Identifying Money Laundering Subgraphs on the Blockchain

Kiwhan Song*
kiwhan@mit.edu
MIT, MIT-IBM Watson AI Lab
Cambridge, MA, USA

Mohamed Ali Dhraief*
mohamed.ali.dhraief@ibm.com
IBM
Zurich, Switzerland

Muhua Xu
muhuaxu@mit.edu
MIT, MIT-IBM Watson AI Lab
Cambridge, MA, USA

Locke Cai
lcai12@mit.edu
MIT, MIT-IBM Watson AI Lab
Cambridge, MA, USA

Xuhao Chen
cxh@mit.edu
MIT, MIT-IBM Watson AI Lab
Cambridge, MA, USA

Arvind
arvind@csail.mit.edu
MIT, MIT-IBM Watson AI Lab
Cambridge, MA, USA

Jie Chen
chenjie@us.ibm.com
IBM Research, MIT-IBM Watson AI
Lab
Cambridge, MA, USA

Abstract

Anti-Money Laundering (AML) involves the identification of money laundering crimes in financial activities, such as cryptocurrency transactions. Recent studies advanced AML through the lens of graph-based machine learning, modeling the web of financial transactions as a graph and developing graph methods to identify suspicious activities. For instance, a recent effort on opensourcing datasets and benchmarks, *Elliptic2*, treats a set of Bitcoin addresses, considered to be controlled by the same entity, as a graph node and transactions among entities as graph edges. This modeling reveals the "shape" of a money laundering scheme—a subgraph on the blockchain, such as a peeling chain or a nested service. Despite the attractive subgraph classification results benchmarked by the paper, competitive methods remain expensive to apply due to the massive size of the graph; moreover, existing methods require candidate subgraphs as inputs which may not be available in practice.

In this work, we introduce *RevTrack*, a graph-based framework that enables large-scale AML analysis with a lower cost and a higher accuracy. The key idea is to track the initial senders and the final receivers of funds; these entities offer a strong indication of the nature (licit vs. suspicious) of their respective subgraph. Based on this framework, we propose *RevClassify*, which is a neural network model for subgraph classification. Additionally, we address the practical problem where subgraph candidates are not given, by proposing *RevFilter*. This method identifies new suspicious subgraphs by iteratively filtering licit transactions, using *RevClassify*. Benchmarking these methods on *Elliptic2*, a new standard for AML, we show that *RevClassify* outperforms state-of-the-art subgraph

classification techniques in both cost and accuracy. Furthermore, we demonstrate the effectiveness of *RevFilter* in discovering new suspicious subgraphs, confirming its utility for practical AML.

CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Applied computing** → **Network forensics**.

Keywords

Artificial Intelligence, Machine Learning, Graph Neural Networks, Subgraph Representation Learning, Financial Forensics, Cryptocurrency, Anti-Money Laundering

1 Introduction

Money laundering, a financial crime, supports a range of destructive activities including human trafficking, drug trafficking, and terrorism. Over 70% of criminal networks use money laundering to fund their activities and conceal their assets [7]. This demonstrates not only the extensive reach of money laundering across various crimes but also the profound consequence, with estimated $3.1 trillion in illicit funds flowed through the global financial system in 2023 [16]. The rise of cryptocurrencies has accelerated this issue, with the number of victims and volume of dollar lost to crypto-frauds doubling [17]. Worse, the pesudo-anonymous nature of the technology offers the illegal activities an additional layer of protection. Combating financial crimes today is crucial and is considered a matter of national security [18].

Driven by the need for better tools for anti-money laundering (AML), the Elliptic dataset (henceforth referred to as *Elliptic1*) was released in 2019, as the largest, publicly accessible, labeled AML/cryptocurrency dataset at the time [23]. *Elliptic1* is a graph,

---

*Both authors contributed equally to this research.

Figure 1: Terminology and examples of money laundering schemes.

comprising 204K crypto transactions as graph nodes and 234K payment flows (the flows of every Bitcoin from one transaction to the next) as directed edges. The machine learning task is node classification—classifying the licit vs. illicit nature of each transaction. While *Elliptic1* gained significant traction in both the machine learning and the AML communities, the task it supports is limited to analyzing individual transactions, leaving the question of studying more complex money laundering schemes unaddressed.

Money laundering schemes involve a series of financial transactions that transform illegally obtained funds to apparently legal accounts. In 2024, *Elliptic2* was released in response to the need for effective tools to unveil such complex patterns on the cryptocurrency blockchain [4]. Besides being a graph that is nearly three orders of magnitude larger than *Elliptic1*, *Elliptic2* models the transaction graph differently and features a different machine learning task—subgraph classification. Specifically, each node represents a financial entity on the blockchain and each edge aggregates transactions between a pair of entities. Thus, a money laundering scheme signifies a subgraph consisting of a series of transactions among multiple entities, likely from criminal to legal.

Representing money laundering schemes as subgraphs allows one to leverage the rapid progress of the field of graph-based machine learning [24, 28] to develop effective AML tools. In particular, subgraph classification is an emerging topic, which extends the thoroughly studied tasks—node classification, edge prediction, and graph classification—to yet another granularity of graphs—subgraphs. Several subgraph neural network methods [1, 2, 22] appear to be good candidates.

However, there are multiple challenges in subgraph AML. First, crypto transaction graphs are massive. The cumulated number of transactions on the blockchain exceeded one billion on May 05, 2024 [5]. Even with address and transaction aggregation, *Elliptic2* includes nearly 50M nodes and 200M edges. Such a large scale costs effective subgraph models, such as GLASS [22], a few days to train without GPUs, while requiring nonstraightforward system engineering efforts to port the training on GPUs [4]. Second, there are

exponentially many subgraphs in a graph; identifying suspicious subgraphs corresponding to money laundering is like finding needles in a haystack. A typical classification method can classify a reasonable amount of instances, but it becomes impractical when the amount is exponential.

In this work, we develop a framework, coined *RevTrack*, which allows efficient classification and discovery of suspicious money laundering subgraphs. A key to this framework is to track the sending and receiving entities of the subgraph rather than the subgraph itself. Doing so allows the use of alternative neural networks (other than graph neural networks) that are easier to train and scale better. Based on this framework, we propose a *RevClassify* method for subgraph classification (classifying a given subgraph) and a *RevFilter* method for identifying potential criminal entities and their money laundering activities (discovering new, suspicious subgraphs). Benchmarking these methods on the *Elliptic2* dataset, we demonstrate their superior performance over strong baselines and their practical utility in AML.

2 Subgraph Representations of Money Laundering

A majority of the participants on the blockchain are "licit" entities; they range from exchanges, wallet providers, miners, to licit services. On the other hand, an "illicit" entity is commonly associated with crimes, such as dark markets, scammers, and hackers. A fundamental assumption of money laundering is that a path connecting an illicit entity to a licit one without a change of ownership of the funds likely represents money laundering by a criminal person or organization. Through layers of transactions (laundering), criminals deposit funds at a legitimate service and evade detection of the illegal source of the funds.

A money laundering scheme may consist of one or multiple such illicit→licit paths; the union of these paths is a subgraph. A known scheme is a "peeling chain" (see the middle illustration of Figure 1), where all the intermediate entities on the path additionally point to

the end of the path. In this case, the ending entity could be an exchange, to which all intermediate entities deposit part of the funds (with the rest sent to the next entity). Another example is a "nested service" (see also Figure 1), where multiple paths starting from separate illicit entities merge on the same "service" entity, which further points to an exchange that is licit. Such services typically have less stringent customer due diligence checks than the exchanges, resulting in their (mis)use for cryptocurrency laundering.

2.1 Terminology and Notation

We illustrate in Figure 1 the main concepts used throughout the paper. The blockchain is modeled as a directed graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, where a node $v \in \mathbb{V}$ is a set of Bitcoin addresses thought to be controlled by a single entity (e.g., a person or organization) and a directed edge $e = (u, v) \in \mathbb{E}$ denotes one or multiple transactions from entity u to v. We call $\mathcal{G}$ the *background graph* when its subgraphs are of concern.

Denote by $\mathcal{H} = (\mathbb{V}_{\mathcal{H}}, \mathbb{E}_{\mathcal{H}})$ a subgraph of $\mathcal{G}$. Each node $v \in \mathbb{V}_{\mathcal{H}}$ with a zero in-degree inside $\mathcal{H}$ is called a *source* and all sources form the set $\mathbb{V}_{source}$. Similarly, each node with a zero out-degree inside $\mathcal{H}$ is called a *sink* and all sinks form the set $\mathbb{V}_{sink}$. A node pointing to any source is called a *sender* and all senders form the set $\mathbb{S}$. Similarly, a node that is pointed to by any sink is called a *receiver* and all receivers form the set $\mathbb{R}$. Note that by definition, $\mathbb{S}$ and $\mathbb{R}$ are outside the subgraph $\mathcal{H}$.

The *Elliptic2* dataset constructs subgraphs for classification with the help of node labeling that is undisclosed. A node is (manually) labeled as *licit*, *illicit*, or in most of the cases, unlabeled, in which case we call it *unknown*. Subgraphs in *Elliptic2* are labeled either *licit* or *suspicious*. Based on the construction procedure described in [4], one can infer that a subgraph in *Elliptic2* is licit if the senders and receivers are all licit, while a subgraph is suspicious if the receivers are licit but the senders are illicit. Suspicious subgraphs are intended to be validated by human analysts to confirm their illicit nature (money laundering).

Note that not every subgraph can be labeled; and illicit subgraphs are scarce among those labeled. It is tempting to predict the subgraph labels based on the labels of the senders and receivers. However, node labels are not provided and reverse engineering them based on the subgraph construction procedure will reveal the labels of only a small fraction of the nodes.

2.2 Two Tasks of Interest

In this work, we are interested in two tasks:

(1) Classify the nature (licit vs. suspicious) of a given subgraph $\mathcal{H}$;
(2) Identify new suspicious subgraphs.

Task (1) is a standard classification problem: Given a set of subgraphs with labels, split it into training/validation/test subsets. Train a model to predict the label of each subgraph by using the training and validation sets and evaluate it on the test set. This task solicits effective machine learning models that can discern the nature of any given subgraph.

The ultimate goal of AML, on the other hand, is to discover all money laundering schemes. Task (1) is infeasible for this goal, for two reasons. First, there are exponentially many (specifically, $2^{|\mathbb{V}|}$) subgraphs; it is impossible to enumerate them and classify one by

Figure 2: Graphlet distribution for licit subgraphs and that for suspicious subgraphs.

one. Second, not every subgraph can be logically labeled; in fact, only a few can be considered licit and even few are suspicious.

Hence, Task (2) is an alternative solution to this goal: Given a set of labeled subgraphs, design a method that can discover new subgraphs that are likely suspicious (among the exponentially many candidates). This task is highly nontrivial. We propose a method by reusing the learned model from Task (1) to accomplish the task.

3 Identifying Money Laundering Subgraphs with Senders and Receivers

Subgraph neural networks are costly. A straightforward approach to telling if a subgraph is a suspicious money laundering scheme is to perform subgraph classification. Several representative neural networks for this approach were explored in [4], suggesting that the effective ones for our use case are quite expensive to train. For example, GLASS [22], using as few as two layers, requires several days to train by using CPUs. This is because GPU training is a complex matter for a background graph as large as *Elliptic2* (∼50M nodes and ∼200M edges) due to the massive memory consumption and it requires nonstraightforward engineering efforts to adapt the node classification workload [12, 13] to subgraph classification workload [4]. While GPU training is the standard for usual neural networks, for graph neural networks it requires special handling because the loss of a data point (i.e., a node) requires the information of not just the data itself but also its neighborhood, which causes debates regarding whether full-graph training or mini-batch training should be used. Moreover, if one uses mini-batch training, adapting batching and neighborhood sampling to subgraphs requires reengineering existing libraries or codebases for effective memory usage [12, 13].

The subgraph structure alone is insufficient for identification. A reason why GLASS, a method that trains a neural network on the entire background graph, performs more effective classification than do methods that train a neural network on individual subgraphs only (e.g., Sub2Vec [1]), is that the internal subgraph structure alone is insufficient for classification. What matters additionally is the border information surrounding but outside the

subgraph. To further illustrate this point, we compute the distribution of graphlets (including 2-node, 3-node, and 4-node graphlets) inside a collection of subgraphs. Figure 2 shows that the distribution for licit subgraphs and that for suspicious ones are rather similar. Although this example is simplified for computational purpose (e.g., ignoring edge directions and larger graphlets), more evidences from the subsequent experiment section confirm that one should look beyond the internal structures for identifying suspicious subgraphs.

Senders and receivers of funds provide a strong hint. A logically useful piece of border information is the senders and receivers, because field wisdom suggests that illicit senders launder money through transferring funds layer by layer to licit accounts. Hence, in the next section, we propose methods to identify money laundering subgraphs with a focal use of sender and receiver information. These entities can be easily extracted by using the definition introduced in Section 2.1 (see also the example in Figure 1): For each subgraph, we first identify nodes where transactions begin and end, namely, *sources* and *sinks*. Then, the nodes outside the subgraph pointing to the sources are *senders* and those being pointed to by the sinks are *receivers*. Note that a subgraph occasionally is not acyclic (e.g., two entities may send funds to each other at different times), which leads to no sources or sinks. In this case, we remove all detected cycles to extract sources and sinks.

4 *RevTrack*: A Tale of Two Methods

We are now ready to introduce *RevTrack*, a framework that analyzes and discovers money laundering subgraphs through tracking senders and receivers. The framework represents a subgraph $\mathcal{H}$ by its sender set $\mathbb{S}$ and its receiver set $\mathbb{R}$. We create "links" between them; these links can be interpreted as an abstraction of the paths between $\mathbb{S}$ and $\mathbb{R}$. *RevTrack* includes two methods: *RevClassify* classifies the nature (licit vs. suspicious) of a given subgraph, while *RevFilter* discovers new suspicious subgraphs through iteratively filtering out deemed licit links. Figure 3 illustrates both methods.

4.1 *RevClassify*: Subgraph Classification

RevClassify classifies a given subgraph $\mathcal{H}$. Given the $(\mathbb{S}, \mathbb{R})$ representation of $\mathcal{H}$, the method generates an embedding vector $\mathbf{h}_{\mathcal{H}}$ for the subgraph, such that a logistic regression of $\mathbf{h}_{\mathcal{H}}$ performs the classification. *RevClassify* requires an expressive architecture to handle the two sets as inputs. Here, we name two architectures, *RevClassify$_{BP}$* and *RevClassify$_{DS}$*, both of which are permutation-invariant with respect to $\mathbb{S}$ and $\mathbb{R}$.

RevClassify$_{BP}$ This architecture builds a fully-connected, directed bipartite graph $\widetilde{\mathcal{H}}$ between $\mathbb{S}$ and $\mathbb{R}$. That is,

$$\widetilde{\mathcal{H}} = (\widetilde{\mathbb{V}}_{\mathcal{H}}, \widetilde{\mathbb{E}}_{\mathcal{H}}), \tag{1}$$

where $\widetilde{\mathbb{V}}_{\mathcal{H}} = \mathbb{S} \cup \mathbb{R}$ and $\widetilde{\mathbb{E}}_{\mathcal{H}} = \{(u,v) \mid u \in \mathbb{S} \land v \in \mathbb{R}\}$. Then, the embedding $\mathbf{h}_{\mathcal{H}}$ is computed by using any Message Passing Neural Network (MPNN) [15, 20, 25], followed by a global pooling layer as the readout:

$$\mathbf{h}_{\mathcal{H}} = \text{READOUT}\left(\text{MPNN}(\widetilde{\mathcal{H}})\right). \tag{2}$$

This architecture effectively runs a graph neural network on an alternative but augmented version of the subgraph, $\widetilde{\mathcal{H}}$, to classify $\mathcal{H}$. It considers information outside $\mathcal{H}$ and is more effective.

RevClassify$_{DS}$ This architecture does not use any graph neural network. Instead, it processes $\mathbb{S}$ and $\mathbb{R}$ separately by using Deep Sets [27], a universal architecture that generates set representations preserving permutation invariance and equivariance:

$$\mathbf{h}_{\mathbb{S}} = \text{DeepSets}(\mathbb{S}) \quad \mathbf{h}_{\mathbb{R}} = \text{DeepSets}(\mathbb{R}). \tag{3}$$

This is suitable for our setting as there is no preference among the senders or receivers. Then, $\mathbf{h}_{\mathbb{S}}$ and $\mathbf{h}_{\mathbb{R}}$ are concatenated and a Multi-Layer Perceptron (MLP) is used to yield the final embedding:

$$\mathbf{h}_{\mathcal{H}} = \text{MLP}\left(\text{CONCAT}(\mathbf{h}_{\mathbb{S}}, \mathbf{h}_{\mathbb{R}})\right). \tag{4}$$

Both *RevClassify$_{BP}$* and *RevClassify$_{DS}$* can learn to classify $\mathcal{H}$ effectively when trained with the binary cross-entropy loss.

4.2 *RevFilter*: Discovering New Suspicious Subgraphs

Although *RevClassify* can accurately classify subgraphs, it assumes that the subgraphs of interest are provided. However, there are in total $2^{|\mathbb{V}|}$ subgraphs and only a tiny fraction of them correspond to money laundering. Hence, it is impractical to enumerate all subgraphs and apply *RevClassify* on them one by one. Rather, there is a strong desire for a recommendation-like system that can efficiently discover potential money laundering without exhaustive search.

To this end, we propose *RevFilter*, an efficient and scalable method that recommends suspicious subgraphs. The method works by iteratively filtering out groups of senders and receivers that are deemed to not contribute to money laundering. Specifically, given an arbitrary set of senders, $\mathbb{S}$, and an arbitrary set of receivers, $\mathbb{R}$, *RevFilter* produces a list of top-k ranked (s, r) pairs, where $s \in \mathbb{S}$, $r \in \mathbb{R}$, and each (s, r) pair is deemed suspicious (i.e., the path connecting s and r is a suspicious subgraph). Remarkably, *RevFilter* leverages a pretrained *RevClassify* classifier, thus eliminating the need for any training or optimization.

Algorithm 1 describes the details. *RevFilter* maintains a list L of sender-set-and-receiver-set pairs. Initially, there is only one pair, $(\mathbb{S}, \mathbb{R})$. In each iteration, we split each pair in the list into four pairs, by bisecting the sender set and the receiver set. Then, all pairs are passed into the pretrained classifier C, which assigns a probability score indicating the likelihood of money laundering between the senders and the receivers. Based on the score, only the top-k ranked pairs are retained for the next iteration. This iterative filtering process is repeated until the size of each pair is reduced to 1-1, suggesting top-k most likely money laundering paths.

This basic version of *RevFilter* can be made more robust with the following enhancements, when suspicious subgraphs are scarce.

(1) Fine-tuning with data augmentation. Note that the pairs passed into the pretrained classifier C have a large sender/receiver set at the beginning. However, this classifier is mostly trained with small pairs, due to the nature of the *Elliptic2* dataset. Thus, to enhance the performance of C, we fine-tune it by using *merged* pairs. These pairs are created by randomly merging sets of senders and receivers, such that the number of sets merged, n_{merge}, follows the exponential distribution $P(n_{merge} = t) = \gamma e^{-\gamma \cdot t}$. Such data augmentation generates a fine-tuning dataset where the size distribution of the pairs is similar to that encountered in the execution of *RevFilter*.

Figure 3: Illustration of the *RevClassify* and *RevFilter* methods.

Algorithm 1 *RevFilter*: Discovering New Suspicious Subgraphs

---

1: **Input:** A pair of sender set and receiver set, $(\mathbb{S}, \mathbb{R})$; Number of pairs to recommend, k; Pretrained classifier C
2: **Output:** Top-k ordered list of suspicious sender-set-and-receiver-set pairs, L
3: $L \leftarrow [(\mathbb{S}, \mathbb{R})]$
4: **repeat**
5:     **for** $(\mathbb{S}_i, \mathbb{R}_i) \in L$ **do**
6:         $\mathbb{S}_i \rightarrow \mathbb{S}_i^1 + \mathbb{S}_i^2, \mathbb{R}_i \rightarrow \mathbb{R}_i^1 + \mathbb{R}_i^2$ ▷ split senders/receivers into two equal-sized groups
7:         $L \leftarrow L - [(\mathbb{S}_i, \mathbb{R}_i)] + \left[(\mathbb{S}_i^1, \mathbb{R}_i^1), (\mathbb{S}_i^1, \mathbb{R}_i^2), (\mathbb{S}_i^2, \mathbb{R}_i^1), (\mathbb{S}_i^2, \mathbb{R}_i^2)\right]$ ▷ replace the pair $(\mathbb{S}_i, \mathbb{R}_i)$ with four new pairs
8:     **end for**
9:     **if** $|L| > k$ **then**
10:         scores $s \leftarrow C(L)$ then REVERSE-SORT(L, key $= s$)   ▷ sort the list L by the scores computed by the classifier C
11:         $L \leftarrow L[:k]$ ▷ keep only the top-k scoring pairs
12:     **end if**
13: **until** $|L| = k$ and $\forall (\mathbb{S}_i, \mathbb{R}_i) \in L, |\mathbb{S}_i| = |\mathbb{R}_i| = 1$ ▷ repeat until all k pairs are 1-1
14: **return** L

---

(2) Keeping more candidates. If the classifier C mistakenly assigns a low score to a pair that contains suspicious money laundering activities during iterations, the suspicious activities are erroneously ignored. Even when the classifier is highly accurate, this could occur through many iterations. Thus, for the initial iteration, we maintain $\alpha_{keep} \times k$ pairs ($\alpha_{keep} > 1$) and gradually decrease this number to k by the end of the iterations, to mitigate accidentally eliminating suspicious pairs. Note that using a large α_{keep} may lead to a trade-off: while it helps preserve potential candidates, it can also result in a decrease in inference speed and a flattening of the iterative filtering process.

Practical Use of RevFilter Algorithm 1 requires an initial sender set $\mathbb{S}$ and a receiver set $\mathbb{R}$ as input. One practical use of *RevFilter* is to identify additional illicit entities given known money laundering subgraphs. The end of these subgraphs are linked to licit entities (such as an exchange). Hence, using these entities to form $\mathbb{R}$ potentially identify new money laundering schemes sharing the same choice of crypto deposits. Additionally, one may partition the node set $\mathbb{V}$ such that one partition is used to form $\mathbb{S}$ each time, balancing the relative sizes of $\mathbb{S}$ and $\mathbb{R}$ while speeding up the inference.

5 Experiments: *RevClassify*

We comprehensively evaluate the effectiveness of the proposed methods—*RevClassify* in this section and *RevFilter* in the next—with the *Elliptic2* dataset [4]. *Elliptic2* was a recently released benchmark, the largest of its kind, which innovatively models AML as a subgraph classification problem. In the dataset, the background graph contains 49,299,864 nodes and 196,215,606 edges. Additionally, there are 121,810 labeled subgraphs, among which 119,092 are licit and 2,718 are suspicious.

5.1 Experiment Setup

Following [4], we split the labeled subgraphs into training, validation, and test sets randomly (80:10:10). In addition to the **full-shot** setting, which uses the entire training set for training, we also explore the **few-shot** setting, where only a fraction of the training set is used, to investigate model behaviors in data-scarce environments. For a given fraction p, we randomly sample p of the suspicious subgraphs as well as licit subgraphs.

5.1.1 Baselines. We compare *RevClassify* with four typical subgraph classification methods. **Sub2Vec** [1] is an early graph embedding method, which samples random walks within a subgraph

Table 1: Accuracy and cost comparisons between *RevClassify* and baselines. "Full-shot" uses the whole training set for training while "few-shot" uses a portion of it. The best results are boldfaced and the second-best are underlined.

Method	Few-shot						Full-shot		Cost			
	3%		10%		30%				Time (min)			Memory (GB)
	PR-AUC ↑	F1 ↑	PR-AUC ↑	F1 ↑	PR-AUC ↑	F1 ↑	PR-AUC ↑	F1 ↑	Preprocess ↓	Train ↓	Inference ↓	Train ↓
Sub2Vec	0.024	0.045	0.024	0.043	0.024	0.043	0.025	0.044	105.4	**2.1**	0.037	2.00
GNN-SEG	0.098	0.109	0.170	0.142	0.194	0.153	0.413	0.206	**7.5**	75.4	1.740	1.81
GNN-PLAIN	0.158	0.105	0.327	0.118	0.440	0.135	0.789	0.660	N/A	25.7	0.069	5.80
GLASS	0.164	0.107	0.350	0.134	0.477	0.149	0.816	0.705	N/A	56.1	0.070	6.23
RevClassify$_{BP}$	<u>0.235</u>	<u>0.140</u>	<u>0.372</u>	<u>0.157</u>	<u>0.599</u>	<u>0.319</u>	<u>0.972</u>	**0.954**	<u>40.4</u>	5.4	<u>0.006</u>	**0.49**
RevClassify$_{DS}$	**0.445**	**0.287**	**0.665**	**0.363**	**0.802**	**0.614**	**0.974**	<u>0.953</u>		<u>3.5</u>	**0.005**	<u>0.51</u>

and uses Paragraph2Vec to learn subgraph embeddings from the sampled walks. **GNN-SEG** is an MPNN that also acts on only the internal structure of the subgraph. **GNN-PLAIN** is another MPNN, but the message passing is conducted on the background graph, such that the subgraph representation draws information additionally outside the subgraph. **GLASS** [22] extends GNN-PLAIN by employing a zero-one labeling trick that distinguishes the nodes inside and outside the subgraph; this method is theoretically proved to be more expressive than GNN-PLAIN.

5.1.2 Evaluation Metrics. Due to the label imbalance and the importance of the suspicious class, we use the binary F1-score (treating suspicious as positive) and the PR-AUC score to evaluate model performance. Additionally, we compare the resource demands of each method, including time and memory usage.

5.1.3 Implementation Details. RevClassify$_{BP}$ employs GIN [25] as the MPNN backbone. *RevClassify$_{DS}$* uses MLP for the invariant layers in Deep Sets [27]. GLASS and Sub2Vec are implemented from the codebases provided by the original authors. GNN-SEG and GNN-PLAIN use the same MPNN backbone as does GLASS. We train all models to minimize the binary cross entropy loss using the Adam optimizer [14] for 1000 epochs for the baseline methods and 150 epochs for *RevClassify*, incorporating early stopping. We tune hyperparameters, including the number of layers, hidden dimensions, type of pooling, learning rate, dropout, and batch size.

The performance of the baselines Sub2Vec, GNN-SEG, and GLASS is (sometimes substantially) improved over that reported by [4]. The results in [4] were obtained by CPU training and ignoring node features, but our results are obtained by GPU training and leveraging node features, whenever possible. Additionally, for GLASS and GNN-PLAIN, we use only one MPNN layer with neighborhood sampling, which significantly reduces the training time and the memory requirement. Moreover, we implement data preprocessing to enable faster data loading and replace the memory-intensive GraphNorm layer [6] with the more light-weight LayerNorm [3]. These modifications boost the speed and accuracy of the baselines, making them stronger competitors. Our experiments are conducted by using a single V100 GPU with 16GB VRAM.

5.2 Results

5.2.1 Accuracy. The results are summarized in Table 1. *RevClassify* outperforms all baselines across the board. Among the baselines, Sub2Vec performs the most poorly, because node features are not used. GNN-PLAIN and GLASS perform significantly better than GNN-SEG, confirming the importance of involving external node information beyond the internal subgraph structure. Our method further significantly improves over GNN-PLAIN and GLASS. Between the two architectures of our method, *RevClassify$_{DS}$* is robustly stronger in the few-shot settings.

5.2.2 Cost. As shown by Table 1, *RevClassify* is considerably more memory-efficient and much faster in inference than the other methods. This efficiency stems from the use of only sender/receiver nodes, whereas GNN-SEG computes with all nodes in the subgraph, and Sub2Vec, GLASS, and GNN-PLAIN involve the background graph. GNN-SEG, GNN-PLAIN, and GLASS are slow to train; on the other hand, Sub2Vec is fast, but it requires extensive preprocessing. The preprocessing of *RevClassify* is not as demanding as Sub2Vec. Moreover, processed senders and receivers are stored in a hash table for reuse and their costs are amortized over subgraphs.

6 Experiments: *RevFilter*

In this section, we evaluate *RevFilter* for discovering new suspicious subgraphs. The objective is to answer the following questions.

- **Q1**: How does *RevFilter* perform compared to baselines?
- **Q2a**: How does the sparsity of money laundering affect the performance of *RevFilter*?
- **Q2b**: How many recommendations (k) are needed for *RevFilter* to discover the majority of money laundering activities?
- **Q3**: (Ablation) Is the iterative filtering in Algorithm 1, as opposed to a one-pass top-k selection, necessary?
- **Q4a**: (Ablation) How does fine-tuning with data augmentation impact the performance of *RevFilter*?
- **Q4b**: (Ablation) How does the keeping of more candidates impact the performance of *RevFilter*?

6.1 Experiment Setup

To the best of our knowledge, there does not exist previous literature experimenting on the same setting. The closest domain is collaborative filtering, which typically recommends items given

Table 2: Performance comparison between *RevFilter* and baselines in eight settings ($n^+ + n^- @k$). The two metrics HR and NDCG are the higher the better.

$n^+ + n^- @k$	1+5@1		1+10@1		1+10@3		1+100@3		3+100@10		3+1000@10		10+1000@100		10+10000@100	
Density (%)	12.54		6.83		6.83		0.92		0.98		0.16		0.18		0.02	
	HR	NDCG	HR	NDCG	HR	NDCG	HR	NDCG	HR	NDCG	HR	NDCG	HR	NDCG	HR	NDCG
MLP	0.4935	0.4935	0.3151	0.3151	<u>0.7383</u>	0.5596	0.0156	0.0105	0.0499	0.0278	0.0000	0.0000	0.0088	0.0034	0.0000	0.0000
NGCF	<u>0.5560</u>	<u>0.5560</u>	<u>0.4831</u>	<u>0.4831</u>	0.6393	0.5723	<u>0.4570</u>	<u>0.4226</u>	<u>0.4199</u>	<u>0.3705</u>	<u>0.3516</u>	<u>0.3291</u>	<u>0.4008</u>	<u>0.3374</u>	<u>0.3590</u>	<u>0.3170</u>
LightGCN	0.5391	0.5391	0.4596	0.4596	0.6862	<u>0.5925</u>	0.4310	0.3661	0.3652	0.3139	0.2452	0.2150	0.2958	0.2556	0.1939	0.1853
RevFilter	**0.9479**	**0.9479**	**0.9479**	**0.9479**	**0.9779**	**0.9424**	**0.9271**	**0.8208**	**0.8826**	**0.6298**	**0.7385**	**0.5291**	**0.8568**	**0.4569**	**0.6646**	**0.3790**

(a) Impact of sparsity (in log-scale)

(b) Impact of k

Figure 4: Comparison between *RevFilter* and baselines regarding sparsity and k.

a user (i.e., **single-user → multi-item**). In contrast, our task involves recommending links (suspicious 1-1 sender-receiver pairs) in a **multi-sender → multi-receiver** setting. Therefore, we adopt the experimental setup and evaluation protocol used in collaborative filtering and recommendation systems [10, 11, 21, 26] but make modifications to suit our task.

6.1.1 Constructing a Test Set. We modify the test set of *Elliptic2* for the recommendation task, which recommends links between senders and receivers corresponding to suspicious money-laundering subgraphs. The modification proceeds as follows:

(1) For every subgraph, identify the senders and receivers. Denote the suspicious set as $\mathcal{D}^+$ and the licit set as $\mathcal{D}^-$.
(2) Filter the suspicious set to include only subgraphs that induce a single link between senders and receivers (i.e., subgraphs with only a single sender and receiver). Call this subset $\mathcal{D}_{1-1}^+$.
(3) Randomly select n^+ subgraphs (links) from $\mathcal{D}_{1-1}^+$ and n^- subgraphs from $\mathcal{D}^-$. Then, collect all the nodes from these subgraphs to construct a fully-connected, bipartite graph $\mathcal{S}$ through linking the senders and receivers. The task is to recommend the n^+ links in $\mathcal{S}$.
(4) Repeat (3) N times to construct a test set of size N. A large N can significantly reduce the performance variance of a method. We take $N = 256$ for all experiments.

We evaluate each method by asking it to recommend top-k links and denote the setting by $\boldsymbol{n^+ + n^- @k}$. The task is more challenging with a smaller n^+ and larger n^-, or with a smaller k. We define the **density** to be $n^+/(|\mathbb{S}||\mathbb{R}|)$ and call the case of small density, **sparse**. We evaluate on different settings with varying n^+, n^-, and k.

6.1.2 Baselines. We compare *RevFilter* against three baselines. **MLP** is a straightforward collaborative filtering method: it produces node embeddings by using MLP and makes recommendations based on the dot product between embeddings. **NGCF** [21] and **Light-GCN** [10] are state-of-the-art recommendation systems based on GCN [15]. Both methods propagate messages in a user-item graph to yield node embeddings; recommendations are made based on the dot product. The distinction between the two lies in their architectures: NGCF incorporates feature transformations and nonlinear activations inside its GCN layers, whereas LightGCN does not.

6.1.3 Evaluation Metrics. To evaluate the performance of top-k recommendation, we employ Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG) [9], which are widely used in collaborative filtering [11, 21]. HR counts the number of ground-truth links appearing in the top-k ranked list and HDCG gives a higher score when the ground-truth links are ranked higher.

6.1.4 Implementation Details. Both architectures of *RevClassify* can serve as the pretrained classifier for *RevFilter*; we use *RevClassify*$_{DS}$ because of its faster inference and robustness (Section 5.2). We also employ the two enhancements mentioned in Section 4.2: we fine-tune the pretrained classifier with merging pairs, setting $\gamma = 0.4$, $1 \leq n_{merge} \leq 20$, and $\alpha_{keep} = 1.5$.

We implement NGCF and LightGCN using publicly available codebases, while incorporating the following changes for our setting. (1) We initialize the embedding layer with node features instead of using the Xavier initialization [8]. (2) We optimize the binary cross entropy loss instead of the BPR loss [19] due to the limited number of labeled links available. We train the baselines

for 150 epochs incorporating early stopping, with hyperparameter (number of layers, dropout, and hidden dimension) searching.

6.2 Results

6.2.1 Performance Comparison With Baselines (Q1). Table 2 presents the comparison of methods in eight diverse settings of $n^+ + n^- @k$. *RevFilter* outperforms all baselines on both metrics. In most cases, *RevFilter* achieves an HR that is 50–100% higher than the second best method, indicating that we can identify 1.5 to 2 times more money-laundering schemes.

6.2.2 Impact of Sparsity (Q2a). As shown in Figure 4a, *RevFilter* demonstrates significant robustness in sparse settings (i.e., low density), where there are few illicit links. For instance, when the density of S decreases from 10^{-1} to 10^{-4} (i.e., 1000x more sparse), *RevFilter* experiences only a $\sim 20\%$ loss in HR, whereas NGCF and LightGCN show a $\sim 50\%$ drop and MLP suffers a $\sim 100\%$ decrease.

6.2.3 Impact of k (Q2b). We present the performance of the methods in the $1 + 1000@k$ setting, with varying k in Figure 4b. *RevFilter* can identify the ground-truth links in more than 90% chance with a small number of recommendations ($k = 10$), whereas the baselines cannot reach 75% with a large number of recommendations ($k = 100$). This result underscores that *RevFilter* can recommend a small yet high-quality set of suspicious subgraphs, significantly reducing the need for human analysts to examine numerous potential money laundering schemes. Note, unlike others, that our method's NDCG decreases as k increases. This is because 1) as k increases, our method's iterative filtering becomes less effective and 2) our method is optimized for filtering rather than ranking links. Therefore, it is important to select a proper k to avoid suboptimal performance associated with an excessively large k.

6.2.4 Iterative Versus One-Pass Filtering (Q3). To verify the usefulness of iterative filtering, we compare against a variant ("No iterations"). In this variant, the outer loop in Algorithm 1 is removed and the filtering is done in a single-pass, by computing scores for every 1-1 sender-receiver pair and recommending the top-k pairs. The results in Table 3 show a significant drop in HR (0.4 to 0.6) for sparse S. This suggests that iterative filtering can mitigate the errors of the base classifier.

6.2.5 Impact of Fine-Tuning With Data Augmentation (Q4a). We investigate the impact of fine-tuning the classifier with augmented data by comparing against the case without fine-tuning ("No fine-tuning"). Table 3 shows that the performance without fine-tuning drops across the board, with more significant decrease in sparse settings. This indicates that fine-tuning improves the classifier's accuracy, particularly for large sender-receiver pairs.

6.2.6 Impact of Keeping More Candidates (Q4b.) To evaluate the effect of keeping more candidates on recommendation performance, we compare our method against the standard case of $\alpha_{keep} = 1$, where exactly k candidates are retained in each iteration. The latter case leads to a slight decline in performance in most of the settings. We speculate that the fine-tuned classifier is already near-perfect, hence the improvement brought in by more candidates is minor. Since keeping more candidates compromises the inference speed, we suggest considering the trade-off when applying a large α_{keep}.

Table 3: Ablation (increasing sparsity from left to right).

Method	1+80@10	1+640@10	1+5120@10	1+10240@10
RevFilter	**0.9710**	0.9128	**0.7969**	**0.7161**
No iterations	0.9609	0.5286	0.1484	0.1367
No fine-tuning	0.8724	0.6563	0.3698	0.3815
$\alpha_{keep} = 1$	0.9688	**0.9167**	0.7891	0.7122

7 Conclusions

Elliptic2 introduced a subgraph approach for AML and it set a new standard for forensic analysis in cryptocurrencies. While the original paper benchmarked a few subgraph classification methods and showed the promise of subgraph modeling, in this paper we advance the state-of-the-art by (1) improving the effectiveness of subgraph classification and (2) developing the capability of new subgraph discovery. *RevClassify* abstracts a transaction subgraph by using its initial senders and end receivers of the funds, significantly reducing the model inference cost and the memory consumption; while *RevFilter* discovers new subgraphs by using an innovative approach inspired by recommendation systems: recommending suspicious links between senders and receivers, which may not be directly connected by a single transaction. We have conducted many empirical evaluations to show that these two approaches significantly outperform strong baselines. An avenue of future research is to discover more complex money laundering schemes (e.g., those involving more than a single sender and receiver). Another avenue is to study the temporal behavior of money laundering and develop more effective methods that leverage both temporal and structural information.

8 Supporting code

Our implementation and pretrained models are available at https://github.com/MITIBMxGraph/RevTrack.

Acknowledgments

References

[1] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. 2018. Sub2vec: Feature learning for subgraphs. In *Advances in Knowledge Discovery and Data Mining: 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part II 22.* Springer, 170–182.

[2] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. 2020. Subgraph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 8017–8029.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[4] Claudio Bellei, Muhua Xu, Ross Phillips, Tom Robinson, Mark Weber, Tim Kaler, Charles E. Leiserson, Arvind, and Jie Chen. 2024. The Shape of Money Laundering: Subgraph Representation Learning on the Blockchain with the Elliptic2 Dataset. In *KDD Workshop on Machine Learning in Finance.*

[5] Blockchain.com. 2024. https://www.blockchain.com/explorer/charts/n-transactions-total

[6] Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-yan Liu, and Liwei Wang. 2021. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning.* PMLR, 1204–1215.

[7] Europol. 2024. The Other Side of the Coin - Analysis of Financial and Economic Crime. https://www.europol.europa.eu/cms/sites/default/files/documents/The%20Other%20Side%20of%20the%20Coin%20-%20Analysis%20of%20Financial%20and%20Economic%20Crime%20%28EN%29.pdf.

[8] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 249–256.

[9] Xiangnan He, Tao Chen, Min-Yen Kan, and Xiao Chen. 2015. Trirank: Review-aware explainable recommendation by modeling aspects. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. 1661–1670.

[10] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 639–648.

[11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[12] Tim Kaler, Alexandros Iliopoulos, Philip Murzynowski, Tao Schardl, Charles E Leiserson, and Jie Chen. 2023. Communication-Efficient Graph Neural Networks with Probabilistic Neighborhood Expansion Analysis and Caching. *Proceedings of Machine Learning and Systems* 5 (2023).

[13] Tim Kaler, Nickolas Stathas, Anne Ouyang, Alexandros-Stavros Iliopoulos, Tao Schardl, Charles E Leiserson, and Jie Chen. 2022. Accelerating training and inference of graph neural networks with fast sampling and pipelining. *Proceedings of Machine Learning and Systems* 4 (2022), 172–189.

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[16] Nasdaq. 2023. Global Financial Crime Report. https://www.nasdaq.com/global-financial-crime-report.

[17] Federal Bureau of Investigation. 2023. 2022 Internet Crime Report. https://www.ic3.gov/Media/PDF/AnnualReport/2022_IC3Report.pdf.

[18] U.S. Department of the Treasury. 2024. 2024 National Money Laundering Risk Assessment. https://home.treasury.gov/system/files/136/2024-National-Money-Laundering-Risk-Assessment.pdf.

[19] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[20] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[21] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[22] Xiyuan Wang and Muhan Zhang. 2021. GLASS: GNN with labeling tricks for subgraph representation learning. In *International Conference on Learning Representations*.

[23] 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. In *2nd KDD Workshop on Anomaly Detection in Finance*.

[24] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.

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

[26] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 974–983.

[27] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. *Advances in neural information processing systems* 30 (2017).

[28] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph Neural Networks: A Review of Methods and Applications. *AI Open* 1 (2020), 57–81.