Catch Me If You Can: Semi-supervised Graph Learning for Spotting Money Laundering

Md. Rezaul Karim, Felix Hermsen, Sisay Adugna Chala, Paola de Perthuis, and Avikarsha Mandal

✦

Abstract—Money laundering is the process where criminals use financial services to move massive amounts of illegal money to untraceable destinations and integrate them into legitimate financial systems. It is very crucial to identify such activities accurately and reliably in order to enforce anti-money laundering (AML). Despite tremendous efforts to AML only a tiny fraction of illicit activities is prevented. From a given graph of money transfers between accounts of a bank, existing approaches attempted to detect money laundering. In particular, some approaches employ structural and behavioural dynamics of dense subgraph detection thereby not taking into consideration that money laundering involves high-volume flows of funds through chains of bank accounts. Some approaches model the transactions in the form of multipartite graphs to detect the complete flow of money from source to destination. However, existing approaches yield lower detection accuracy, making them less reliable. In this paper, we employ semi-supervised graph learning techniques on graphs of financial transactions in order to identify nodes involved in potential money laundering. Experimental results[1] suggest that our approach can sport money laundering from real and synthetic transaction graphs.

Index Terms—Money laundering, Graph embedding, Machine learning.

1 INTRODUCTION

Money laundering is an alarming problem globally. It causes approximately 1.6 Trillion USD corresponding to 2.7% of the global GDP is laundered every year [1], [2]. Criminals involved in money laundering activities, often hide the original sources of illegal money by using the funds in casinos or real estate purchases, or by overvaluing legitimate invoices. There are many ways of money laundering, yet generally, it is composed of three primary steps: *placement*, *layering*, and *integration*. Placement is about introducing dirty money into existing financial systems. Layering is the process of carrying out complex transactions to hide the source of the funds. The integration is about withdrawing the proceeds from a destination bank account before using the fund for legitimate activities.

Anti-money laundering (AML) is the task of preventing criminals from moving illicit funds through the financial system. AML is often perceived through regulatory compliance since the burden of forensic analysis falls primarily on financial institutions that are responsible to comply with Know Your Customer (KYC) standards, monitoring transactions, shutting down or restricting accounts deemed suspect, and submitting timely Suspicious Activities Reports (SARs) to law enforcement agencies. These are typically carried out in a five-step process:

1) introducing a compliance organization within the company including formal AML training for employees.
2) emphasising and execution of KYC onboarding and profile maintenance procedures.
3) account activity oversight and constraints via transaction monitoring systems (TMS).
4) manual review of flagged accounts and transactions.
5) filing of SARs to law enforcement and corresponding restrictive action against suspect accounts.

Transaction monitoring systems are predominantly rules-based thresholding protocols tuned for the volume and velocity of transactions with tiered escalation procedures. Thorough analysis is carried out using sophisticated techniques to determine whether or not a SAR need to be filed and the account in question suspended. There are two main topologies to represent how the money-laundering is carried out: The first topology involves one sender account and one receiver account, and many intermediary accounts forming a column. The funds are divided by the sender among intermediary accounts, which then resend the funds to the receiver account. The second topology consists of one sender and one receiver and a row of intermediary accounts. The intermediary accounts pass the entire amount[2] of funds to the next account in the row until they are received at the receiver account.

A money transfer or transaction graph can be constructed such that a single account is represented as a vertex and a single transaction between two accounts is represented as an edge. Besides, a group of accounts can be represented as a vertex (under a holding company or inferred to shareowners), while an edge can represent the aggregate transaction volume with a neighbouring node over a period of time. Further, a node entity might be a single account or a set of associated accounts in AML transaction monitoring.

Identifying money launders from such a graph is very challenging. Automatically or semi-automatically spotting culprits[3] from a massive graph data by mapping billions of edges between millions of entities (nodes) needs a scalable

Md. Rezaul Karim is with Fraunhofer FIT & RWTH Aachen University
Felix Hermsen, Sisay A. Chala, & Avikarsha Mandal are with Fraunhofer FIT
Paola de Perthuis is with Cosmian, France.

[1] Codes to appear at: https://github.com/AwesomeDeepAI/Graph-based-money-laundering-detection

[2] Sometimes minus the optional commission.   [3] That could potentially involve money laundering activities.

and efficient method. Nevertheless, criminals can be quite sophisticated in masking the true nature of their transactions with complicated account layering[4] or multi-hop transactions, making the identification of money laundering extremely difficult and computationally complex problem. Consequently, using existing methods or tools, only a tiny fraction of illicit activities is prevented despite of tremendous efforts to AML, while penalties for failed AML compliance are severe. Therefore, it is of uttermost importance for any country or financial organization to identify such activities accurately and reliably in order to enforce AML.

Further, recent approaches that employed non-graph-based models mostly benefit from using additional features generated by graph-based models such as graph neural networks (GNNs) [3]. These inspired us to combine the power of graph analytics with tree-based ensemble models, thereby modelling both spatial and temporal information in a large-scale graph, in semi-supervised learning settings [4]. We hypothesize that similar to network analysis that involves predictions over nodes and edges (e.g., predicting most probable labels of nodes in a network), nodes showing distinct characteristics from regular nodes can be classified as potential money launders in a transaction graph, by modelling it as a node classification problem. In the end, we employ a semi-supervised graph learning technique on the graph of financial transactions in order to identify nodes involved in potential money laundering.

2 RELATED WORKS

Several methods have been proposed to accurately identify AML activities [6]. The earliest approaches predominantly relied on rule-based classification. For example, Rajput et al. [7] developed an ontology-based expert system to detect suspicious transactions. However, rule-based algorithms are easy to be evaded by fraudsters [6]. Therefore, graph and ML-based approaches have emerged. Michalak et al. [8] used fuzzy matching to capture subgraphs that are more likely to contain suspicious accounts involved in fraudulent activities. A few other approaches tried to assess if the capital flow is involved in money laundering activities using radial basis function (RBF) neural networks calculating w.r.t time-to-times [6]. From a graph of money transfers between accounts, existing approaches attempted to detect money laundering activities by employing a variety of methods starting from simple logistic regression (LR), support vector machines (SVM) [9], random forest (RF), and multilayer perceptron (MLP) to more sophisticated approach that are based on GNNs [3]. Some earlier approaches [10], [11] employed a clustering-based method to detect money laundering activities by grouping transactions into clusters. Soltani et al. [5] proposed a money laundering detection algorithm. This structural similarity-based approach finds pairs of transactions with common attributes and behaviours that potentially involve money laundering activities.

Money laundering usually involves high-volume flows of funds through chains of bank accounts [6]. However, approaches that employ structural and behavioural dynamics of dense subgraph detection do not take it into account [6].

Methods that do not perform flow tracking, yield lower detection accuracy and hence cannot provide theoretical guarantees, while the flow across multiple nodes is important for accuracy and robustness against camouflage in the money laundering activities [12]. However, real-world data is typically either unlabeled, or has noisy, or sparse labels, whereas these algorithms detect money laundering activities in a supervised manner, suffering from highly skewed labels and limited adaptability. Further, money laundering activities often involve cash flow relationships between entities, i.e., network structures [6]. Therefore, some approaches model the transactions in the form of multipartite graphs to detect the complete flow of money from source to destination [6] in an unsupervised manner based on graphs. In particular, FlowScope [6] is a flow-based approach for detecting money laundering behaviour to detect the chains of transactions.

A knowledge graph (KG) can be formed in a similar fashion, where nodes represent entities and edges represent binary relations between those entities [13]. More formally, $\mathcal{G} = \{E, R, T\}$, where $\mathcal{G}$ is a labelled and directed multi-graph, and E, R, T are the sets of entities, relations, and triples, respectively. Each triple[5] in the KG can be formalized as $(h, r, t) \in T$, where $h \in E$ is the head node, $t \in E$ is the tail node, and $r \in R$ is the edge connecting h and t (i.e., the relation r holds between h and t) [13]. Recently, graph analytics techniques on such KGs have emerged as an increasingly important means for AML analysis. In particular, GNN architectures such as graph convolutional networks (GCN) [4], GraphSAGE [14], and FastGCN [15] have emerged [5], showing efficiency and scalability in graph-based representation learning [15], [16]. The reason is that in low-labelled dataset scenarios, unsupervised techniques can learn low-dimensional meaningful representations of nodes by leveraging the graph structure and features. An unsupervised GE model learns embeddings of unlabeled graph nodes. Further, literature has shown that the inclusion of non-local information – specifically information about the neighbours of a centre node always improves the performance of each model, non-graph-based models mostly benefit from using additional features given by a GE model. This makes graph learning-based approaches quite promising due to their ability to understand complex patterns and feature discovery capabilities.

Generating quality feature vectors using appropriate graph embedding (GE) methods play a significant role in any downstream learning task. Numerous GE methods have been proposed to date [3]. A GE technique aims to map a KG into a dense, low-, feature space, which is capable of preserving as much structure and property information of the graph as possible and aiding in calculations of the entities and relations[17]. A GE model consists of three steps: representing entities and relations, defining a scoring function, and learning entity and relation representation [18]. Translation-based embedding techniques such as TransD [19] and TransE [20] are the earliest embedding methods that represent the neighbourhood of nodes as well as the kind of relations that exist to the neighbouring nodes.

Using translation-based embedding methods, embeddings are generated by treating relations as translations

---

[4] In order to confuse AML tools and methods.

[5] The notation $< head, relation, tail >$ is called resource description framework (RDF) - a W3C standard for serializing data.
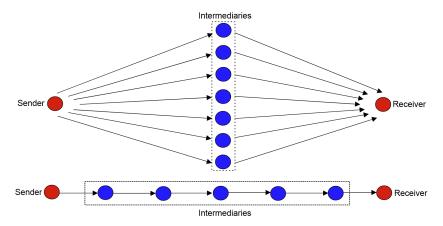
Fig. 1: Two common topologies of money laundering networks (recreated based on literature [5])

from the head entity to the tail entity [21]. Embeddings are created such that $\mathbf{h} \bigoplus \mathbf{r} \approx \mathbf{t}$, by preserving a proximity measure defined on graph $\mathcal{G}^6$. The resultant embedding vectors for the entities and relations in in the KG are a denser and more efficient representation of the domain that can more easily be used for many downstream tasks. However, most translation-based embeddings are limited in their capacity to model complex and diverse objects, including important properties of relations, such as symmetric, transitive, one-many, many-to-one, and many-many relations in KGs [22]. Therefore, GE methods such as RDF2Vec[23], SimpleIE [24], KGloVe [25], and CrossE [26] have been proposed.

On the other hand, the GraphSAGE model learns embeddings of unlabeled graph nodes by leveraging the graph structure and features of the nodes. Being an inductive embedding technique, GraphSAGE allows extracting embeddings of unseen nodes, without the need to re-training. Unlike another embedding model such as Node2Vec which learns a look-up table of node embeddings instead of training individual embeddings for each node, GraphSAGE learns a function that generates embeddings by sampling and aggregating attributes from each node's local neighbourhood and combines those with the node's own attributes [14]. As real-life transaction network graphs are large-scale, some works were published with a view on scalability. In particular, FastGCN outperformed GCN and GraphSAGE in various benchmark datasets by as much as two orders of magnitude without sacrificing accuracy [27]. Weber et al. [27] trained GCN and FastGCN models on a large synthetic graph of 1 million nodes and 9 million edges. Although the GCN model significantly outperformed LR, the RF model turns out to be the best classifier, even outperforming the GCN. Another recent work [28] using a two-layer GCN model on a real transaction graph called *Elliptic* from Bitcoin blockchain, shows that the GCN model outperforms linear models like LR, comparable to MLP, but underperforms in comparison with RF. They showed that the inclusion of non-local information – specifically information about the neighbours of a centre node always improves the performance of each model.

The majority of these models take into consideration only spatial information, thereby largely ignoring tem-

poral information. However, transaction graphs are dynamic, where each transaction has an associated timestamp. To learn knowledge from such a dynamic graph, EvolveGCN [29] is proposed to extract node embeddings by coupling both spatial- and temporal information. EvolveGCN evolves along the temporal axis, where a recurrent neural network (RNN) is used for evolving the GCN model's parameters. EvolveGCN is flexible for modelling temporal data since it is not reliant on node embeddings. Dynamic graph transformer (DGT) [30] is another approach in which the transformer model is composed of two modules: the *transformer* and the *pooling*. The transformer module captures the cross-domain knowledge by the attention mechanism, where the final attention layer generates the informative node embeddings.

3 METHODS

We employ a semi-supervised graph learning technique on transaction graphs in order to identify nodes involved in potential money laundering. For this, we employ both pipeline and *end-to-end* approaches. For the former, an embedding model is first trained to generate node embeddings that are used to train the ERT, GBT, and RF classifiers. For the latter, the node classification is performed in an end-to-end setting, without requiring training any separate classifiers.

3.1 Problem formulation

Let $\mathcal{G} = (V, E)$ be a money transfer or transaction graph, where nodes V represent accounts and edges E represent transfers. Let $V = X \cup W \cup Y$, where W is the inner accounts of the bank, and X and Y are sets of outer accounts. X is the set of accounts that have the net transfer of money into the bank, and Y is the set of accounts that have a net transfer out of the bank. An edge $(i, j) \in E$ indicates that account v_i transfers money into another account v_j for $v_i, v_j \in V$ and e_{ij} is the amount of money transferred. Based on this setting, a directed KG can be represented as triplet facts $(h, r, t) \in F$ such that $\mathcal{G} = (V, E, F)$ to denote a link $r \in R$ between the head $\in V$ and the tail $t \in E$.

Now using annotations from the alerts, the task in a semi-supervised learning setting is embedding the nodes into a lower dimensional vector space, followed by using the

---

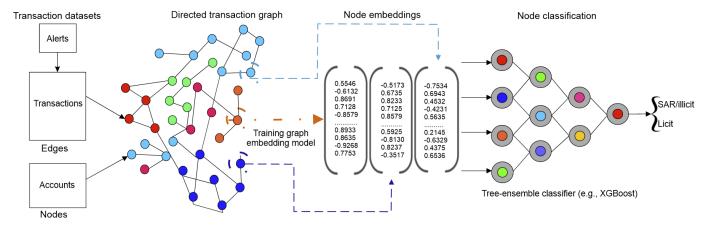6. The $\bigoplus$ operator could be different w.r.t GE models.

Fig. 2: Workflow of our pipeline method for identifying nodes potentially involved in money laundering activities

embedding vectors to train a binary classifier to predict the suspiciousness of a given target node in the graph via direct or indirect connections to nodes known to be suspicious. Let the embedding model Γ embed each node of graph $\mathcal{G}$ into a lower dimensional vector space, yielding a set of vectors $\vec{V}$. More formally, given a graph $\mathcal{G}$, a node embedding is a mapping $\Gamma : v_i \rightarrow \vec{v_i} \in \mathbb{R}, \forall i \in [N]$ by capturing the information of the graph[7], where d is the dimension of the embeddings and N is the number of nodes. The task is now to train a classifier f on $\vec{V}$ in order to predict if a node is of suspiciousness, where the prediction $\hat{y}_i$ for embedding vector $\vec{v_i}$ for the ith node can be defined as follows:

$$\hat{y}_i = f(\vec{v_i}) = \begin{cases} 1, & \text{if flagged, e.g., SAR or illicit} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

For the sake of employing a semi-supervised learning paradigm, we remove a certain per cent of the nodes (including edges from these nodes to any other nodes in the network), followed by training a GE model on the reduced sub-graph. During the inferencing, we generate the embeddings of the removed nodes using the trained GE model that are subsequently used to predict the labels of the nodes originally held out after re-inserting them in the network.

3.2 Generating directed graphs

We generate a directed transactional graph in five steps:

- **Step-1:** Load transactions, alerts[8], and party datasets.
- **Step-2:** Define user defined utility functions for preprocessing and encoding categorical features.
- **Step-3:** Generate graph nodes and edges.
- **Step-4:** Form a directed graph for nodes and edges.
- **Step-5:** Annotate the nodes with alerts datasets or additional features[9].

Examples of different types of transactions that are flagged with "alert types" are shown in fig. 3.

3.3 Graph embeddings

Since ML classifiers do typically expect their input as fixed-length vectors, we employ different unsupervised graph representation learning techniques to generate node embeddings using the Word2vec, Node2vec, Attri2Vec, GraphSAGE, and DGT models that represent the neighbourhood of a node and their relations to the neighbouring nodes.

Using *Word2vec* and *Node2Vec*, a corpus of text $\mathcal{C}$ is generated by performing uniform random walks starting from each entity in the graph [31]. Then, $\mathcal{C}$ of edge-labelled random walks are used as the input for learning embeddings of each node using the skip-gram (SG) Word2vec [32] model. From a given a sequence of facts $(w_1, w_2, ..., w_n) \in \mathcal{C}$, the SG model aims to maximize the average log probability L_p according to the context within the fixed-size window [32]:

$$L_p = \frac{1}{N} \sum_{n=1}^{N} \sum_{-c \leq j \leq c, j \neq 0} \log p\left(w_{n+j} | w_n\right), \quad (2)$$

where c represents a context. To define $p\left(w_{n+j} | w_n\right)$, we use negative sampling by replacing $\log p\left(w_O | w_I\right)$ with a function to discriminate target words (w_o) from a noise distribution $P_n(w)$ drawing k words from $P_n(w)$ [21]:

$$\log \sigma \left(v_{w_O}'^{\top} v_{w_I}\right) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma \left(-v_{w_i}'^{\top} v_{w_I}\right)\right]. \quad (3)$$

The embedding of a concept c occurring in corpus $\mathcal{C}$ is the vector $\vec{v_s}$ in eq. (3) derived by maximizing eq. (2). Technically both Word2vec and Node2Vec algorithms follow is a 2-step representation learning algorithm: i) using second-order random walks to generate sentences from a graph[10], ii) the corpus is then used to learn an embedding vector for each node in the graph. Each node id is considered a unique word/token in a dictionary that has a size equal to the number of nodes N in our graph $\mathcal{G}$.

The Attri2Vec model [33] is trained to learn node representations by performing linear/non-linear mapping on node content attributes. To make the learned node representations respect structural similarity, DeepWalk/Node2Vec learning mechanism is employed to make nodes sharing

---

[7] Depending on different embedding methods Γ and embedding dimensions, different embeddings vector can be generated for the entities. [8] e.g., containing falgs like SARs or illicitness. [9] i.e., labelled SARs indicating if a node was part of a previously known money laundering scheme or illicit.

[10] A sentence is a list of node ids. A corpus is the set of all sentences.

similar random walk context nodes represented closely in the subspace. For each (target, context) node pair (v_i, v_j) from random walks, Attri2Vec learns the representation $\vec{v}_i$ for the target node v_i by using it to predict the existence of context node v_j based on a three-layer neural network [33]. The representation of a node $\vec{v}_i$ in the hidden layer is then obtained by multiplying its raw content feature vector in the input layer with the input-to-hidden weight matrix W_{in}, followed by an activation function [33].

For a given large set of "positive" (target, context) node pairs generated from random walks and an equally large set of "negative" node pairs that are randomly selected from graph $\mathcal{G}$ according to a certain distribution, GraphSAGE learns a binary classifier that predicts whether arbitrary node pairs are likely to co-occur in a random walk performed on the graph [14]. The GE models we train so far take into consideration only spatial/structural information, thereby not considering temporal information. Therefore, we learn knowledge from such a dynamic graph, with the hypothesis that the anti-money laundering could be benefited from it since DGT is able to couple both spatial- and temporal information capturing simultaneously [30].

Taking into consideration the dynamic nature of the transaction graph, let node v_1^t and v_2^t be involved in a transfer at time t, where their common connections have had several transactions in the previous timestamps. Then, this temporal relation can be represented as $u_1^{t-1} - u_2^{t-1}$ and $u_1^{t-2} - u_2^{t-2}$ [30]. To extract spatial-temporal knowledge, encodings of the nodes are aggregated within a substructure node set into node embeddings. Attention layers are utilized to exchange the information of different nodes, where a single attention layer is represented as follows [30]:

$$\mathbf{H}^{(l)} = \mathrm{att}\left(\mathbf{H}^{(l-1)}\right) = \mathrm{softmax}\left(\frac{\mathbf{Q}^{(l)}\mathbf{K}^{(l)\top}}{\sqrt{d}}\right)\mathbf{V}^{(l)}, \quad (4)$$

where $\mathbf{H}^{(l)}$ and $\mathbf{H}^{(l-1)}$ is the output embedding for the l and $(l-1)^{th}$ layer, respectively; d is the dimension of node embedding, att signifies the self-attention operation; $\mathbf{Q}^{(l)}, \mathbf{K}^{(l)}, \mathbf{V}^{(l)} \in \mathbb{R}^{(\tau(\tilde{k}+2))\times d}$ are the query-, key-, and value matrices for feature transformation and information exchange, respectively that can be represented as [30]:

$$\begin{cases} \mathbf{Q}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}_Q^{(l)}, \\ \mathbf{K}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}_K^{(l)}, \\ \mathbf{V}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}_V^{(l)}, \end{cases} \quad (5)$$

where $\mathbf{W}_Q^{(l)}, \mathbf{W}_K^{(l)}, \mathbf{W}_V^{(l)} \in \mathbb{R}^{d\times d}$ are the learnable parameter matrices of the l-th attention layer. In an attention layer, $\mathbf{Q}^{(l)}$ and $\mathbf{K}^{(l)}$ calculate the contributions of different nodes' embeddings, while $\mathbf{V}^{(l)}$ projects the input into a new feature space that is combined as of eq. (4) to acquire the output embedding of each node by aggregating the information of all nodes adaptively [30].

The input of the transformer module $\mathbf{H}^{(0)}$ represents the encoding matrix of the target edge $\mathbf{X}\left(e_{\mathrm{tgt}}^t\right)$ by setting $d = d_{enc}$ to align the dimension. The output of the final attention layer $\mathbf{H}^{(L)}$ is extracted as the output node embedding matrix $\tilde{\mathbf{Z}}$ of the transformer module, where each row represents an embedding vector of a node [30].

3.4 Training of classifiers

We train decision trees (DTs) and their ensemble models such as RF, extremely randomized trees (ERT), and gradient-boosted trees (GBT) on learned embeddings. DTs exploit tree structures, where internal nodes represent feature values w.r.t boolean conditions and leaf nodes represent predicted labels. DT iteratively splits X^{*11} into multiple subsets w.r.t to threshold values of features at each node until each subset contains instances from one class only. Each branch in a DT represents a possible outcome, where the relationship between prediction $\hat{y}_i^*$ and feature $\vec{x}_i^*$ can be defined as [34]:

$$\hat{y}_i^* = f\left(x_i^*\right) = \sum_{j=1}^{N} c_j I\left\{X_i^* \in R_j\right\}, \quad (6)$$

where each sample x^* reaches exactly one leaf node, R_j is the subset of the data representing the combination of rules at each internal node, and $I\{.\}$ is an identity function [34]. In the case of tree ensemble models, the prediction function $f(\mathrm{x}^*)$ is defined as the sum of individual feature contributions plus the average contribution for the initial node in a DT for the dataset and K possible class labels that change along the prediction path after every split, along with the information which a feature caused the split [35]:

$$f(x) = c_{full} + \sum_{k=1}^{M} \sigma(x,k), \quad (7)$$

where c_{full} is the average of the entire training set X dataset (initial node), M is the total number of features.

For graph-based node classification, which is technically predicting the label of a node u at time t, we follow the usual training procedures for EvolveGCN [29] and FastGCN [15], followed by the standard GCN-like approach: the activation function of the last graph convolution layer is set to sigmoid so that h_t^u is a probability vector over two probable classes.

4 EXPERIMENTS

In this section, we report and analyse experiment results.

4.1 Datasets

We tested approach on two datasets: *AMLSim* and *Elliptic dataset*. The AMLSim is a multi-agent simulation platform tailored for an AML problem, where each agent behaves as a bank account transferring money to other agent accounts, and where a small number of agents conduct nefarious activity modelled on real-world patterns. we generate a dynamic directed transaction graph containing semi-realistic suspicious activities, based on the following information and graph generation process described in section 3.2:

- **Accounts**: contains the information about all the bank accounts whose transactions are monitored.
- **Alerts**: contains the list of transactions that triggered an alert according to AML guidelines.
- **Transactions**: contains the list of all the transactions with information about sender and receiver accounts.

---

[11] Let it represents the set of embedding vectors $\vec{V}$.

(a) Gather scatter

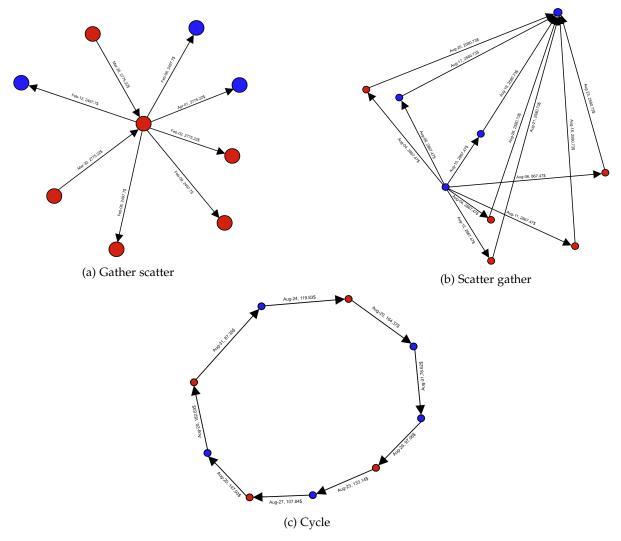
(b) Scatter gather

(c) Cycle

Fig. 3: Different (alter) types of transactions in the graph

Each node in the graph represents an account with attributes such as account number, account type, owner name, and date/time created. Nodes are designated with *cash in*, '*cash out*, *debit*, *payment*, *transfer* or *deposit* activities and are of either organization or individual types. Each edge has a transaction ID, amount, and time stamp. The data is sparsely labelled with flagged transactions[12] and SARs[13].

The Elliptic dataset [14] is a graph network of Bitcoin transactions with handcrafted features. All features are constructed using only publicly available information. This anonymized data set is a transaction graph collected from the Bitcoin blockchain. The Elliptic Dataset maps Bitcoin transactions to real entities in two categories [28]:

- **Licit**: Exchanges, wallet providers, miners, licit services.
- **Illicit**: Scams, malware, terrorist organizations, ransomware, Ponzi schemes, etc.

The graph contains 203,769 node transactions and 234,355 directed edge payments flow out of which 2% are illicit, 21% are licit. The remaining 77% samples are labelled as 'unknown' transactions. Each node has 166 features as-

sociated: the first 94 features represent local information[15] The remaining 72 features represent aggregated features that are obtained using transaction information one-hop backwards/forwards from the centre node[16]. Time steps are associated with each node, representing an estimated time when the transaction is confirmed. There are 49 distinct timesteps evenly spaced with an interval of 2 weeks.

4.2 Experiment settings

We use open source StellarGraph[17] library to compute node embeddings and are based on BiasesRandomWalk and Word2Vec from the gensim library. The DGT model is trained based on its PyTorch implementation'[18]. The RF and ERT models are based on scikit-learn's ensemble methods. As of GBT, we train its XGBoost implementation[19]. While RDF2Vec was trained using a skip-gram model by setting a window size of 5 with graph walk at depth 5 and 500 walks

---

[12] Violating both volume and velocity rules.   [13] Confirmed suspiciousness.
[14] https://www.kaggle.com/datasets/ellipticco/elliptic-data-set

[15] Timestep, number of inputs/outputs, transaction fee, output volume and aggregated figures, e.g., average BTC received/spent by inputs/outputs, the average number of incoming/outgoing transactions.   [16] Max/min standard deviation and correlation coefficients of neighbour transactions w.r.t number of inputs/outputs, transaction fee, etc.   [17] https://github.com/stellargraph/stellargraph
[18] https://github.com/yuetan031/TADDY_pytorch
[19] https://xgboost.readthedocs.io/

TABLE 1: Pipeline methods on AMLSim

Emb. model	Classifier	AUPR	F1-score	MCC
Node2Vec	ERT	0.746	0.753	0.643
	RF	0.751	0.760	0.651
	XGBoost	0.806	0.801	0.752
Attri2Vec	ERT	0.762	0.783	0.651
	RF	0.775	0.782	0.669
	XGBoost	0.815	0.810	0.7692
GraphSAGE	ERT	0.786	0.801	0.679
	RF	0.802	0.804	0.675
	XGBoost	0.815	0.816	0.701
DGT	ERT	0.797	0.803	0.671
	RF	0.813	0.825	0.693
	XGBoost	**0.833**	**0.832**	**0.715**

TABLE 2: End-to-end methods on AMLSim

Model	AUPR	F1-score	MCC
Skip-GCN	0.834 (0.792)	0.915 (0.875)	0.881 (0.763)
FastGCN	0.841 (0.804)	0.927 (0.890)	0.903 (0.781)
EvolveGCN	**0.869 (0.813)**	**0.934 (0.902)**	**0.891 (0.773)**

TABLE 3: Pipeline methods on Elliptic

Emb. model	Classifier	AUPR	F1-score	MCC
DGT	ERT	0.885	0.854	0.726
	RF	0.907	0.897	0.753
	XGBoost	0.918	0.915	0.792
GraphSAGE	ERT	0.874	0.865	0.743
	RF	0.891	0.882	0.778
	XGBoost	0.912	0.905	0.782
Attri2Vec	ERT	0.815	0.824	0.653
	RF	0.821	0.832	0.665
	XGBoost	0.902	0.894	0.673
Node2Vec	ERT	0.792	0.805	0.617
	RF	0.806	0.817	0.622
	XGBoost	0.885	0.874	0.662

per entity. To observe the efforts of embedding dimension, for each embedding model, the value of d is selected from $\{32, 64, 128, 256, 300\}$. As for the DGT model, number of layers L is selected from $\{1, 2, 3, 4, 5\}$.

As for the pipeline approach, Node2vec, Attri2Vec, GraphSAGE, and DGT models are first trained to generate node embeddings that are then used to train classifiers ERT, GBT, and RF. As of end-to-end approach, we train GCN [4] with batched training, FastGCN [15] that reduces the training cost through neighbourhood sampling, and EvolveGCN to capture the dynamism by evolving GCN parameters. Both models were configured to have 128 hidden units and trained with AdaGrad with varying learning rates and batch sizes. Further, since classes are imbalanced in the case of both datasets, we trained GCN, FastGCN, and EvolveGCN models using a *weighted cross-entropy loss* to provide higher importance to minority class (i.e., illicit/SARs samples).

For each experiment, we remove 10 to 20% of the nodes, followed by training the GE models on the reduced subgraph. During the inferencing, we generate the embeddings of the removed nodes using the trained GE model that are subsequently used to predict the labels of the nodes originally held out after re-inserting them in the network. Thus, for each experiment, 80% of the data is used for the training and validating in which the best hyperparameters were produced via random search and 5-fold cross-validation. We used the area under the precision-recall curve (AUPR), and Matthias correlation coefficient (MCC) along with the AUC and F1-scores to measure the performance of each classifier.

4.3 Analysis of node classifications

Table 1 and table 3 summarizes the results of the prediction task based on pipeline methods. As seen, the *DGT+XGBoost* combination outperformed all other combinations, covering both datasets. The ROC curves show that all the classifiers perform worse when trained on embeddings generated by the Node2Vec and Attri2Vec models. The performance of GraphSAGE+XGBoost is comparable to DGT+XGBoost.

However, a general observation is that the end-to-end methods outperformed each and every pipeline methods. In particular, EvolveGCN outperforms all the pipeline methods with tree-ensemble classifiers, indicating the effectiveness of end-to-end methods compared over pipeline methods. Further, EvolveGCN consistently outperforms both Skip-GCN and FastGCN, although the improvement is not very substantial for both datasets.

4.4 Effects of temporal information

As for the AMLSim dataset, the XGBoost model mostly benefited from the temporal information captured by the DGT embedding model. Subsequently, this pipeline method outperformed other methods of this class. This is also the case for the Elliptic dataset, where same method outperformed GraphSAGE+XGboost. These results indicate that the dynamic models are more effective. A potential reason could be that the input features are already quite informative and on top of that GNNs or transformer were able to extract more abstract features that are relevant in distinguishing licit and illicit nodes. Overall, the representation learning capability of these models from input features is clearly reflected in the classification results.

4.5 Effects of nodes embeddings + feature space

A recent approach [29] showed that aggregated information may lead higher F1 scores for anomaly detection like tasks. Inspired by this, we extended the feature space by combining node embeddings obtained from each embedding models and retrained the models. As shown in fig. 6, the node classification accuracy is slightly improves, by taking the global graph structure in this context into consideration. This somewhat becomes comparable to end-to-end methods. The micro averages for all pipeline approach is higher than 0.93. However, they are not very informative for highly imbalanced datasets. In the case of financial crime forensics, the minority illicit class is of main interest. Therefore, we plot the minority F1 scores for both datasets.

5 CONCLUSION AND OUTLOOK

In this paper, we employed semi-supervised graph learning techniques on graphs of financial transactions in order to identify nodes involved in potential money laundering. We use SARs annotations from alerts as the ground truths. We trained different GE models such as Word2vec, Node2vec, Attri2Vec, GraphSAGE, and DGT models that embed the

TABLE 4: End-to-end methods on Elliptic

Model	AUPR	F1-score	MCC
Skip-GCN	0.928 (0.793)	0.916 (0.873)	0.854 (0.763)
FastGCN	0.933 (0.805)	0.925 (0.881)	0.875 (0.781)
EvolveGCN	**0.941 (0.813)**	**0.934 (0.891)**	**0.891 (0.773)**

(a) Pipeline methods

(b) End-to-end methods

Fig. 4: Sensitivity vs. specificity for AMLSim dataset

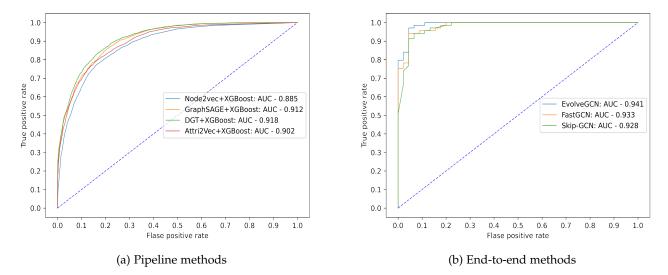(a) Pipeline methods

(b) End-to-end methods

Fig. 5: Sensitivity vs. specificity for Elliptic dataset

nodes into a lower dimensional vector space. Then, using the embeddings, we trained RF, GBT, and ERT classifiers to predict the suspiciousness of a given target node in the graph via direct or indirect connections to nodes known to be suspicious. Further, we critically reviewed existing AML methods and outlined their potential limitations.

With the rapid development of a cashless society engaged in global economic exchange, the advent of cryptocurrency has catalyzed a paradigm shift in peer-to-peer transactions and extranational financial governance. Cryptocurrencies not only impose great challenges to AML but also increase difficulty across cryptocurrency types. Another challenge lies involving temporal dynamics with the emergence/disappearance of new entities in the blockchain. For example, Weber et al. [27] have shown that at time step the market may appear to follow *Dark Market shutdown*. In such a situation, no models (including EvolveGCN or

DGT) would be able to capture such high volatility and consequently may not perform well.

AML detection mechanisms as explored in this study are not immune to attacks, while deep models may lack robustness under adversarial attacks. A transactional dataset itself might be viewed as private since the transactional dataset contains sensitive personal information and data with a high monetary value for the organisation possessing it. Thus, any form of privacy attack that could potentially disclose privacy-sensitive information could be very costly. Moreover, an adversary might craft[20] a series of specific transactions to fool the AML detection system to classify an illicit transaction to be licit. Such adversarial attacks are real security threats for any financial systems [36]. On the other hand, since the computed embeddings might need

---

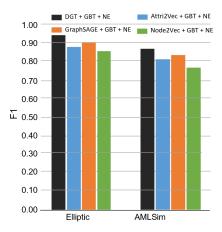[20] e.g., by adding noise or minor perturbations in the embedding space.

Fig. 6: Effects of combining node embeddings with features

to be shared, which an attacker could apply techniques to reverse[21] to original input or link back to a natural person. Further, a deployed AML detection model could be directly attacked too. In particular, membership inference attacks [37] can be introduced in order to disclose whether a particular training record was used to train the model.

In future studies, we will expand on this research and apply realistic attacks on AML detection algorithms in order to assess whether sensitive information can be disclosed or fool the model to hide illicit transactions. Based on that, we will also explore defence strategies to mitigate these attacks. One popular defence strategy against privacy attacks is differential privacy [38]. Our main results learn from graphs which are totally available, such as graphs of banking transactions in blockchains where these are public. However, there are also many real-life scenario in classical banking where money-launderers take advantage of banks by not disclosing private transactions to split the laundered amounts into intermediates across different banks to go undetected (cf. the model of[5]), and we would also like to provide tools to grant financial authorities the ability to detect such patterns with these privacy constraints, as a complementary approach to the more efficient learning methods on public graphs presented in previous sections. Work on this topic has already been done in literature [39], to give a cryptographic functional encryption scheme to allow [5]'s similarity calculations to be performed by financial authorities on data from concurrent banks. However, size of embedding vectors on which the similarity scores are calculated are linear w.r.t number of existing bank accounts, which would be huge in a practical use-case. Using graph embedding techniques the way we have in previous sections could drastically improve these sizes.

One of the challenges in this context is for banks to provide embeddings of each of their nodes (the bank accounts), from the graph of transactions, knowing only the subgraph of the transactions made to and from the accounts they control. Because of this, we will perform an embedding depending only on nodes' first neighbors, those they are directly having transactions with. This is also relevant with respect to the corresponding first topological pattern in fig. 1, as we are considering intermediate nodes split across

different banks but having the same direct neighbors, and this is the sole criteria upon which we would want to have similar node embeddings. Hence, we use a variant of fast random projection embeddings that we describe with the following algorithm steps, to get the incoming (resp. outgoing) transaction embeddings:

1) Each node n in the subgraph and its direct neighbors gets an initial random embedding vector value $e_{n,0}$ from a public hash function on the node, instantiated with a public seed (and with no null outputs).
2) For each node n in the subgraph, denoting N the set of its direct neighbors, and $w_{i \to j}$ the amounts sent from account i to account j (i.e. the weight on the oriented edge from i to j, for any nodes i, j, and setting that weight to 0 when the nodes are not connected), n get a new embedding vector: $e_{n,1} := \sum_{m \in N} e_{m,0} \cdot w_{m \to n}$ (and respectively, for the outgoing transaction embeddings: $e_{n,1} := \sum_{m \in N} e_{m,0} \cdot w_{n \to m}$).
3) The vectors for the previous step are then normalized; each node n in the subgraph gets the embedding vector: $e_{n,3} := \frac{e_{n,2}}{||e_{n,2}||_2}$, where $||.||_2$ denotes the euclidian norm. This vector is then returned as the embedding vector $e_{n,\text{in}}$ (resp. $e_{n,\text{out}}$ for outgoing transactions).

We then apply the inner-products and similarity calculations similar to literature [5], [39] on these embedding vectors; to compare nodes n from bank 1 and m from bank 2, we compute: $\sigma(n,m) = \langle e_{n,\text{in}}; e_{m,\text{in}} \rangle \times \langle e_{n,\text{out}}; e_{m,\text{out}} \rangle$, and deem that nodes with high similarity values should be labeled as suspicious accounts which might be part of the same money-laundering network, having the bulk of their transactions from and to the same neighbors.

ACKNOWLEDGMENT

REFERENCES

[1] R. Frumerie, "Money laundering detection using tree boosting and graph learning algorithms," 2021.
[2] U. Economic and S. Council, "Tax abuse, money laundering and corruption plague global finance," 2020.
[3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.
[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[5] R. Soltani, U. T. Nguyen, Y. Yang, M. Faghani, A. Yagoub, and A. An, "A new algorithm for money laundering detection based on structural similarity," in *2016 IEEE 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 2016, pp. 1–7.
[6] 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*, vol. 34, no. 04, 2020, pp. 4731–4738.
[7] Q. Rajput, N. S. Khan, A. Larik, and S. Haider, "Ontology based expert-system for suspicious transactions detection," *Computer and Information Science*, vol. 7, no. 1, p. 103, 2014.
[8] K. Michalak and J. Korczak, "Graph mining approach to suspicious transaction detection," in *2011 Federated conference on computer science and information systems (FedCSIS)*. IEEE, 2011, pp. 69–75.

---

[21] e.g., by employing a sophisticated approach for input reconstruct.

[9] J. Tang and J. Yin, "Developing an intelligent data discriminating system of anti-money laundering based on svm," in *2005 International conference on machine learning and cybernetics*, vol. 6. IEEE, 2005, pp. 3453–3457.

[10] A. Awasthi, "Clustering algorithms for anti-money laundering using graph theory and social network analysis," *Universitat Autònoma de Barcelona*, 2012.

[11] N. A. Le Khac and M.-T. Kechadi, "Application of data mining for anti-money laundering detection: A case study," in *2010 IEEE international conference on data mining workshops*. IEEE, 2010, pp. 577–584.

[12] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos, "Fraudar: Bounding graph fraud in the face of camouflage," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 895–904.

[13] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier *et al.*, "Knowledge graphs," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–37, 2021.

[14] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[15] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

[16] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, pp. 1–23, 2019.

[17] Y. Dai, S. Wang, N. N. Xiong, and W. Guo, "A survey on knowledge graph embedding: Approaches, applications and benchmarks," *Electronics*, vol. 9, no. 5, p. 750, 2020.

[18] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, pp. 2181–2187.

[19] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, "Knowledge graph embedding via dynamic mapping matrix," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 687–696.

[20] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2787–2795.

[21] M. R. Karim, M. Cochez, J. B. Jares, M. Uddin, O. Beyan, and S. Decker, "Drug-drug interaction prediction based on knowledge graph embeddings and convolutional-lstm network," in *Proceedings of the 10th ACM international conference on bioinformatics, computational biology and health informatics*, 2019, pp. 113–123.

[22] J. Feng, M. Huang, M. Wang, M. Zhou, Y. Hao, and X. Zhu, "Knowledge graph embedding by flexible translation," in *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*, ser. KR'16. AAAI Press, 2016, pp. 557–560.

[23] P. Ristoski and H. Paulheim, "Rdf2vec: Rdf graph embeddings for data mining," in *The Semantic Web – ISWC 2016*, P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck, and Y. Gil, Eds. Cham: Springer International Publishing, 2016, pp. 498–514.

[24] S. M. Kazemi and D. Poole, "Simple embedding for link prediction in knowledge graphs," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 4284–4295.

[25] M. Cochez, P. Ristoski, and H. Paulheim, "Global RDF vector space embeddings," in *The Semantic Web – ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017*, C. d'Amato, M. Fernandez *et al.*, Eds. Cham: Springer International Publishing, 2017, pp. 190–207.

[26] W. Zhang, B. Paudel, W. Zhang, A. Bernstein, and H. Chen, "Interaction embeddings for prediction and explanation in knowledge graphs," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, ser. WSDM '19. New York, NY, USA: ACM, 2019, pp. 96–104.

[27] M. Weber, J. Chen, T. Suzumura, A. Pareja, T. Ma, H. Kanezashi, T. Kaler, C. E. Leiserson, and T. B. Schardl, "Scalable graph learning for anti-money laundering: A first look," *arXiv preprint arXiv:1812.00076*, 2018.

[28] 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," *arXiv preprint arXiv:1908.02591*, 2019.

[29] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5363–5370.

[30] Y. Liu, S. Pan, Y. G. Wang, F. Xiong, L. Wang, Q. Chen, and V. C. Lee, "Anomaly detection in dynamic graphs via transformer," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[31] M. Cochez, P. Ristoski, S. P. Ponzetto, and H. Paulheim, "Biased graph walks for RDF graph embeddings," in *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics*, ser. WIMS '17. New York, NY, USA: ACM, 2017, pp. 21:1–21:12.

[32] T. Mikolov, K. Chen, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[33] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Attributed network embedding via subspace discovery," *Data Mining and Knowledge Discovery*, vol. 33, no. 6, pp. 1953–1980, 2019.

[34] F. Di Castro and E. Bertini, "Surrogate decision tree visualization." in *IUI Workshops*, 2019.

[35] F. Al-Obeidat, Á. Rocha, M. Akram, S. Razzaq, and F. Maqbool, "(cdrgi)-cancer detection through relevant genes identification," *Neural Computing and Applications*, pp. 1–8, 2021.

[36] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, "A survey of deep learning methods for cyber security," *Information*, vol. 10, no. 4, p. 122, 2019.

[37] H. Hu, Z. Salcic, L. Sun, G. Dobbie, P. S. Yu, and X. Zhang, "Membership inference attacks on machine learning: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–37, 2022.

[38] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[39] P. de Perthuis and D. Pointcheval, "Two-client inner-product functional encryption with an application to money-laundering detection," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 725–737. [Online]. Available: https://doi.org/10.1145/3548606.3559374