distribution of Anomaly Scores from the IF Model on Real-World Dataset.



Figure 6.15: Scatter plot of Isolation Forest scores versus transaction amounts, based on a 50% sample of the data.

WCC	# Nodes	% of High-Risk Nodes	# Edges	% of High-Risk Edges
1	238,890	94.33%	399,636	96.87%
2	55	0.02%	89	0.02%
3	22	0.01%	22	0.01%

Table 6.8: Statistics for the Top 3 Largest Connected Components in the Graph generated by Anomalous Transactions.

6.4. Summary

In this chapter, we evaluated our self-supervised anomaly detection pipeline for money laundering, which integrates an Isolation Forest (IF) for initial filtering and normal transaction identification, followed by a GAE model applied to a temporal graph of sequential transactions. The IF model proved effective in isolating outliers, highlighting the importance of thorough data preparation and quality feature engineering. The GAE model then leveraged this filtered graph of normal transactions to detect anomalies efficiently. Post-processing of anomaly scores further refined the model’s accuracy. This pipeline is fully unsupervised and self-supervised, requiring no labeled data and learning patterns internally from the data itself to predict money laundering pattern transaction instances.

7 | Conclusions and future developments

7.1. Conclusions

In this research, we proposed a novel, self-supervised anomaly detection pipeline for Anti-Money Laundering (AML) that integrates unsupervised transaction filtering with graph-based anomaly detection on a temporal graph of sequential transactions. The entire approach was developed to answer core research questions.

- **RQ1:** Our results show that combining an Isolation Forest for transaction filtering with a GNN-based model on transaction graphs allows us to capture complex money laundering patterns. This integrated approach optimizes anomaly detection by isolating a large portion of normal transactions with high confidence, thus enabling us to focus model learning on potentially anomalous behaviors in a computationally efficient way.
- **RQ2:** By constructing a temporal graph of sequential transactions, we achieved a robust, unsupervised representation of transaction behavior over time. This temporal graph allows us to leverage both transaction connectivity and time-sensitive patterns in anomaly detection, capturing suspicious behaviors that would be missed in a static graph representation.
- **RQ2.1:** The temporal graph scales well with larger datasets, though its effectiveness depends on dataset quality and temporal granularity. This method requires a sufficiently large and high-quality dataset of normal transactions to produce accurate, interpretable results.
- **RQ3:** Our evaluation confirms that a GNN-based GAE model is effective in learning representations from normal transaction behavior, helping to identify anomalous patterns. The approach is scalable, though enhancements in processing power and optimization techniques would further improve performance in larger-scale financial

transaction datasets.

- **RQ3.1:** Scalability was addressed with efficient frameworks such as PySpark. However, additional refinements could enhance its application in large-scale financial data environments, enabling the model to process transaction batches regularly and efficiently.

This study proposes a feasible self-supervised approach for detecting suspicious transactions in contexts with limited labeled data. While the current model does not capture all true positives, it establishes a first foundation for future improvements in anomaly detection, with the potential for enhancing both accuracy and coverage.

7.2. Limitations

While our pipeline demonstrates promising results, several limitations remain. Interpretability of anomaly scores is a challenge, as directly flagging transactions as suspicious without detailed reasoning can hinder both model adoption and interpretability. The model currently relies mainly on transaction amounts as a risk indicator; access to additional indicators—such as account profiles, counterparty histories, and transaction types—would improve both precision and interpretability. Data quality is also crucial, as high-quality, balanced datasets are essential for accurate anomaly detection, and in practice, financial institutions often use rule-based systems or risk scorecards to define a robust baseline of normal transactions. Lastly, although the pipeline is designed for large datasets, it would benefit from further optimization to fully scale to production-level transaction volumes. With additional resources, the system could be refined to process large transaction batches in real-time or near real-time, enhancing its suitability for production environments.

7.3. Future Directions

The AML landscape is continually evolving, offering numerous avenues to advance this approach. Expanding the framework to a semi-supervised model by incorporating a small portion of labeled data could improve the model's ability to differentiate laundering from legitimate transactions, especially in training GNNs. Additionally, developing models tailored to specific money laundering typologies would enhance detection by focusing on particular transaction patterns, such as structuring, layering, and smurfing, each of which exhibits unique characteristics. The temporal graph itself offers further potential, as we could explore different ways of utilizing it—such as incorporating edge features or

temporal patterns—to enrich transaction relationships and uncover more nuanced behaviors. Further, testing on additional datasets and a high-quality, real-world transaction dataset, ideally manually labeled or already labeled, would enable a more comprehensive assessment of performance and practicality in real-world applications. Real-time and streaming analysis also holds promise, as detecting or blocking anomalous transactions of money laundering in real-time could substantially enhance monitoring effectiveness and save losses for banks and society. By applying process mining techniques to track sequences of events at an entity level, we could achieve a broader perspective on transaction flows and better highlight suspicious patterns. Ultimately, ongoing research and development in these areas would position self-supervised anomaly detection as a more effective, adaptable solution to meet real-world AML demands and enhance financial crime detection systems.

Bibliography

- [1] J. K. Afriyie, K. Tawiah, W. A. Pels, S. Addai-Henne, H. A. Dwamena, E. O. Owiredu, S. A. Ayeh, and J. Eshun. A supervised machine learning algorithm for detecting and predicting fraud in credit card transactions. *Decision Analytics Journal*, 6:100163, 2023. ISSN 2772-6622. doi: <https://doi.org/10.1016/j.dajour.2023.100163>. URL <https://www.sciencedirect.com/science/article/pii/S2772662223000036>.
- [2] L. Akoglu, H. Tong, and D. Koutra. Graph-based anomaly detection and description: A survey, 2014. URL <https://arxiv.org/abs/1404.4679>.
- [3] E. Altman, J. Blanuša, L. von Niederhäusern, B. Egressy, A. Anghel, and K. Atasu. Realistic Synthetic Financial Transactions for Anti-Money Laundering Models, 2024.
- [4] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders, 2021. URL <https://arxiv.org/abs/2003.05991>.
- [5] L. Bionducci, A. Botta, P. Bruno, O. Denecker, C. Gathinji, R. Jain, M.-C. Nadeau, and B. Sattanathan. On the cusp of the next payments era: Future opportunities for banks, Sep 2023. URL <https://www.mckinsey.com/industries/financial-services/our-insights/the-2023-mckinsey-global-payments-report>.
- [6] J. Blanuša, M. C. Baraja, A. Anghel, L. von Niederhäusern, E. Altman, H. Pozidis, and K. Atasu. Graph feature preprocessor: Real-time extraction of subgraph-based features from transaction graphs, 2024.
- [7] M. Cardoso, P. Saleiro, and P. Bizarro. Laundrograph: Self-supervised graph representation learning for anti-money laundering, 2022.
- [8] M. Carletti, M. Terzi, and G. A. Susto. Interpretable anomaly detection with diffi: Depth-based isolation forest feature importance, 2021. URL <https://arxiv.org/abs/2007.11117>.
- [9] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings*

- of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, Aug. 2016. doi: 10.1145/2939672.2939785. URL <http://dx.doi.org/10.1145/2939672.2939785>.
- [10] Z. Chen, L. D. Van Khoa, E. N. Teoh, A. Nazir, E. K. Karuppiah, and K. S. Lam. Machine learning techniques for anti-money laundering (AML) solutions in suspicious transaction detection: a review. *Knowl. Inf. Syst.*, 57(2):245–285, Nov. 2018.
- [11] Z. Chen, W. M. Soliman, A. Nazir, and M. Shorfuzzaman. Variational autoencoders and wasserstein generative adversarial networks for improving the anti-money laundering process. *IEEE Access*, 9:83762–83785, 2021. doi: 10.1109/ACCESS.2021.3086359.
- [12] A. F. Colladon and E. Remondi. Using social network analysis to prevent money laundering. *CoRR*, abs/2105.05793, 2021. URL <https://arxiv.org/abs/2105.05793>.
- [13] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, sep 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL <https://doi.org/10.1023/A:1022627411411>.
- [14] B. Deprez, T. Vanderschueren, B. Baesens, T. Verdonck, and W. Verbeke. Network analytics for anti-money laundering – a systematic literature review and experimental evaluation, 2024.
- [15] B. Deprez, T. Vanderschueren, B. Baesens, T. Verdonck, and W. Verbeke. Network analytics for anti-money laundering – a systematic literature review and experimental evaluation, 2024. URL <https://arxiv.org/abs/2405.19383>.
- [16] K. Ding, J. Li, R. Bhanushali, and H. Liu. Deep anomaly detection on attributed networks. In *SIAM International Conference on Data Mining (SDM)*, 2019.
- [17] X. Du, J. Yu, Z. Chu, L. Jin, and J. Chen. Graph autoencoder-based unsupervised outlier detection. *Information Sciences*, 608:532–550, 2022. ISSN 0020-0255. doi: <https://doi.org/10.1016/j.ins.2022.06.039>. URL <https://www.sciencedirect.com/science/article/pii/S0020025522006338>.
- [18] J. G. Dy and C. E. Brodley. Feature selection for unsupervised learning. *The Journal of Machine Learning Research*, 5:845–889, 2004.
- [19] A. N. Eddin, J. Bono, D. Aparício, D. Polido, J. T. Ascensão, P. Bizarro, and P. Ribeiro. Anti-money laundering alert optimization using machine learning with graphs, 2022.

- [20] B. Egressy, L. Von Niederhäusern, J. Blanuša, E. Altman, R. Wattenhofer, and K. Atasu. Provably powerful graph neural networks for directed multigraphs. *Proc. Conf. AAAI Artif. Intell.*, 38(10):11838–11846, Mar. 2024.
- [21] FIU Nederland. What is money laundering?, 2024. URL <https://www.fiu-nederland.nl/en/home/what-is-money-laundering>. Accessed: 2024-09-16.
- [22] F. A. T. Force. International standards on combating money laundering and the financing of terrorism & proliferation, 2012–2023. URL <https://www.fatf-gafi.org/en/publications/Fatfrecommendations/Fatf-recommendations.html>. Accessed: September 4, 2024.
- [23] F. A. T. Force. Professional money laundering, 2018. URL <https://www.fatf-gafi.org/en/publications/Methodsandtrends/Professional-money-laundering.html>. Accessed: September 4, 2024.
- [24] D. Fourure, M. U. Javaid, N. Posocco, and S. Tihon. *Anomaly Detection: How to Artificially Increase Your F1-Score with a Biased Evaluation Protocol*, page 3–18. Springer International Publishing, 2021. ISBN 9783030865146. doi: 10.1007/978-3-030-86514-6_1. URL http://dx.doi.org/10.1007/978-3-030-86514-6_1.
- [25] Z. Gao. Application of cluster-based local outlier factor algorithm in anti-money laundering. In *2009 International Conference on Management and Service Science*, pages 1–4, 2009. doi: 10.1109/ICMSS.2009.5302396.
- [26] Grand View Research. Anti-money laundering market size and share report, 2030, 2024. URL <https://www.grandviewresearch.com/industry-analysis/anti-money-laundering-market>. Accessed: 2024-09-16.
- [27] W. L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2010.
- [28] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.
- [29] W. Hilal, S. A. Gadsden, and J. Yawney. Financial fraud: A review of anomaly detection techniques and recent advances. *Expert Systems with Applications*, 193:116429, 2022. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2021.116429>. URL <https://www.sciencedirect.com/science/article/pii/S0957417421017164>.
- [30] R. Jensen, J. Ferwerda, K. Jørgensen, et al. A synthetic data set to benchmark

- anti-money laundering methods. *Scientific Data*, 10(1):661, 2023. doi: 10.1038/s41597-023-02569-2. URL <https://doi.org/10.1038/s41597-023-02569-2>.
- [31] M. R. Karim, F. Hermsen, S. A. Chala, P. de Perthuis, and A. Mandal. Catch me if you can: Semi-supervised graph learning for spotting money laundering, 2023.
- [32] L. Keyan and Y. Tingting. An improved support-vector network model for anti-money laundering. In *2011 Fifth International Conference on Management of e-Commerce and e-Government*, pages 193–196, 2011. doi: 10.1109/ICMeCG.2011.50.
- [33] H. Kim, B. S. Lee, W.-Y. Shin, and S. Lim. Graph anomaly detection with graph neural networks: Current status and challenges, 2022. URL <https://arxiv.org/abs/2209.14930>.
- [34] T. N. Kipf and M. Welling. Variational graph auto-encoders, 2016. URL <https://arxiv.org/abs/1611.07308>.
- [35] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks, 2017. URL <https://arxiv.org/abs/1609.02907>.
- [36] D. V. Kute, B. Pradhan, N. Shukla, and A. Al-Amri. Deep learning and explainable artificial intelligence techniques applied for detecting money laundering—a critical review. *IEEE Access*, 9:82300–82317, 2021.
- [37] P. B. Lamichhane and W. Eberle. Anomaly detection in graph structured data: A survey, 2024. URL <https://arxiv.org/abs/2405.06172>.
- [38] A. S. Larik and S. Haider. Clustering based anomalous transaction reporting. *Procedia Computer Science*, 3:606–610, 2011. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2010.12.101>. URL <https://www.sciencedirect.com/science/article/pii/S187705091000476X>. World Conference on Information Technology.
- [39] X. Li, X. Cao, X. Qiu, J. Zhao, and J. Zheng. Intelligent anti-money laundering solution based upon novel community detection in massive transaction networks on spark. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*, pages 176–181, 2017. doi: 10.1109/CBD.2017.38.
- [40] X. Li, S. Liu, Z. Li, X. Han, C. Shi, B. Hooi, H. Huang, and X. Cheng. Flowscope: Spotting money laundering based on graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4731–4738, 2020.
- [41] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE*

- International Conference on Data Mining*, pages 413–422, 2008. doi: 10.1109/ICDM.2008.17.
- [42] J. Liu, C. Yin, H. Wang, X. Wu, D. Lan, L. Zhou, and C. Ge. Graph embedding-based money laundering detection for ethereum. *Electronics*, 12(14), 2023. ISSN 2079-9292. URL <https://www.mdpi.com/2079-9292/12/14/3180>.
- [43] K. Liu, Y. Dou, X. Ding, X. Hu, R. Zhang, H. Peng, L. Sun, and P. S. Yu. Pygod: A python library for graph outlier detection, 2024. URL <https://arxiv.org/abs/2204.12095>.
- [44] Y. Liu, Z. Li, S. Pan, C. Gong, C. Zhou, and G. Karypis. Anomaly detection on attributed networks via contrastive self-supervised learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(6):2378–2392, June 2022. ISSN 2162-2388. doi: 10.1109/tnnls.2021.3068344. URL <http://dx.doi.org/10.1109/TNNLS.2021.3068344>.
- [45] W. W. Lo, G. K. Kulatilleke, M. Sarhan, S. Layeghy, and M. Portmann. Inspection-1: self-supervised gnn node embeddings for money laundering detection in bitcoin. *Applied Intelligence*, 53(16):19406–19417, Mar. 2023. ISSN 1573-7497. doi: 10.1007/s10489-023-04504-9. URL <http://dx.doi.org/10.1007/s10489-023-04504-9>.
- [46] S. Misra, S. Thakur, M. Ghosh, and S. K. Saha. An autoencoder based model for detecting fraudulent credit card transaction. *Procedia Computer Science*, 167:254–262, 2020. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2020.03.219>. URL <https://www.sciencedirect.com/science/article/pii/S1877050920306840>. International Conference on Computational Intelligence and Data Science.
- [47] A. M. Mubarek and E. Adah. Multilayer perceptron neural network technique for fraud detection. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 383–387, 2017. doi: 10.1109/UBMK.2017.8093417.
- [48] Nederlandse Vereniging van Banken. Transactie monitoring nederland unieke stap in strijd tegen witwassen en terrorismefinanciering, 2020. URL <https://www.nvb.nl/nieuws/transactie-monitoring-nederland-unieke-stap-in-strijd-tegen-witwassen-en-terror>. Accessed: 2024-09-17.
- [49] B. Oztas, D. Cetinkaya, F. Adedoyin, M. Budka, G. Aksu, and H. Dogan. Transaction monitoring in anti-money laundering: A qualitative analysis and points of view from industry. *Future Generation Computer Systems*, 159:161–171, 2024.

- ISSN 0167-739X. doi: <https://doi.org/10.1016/j.future.2024.05.027>. URL <https://www.sciencedirect.com/science/article/pii/S0167739X24002607>.
- [50] Q. Rajput, N. S. Khan, A. Larik, and S. Haider. Ontology based expert-system for suspicious transactions detection. *Comput. Inf. Sci.*, 7(1), Jan. 2014.
- [51] S. Raza and S. Haider. Suspicious activity reporting using dynamic bayesian networks. *Procedia Computer Science*, 3:987–991, 2011. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2010.12.162>. URL <https://www.sciencedirect.com/science/article/pii/S1877050910005375>. World Conference on Information Technology.
- [52] Y. Sahin and E. Duman. Detecting credit card fraud by decision trees and support vector machines. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 1–6, 2011.
- [53] SAS Institute Inc. Modernize your anti-money laundering program with ai and machine learning in the cloud, 2020. URL <https://www.sas.com/content/dam/SAS/documents/briefs/solution-brief/en/anti-money-laundering-111520.pdf>. Accessed: 2024-09-16.
- [54] A. K. Shaikh, M. Al-Shamli, and A. Nazir. Designing a relational model to identify relationships between suspicious customers in anti-money laundering (aml) using social network analysis (sna). *Journal of Big Data*, 8(1), Jan. 2021. ISSN 2196-1115. doi: 10.1186/s40537-021-00411-3. URL <http://dx.doi.org/10.1186/s40537-021-00411-3>.
- [55] M. Starnini, C. E. Tsourakakis, M. Zamanipour, A. Panisson, W. Allasia, M. Fornasiero, L. L. Puma, V. Ricci, S. Ronchiadin, A. Ugrinoska, M. Varetto, and D. Moncalvo. *Smurf-Based Anti-money Laundering in Time-Evolving Transaction Networks*, page 171–186. Springer International Publishing, 2021. ISBN 9783030865146. doi: 10.1007/978-3-030-86514-6_11. URL http://dx.doi.org/10.1007/978-3-030-86514-6_11.
- [56] E. Stripling, B. Baesens, B. Chizi, and S. vanden Broucke. Isolation-based conditional anomaly detection on mixed-attribute data to uncover workers’ compensation fraud. *Decision Support Systems*, 111:13–26, 2018. ISSN 0167-9236. doi: <https://doi.org/10.1016/j.dss.2018.04.001>. URL <https://www.sciencedirect.com/science/article/pii/S016792361830068X>.
- [57] T. Suzumura and H. Kanezashi. Anti-Money Laundering Datasets: InPlusLab anti-money laundering datadatasets. <http://github.com/IBM/AMLSim/>, 2021.

- [58] H. Tariq and M. Hassani. Topology-Agnostic Detection of Temporal Money Laundering Flows in Billion-Scale Transactions, 2023.
- [59] S. Thudumu, P. Branch, J. Jin, and J. Singh. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, 7(1), July 2020. ISSN 2196-1115. doi: 10.1186/s40537-020-00320-x. URL <http://dx.doi.org/10.1186/s40537-020-00320-x>.
- [60] United Nations Office on Drugs and Crime. Overview - money laundering, 2024. URL <https://www.unodc.org/unodc/en/money-laundering/overview.html>. Accessed: 2024-09-16.
- [61] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2018. URL <https://arxiv.org/abs/1710.10903>.
- [62] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax, 2018. URL <https://arxiv.org/abs/1809.10341>.
- [63] X. Wang, B. Jin, Y. Du, P. Cui, Y. Tan, and Y. Yang. One-class graph neural networks for anomaly detection in attributed networks. *Neural Computing and Applications*, 33(18):12073–12085, Mar. 2021. ISSN 1433-3058. doi: 10.1007/s00521-021-05924-9. URL <http://dx.doi.org/10.1007/s00521-021-05924-9>.
- [64] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics, 2019. URL <https://arxiv.org/abs/1908.02591>.
- [65] Y. Xie, Z. Xu, J. Zhang, Z. Wang, and S. Ji. Self-supervised learning of graph neural networks: A unified review, 2022. URL <https://arxiv.org/abs/2102.10757>.
- [66] G. Yang, X. Liu, and B. Li. Anti-money laundering supervision by intelligent algorithm. *Computers & Security*, 132:103344, 2023. ISSN 0167-4048. doi: <https://doi.org/10.1016/j.cose.2023.103344>. URL <https://www.sciencedirect.com/science/article/pii/S0167404823002547>.
- [67] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. ISSN 2666-6510. doi: <https://doi.org/10.1016/j.aiopen.2021.01.001>. URL <https://www.sciencedirect.com/science/article/pii/S2666651021000012>.

List of Figures

1.1	Network of banks and money flows.	2
2.1	Illustration of the money laundering stages.	6
2.2	Illustration of different types of anomalies in data.	9
2.3	Sample of financial transaction data.	11
2.4	Common money laundering patterns injected in synthetic datasets.	11
2.5	Comparison between a directed multigraph and a directed aggregated graph.	13
2.6	Edge transformation from 1 st to 2 nd order.	14
2.7	Building a temporal graph of sequential transactions.	15
2.8	Illustration of graph representation learning through node embedding.	16
3.1	General workflow of rule-based suspicious activity reporting.	20
3.2	Illustration of Support Vector Machine classification.	22
3.3	Ensemble techniques in Machine Learning.	22
3.4	Overview of the Graph-Feature Preprocessor workflow.	23
3.5	Illustration of a typical Autoencoder architecture.	24
3.6	Visualization of dense money flows in a tripartite transaction graph.	25
3.7	Visualization of smurf-like subgraphs.	26
3.8	Types of graph-based anomaly detection.	27
3.9	Workflow of GNN-based anomaly detection algorithms.	29
4.1	High-level overview of the framework.	32
4.2	Detailed overview of the framework.	33
4.3	Feature engineering process for transaction data.	35
4.4	Graph Feature Preprocessor feature enrichment process.	37
4.5	Principle of Isolation Forest.	40
4.6	Ensemble of Isolation Forest trees.	41
4.7	Illustration of the thresholding operation on anomaly scores.	42
5.1	Sample of realistic synthetic transactions.	53

6.1	Confusion Matrices for the HI-Small and LI-Small Datasets.	63
6.2	AUC-ROC Curves for HI-Small and LI-Small Datasets.	63
6.3	Scores for real <i>Normal</i> Transactions (blue) and <i>Anomalous</i> Transactions (orange).	63
6.4	AUC-ROC Curve Comparison for the HI-Small Dataset.	65
6.5	AUC-ROC Curve Comparison for the LI-Small Dataset.	66
6.6	GAE Model - Confusion Matrix for HI-Small and LI-Small Datasets. . . .	67
6.7	OCGNN Model - Confusion Matrix for HI-Small and LI-Small Datasets. . .	67
6.8	CoLA Model - Confusion Matrix for HI-Small and LI-Small Datasets. . . .	68
6.9	Model Raw Output Scores for HI-Small dataset.	68
6.10	Model Raw Output Scores for LI-Small dataset.	69
6.11	Increase of the total number of graph edges across different time-window parameters.	70
6.12	Edge count for every edge type and fixed time window.	71
6.13	Total training time for each model over 100 epochs.	72
6.14	Distribution of Anomaly Scores from the IF Model on Real-World Dataset. .	72
6.15	Scatter plot of Isolation Forest scores versus transaction amounts, based on a 50% sample of the data.	73

List of Tables

2.1	Summary of the notations used in the definitions.	17
3.1	Relevant state-of-the-art anomaly detection approaches in AML.	30
4.1	Overview of GFP Features.	38
5.1	Summary of descriptive statistics for the available datasets.	52
5.2	Statistics of synthetic transaction datasets.	52
5.3	The patterns modeled in the IBM Transactions for AML datasets.	55
5.4	Sample of real aggregated transactions.	55
6.1	Confusion matrix.	60
6.2	Dataset Statistics Before and After Pre-processing and Filtering.	62
6.3	Performance Metrics (%) for HI-Small and LI-Small Datasets.	62
6.4	AUC-ROC Performance (%) of GNN-based Models on Synthetic Datasets.	64
6.5	Total number of temporal graph edges across different time-window parameters.	69
6.6	Training time per epoch (in seconds) for different models on each dataset.	69
6.7	Statistics for the Top 3 Largest Connected Components in the Graph.	71
6.8	Statistics for the Top 3 Largest Connected Components in the Graph generated by Anomalous Transactions.	73

Acknowledgements

This thesis represents the final achievement of the EIT Digital Double Degree Master program between Politecnico di Milano and Eindhoven University of Technology. This two-year journey has been extremely formative and influential in shaping my knowledge and fueling my interests. First and foremost I would like to express my sincere gratitude to the selection committee who granted me the privilege to participate in such an international experience.

A special thanks goes to my supervisor Marwan Hassani, for the opportunity to participate in this project and work with TMNL, a unique company for transaction monitoring in the Netherlands. To him, I extend my heartfelt thanks for the patience, meticulous scheduling, strength, and motivation he provided me throughout this journey. He was instrumental in helping me navigate the challenges that arose during these months.

I would also like to extend my thanks to my company supervisor Haseeb Tariq, a truly exceptional data scientist and mentor. His daily guidance was invaluable, as he not only was a source of inspiration for the project but also helped solve technical issues and addressed my doubts. This work would not have been possible without his expertise and guidance.

My family deserves special thanks for their unwavering support, and encouragement, which reminded me to never give up. They provided the foundation for all of my achievements and made everything possible for me.

I am grateful to everyone I collaborated with over these two years and to the people I spent my time with. While they may have been a distraction at times, they helped me forge unforgettable memories. Their inspiration and presence have made me a better person, for which I am deeply thankful.

