Graph Feature Preprocessor: Real-time Subgraph-based Feature Extraction for Financial Crime Detection

Jovan Blanuša
IBM Research Europe
Switzerland
jov@zurich.ibm.com

Maximo Cravero Baraja
Caltech
USA
mcravero@caltech.edu

Andreea Anghel
IBM Research Europe
Switzerland
aan@zurich.ibm.com

Luc von Niederhäusern
IBM Research Europe
Switzerland
lvn@zurich.ibm.com

Erik Altman
IBM Watson Research
USA
ealtman@us.ibm.com

Haris Pozidis
IBM Research Europe
Switzerland
hap@zurich.ibm.com

Kubilay Atasu
TU Delft
Netherlands
kubilay.atasu@tudelft.nl

Abstract

In this paper, we present *Graph Feature Preprocessor*, a software library for detecting typical money laundering patterns in financial transaction graphs in real time. These patterns are used to produce a rich set of transaction features for downstream machine learning training and inference tasks such as detection of fraudulent financial transactions. We show that our enriched transaction features dramatically improve the prediction accuracy of gradient-boosting-based machine learning models. Our library exploits multicore parallelism, maintains a dynamic in-memory graph, and efficiently mines subgraph patterns in the incoming transaction stream, which enables it to be operated in a streaming manner. Our solution, which combines our Graph Feature Preprocessor and gradient-boosting-based machine learning models, can detect illicit transactions with higher minority-class F1 scores than standard graph neural networks in anti-money laundering and phishing datasets. In addition, the end-to-end throughput rate of our solution executed on a multicore CPU outperforms the graph neural network baselines executed on a powerful V100 GPU. Overall, the combination of high accuracy, a high throughput rate, and low latency of our solution demonstrates the practical value of our library in real-world applications.

1 Introduction

Financial transactions serve as records documenting the movement of financial funds between accounts. Typically, these transactions are captured in a tabular format, where each row represents a distinct financial transaction, and columns represent basic transaction features such as timestamp, source account, target account, amount
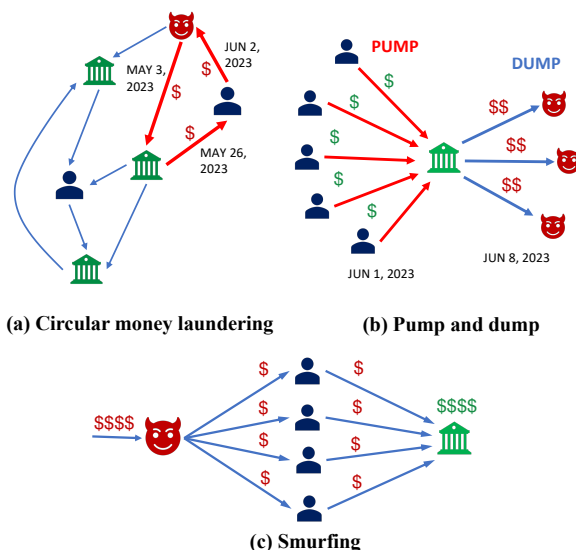
(a) Circular money laundering

(b) Pump and dump

(c) Smurfing

Figure 1: Crime patterns in financial transaction graphs.

transferred, currency, and payment type [1]. While this tabular representation offers a structured view of the data, a more insightful approach emerges when financial transactions are represented as graphs by treating transactions as edges and accounts as vertices of a graph, as illustrated in Figure 1. Such a graph representation enables analysts to uncover insights that may not be immediately apparent in tabular formats. As a result, financial transaction graphs facilitate the efficient analysis and interpretation of complex financial data, aiding in the detection of financial crime [19, 49].

Subgraph patterns in financial transaction graphs can often serve as indicators of financial crime. A *simple cycle* [48], depicted in Figure 1a, is one such pattern and represents a sequence of transactions that transfer funds from one bank account back to the same account. Such a cycle can be an indicator of financial crimes such as money laundering, tax avoidance [28, 69], credit card frauds [49, 56], or circular trading used for stock price manipulation [34, 36, 52]. In addition, a *gather-scatter* pattern, illustrated in Figure 1b, can suggest a *pump and dump* stock manipulation scheme [49]. In this scheme, the stock price of a company is artificially increased through the use of social media to attract other traders for investment. After
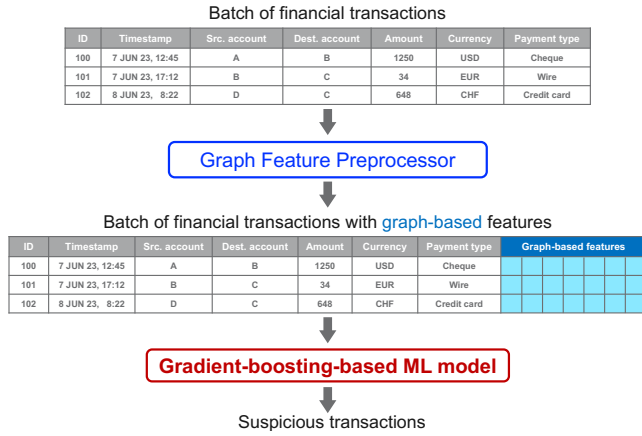
Figure 2: The overview of our graph ML pipeline for the detection of suspicious financial transactions.

the stock price rises sufficiently, malicious traders sell the stocks. Due to the artificially inflated stock price, its value drops, and other traders suffer financial losses. Furthermore, a *scatter-gather* pattern, depicted in Figure 1c, can represent a money laundering tactic called *smurfing* [19, 39, 42, 43, 62, 67], in which a malicious actor employs several intermediary accounts (blue nodes in Figure 1c) to integrate small sums of illicit funds into the legal banking system. Similarly, in cryptocurrency transaction networks, criminals use sophisticated mixing and shuffling schemes to obfuscate the trace of their activities [44]. Such schemes can usually be represented by subgraph structures [15, 64, 73, 76]. The discovery of such suspicious subgraph patterns may enable locating and stopping criminal activities and their perpetrators.

Rapid detection and processing of suspicious financial transactions are important to avoid financial losses. As financial data is often represented in a tabular format [1], the fastest and most accurate machine learning models [27] for this input format are gradient-boosting-based models [14, 38]. However, these models cannot take into account the underlying graph structure and cannot discover graph patterns that could be associated with financial crime. Furthermore, a limited set of basic features associated with financial transactions (see Figure 2) does not provide sufficient information to gradient-boosting-based models for detecting suspicious transactions with sufficient accuracy. As a result, the detection of suspicious transactions using these methods poses a challenge.

To overcome the aforementioned limitations, we propose a solution shown in Figure 2. Specifically, we develop the *Graph Feature Preprocessor* (GFP) library to produce a rich set of graph-based features for financial transactions. Our library searches for typical financial crime patterns, such as money laundering cycles and scatter-gather patterns (see Figure 1), and encodes these graph patterns into additional columns (i.e., features) of the transaction table. The transaction table enriched with the graph-based features is then forwarded to a pre-trained gradient-boosting-based machine learning model that performs the classification of financial transactions and detects suspicious transactions. As a result, the machine learning model is provided with additional transaction features extracted from the financial transaction graph, which facilitates the detection of transactions associated with financial crime.
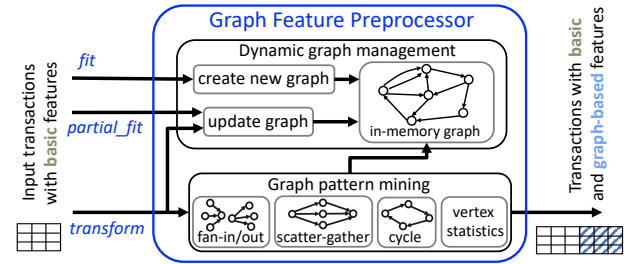

Figure 3: Our Graph Feature Preprocessor is offered as a scikit-learn preprocessor with *fit* and *transform* methods.

Our contributions can be summarised as follows:

- We present a graph-based feature extraction library called Graph Feature Preprocessor for enriching the feature set of edges in financial transaction graphs by enumerating suspicious subgraph patterns in graphs as well as by computing various statistical properties of graph vertices. We then use this library to develop a graph machine learning (graph ML) pipeline for monitoring financial transaction networks. Section 2 introduces this library.

- We conduct experiments that demonstrate an improvement of up to 36% in the minority-class F1 score compared to graph neural network (GNN) baselines [11, 20, 31] for money laundering detection tasks. In addition, we demonstrate that our graph ML pipeline executed using 32 cores of an Intel Xeon processor achieves higher throughput rates compared to those GNN baselines executed on an NVIDIA Tesla V100 GPU. Our experimental evaluation is presented in Section 4.

The GFP library is publicly available on PyPI as part of Snap ML [59–61]. In addition, it is offered with IBM[1] mainframe software products *Cloud Pak for Data on Z* [33] and *AI Toolkit for IBM Z and LinuxONE* [32]. Furthermore, an *AI on IBM Z Anti-Money Laundering Solution Template* [63], which demonstrates how to develop and deploy a graph ML pipeline with GFP using an IBM Z environment, is open-sourced and publicly available[2].

2 Graph Feature Preprocessor

An overview of the Graph Feature Preprocessor (GFP) is given in Figure 3. It operates in a streaming fashion, receiving as input a batch of transactions with only basic features, such as in Figure 2, and producing additional graph-based features as output. GFP stores past financial transactions in an in-memory graph, which is dynamically updated as new transactions are received. The graph-based features are computed by enumerating subgraph patterns in the graph and by generating statistical properties of the accounts stored in that graph. GFP can compute the graph-based features across several CPU cores in parallel, which, together with the dynamic graph representation, enables real-time feature extraction.

We have implemented GFP as a scikit-learn preprocessor with the *fit/transform* interface [66] and made it publicly available on PyPI as part of the Snap ML package [59–61]. The main functionality of GFP is implemented by the *transform* function, which is illustrated

---

[1]IBM, the IBM logo, and IBM Cloud Pak are trademarks or registered trademarks of International Business Machines Corporation, in the United States and/or other countries.
[2]https://github.com/ambitus/aionz-st-anti-money-laundering

in Figure 3. This function inserts a batch of input transactions into the in-memory graph and computes graph-based features for these transactions. Creating the initial in-memory graph is performed by providing some past transactions as an input to the *fit* function. The existing in-memory graph can be updated without computing any graph features by using the *partial_fit* function. Other standard preprocessor functions supported by GFP are described in the publicly available documentation [60]. In the rest of this section, we describe the dynamic graph management and graph pattern mining components of GFP (see Figure 3), and we describe how the graph-based features produced by the library are encoded.

2.1 Dynamic Graph Management

The dynamic graph management component in GFP uses an in-memory graph to represent the financial transaction network. In this scenario, each account is treated as a graph vertex, and each transaction represents an edge from its source account to its destination account. As financial transactions typically include a *timestamp* indicating when a transaction was created (see Figure 2), financial transaction graphs are considered *temporal graphs* [30]. Furthermore, financial transaction graphs are also *multigraphs* [3], as there can be several *parallel edges*, i.e., edges that connect the same pair of source and destination vertices. Hence, our in-memory graph must be capable of representing temporal multigraphs.

To enable the seamless processing of transactions in a streaming fashion, our in-memory graph must support the insertion of new transactions and the removal of outdated transactions. We define new transactions as those with timestamps greater than the timestamp of any transaction currently in the in-memory graph. Outdated transactions are identified as those with timestamps smaller than a value $t_{now} - \delta$, where t_{now} represents the largest timestamp among the transactions in the in-memory graph and δ denotes a user-defined time window. Consequently, the in-memory graph retains only transactions that fall within the time window $[t_{now} - \delta : t_{now}]$, effectively constraining its memory usage.

Our in-memory graph comprises two main data structures: a *transaction log* and an *index*. The transaction log, implemented as a double-ended queue, maintains a list of edges sorted in ascending order of their timestamps. This data structure facilitates the detection and removal of outdated edges by supporting an $O(1)$ operation for removing the edge with the smallest timestamp. The index data structure employs an *adjacency list* representation to enable fast access to the neighbours of a vertex [18]. Implemented as a vector of hash maps [58], each entry in the vector represents a vertex v, and the hash map associated with that vertex v signifies the adjacency list of v. Vertices are internally mapped to integers in the range of $0, 1, \ldots, n - 1$, where n is the number of vertices in the graph. These integers are used to access the adjacency list of a vertex v in this vector. Furthermore, each edge can be accessed in $O(1)$ time using the index, facilitating traversal through the graph, as required by the graph pattern mining component.

To support the maintenance of parallel edges in the index, each entry in an adjacency list of the vertex v, representing a neighbour u of the vertex v, also contains a list of edges connecting v with u, referred to as the *parallel edge list*. The edges in this list, also implemented as a double-ended queue, are represented with their

---

Algorithm 1: ScatterGatherStream $(\mathcal{G}(\mathcal{V}, \mathcal{E}), batch, \delta_p)$

Input: $\mathcal{G}$ - the input graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$
$\quad\quad\quad$ *batch* - a batch of edges; δ_p - the time window

1 **parallel foreach** $(u \to v, t_{uv}) : batch$ **do**
2 $\quad$ TW $= [t_{uv} - \delta_p : t_{uv}]$; $\quad\quad$ ▷ Time window of size δ_p
$\quad$ // The first phase
3 $\quad$ $N_u^+ = \{\forall x \mid (u \to x, t_s) \in \mathcal{E} \land t_s \in$ TW $\}$;
4 $\quad$ $N_v^+ = \{\forall x \mid (v \to x, t_s) \in \mathcal{E} \land t_s \in$ TW $\}$;
5 $\quad$ **parallel foreach** w : N_v^+ **do**
6 $\quad\quad$ $N_w^- = \{\forall x \mid (x \to w, t_s) \in \mathcal{E} \land t_s \in$ TW $\}$;
7 $\quad\quad$ $I = N_u^+ \cap N_w^-$;
8 $\quad\quad$ **if** $|I| \geq 2$ **then** report scatter-gather pattern $\{u, I, w\}$;
$\quad$ // The second phase
9 $\quad$ $N_u^- = \{\forall x \mid (x \to u, t_s) \in \mathcal{E} \land t_s \in$ TW $\}$;
10 $\quad$ $N_v^- = \{\forall x \mid (x \to v, t_s) \in \mathcal{E} \land t_s \in$ TW $\}$;
11 $\quad$ **parallel foreach** w : N_u^- **do**
12 $\quad\quad$ $N_w^+ = \{\forall x \mid (w \to x, t_s) \in \mathcal{E} \land t_s \in$ TW $\}$;
13 $\quad\quad$ $I = N_v^- \cap N_w^+$;
14 $\quad\quad$ **if** $|I| \geq 2$ **then** report scatter-gather pattern $\{w, I, v\}$;

ID and timestamp, sorted in ascending order of their timestamps. For this reason, the operations of inserting new edges and removing the outdated edges can be performed in $O(1)$ time.

2.2 Graph Pattern Mining

The task of the graph pattern mining component is to produce graph-based features for edges forwarded to the library through the *transform* function. Two types of graph-based features are supported: *i)* graph-pattern-based features and *ii)* vertex-statistics-based features.

Graph-pattern-based features are computed by extracting graph patterns from the in-memory graph that contain one of the forwarded edges. Our library extracts the following graph patterns: fan-in, fan-out, scatter-gather, gather-scatter, simple cycle, and temporal cycle. Fan-in and fan-out patterns refer to patterns defined by a vertex v and all of its incoming and outgoing edges, respectively. A *gather-scatter* pattern combines a fan-in pattern of the vertex v with a fan-out pattern of the same vertex v, as illustrated in Figure 1b [67]. A fan-out pattern of a vertex v and a fan-in pattern of a vertex u form a *scatter-gather* pattern, depicted in Figure 1c, if the fan-out and the fan-in patterns connect vertices v and u, respectively, to the same set of intermediate vertices [67] (blue vertices in Figure 1c). A simple cycle is a path from vertex v to the same vertex v without repeated vertices except for the first and last vertex. Finally, a temporal cycle is a simple cycle with edges ordered in time.

To compute graph-pattern-based features in a streaming manner, our library enumerates new patterns that are formed after inserting the input batch of edges into the graph. The fan-in and fan-out pattern features of a vertex v that belongs to the input batch are determined by counting the number of outgoing and incoming vertices of v, respectively. These features can be determined in $O(1)$ time by simply querying the size of the hash maps that are implementing the adjacency lists of the vertex v in our index data
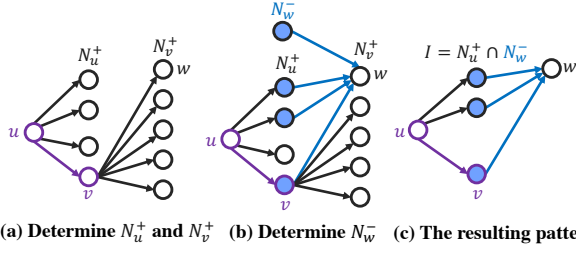
(a) Determine N_u^+ and N_v^+ **(b) Determine N_w^-** **(c) The resulting pattern**

Figure 4: Enumeration of scatter-gather patterns that contain the edge $u \rightarrow v$ with v being an intermediate vertex.

structure (see Section 2.1). A gather-scatter pattern is detected implicitly if the fan-in and fan-out of a vertex v are at least two. Due to space constraints, we omit the description of our algorithm for finding scatter-gather patterns in a streaming manner.

To enumerate simple cycles and temporal cycles in a streaming manner, we use fine-grained parallel algorithms introduced in Blanuša et al. [6, 7]. These algorithms enable the search for cycles that start from a single edge or a small batch of edges in parallel using several threads. The benefit of these algorithms is that they can process transactions in small batches with high throughput. For instance, if the computation of cycles is parallelised by adopting the *coarse-grained* parallel approach, recursive cycle search for each edge of a batch is performed by a different thread. However, as shown in Blanuša et al [6, 7] using the coarse-grained approach might result in a suboptimal solution due to the potential workload imbalance across threads. In contrast, *fine-grained* cycle enumeration algorithms are able to execute the recursive cycle search from a single edge using several threads, thereby increasing the parallelism. As a result, even if the input batch contains one transaction, our library would be able to parallelise the search for cycles.

To compute scatter-gather pattern in a streaming manner, we use our algorithm illustrated in Figure 4 and presented in Algorithm 1. In this algorithm, $(u \rightarrow v, t_{uv})$ denotes a temporal edge with source vertex u, target vertex v and timestamp t_{uv}. This algorithm processes each edge $(u \rightarrow v, t_{uv})$ in the input batch by searching for all scatter-gather patterns that include that edge. The first and second phase of this algorithm search for scatter-gather patterns that contain v and u as an intermediate vertex, respectively. In the first phase, we first determine the outgoing neighbours of u and v, denoted as N_u^+ and N_v^+, respectively, as shown in Figure 4a. Then, for each outgoing neighbour w of v, we search for incoming neighbours N_w^- of the vertex w, which are represented as filled circles in Figure 4b. Afterwards, we perform a set intersection between N_u^+ and N_w^- to find the intermediate vertices I of a scatter gather pattern. Finally, the algorithm reports the resulting scatter-gather pattern defined with vertices u, w, and I, as shown in Figure 4c. The second phase of this algorithm, presented in lines 9–14 of Algorithm 1, is analogous to the first phase, and we omit its description for brevity. Note that this algorithm can be parallelised in a fine-grained manner by parallelising its loops, as shown in Algorithm 1.

Apart from parallelisation, another method to reduce the time required to find graph patterns is to impose time-window constraints. In this case, a time window parameter δ_p can be specified for each graph pattern, in which case the library searches only for patterns whose edges have timestamps greater than or equal to $t_{now} - \delta_p$,
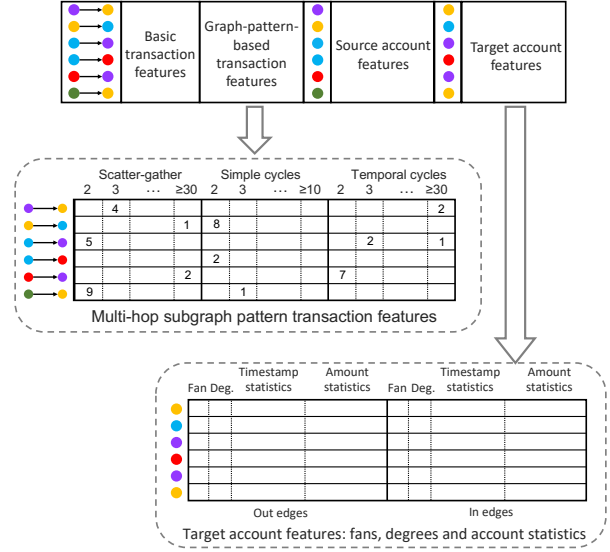

Figure 5: Feature encoding: scatter-gather patterns are binned according to the number of intermediate vertices they have, and cycles are binned according to their length.

where t_{now} represents the largest timestamp among the edges in the in-memory graph. Additionally, the search for simple cycles can be constrained by limiting their maximal length.

Vertex-statistics-based features are computed for the vertices that appear in the input batch of edges. For each such vertex v, some predefined statistical property can be computed using a selected basic feature associated with the outgoing edges of v and its incoming edges. The statistical properties currently supported by our library are: sum, mean, minimum, maximum, median, variance, skew, and kurtosis [41]. For instance, if "Amount" is the selected basic feature used for the calculation of statistical properties, the statistical features include the average and total amount of money an account received or sent. Combining different statistical feature types with different user-specified basic features in this way extends the feature space significantly.

Vertex-statistics-based features can be determined in a streaming manner through incremental computation. For this purpose, our library maintains second, third, and fourth central moments for each vertex of the graph and for each basic feature used for calculating account statistics (e.g., "Amount"). After inserting or removing an edge $u \rightarrow v$, all central moments for u and v are updated incrementally [25, 70]. These central moments are then used to compute the following statistical features: sum, mean, variance, skew, and kurtosis [41]. Note that the computation of each aforementioned statistical feature can be performed in $O(1)$ time. Other statistical features, i.e., minimum, maximum, and median, are simply computed by iterating through the incident edges of a vertex, which is executed in $O(\Delta)$ time per statistical feature, where Δ is the maximum degree of a vertex in the graph.

2.3 Feature Encoding

The encoding of the features produced by the *transform* function of GFP is shown in Figure 5. Each row of the output feature table stores the feature vector of a single transaction. Across different

Table 1: Datasets used in the experiments.

Dataset	# nodes	# edges	illicit rate	time span
AML HI Small	0.5 M	5 M	0.102%	10 days
AML HI Medium	2.1 M	32 M	0.110%	16 days
AML HI Large	2.1 M	180 M	0.124%	97 days
AML LI Small	0.7 M	7 M	0.051%	10 days
AML LI Medium	2.1 M	32 M	0.051%	16 days
AML LI Large	2.1 M	180 M	0.057%	97 days
ETH Phishing	2.9 M	13 M	0.278%	1261 days

columns of a feature vector, there are basic transaction features, graph-pattern-based transaction features, and the account features of the source and the destination account of the transaction. The account features consist of vertex-statistics-based features and features based on fan-in and fan-out patterns, both of which are single-hop patterns. Features based on fan-in and fan-out patterns are computed for each account v and represent the number of accounts connected to v in those patterns. Graph-pattern-based transaction features are computed using multi-hop subgraph patterns: scatter-gather, hop-constrained simple cycles, and temporal cycles. For each transaction, our library reports the number of multi-hop subgraph patterns of different sizes that this transaction is part of. Example features based on multi-hop subgraph patterns are given in Figure 5, where the first transaction participates in 4 scatter-gather patterns with 3 intermediate vertices and in 2 temporal cycles with 30 or more edges. Even though these multi-hop subgraph patterns can also be used to compute account features, computing them as transaction features provides more compact feature vectors.

3 Experimental setup

Datasets. Table 1 presents the datasets used in the evaluation. The AML datasets are publicly available synthetic AML datasets produced by the *AMLworld* generator[1]. These datasets contain transactions labelled as licit or illicit, and, thus, they can be directly used with our graph ML pipeline that performs transaction classification. The datasets are available in two variants: one with a higher illicit rate (AML HI) and one with a lower illicit rate (AML LI). In addition, we use the ETH Phishing dataset, which is a real-world Ethereum dataset [13, 77] with 1, 165 accounts labelled as phishing. To enable transaction classification using the ETH Phishing dataset, we label a transaction of this dataset as phishing if its destination account is labelled as phishing. As a result, 0.278% of Ethereum transactions are labelled as phishing.

Baselines. We use LightGBM (version 3.1.1) [38] and XGBoost (version 1.7.5) [14] boosting machines, which are widely-used ML models for tabular data, as machine learning models for our graph ML pipeline. We compare our graph ML pipeline with LightGBM and XGBoost models trained exclusively using basic features, without incorporating features generated by our Graph Feature Preprocessor. To perform hyper-parameter tuning of these models, we employ a successive halving model tuning approach [35]. As additional baselines, we use the following graph neural networks (GNNs): Graph Isomorphism Network (GIN) [31, 78], GIN with edge updates (GIN+EU) [4, 11], and Principal Neighbourhood Aggregation (PNA) [20, 72]. GIN+EU baseline is similar to LaundroGraph [11], which is a GNN specifically designed for anti-money laundering.
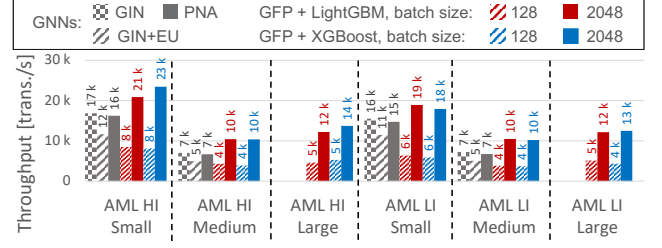
Figure 6: Our graph ML pipeline has higher throughput compared to GNN baselines executed on a V100 GPU.

The accuracy results for these GNNs on the AML datasets are obtained from Altman et al. [1]. Furthermore, all of the baselines, as well as our graph ML pipeline, are trained without the source and destination account IDs of the transactions. This prevents the models from identifying money laundering transactions based on the memorisation of account IDs.

Graph Feature Preprocessor setup. We configure GFP to extract the graph-based features in the following way. The features are extracted from the AML datasets using a time window of six hours for scatter-gather patterns and a time window of one day for the rest of the graph-based features. We specify a cycle-length constraint of 10 for simple cycle enumeration. We use the "Amount" and "Timestamp" fields of the basic transaction features to generate the vertex-statistics-based features. Feature extraction from the ETH Phishing dataset is performed using a 20-day time window for all graph-based features. In addition, we disable the generation of temporal cycles and specify a hop constraint of 5 for simple cycle enumeration. We use the "Amount", "Timestamp", and "Block Nr." fields of the basic transaction features to generate the account statistics. We selected these parameters after some careful exploration aimed at finding the best trade-offs between the throughput of GFP and the accuracy of the ML models used for scoring.

Data split. To tune the parameters of the models and to test the model generalisation performance, we split the input data into train, validation, and test sets. The train and validation sets are used by the successive halving scheme to tune the model, while the test set is used for the final evaluation of the model. For AML datasets, the splitting is performed such that 60% of transactions with the smallest timestamps is selected as a training set, the next 20% transactions with the smallest timestamps excluding the ones from the training set are selected as a validation set, and the rest are selected as the test set. For the ETH dataset, we define the timestamp of an account as the minimum timestamp among the transactions that involve this account and split the accounts of the dataset such that 65% of the accounts with the smallest timestamp exist only in the training set, the next 15% of the accounts exist only in the validation dataset, and the rest are in the test set. Splitting the datasets in the aforementioned way prevents data leakage in our experiments.

4 Results

In this section, we evaluate the accuracy of our graph ML pipeline and other baselines trained on the datasets from Table 1. We refer to our graph ML pipeline that uses LightGBM and XGBoost as GFP+LightGBM and GFP+XGBoost, respectively. As a measure of

Table 2: Minority class F1 scores (%) of the money laundering detection task using the AML datasets and the phishing detection task using the ETH Phishing dataset. NA stands for not available.

Model	batch size	AML HI			AML LI			batch size	ETH Phishing
		Small	Medium	Large	Small	Medium	Large		
GIN [31]	∞	28.70 ± 1.13	42.30 ± 0.44	NA	7.90 ± 2.78	3.86 ± 3.62	NA	∞	26.92 ± 7.52
GIN+EU [4, 11]	∞	47.73 ± 7.86	49.26 ± 4.02	NA	20.62 ± 2.41	6.19 ± 8.32	NA	∞	33.92 ± 7.34
PNA [20]	∞	56.77 ± 2.41	59.71 ± 1.91	NA	16.45 ± 1.46	27.73 ± 1.65	NA	∞	51.49 ± 4.29
LightGBM [38]	—	21.30 ± 0.30	18.60 ± 0.10	24.50 ± 0.20	2.05 ± 0.81	3.3 ± 0.48	4.04 ± 0.16	—	13.74 ± 0.54
GFP+LightGBM	128	62.86 ± 0.25	59.48 ± 0.15	58.03 ± 0.19	20.83 ± 1.50	24.74 ± 0.46	23.67 ± 0.11	128	40.17 ± 0.22
GFP+LightGBM	2048	60.52 ± 0.59	56.12 ± 0.37	54.76 ± 0.08	17.99 ± 0.60	21.06 ± 0.08	22.65 ± 0.59	∞	51.00 ± 1.01
XGBoost [14]	—	19.75 ± 0.89	20.10 ± 0.22	10.61 ± 6.73	0.21 ± 0.22	0.40 ± 0.14	0.00 ± 0.00	—	15.52 ± 0.15
GFP+XGBoost	128	63.23 ± 0.17	65.69 ± 0.26	42.68 ± 12.93	27.28 ± 0.69	31.03 ± 0.22	24.23 ± 0.12	128	37.01 ± 2.45
GFP+XGBoost	2048	64.77 ± 0.47	59.19 ± 0.29	56.88 ± 0.21	28.25 ± 0.80	21.36 ± 0.90	22.64 ± 0.15	∞	49.40 ± 0.54
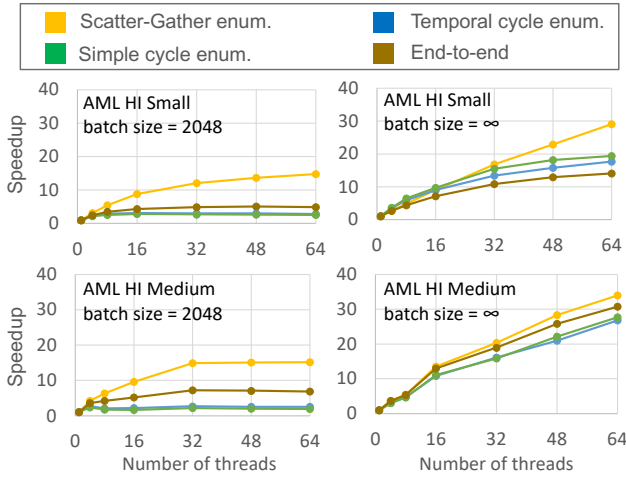
Figure 7: Scalability of executing different parts of our GFP library, as well as its end-to-end execution. The speedup is relative to the single-threaded execution.

accuracy, we use the minority-class F1 score. The F1 scores reported are averaged across five different runs. The standard deviation of the F1 score is also reported for each experiment.

Our graph ML pipeline requires transactions to arrive in batches. For the AML datasets, the graph ML pipeline uses batch sizes of 128 and 2048. In addition, for the ETH Phishing dataset, graph feature extraction is performed using batch sizes of 128 and ∞. When using a batch size of ∞, all the transactions of the test set are made available to GFP in a single batch. Using a batch size of ∞ essentially corresponds to an offline solution and, in principle, can lead to better accuracy because, in this case, the future transactions are also visible during feature extraction. However, if real-time processing capability is required by an application, the batch size will have to be constrained. Note that GNN baselines require the entire dataset to be available in memory, making it effectively an offline solution with batch size ∞.

AML results. The minority class F1 scores of the ML models that perform laundering detection using AML datasets are shown in Table 2. Clearly, our graph-based features lead to significant improvements in the F1 scores achieved by gradient-boosting models. Without our graph-based features, the maximum F1 score that LightGBM and XGBoost achieve is 24.5% for the AML HI datasets

and 4.04% for the AML LI datasets. The reason for this low accuracy is that the labels in AML datasets are highly imbalanced, and the number of illicit transactions in these datasets is at most 0.13% of the total number of transactions (see Table 1). Our graph ML pipeline, in which LightGBM and XGBoost models use our graph-based features in addition to basic features, achieves up to a 46% higher F1 scores than the models that use only basic features. Furthermore, our graph ML pipeline that uses XGBoost models consistently achieves higher F1 scores than GNN baselines. Compared to PNA, the GNN baseline with the highest accuracy, our graph ML pipeline with XGBoost achieves up to an 8% higher F1 score for AML HI datasets and up to an 11.8% higher F1 score for LI datasets.

The effect of different types of graph-based features produced by GFP on the accuracy of our graph ML pipeline for the AML task is shown in Table 3. We observe that including graph features based on fan-in and fan-out patterns already improves the minority class F1 score by more than 30% compared to the case that uses only basic transaction features. Including multi-hop graph pattern features, i.e., features based on cycles and scatter-gather patterns, further improves the F1 score by up to 4%. Finally, by incorporating vertex-statistics-based features produced by GFP, our graph ML pipeline is able to achieve higher accuracy compared to the PNA baseline (see Table 2). Thus, each type of graph-based feature contributes to the overall accuracy of our graph ML pipeline.

Figure 6 shows the throughput of our graph ML pipeline and GNN baselines. The performance of our graph ML pipeline is evaluated using 64 software threads of the Cascade Lake Intel Xeon Processor available from IBM Cloud [17], and the performance of GNN baselines is evaluated on an NVIDIA Tesla V100 GPU. We observe that our graph ML pipeline is able to achieve higher throughput than GNN baselines when it receives transactions in batches of 2048. This throughput is the result of the scalable parallel graph pattern mining algorithms that GFP uses, as shown in Figure 7. This figure also shows that our streaming scatter-gather algorithm, introduced in Section 2.2, scales almost linearly with the number of software threads when batch size is infinity. As a result of this scalability, the average latency of processing batches of 128 and 2048 transactions from the AML dataset is 30 ms and 148 ms, respectively. Being able to process a batch of transactions with low latency makes GFP suitable for real-time processing.

Explainability. A benefit of our graph ML pipeline is that it produces explainable results. Using the SHAP library [47], we can

Table 3: Minority class F1 scores (%) of our graph ML pipeline demonstrating the effect of different graph-based features produced by GFP on the accuracy of money laundering detection. Multi-hop pattern features include features based on simple cycles, temporal cycles, and scatter-gather patterns.

Dataset	AML HI Small				AML HI Medium				ETH Phishing	
Model	LightGBM		XGBoost		LightGBM		XGBoost		LightGBM	XGBoost
batch size	128	2048	128	2048	128	2048	128	2048	∞	∞
basic features	21.30 ± 0.30	21.30 ± 0.30	19.75 ± 0.89	19.75 ± 0.89	18.60 ± 0.10	18.60 ± 0.10	20.10 ± 0.22	20.10 ± 0.22	13.74 ± 0.54	15.52 ± 0.15
+ fan-in/fan-out features	50.85 ± 0.83	49.73 ± 1.20	56.88 ± 0.66	59.71 ± 0.07	46.71 ± 0.17	50.59 ± 0.36	53.00 ± 0.08	55.25 ± 0.19	35.92 ± 1.96	40.46 ± 0.94
+ multi-hop pattern features	54.66 ± 0.39	55.54 ± 0.55	58.60 ± 0.15	61.01 ± 0.24	47.47 ± 0.21	51.40 ± 0.15	55.42 ± 0.23	55.92 ± 0.26	39.46 ± 0.27	42.76 ± 0.48
+ vertex-statistic-based features	62.86 ± 0.25	60.52 ± 0.59	63.23 ± 0.17	64.77 ± 0.47	59.48 ± 0.15	56.12 ± 0.37	65.70 ± 0.26	59.19 ± 0.29	51.00 ± 1.01	49.40 ± 0.54
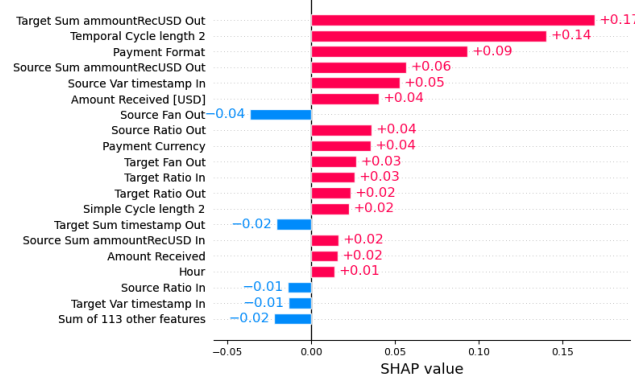
Figure 8: Importance of features used by our GFP+LightGBM setup to flag an AML HI Small transaction as illicit.

obtain the importance of features a gradient-boosting-based model used to flag a transaction as illicit. For example, the two most important features used for flagging a transaction as illicit in Figure 8 are the number of two-hop temporal cycles (*Temporal Cycle length 2*) and a vertex statistics feature, which represents the sum of money the target account received (*Target Sum ammountRecUSD Out*). Explaining the decision is critical for increasing trust in a fraud detection system because it allows analysts to verify the decisions of the system as needed.

ETH Phishing results. Table 2 also shows the minority class F1 scores achieved by the ML models we trained on the ETH Phishing dataset to perform phishing detection. When using a batch size of 128, our graph-based features enable F1-score improvements exceeding 20% for both LightGBM and XGBoost. Setting the batch size to ∞ further improves the F1 score of LightGBM to 51%. In that case, LightGBM with our graph-based features outperforms the GIN+EU baseline by 10% and achieves competitive accuracy with PNA. However, increasing the batch size from 128 to ∞ effectively makes our graph ML pipeline an offline solution. In general, the optimal configuration of GFP depends on the requirements of the end application and might require trading off performance for accuracy.

5 Related Work

Graph machine learning has applications in many different fields, including financial transaction network analysis [12, 44, 50, 74], fraud detection [2, 10, 21, 22, 45, 80], drug discovery [26], molecular property prediction [79], genomics [65], recommender systems [23], social network analysis [5, 24], and relation prediction in knowledge graphs [55]. Fraud detection systems TitAnt [10] and Eddin et al. [22] are graph machine learning systems that extract features

from transaction graphs by generating node embeddings [53] or by performing random walks [51] in graphs. These features are then used by machine learning models to predict whether an incoming transaction is fraudulent or not.

Graph neural networks (GNNs) [8, 11, 29, 40, 45, 71, 74, 78] are powerful tools that can be used for the purpose of financial crime detection. Cardoso et al. [11] and Weber et al. [75] apply GNN to the anti-money laundering problem, Kanezashi et al. [37] apply GNN to the phishing detection problem on the Ethereum blockchain, and Rao et al. [57] uses a GNN to detect fraudulent transactions. Graph Substructure Network, proposed by Bouritsas et al. [8], takes advantage of pre-calculated subgraph pattern counts to improve the expressivity of GNNs. GNNs could also be used to count subgraph patterns, such as in Chen et al. [16], which could enable detecting patterns associated with financial crime. In contrast to our work, GNNs cannot straightforwardly operate in a streaming manner and require the entire dataset to be available at the time of testing.

6 Conclusions

We presented Graph Feature Preprocessor (GFP), a software library for fast feature extraction from dynamically changing transaction graphs. To achieve fast feature extraction, our library leverages an in-memory dynamic multigraph representation as well as fine-grained parallel subgraph enumeration algorithms. GFP enables our graph ML pipeline to operate in a streaming manner with low per-batch latency and higher throughput compared to the GNN baselines presented in the experiments. This capability makes GFP suitable for scenarios that require real-time processing.

We have also shown that the graph-based features generated by GFP can significantly improve the accuracy of gradient-boosting-based machine learning models. The graph-based features improve the minority class F1 score of gradient-boosting-based machine learning models by up to 46% for the synthetic AML datasets and by up to 35% for a real-world phishing detection dataset extracted from Ethereum. Furthermore, we show that our solution achieves up to a 36% higher F1 score than GNN baselines for the AML task. In particular, our graph ML pipeline achieves up to a 24% higher minority-class F1 score compared to the GIN+EU baseline with the similar architecture to LaundroGraph [11], which is a GNN designed specifically for anti-money laundering.

The application scope of our GFP library is not limited to money laundering detection. Given that a cycle in a graph can be an indicator of tax avoidance [28], circular trading [34, 36, 52], and credit card frauds [49, 56], a GFP could also help to detect these types of frauds. However, the reliance on pre-defined subgraph patterns, such as cycles, is one drawback of this library, which we plan to address as

part of the future work by adding the support for subgraph matching using user-defined subgraph patterns in GFP [68]. Furthermore, we plan to add support for feature extraction based on additional subgraph patterns, such as cliques [9] and bicliques [54]. Being able to enumerate these patterns could enable the detection of close-knit communities [46] as well as stacked money laundering patterns [1] encountered in various different financial crime scenarios.

Acknowledgments

References

[1] Erik Altman, Jovan Blanuša, Luc von Niederhäusern, Béni Egressy, Andreea Anghel, and Kubilay Atasu. 2023. Realistic Synthetic Financial Transactions for Anti-Money Laundering Models. In *NeurIPS'23, Datasets and Benchmarks Track*.

[2] Amazon. 2023. Amazon Fraud Detector. https://aws.amazon.com/fraud-detector/ Accessed: 2023-01-10.

[3] V K Balakrishnan. 1997. *Graph Theory*. McGraw-Hill Professional, New York, NY.

[4] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[5] Austin R. Benson, David F. Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353, 6295 (2016), 163–166. https://doi.org/10.1126/science.aad9029

[6] Jovan Blanuša, Paolo Ienne, and Kubilay Atasu. 2022. Scalable Fine-Grained Parallel Cycle Enumeration Algorithms. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, Philadelphia PA USA, 247–258. https://doi.org/10.1145/3490148.3538585

[7] Jovan Blanuša, Kubilay Atasu, and Paolo Ienne. 2023. Fast Parallel Algorithms for Enumeration of Simple, Temporal, and Hop-constrained Cycles. *ACM Trans. Parallel Comput.* 10, 3 (Sept. 2023), 1–35. https://doi.org/10.1145/3611642

[8] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. 2023. Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting. *IEEE Trans. Pattern Anal. Mach. Intell.* 45, 1 (Jan. 2023), 657–668. https://doi.org/10.1109/TPAMI.2022.3154319

[9] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (Sept. 1973), 575–577. https://doi.org/10.1145/362342.362367

[10] Shaosheng Cao, XinXing Yang, Cen Chen, Jun Zhou, Xiaolong Li, and Yuan Qi. 2019. TitAnt: online real-time transaction fraud detection in Ant Financial. *PVLDB* 12, 12 (Aug. 2019), 2082–2093. https://doi.org/10.14778/3352063.3352126

[11] Mário Cardoso, Pedro Saleiro, and Pedro Bizarro. 2022. LaundroGraph: Self-Supervised Graph Representation Learning for Anti-Money Laundering. In *Proceedings of the Third ACM International Conference on AI in Finance*. 130–138.

[12] Tao-Hung Chang and Davor Svetinovic. 2020. Improving Bitcoin Ownership Identification Using Transaction Patterns Analysis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 50, 1 (2020), 9–20. https://doi.org/10.1109/TSMC.2018.2867497

[13] Liang Chen, Jiaying Peng, Yang Liu, Jintang Li, Fenfang Xie, and Zibin Zheng. 2019. XBLOCK Blockchain Datasets: InPlusLab Ethereum Phishing Detection Datasets. http://xblock.pro/ethereum/.

[14] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) (*KDD '16*). ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[15] Xucan Chen, Mohammad Al Hasan, Xintao Wu, Pavel Skums, Mohammad Javad Feizollahi, Marie Ouellet, Eric L. Sevigny, David Maimon, and Yubao Wu. 2019. Characteristics of Bitcoin Transactions on Cryptomarkets. In *Security, Privacy, and Anonymity in Computation, Communication, and Storage*, Guojun Wang, Jun Feng, Md Zakirul Alam Bhuiyan, and Rongxing Lu (Eds.). Vol. 11611. Springer International Publishing, Cham, 261–276. https://doi.org/10.1007/978-3-030-24907-6_20 Series Title: Lecture Notes in Computer Science.

[16] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. 2020. Can Graph Neural Networks Count Substructures?. In *NeurIPS 2020, December 6-12, 2020,*

[17] *virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.).

[17] IBM Cloud. 2024. IBM Cloud Docs - Virtual Private Cloud (VPC). https://cloud.ibm.com/docs/vpc Accessed: 2024-02-08.

[18] Thomas H. Cormen (Ed.). 2009. *Introduction to algorithms* (3rd ed ed.). MIT Press, Cambridge, Mass. OCLC: ocn311310321.

[19] Livio Corselli. 2023. Italy: money transfer, money laundering and intermediary liability. *JFC* 30, 2 (Feb. 2023), 377–388. https://doi.org/10.1108/JFC-10-2019-0137

[20] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. 2020. Principal Neighbourhood Aggregation for Graph Nets. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 13260–13271.

[21] Andras Cser, Merritt Maxix, Caroline Provost, and Peggy Dostie. 2022. *The Forrester Wave™: Anti-Money-Laundering Solutions, Q3 2022*. Technical Report. Forrester. 1–10 pages. https://www.forrester.com/report/the-forrester-wave-tm-anti-money-laundering-solutions-q3-2022/RES176346 Accessed: 2023-01-10.

[22] Ahmad Naser Eddin, Jacopo Bono, David Aparício, David Polido, João Tiago Ascensão, Pedro Bizarro, and Pedro Ribeiro. 2022. Anti-Money Laundering Alert Optimization Using Machine Learning with Graphs. arXiv:2112.07508 [cs].

[23] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) (*WWW '18*). 1775–1784. https://doi.org/10.1145/3178876.3186183

[24] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph Neural Networks for Social Recommendation. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. ACM, 417–426. https://doi.org/10.1145/3308558.3313488

[25] Tony Finch. 2009. Incremental calculation of weighted mean and variance. (01 2009), 1–8.

[26] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy B R Hayter, Richard Vickers, Charles Roberts, Jian Tang, David Roblin, Tom L Blundell, Michael M Bronstein, and Jake P Taylor-King. 2021. Utilizing graph machine learning within drug discovery and development. *Briefings in Bioinformatics* 22, 6 (05 2021). https://doi.org/10.1093/bib/bbab159

[27] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data?. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 507–520.

[28] László Hajdu and Miklós Krész. 2020. Temporal Network Analytics for Fraud Detection in the Banking Sector. In *ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium*. Vol. 1260. Springer International Publishing, Cham, 145–157. https://doi.org/10.1007/978-3-030-55814-7_12 Series Title: Communications in Computer and Information Science.

[29] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[30] Petter Holme and Jari Saramäki. 2012. Temporal networks. *Physics Reports* 519, 3 (Oct. 2012), 97–125. https://doi.org/10.1016/j.physrep.2012.03.001

[31] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265* (2019).

[32] IBM. 2023. AI Toolkit for IBM Z and LinuxONE. https://www.ibm.com/products/ai-toolkit-for-z-and-linuxone Accessed: 2024-01-25.

[33] IBM. 2023. Cloud Pak for Data. https://www.ibm.com/products/cloud-pak-for-data Accessed: 2023-02-21.

[34] Md. Nazrul Islam, S. M. Rafizul Haque, Kaji Masudul Alam, and Md. Tarikuzzaman. 2009. An approach to improve collusion set detection using MCL algorithm. In *2009 12th International Conference on Computers and Information Technology*. IEEE, Dhaka, Bangladesh, 237–242. https://doi.org/10.1109/ICCIT.2009.5407133

[35] Kevin Jamieson and Robert Nowak. 2014. Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting. In *2014 48th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, Princeton, NJ, USA, 1–6. https://doi.org/10.1109/CISS.2014.6814096

[36] Zhi-Qiang Jiang, Wen-Jie Xie, Xiong Xiong, Wei Zhang, Yong-Jie Zhang, and Wei-Xing Zhou. 2013. Trading networks, abnormal motifs and stock manipulation. *Quantitative Finance Letters* 1, 1 (Dec. 2013), 1–8. doi:10.1080/21649502.2013.802877.

[37] Hiroki Kanezashi, Toyotaro Suzumura, Xin Liu, and Takahiro Hirofuchi. 2022. Ethereum Fraud Detection with Heterogeneous Graph Neural Networks. arXiv:2203.12363 [cs].

[38] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[39] Nancy Kinnison and John Madinger (Eds.). 2011. *Money Laundering: A Guide for Criminal Investigators, Third Edition*. Routledge, Boston, MA.

[40] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

[41] Stephen Kokoska and Daniel Zwillinger. 2000. *CRC Standard Probability and Statistics Tables and Formulae, Student Edition* (0 ed.). CRC Press. https://doi.org/10.1201/b16923

[42] Meng-Chieh Lee, Yue Zhao, Aluna Wang, Pierre Jinghong Liang, Leman Akoglu, Vincent S. Tseng, and Christos Faloutsos. 2020. AutoAudit: Mining Accounting and Time-Evolving Graphs. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, Atlanta, GA, USA, 950–956. https://doi.org/10.1109/BigData50022.2020.9378346

[43] Xiangfeng Li, Shenghua Liu, Zifeng Li, Xiaotian Han, Chuan Shi, Bryan Hooi, He Huang, and Xueqi Cheng. 2020. FlowScope: Spotting Money Laundering Based on Graphs. *AAAI* 34, 04 (April 2020), 4731–4738. https://doi.org/10.1609/aaai.v34i04.5906

[44] Xiao Fan Liu, Xin-Jian Jiang, Si-Hao Liu, and Chi Kong Tse. 2021. Knowledge Discovery in Cryptocurrency Transactions: A Survey. *IEEE Access* 9 (2021), 37229–37254. https://doi.org/10.1109/ACCESS.2021.3062652

[45] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2021. Pick and Choose: A GNN-Based Imbalanced Learning Approach for Fraud Detection. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 3168–3177. https://doi.org/10.1145/3442381.3449989

[46] Zhenqi Lu, Johan Wahlström, and Arye Nehorai. 2018. Community Detection in Complex Networks via Clique Conductance. *Sci Rep* 8, 1 (Dec. 2018), 5982. https://doi.org/10.1038/s41598-018-23932-z

[47] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 4765–4774.

[48] Prabhaker Mateti and Narsingh Deo. 1976. On Algorithms for Enumerating All Circuits of a Graph. *SIAM J. Comput.* 5, 1 (March 1976), 90–99. https://doi.org/10.1137/0205007

[49] Jack Nicholls, Aditya Kuppa, and Nhien-An Le-Khac. 2021. Financial Cybercrime: A Comprehensive Survey of Deep Learning Approaches to Tackle the Evolving Financial Crime Landscape. *IEEE Access* 9 (2021), 163965–163986. https://doi.org/10.1109/ACCESS.2021.3134076

[50] Jack Nicholls, Aditya Kuppa, and Nhien-An Le-Khac. 2021. Financial Cybercrime: A Comprehensive Survey of Deep Learning Approaches to Tackle the Evolving Financial Crime Landscape. *IEEE Access* 9 (2021), 163965–163986. https://doi.org/10.1109/ACCESS.2021.3134076

[51] Catarina Oliveira, João Torres, Maria Inês Silva, David Aparício, João Tiago Ascensão, and Pedro Bizarro. 2021. GuiltyWalker: Distance to illicit nodes in the Bitcoin network. arXiv:2102.05373 [cs].

[52] Girish Keshav Palshikar and Manoj M. Apte. 2008. Collusion set detection using graph clustering. *Data Min Knowl Disc* 16, 2 (April 2008), 135–164. https://doi.org/10.1007/s10618-007-0076-8

[53] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, New York New York USA, 701–710. https://doi.org/10.1145/2623330.2623732

[54] Erich Prisner. 2000. Bicliques in Graphs I: Bounds on Their Number. *Combinatorica* 20, 1 (Jan. 2000), 109–117. https://doi.org/10.1007/s004930070035

[55] Xiao Qin, Nasrullah Sheikh, Berthold Reinwald, and Lingfei Wu. 2021. Relation-aware Graph Attention Model with Adaptive Self-adversarial Training. In *AAAI'21*. AAAI Press, 9368–9376.

[56] Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. 2018. Real-time constrained cycle detection in large dynamic graphs. *PVLDB* 11, 12 (Aug. 2018), 1876–1888. doi: 10.14778/3229863.3229874.

[57] Susie Xi Rao, Shuai Zhang, Zhichao Han, Zitao Zhang, Wei Min, Zhiyao Chen, Yinan Shan, Yang Zhao, and Ce Zhang. 2021. xFraud: explainable fraud transaction detection. *PVLDB* 15, 3 (Nov. 2021), 427–436. https://doi.org/10.14778/3494124.3494128

[58] C++ reference. 2023. std::unordered_map. https://en.cppreference.com/w/cpp/container/unordered_map Accessed: 2023-02-21.

[59] IBM Research. 2022. Graph Feature Preprocessor Public Examples. https://github.com/IBM/snapml-examples/blob/main/examples/graph_feature_preprocessor/graph_feature_preprocessor.ipynb Accessed: 2023-03-3.

[60] IBM Research. 2022. Graph Feature PreprocessorDocumentation. https://snapml.readthedocs.io/en/latest/graph_preprocessor.html Accessed: 2023-01-10.

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

[62] Peter Reuter and Edwin M. Truman. 2004. *Chasing Dirty Money: The Fight Against Money Laundering*. Institute for International Economics, Washington, DC, Chapter Money Laundering: Methods and Markets.

[63] Evan Rivera, Jovan Blanuša, Jawaharlal Rajan, Alexis Landis, and Haris Pozidis. 2024. AI on IBM Z Anti-Money Laundering Solution Template. https://github.com/ambitus/aionz-st-anti-money-laundering Accessed: 2024-10-02.

[64] Viktoria Ronge, Christoph Egger, Russell W. F. Lai, Dominique Schröder, and Hoover H. F. Yin. 2021. Foundations of Ring Sampling. *Proceedings on Privacy Enhancing Technologies* 2021, 3 (July 2021), 265–288. https://doi.org/10.2478/popets-2021-0047

[65] Roman Schulte-Sasse, Stefan Budach, Denes Hnisz, and Annalisa Marsico. 2021. Integration of multiomics data with graph convolutional networks to identify new cancer genes and their associated molecular mechanisms. *Nature Machine Intelligence* 3, 6 (2021), 513–526. https://doi.org/10.1038/s42256-021-00325-y

[66] scikit-learn developers. 2022. scikit-learn: Preprocessing Data. https://scikit-learn.org/stable/modules/preprocessing.html Accessed: 2023-01-16.

[67] Michele Starnini and Charalampos E. Tsourakakis et al. 2021. Smurf-Based Anti-money Laundering in Time-Evolving Transaction Networks. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track*. Vol. 12978. Springer International Publishing, Cham, 171–186. https://doi.org/10.1007/978-3-030-86514-6_11

[68] Shixuan Sun and Qiong Luo. 2020. In-Memory Subgraph Matching: An In-depth Study. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. ACM, Portland OR USA, 1083–1098. https://doi.org/10.1145/3318464.3380581

[69] Toyotaro Suzumura and Hiroki Kanezashi. 2021. Anti-Money Laundering Datasets: InPlusLab Anti-Money Laundering DataDatasets. http://github.com/IBM/AMLSim/.

[70] Katharina Tschumitschew and Frank Klawonn. 2012. Incremental Statistical Measures. In *Learning in Non-Stationary Environments*, Moamar Sayed-Mouchaweh and Edwin Lughofer (Eds.). Springer New York, New York, NY, 21–55. https://doi.org/10.1007/978-1-4419-8020-5_2

[71] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations* (2018).

[72] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. *ICLR (Poster)* 2, 3 (2019), 4.

[73] Samourai Wallet. 2021. Whirlpool Coinjoin. https://samouraiwallet.com/whirlpool

[74] Jianian Wang, Sheng Zhang, Yanghua Xiao, and Rui Song. 2021. A Review on Graph Neural Network Methods in Financial Applications. *CoRR* abs/2111.15367 (2021). arXiv:2111.15367

[75] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591* (2019).

[76] Jiajing Wu, Jieli Liu, Weili Chen, Huawei Huang, Zibin Zheng, and Yan Zhang. 2021. Detecting Mixing Services via Mining Bitcoin Transaction Network With Hybrid Motifs. *IEEE Trans. Syst. Man Cybern, Syst.* (2021), 1–13. https://doi.org/10.1109/TSMC.2021.3049278

[77] Xblock. 2024. Ethereum Phishing Transaction Network. https://www.kaggle.com/datasets/xblock/ethereum-phishing-transaction-network Accessed: 2023-01-27.

[78] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[79] Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. 2021. Motif-based Graph Self-Supervised Learning for Molecular Property Prediction. *CoRR* abs/2110.00987 (2021). arXiv:2110.00987

[80] Yongchun Zhu, Dongbo Xi, Bowen Song, Fuzhen Zhuang, Shuai Chen, Xi Gu, and Qing He. 2020. Modeling Users' Behavior Sequences with Hierarchical Explainable Network for Cross-domain Fraud Detection. In *Proceedings of The Web Conference 2020*. ACM, Taipei Taiwan, 928–938. https://doi.org/10.1145/3366423.3380172