The Impact of Realistic Laundering Subgraph Perturbations on Graph Neural Network Based Anti-Money Laundering Systems

Trustworthy Financial Crime Analytics

Tom Joshua Clark[1]

Supervisor(s): Dr. Zeki Erkin[1]**, Dr. Kubilay Atasu**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Tom Joshua Clark
Final project course: CSE3000 Research Project
Thesis committee: Dr. Zeki Erkin, Dr. Kubilay Atasu, Dr. Megha Khosla

Abstract

As financial institutions adopt more sophisticated Anti-Money Laundering (AML) techniques, such as the deployment of Graph Neural Networks (GNNs) to detect patterns, laundering behavior is likely to evolve. In this paper, we present a novel perturbation framework that models laundering as an evasion-based, restricted black-box process. Our tool systematically alters labeled laundering subgraphs through a set of parameterized graph actions (intermediary injection, merging, and splitting) designed to simulate realistic laundering adaptations. We apply our framework to one of the AMLWorld synthetic transaction datasets to generate multiple perturbed versions defined by a set of parameterized preset configuration files. We then evaluate the impact of these perturbations on two MEGA-GNN variants of the current state-of-the-art in temporal multigraph-compatible GNN architectures. Our results show that realistic structural perturbations can impact performance and serve as a valuable tool to evaluate model adaptability and robustness. Our work aims to contribute to a deeper understanding of the evolutionary dynamics between AML systems and laundering behavior.

1 Introduction

In this section, we provide an overview of the research context and our motivation. We begin by looking at the limitations of current Anti-Money Laundering (AML) methods in detecting financial crime, and then briefly discuss the future of AML investigations, where access to complete transaction graphs (whether by regulatory shifts or privacy-preserving technologies) could lead to more effective detection of complex laundering schemes and consequently influence the evolution of laundering tactics. Finally, we describe our contribution, where we approach AML from the perspective of an adversary and introduce a framework to simulate and evaluate the impact of evolving laundering behavior on graph-based detection systems.

1.1 Financial Crime

Financial crime encompasses a broad spectrum of illicit activities, including fraud, bribery, tax evasion, and money laundering. As financial networks grow more complex and interconnected, detecting and disrupting these crimes has become increasingly difficult. According to the United Nations Office on Drugs and Crime, the estimated amount of money laundered worldwide in one year is 2–5% of global GDP [1]. Traditionally, financial institutions have used Know Your Customer (KYC) checks and rule-based monitoring methods [2], which rely on predefined thresholds and struggle to detect more elaborate money laundering tactics. This has led to an emerging interest in using machine learning methods for pattern recognition and anomaly detection, such as Graph Neural Networks (GNNs), which can analyze more complex relationships between entities [3].

1.2 The Future of AML

Whether driven by regulatory changes that weaken privacy protections or the adoption and viability of privacy-preserving technologies that allow for shared data access, it is becoming increasingly plausible that AML investigations in the near future will have insights into a comprehensive transaction graph rather than only what is available from their institution. With access to all transaction data, institutions would be better positioned to detect cross-border schemes and complex money laundering patterns and be better able to leverage graph-based anomaly detection and ML techniques. As AML investigations become more effective, money laundering tactics are likely to evolve, leading to the emergence of more complex laundering patterns.

1.3 Contribution

The existing literature on Anti-Money Laundering largely focuses on privacy-preserving methods or improving detection techniques through machine learning and graph-based models to identify suspicious patterns [4]. However, none of the research approaches AML from the perspective of a motivated adversary who can strategically alter the structure of their accounts and transactions to evade detection, and most overlook how the widespread use of AML systems with access to a full transaction graph could fundamentally alter money laundering tactics. Our paper will instead introduce a framework for systematically and realistically perturbing laundering subgraphs, and demonstrate how these perturbations affect the performance of state-of-the-art GNNs to help us better understand how advancements in detection might influence the evolution of financial crime and the value of robustness testing in developing more resilient and adaptive GNNs.

1.4 Outline

The rest of this paper is structured as follows: In Section 2, we provide an overview of the key concepts and background necessary to understand the paper. Section 3 details our contribution to the research area and outlines our methodology. In Section 4, we describe the experimental setup used in our research and report on the results. Section 5 will analyze and reflect on the results of our research. In Section 6, we will conclude the paper and discuss future work. Finally, Section 7 will address responsible research considerations.

2 Background

In this section, we give an overview of the key concepts relevant to our research. We begin by introducing Graph Neural Networks (GNNs) and how they are used to detect more complex laundering behavior. Then, we describe the common typologies of money laundering patterns in financial transaction graphs. Next, we explain how synthetic data is used to provide realistic transaction graphs for training and testing GNN models. Finally, we introduce the concept of adversarial attacks and their effectiveness on neural networks.

2.1 Graph Neural Networks

By using graph structure along with node and edge features, Graph Neural Networks are an effective tool for analyzing relational data. To update its state, each node in the message-passing framework aggregates information from its neighbors based on their connectivity, edge features, and node features [5].
In recent years, the application of Graph Neural Networks (GNNs) to financial crime detection has attracted growing attention [6], but despite their effectiveness, standard GNNs often struggle to detect cycles and work with directed multigraphs as they only aggregate messages at the node level, making them limited in their ability to detect subgraph patterns [7]. Recent work on GNN architectures has approached these problems with adaptations such as port numbering, reverse message passing, and multi-edge aggregations [8; 7].

2.2 Money Laundering Patterns

Transaction data can be modeled as graphs, where nodes represent financial entities and edges represent transactions between them. Within this graph representation, several common money laundering patterns emerge, as shown in Figure 1:

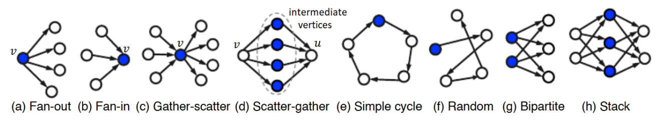

Figure 1: Laundering Patterns [9]

Much of the current literature on graph learning algorithms focuses on identification and how GNNs can learn from fraud patterns [10; 11].

2.3 Synthetic Data

Access to real-world financial transaction data is limited because of privacy regulations and, when available, is often unlabeled or only partially labeled [9], making it difficult to train or evaluate GNN models on realistic laundering scenarios. Synthetic data generators address this by creating configurable transaction graphs that can model money laundering patterns and legitimate behavior. More realistic transaction graphs can be created by more sophisticated synthetic data generators like AMLWorld, which use multi-agent-based modeling to simulate how individual actors interact within a financial network [9]. Available datasets are useful for training and benchmarking GNNs, but state-of-the-art data generators are not made publicly available, leading to a lack of support for controlled experimentation on specific laundering typologies or the modeling of evolving behavior.

2.4 Adversarial Attacks

The sensitivity of neural networks can be exploited to cause models to misclassify elements by making small and intentional perturbations to the input data [12]. These attacks are typically either:
1. Poisoning attacks: target the model's training phase by injecting malicious data.
2. Evasion attacks: target the model's inference phase by modifying the data seen by the trained model to trick it into misclassifying the data.
In white-box attacks, the adversary has access to complete information about the model, including parameters, input data, and labels, while in black-box attacks, the adversary can only observe and interact with the model's inputs and outputs. In a restricted black-box attack, the adversary has the additional constraint of only controlling a subset of nodes in the graph [13]. Research shows that for neural networks using graph data, the accuracy of node classification significantly drops even when performing only a few perturbations [14]. Notably, adversarial examples that evade detection on one model often also evade detection on other models trained for the same task [15].

3 Contribution and Methodology

In this section, we place our contribution in the context of the broader research area and lay out our methodology. We begin by detailing our intended contributions and motivation, before providing a comparison of the existing transaction datasets and justifying which we use. Next, we introduce the Graph Perturbation Framework and describe the "Parameterized Actions" model. Finally, we briefly outline the evaluation methodology that we use for the experiment, which we elaborate on in Section 4.

3.1 Contribution

Although adversarial attack frameworks on GNNs have been well studied [16], including recently in the context of fraud detection [17], the concept of modeling the evolution of money laundering patterns as adversarial behavior and perturbing labeled data is a novel approach in AML graph learning. We reframe money laundering as an evasion-based, restricted black-box adversarial attack, where a criminal without knowledge about a model's parameters can manipulate nodes and edges within their control to evade detection by maximizing the misclassification of illicit transactions.

We present a framework for systematically making graph perturbations with configurable parameters. It is important that these perturbations are carefully targeted to only affect nodes and edges that an adversary would realistically control, and that they increase or decrease complexity while preserving the overall shape of the patterns. The value of our work is twofold:

1. Tooling Contribution: We introduce a graph perturbation tool with configurable parameters that allows researchers to simulate how an adversary may plausibly alter their laundering patterns. This enables "what-if" simulations and training of GNNs on new data by altering available datasets.

2. Empirical Contribution: We run an experiment with four distinct preset parameter setups to analyze how graph perturbations that simulate adding or reducing complexity to known laundering typologies affect the performance of pre-trained GNNs.

3.2 Overview of Transaction Datasets

We do a brief comparison of some different commonly used datasets in research.

The Elliptic Dataset [18] is a partially labeled graph built on real-world Bitcoin transactions. It is valuable as it contains real-world data, but is limited due to only a small subset of nodes being annotated with ground-truth labels, which restricts supervised learning and evaluation. Additionally, the only detectable laundering patterns are those already known and labeled, which limits the ability to discover or evaluate against novel or evolving behaviors.

AMLSim [19] is a multi-agent synthetic data generator that produces fully labeled transaction graphs. It is limited by the diversity of money laundering patterns it can produce and by how accurately the generator simulates the complexity of real-world financial behavior.

AMLWorld [9] is a set of fully labeled synthetic datasets generated using a more advanced proprietary multi-agent simulator. We build our graph perturbation framework on AMLWorld because it directly implements the laundering typologies we consider, and because the underlying generator is not open source, making it a compelling case for perturbation-based modifications.

3.3 Graph Perturbation Framework Setup

The algorithm begins by clustering nodes and edges labeled "Is Laundering", essentially creating subgraphs that represent the laundering patterns within the larger transaction graph. Any node connected to a laundering edge is considered a laundering node, and we make the assumption that all nodes and edges within each of these laundering networks are under the control of an adversary, who can manipulate the shape and flow of these patterns (they may correspond to shell companies, money mules, or layering entities in a laundering network). The algorithm then further decomposes the graph by identifying all "laundering-adjacent" transactions, made up of the nodes and edges that have a non-laundering transaction going to or coming from the laundering subgraphs, such that we only ever have to perform computations on targeted subsets of the complete graph. The algorithm then makes perturbations to these identified subgraphs based on three actions, which are outlined in the next section. We emphasize that we do not target any nodes or edges outside of the laundering subgraphs, and only make perturbations to subgraphs in isolation, so as to ensure that each money laundering cluster is modified and evaluated individually without the malicious entity influencing other parts of the transaction graph. We also ensure that the source and destination of illicit funds within a laundering subgraph remain roughly the same, as these reflect the real-world directional flow of illicit funds and cannot be arbitrarily changed. Finally, we only apply our perturbations to the test split, since under our threat model, the adversary would not have access to the transactions from other splits.

3.4 Parameterized Actions and Configurability

Our perturbation framework implements parameterized actions designed to simulate realistic adaptations of money laundering behavior that can be used independently or in combination. The user is prompted to provide the file path to a configuration file, which centralizes all parameters and settings for ease of use and flexible control over perturbations. Users can configure the following settings:

- `inputfile:` path to the input CSV file containing the original transaction graph.
- `outputfile:` path to the output CSV file where the perturbed graph will be saved.
- `modification_filter:` when multiple actions are applied consecutively, this determines which edges subsequent actions will apply perturbations to.
- `visualization:` whether to visualize the input graph and resulting graph. Only enable for small input files.
- `seed:` optional seed for reproducibility.
- `actions:` an array of JSON objects where each defines a single action. Each object supports the following fields:

Action 1: Inject Intermediary Nodes (Depth Perturbation)
This action adds one or more newly created intermediary nodes between two nodes connected by an edge.
On its own, this simulates the layering stage, where transactions are routed through multiple entities to obscure the laundering trail. Intermediary injection increases the depth of the graph, which GNNs struggle with due to difficulties aggregating information over longer paths [7].

- `percent`: sets the percentage of eligible edges or nodes to which we inject nodes, allowing the user to control the intensity of perturbations.
- `perturb_by_edges`: if true, injects intermediary nodes on a per-edge basis. If false, injects intermediaries between two connected nodes and reroutes all edges between them through the intermediary. See Figure 2.
- `intermediary_depth`: number of intermediary nodes to inject per selected edge or node.

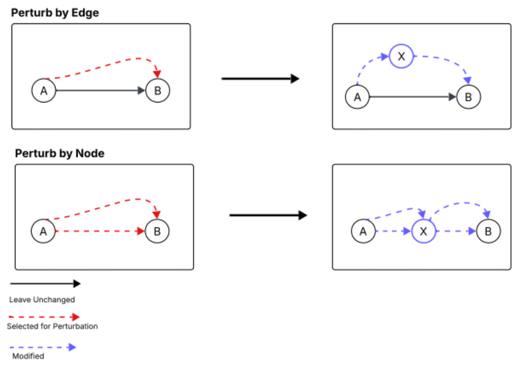

Figure 2: Inject Intermediary

Action 2: Merge Nodes (Consolidation)
This action consolidates multiple laundering nodes into a single node.
On its own, this simulates integration, where funds are gathered into fewer accounts. Merging reduces graph complexity and changes degree distributions, hiding identifiable flow patterns.

- `percent`: sets the percentage of eligible edges or nodes to which we mark to be merged, allowing the user to control the intensity of perturbations. Note that it will not merge clusters with two or fewer nodes.
- `perturb_by_edges`: if true, merges nodes that are directly connected by an edge. If false, merges nodes that share a common neighbor (eg two nodes both connected to the same central node). This does not reduce the total number of edges in the graph. See Figure 3.

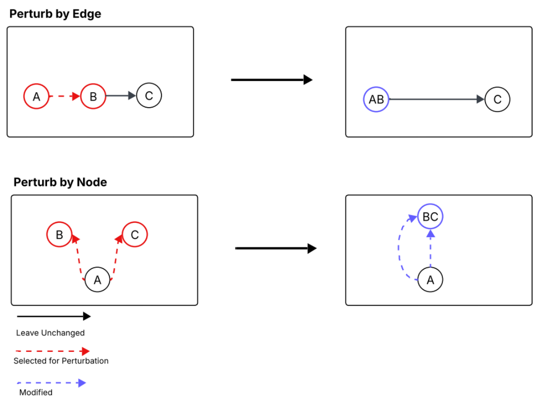

Figure 3: Merge

Action 3: Split Nodes (Fragmentation)
This action fragments laundering nodes into multiple smaller nodes by redistributing and creating edges.
On its own, this simulates smurfing, where funds are broken down into smaller amounts and distributed across different accounts. Splitting increases the number of nodes and edges in the graph, which can obscure laundering paths and create noise.

- `percent`: sets the percentage of eligible nodes to split or edges to split from, allowing the user to control the intensity of perturbations.
- `perturb_by_edges`: if true, splits nodes by individually rerouting edges to new nodes. If false, clones target nodes and redistributes incoming and outgoing edges from the original, resulting in a complete bipartite connection between sources and destinations for both original nodes and clones. See Figure 4.
- `split_depth`: number of new nodes to create from each selected node (eg if set to 2, one node will be split into three).

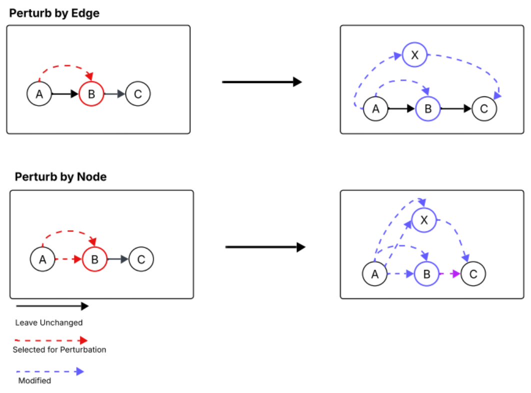

Figure 4: Split

Different configurations and combinations of these actions can be used to simulate adding or reducing complexity to the known money laundering patterns we have identified in Subsection 2.2. Table 1 below illustrates how our parameterized actions align with the AMLWorld patterns:

Laundering Pattern	Action Configuration
Fan-Out	Split by Node
Fan-In	Merge by Node or Split by Node
Gather-Scatter	Merge by Node followed by Split with split_depth > 1
Scatter-Gather	Inject Intermediary followed by Split by Node with modification_filter: modified
Simple-Cycle/Random	Lengthen with Inject Intermediary with intermediary_depth > 1
Bipartite	Split by Node
Stack	Inject Intermediary by Edge followed by Split by Node

Table 1: Laundering Patterns and Action Configurations required to simulate

Our actions were carefully designed to increase or decrease the complexity of patterns while keeping the underlying shape of the patterns intact. The only action with a risk of pattern destruction is Merge in the case of Scatter-Gather/Gather-Scatter or Fan-In/Fan-Out, as it can reduce the number of nodes to below the minimum required to form these patterns.

3.5 Evaluation

Our empirical contribution is performed by applying four distinct "preset" parameter configurations (which we will detail and motivate in the next section) to the chosen dataset. Each preset defines a different perturbation strategy to simulate plausible changes in money laundering behavior.
We run inference using two pre-trained MEGA-GNN models on variations of five datasets: the original AMLWorld HI_Small dataset, which consists of 5 million transactions, 5,100 of which are laundering, and 78 perturbed variants generated by applying four different presets. Specifically, Presets 1–3 were each applied at four perturbation levels across six random seeds ($3 \times 4 \times 6$), and Preset 4 was applied without varying the perturbation percentage across six seeds (1×6). We then conduct a quantitative and qualitative analysis on how these perturbations impact the performance of the model. We measure the extent to which the F1 score degrades under different perturbation strategies.

4 Experimental Setup and Results

In this section, we describe the setup used to evaluate the effects of our graph perturbation framework. First, we explain our choice of GNN architecture and introduce the pre-trained models used for inference. Then we define and motivate our parameterized action presets and explain how they are applied to the AMLWorld dataset to produce the perturbed variants. Finally, we present the performance of the GNN models on both the original and perturbed datasets, and compare the F1 scores across different laundering pattern complexities.

4.1 Selected GNN architecture

Financial transaction datasets are modeled as temporal multigraphs, which comprise the complex money laundering patterns found in the AMLWorld datasets that we modify. There have been comparatively few GNNs introduced specifically as solutions to this, such as ADAMM [20], Multi-GNN [7], FraudGT [10], and MEGA-GNN [8], as these are the only ones capable of directly handling directed graphs with multiedges and self-loops [20]. We use MEGA-GNN as our GNN architecture because it is the current state-of-the-art, outperforming other solutions by up to 13% on AML datasets [8]. Specifically, we run a MEGA-GNN Graph Isomorphism Network (GIN) model and a MEGA-GNN Principal Neighbourhood Aggregation (PNA) model that have been pre-trained on the "HI-Small_Trans.csv" AMLWorld dataset with the following settings:

```
python main.py --data Small_HI --model
{MODEL} --emlps --reverse_mp --ego
--flatten_edges --edge_agg_type {MODEL}
--n_epochs 80 --save_model --task edge_class
--unique_name {NAME}
```

where {MODEL} is either `gin` or `pna`.

4.2 Action Parameter Presets

We introduce four distinct presets to contrast the impact on GNN performance for different structurally realistic laundering behaviors (see Appendix B, Figure 7):

Preset 1-3 (7a)(7b)(7c): Apply a single action each: Inject Intermediary by edge for Preset 1, Merge by edge for Preset 2, and Split by node for Preset 3.
To create the different datasets, we vary the perturbation percentage across four levels (0.2, 0.4, 0.6, 0.8), and for each level, we apply six different seeds (50, 100, 150, 200, 250, and 300), resulting in 24 perturbed datasets per preset.

Preset 4 (7d): Applies two actions sequentially. First, injects intermediaries into 60% of edges, and then splits 40% of nodes that have an incoming or outgoing edge modified or created by the injection action.

These actions add structural complexity to the shape of clusters while maintaining the motifs of common money laundering patterns. It is trivial to see how edge injection followed by node splitting can result in fan-out or fan-in patterns, but we also demonstrate in Figure 5 how it can result in scatter-gather:

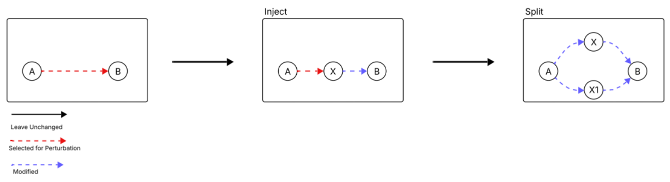

Figure 5: Scatter-Gather Pattern simulated with Preset 4

For this preset, only the seed is varied (50, 100, 150, 200, 250, and 300), yielding six perturbed datasets, for a total of 78 perturbed datasets, plus the original unperturbed HI-Small_Trans.csv

By introducing depth perturbations and fragmentation, we disguise direct transactional relationships, impeding the model's ability to aggregate signals effectively, as GNNs are known to struggle with detecting long cycles and spread out patterns [7].
Each of these presets distort the original topology and shape of the laundering subgraphs, making it harder for GNNs to identify suspicious transactions.

4.3 Procedure

We use AMLWorld HI-Small_Trans.csv as our base dataset, which contains realistic, labeled financial transactions and the money laundering patterns we identified in the background section. We then train two MEGA-GNN models (GIN, PNA) on the unmodified dataset as described in Subsection 4.1. Next, using our graph perturbation tool, we generate `3 (presets) * 4 (perturbation levels) * 6 (seeds) + 1 (preset) * 6 (seeds) = 78` modified versions of the test split of the original dataset, the exact configurations of which are detailed in Subsection 4.2.
Next, we recombine the splits and convert the datasets into the format used by the GNN architecture with:

```
python format_kaggle_files.py
data\{DIR}\augmented.csv
```

where `{DIR}` is the directory the dataset is stored in.
See Appendix A.2 for the full set of perturbation metrics at each percentage level, which includes statistics about the number of clusters affected, laundering edges perturbed, and the percentage change in laundering transactions.

Using the two models that have been pre-trained on the original data, we run inference on the 78 datasets, as well as the original dataset six times for fair comparison and standard deviation calculation, yielding a total of 168 sets of results:

```
python main.py --data{DATASET} --inference
--model {MODEL} --emlps --reverse_mp --ego
--flatten_edges --edge_agg_type {MODEL}
--task edge_class --unique_name {MODEL}
```

We then compare the resulting F1 scores across the datasets to assess the impact of each perturbation strategy on model performance.

4.4 Results

Transaction Dataset	**GIN**	**PNA**
HI-Small (Original)	73.32% ± 0.06	76.41% ± 0.09
Preset 4 (Inject+Split)	56.54% ± 0.75	62.07% ± 0.52

Table 2: F1 Scores (%) for GIN and PNA on original dataset and dataset perturbed by Preset 4

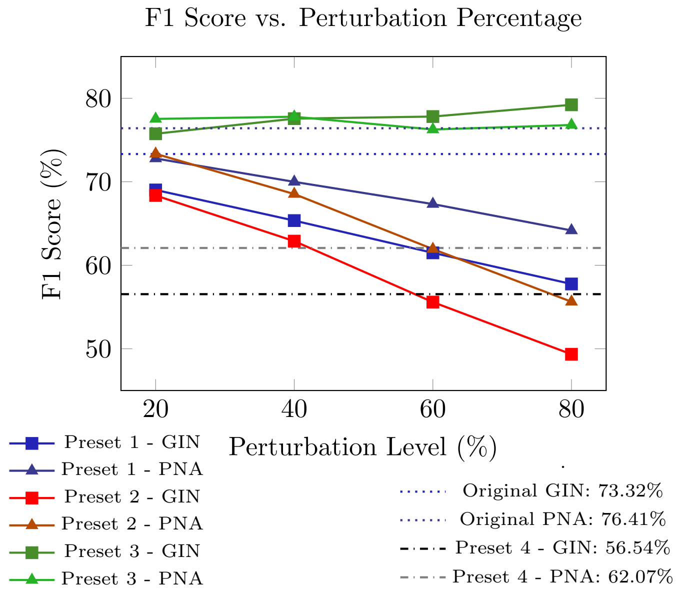

Figure 6: Test F1 Scores (%) across perturbation levels

The results in Table 2 show the performance of the two MEGA-GNN variants (GIN and PNA) on the original dataset and the perturbed datasets produced using Preset 4 (Inject followed by Split). Our results on the unperturbed dataset are consistent with the results obtained in the MEGA-GNN paper [8].

The results in Figure 6 show a plot of the F1 scores across perturbation levels for each preset. Increasing the perturbation level in Preset 1 (Inject) and Preset 2 (Merge) causes a decrease in performance for both models, with Preset 2 (Merge) having the most negative impact (especially at higher perturbation which results in significant information loss). Preset 3 (Split) slightly improves the performance of the GIN model and has little effect on the PNA model.

The F1 score for Preset 1 (Inject) at perturbation level 60% is 61.49% for the GIN model and 67.33% for the PNA model, which are higher than the Preset 4 (Inject at perturbation level 60% followed by Split at perturbation level 40%) scores of 56.54% for the GIN model and 62.07% for the PNA model, which shows that splitting does reduce performance when applied to edges that have been perturbed by a previous action.

See Appendix A.1 for complete results for each preset, including F1 score, precision, recall, and their corresponding standard deviations.

5 Discussion

In this section, we interpret the results of our experiment in the context of realistic adversarial strategies. We then revisit the threat model and discuss how well our perturbation framework captures the behavior of laundering evolution in transaction graphs. Next, we examine the broader significance of our results and the applications of our tool. Finally, we discuss the limitations of our perturbation framework and reflect on our results.

5.1 Interpretation of Results

Our results show that structural perturbations do impact and degrade the performance of GNN models while maintaining the operational plausibility and realism of laundering patterns. By carefully designing perturbations, we can modify existing AML datasets while preserving their underlying pattern shapes, allowing us to test the adaptability of GNNs against a much wider range of datasets. MEGA-GNN, the current state-of-the-art in AML architectures, is already fairly robust against increasing the complexity of laundering patterns within transaction datasets, as evidenced by its relatively high F1 scores, even under heavier perturbations.

Increasing structural complexity decreases performance, though not drastically, because the underlying patterns are not destroyed, which is why high perturbation levels of Preset 2 (Merge), where information is removed, does more significantly degrade performance.
Interestingly, Preset 3 (Split) leads to a slight increase in performance for both models, though the GIN variant shows a more noticeable improvement than the PNA. This is likely because splitting alone preserves all original information and introduces cloned nodes, which increases the neighborhood size, especially for larger and more connected laundering patterns.
In Preset 4, when Split is applied after Inject (only to nodes with edges perturbed by Inject), it results in lower performance than applying Preset 1 (Inject) at the same perturbation level, which suggests that splitting does decrease performance when it amplifies perturbations introduced by other actions.
The PNA model performed considerably better than the GIN model on both the original dataset and the perturbed datasets, which is expected since it combines multiple aggregation functions and applies degree-scalers, allowing it to adapt to varying graph topologies [21]. It is also expected that the PNA model was slightly less negatively impacted by Preset 1 (Inject), since a model that is more capable of extracting information from neighborhoods should be more resistant to added complexity.

5.2 Threat Model

As explained in Subsection 3.3, our perturbations are only applied to known laundering subgraphs, so we can assume that all nodes and edges we apply perturbations to are directly controlled by the adversary, which supports the plausibility and relevance of our threat model.
Furthermore, each of the three perturbation actions in our framework was designed not to destroy the shape of patterns, and to simulate established and plausible money laundering tactics:
Intermediary Node Injection models the use of layering to obscure the origin and destination.
Merging Nodes models integration, where funds are re-aggregated into fewer accounts.
Splitting Nodes models smurfing, in which large sums of money are split into smaller amounts.
Because these perturbations are only applied to laundering subgraphs and correspond to known, real-world tactics, we argue that our framework effectively simulates realistic laundering behavior. Our framework allows us to explore how laundering strategies may evolve in response to improved detection systems, since as models become more effective at identifying known patterns, adversaries are likely to adapt their laundering strategies to evade detection. Our framework provides a systematic way to simulate potential behavior, which may provide insights into the potential direction of future laundering techniques.

5.3 Broader Implications

Our perturbation framework serves as a tool to generate data for "what-if" scenarios, which allows us to investigate how different laundering strategies may change in response to improvements in graph-based detection systems. By carefully designing structural graph perturbations, we can evaluate the adaptability of GNN models against a variety of modifications to laundering patterns, which can help anticipate evolving tactics in money laundering behavior. Since training models on current-day datasets relies on the assumption that behavior remains constant over time, we reiterate the need for flexible and customizable synthetic data generators. Our research contributes to a deeper understanding of the evolutionary dynamics between AML systems and laundering strategies by supporting the development of AML tools that are more resilient to previously unseen or deliberately evasive laundering patterns.

5.4 Reflection

Our framework provides a valuable approach to examining how laundering behavior may evolve. By modeling systematic pattern perturbations, we move toward a more proactive approach to AML design, where we use simulation to anticipate previously unmodeled threats rather than only reacting to and detecting currently known patterns.
On the other hand, our tool is limited in its flexibility by being restricted to only three types of perturbation actions, which are applied randomly within laundering subgraphs based on the parameters set in the configuration file. There is currently no way to simulate conditional or rule-based perturbations aside from our modification filter, which only applies to successive actions. Real-world laundering may be more context-dependent, which might make it more challenging to reproduce realistically through our actions.
Furthermore, while our actions respect causality, we do not implement settings to manipulate timestamps of modified or created transactions, which real-world laundering may rely on.

Our experiment succeeds in demonstrating the value of our perturbation framework and contributes to the broader literature that examines the effects of structural perturbations on GNNs. Furthermore, by using a seed parameter that can be set in the configuration file and providing automation scripts for perturbation and inference in our repository, we ensure the reproducibility of our research. On the other hand, our experimental setup could benefit from applying our perturbations to a broader range of graph sizes, laundering typologies, or GNN architectures. Additionally, the presets used to define our perturbations could have been more purposefully designed and motivated. To better investigate how different perturbations impact model performance, future experiments could develop more established presets that reflect researched laundering strategies.

6 Conclusions and Future Work

Our research fills a gap in the current AML literature by approaching the problem from the perspective of a motivated adversary who can modify the structure of their laundering patterns to evade detection. Traditional AML research often overlooks how the widespread use of GNNs with access to comprehensive transaction graphs can fundamentally alter money laundering behavior. We address this by introducing a novel framework for systematically making realistic perturbations to laundering subgraphs and evaluating how these perturbations affect the performance of state-of-the-art GNNs.

A clear next step in our research would be to design presets with more action combinations and depth perturbations to test at what level of complexity performance begins to deteriorate more rapidly. Additionally, future work could expand on the capabilities of our perturbation framework to simulate more varied adversary behavior by exploring how perturbations to specific laundering patterns (rather than to all laundering subgraphs) affect detectability. Furthermore, additional actions or settings could be added to rewire nodes or manipulate the flow of laundered funds. Currently, nodes created by our actions do not interact with any nodes that are not part of their laundering subgraph and do not exhibit realistic behavior other than what is inherited by the nodes they are connected to. Our framework could benefit from implementing rule-based perturbations that apply consistent modifications across all instances of identified patterns, allowing for more controlled adversarial simulations. Finally, more experiments could be designed to investigate how different actions and types of perturbations affect the resilience and generalization capabilities of various GNN architectures.

We began this research with the goal of investigating the extent to which laundering subgraph perturbations affects GNN performance, to which our main takeaway is that realistic graph perturbations can degrade model performance while maintaining plausible behavior by preserving the shape of patterns, which highlights the importance of evaluating model adaptability under various structural changes. The lack of customizable, realistic, and fully labeled datasets remains a significant issue in AML research, which our framework addresses by enabling "what-if" simulations of potential laundering behavior through the systematic perturbation of existing synthetic datasets.
We advocate for a more proactive approach to designing AML systems by utilizing robustness testing to contribute to the development of solutions that are more resistant to the continuously evolving landscape of financial crime.

7 Responsible Research

7.1 Environmental Impact

Training GNNs requires significant computational resources. For our experiments, all training (80 epochs per model) was performed on a node with a single V100 GPU. Additionally, to reduce energy consumption, we tested and debugged our perturbation tool on small subgraphs made from HI-Small_Patterns locally before using the full HI-Small_trans dataset and running inference on the DelftBlue HPC cluster.

7.2 Reproducibility

It is important to us that our results can be independently verified, so both the perturbation framework and the pretrained GNN models used in our experiments are available at `https://github.com/tjcleric/AMLTerraform`.
This repository contains the perturbation tool (terraform.py), the JSON configuration files (preset1.json, preset2.json, preset3.json, and preset4.json), the saved model checkpoints (checkpoint_GIN.tar and checkpoint_PNA.tar), and miscellaneous scripts, including automation tools for dataset generation and inference (automate_perturbation.py and automate_inference.py). Model training was done on DelftBlue's V100 GPU node.

7.3 Ethical Consideration and Potential for Misuse

This research project introduces a framework for simulating money laundering behavior by applying perturbations to known patterns, but by itself, it does not help malicious actors create novel laundering schemes. We only provide a systematic way to alter existing patterns and observe how a GNN model responds to these changes, which a motivated adversary would be able to do without our tool simply by inserting their own crafted laundering patterns into the dataset. Our contribution lies in automating and parameterizing this process for research purposes, and in evaluating how state-of-the-art GNN architectures respond to these perturbations, to support the improvement of laundering detection capabilities.

References

[1] United Nations Office on Drugs and Crime, “Money laundering,” accessed: 2025-05-13. [Online]. Available: https://www.unodc.org/unodc/en/money-laundering/overview.html

[2] European Commission, “Anti-money laundering and countering the financing of terrorism at eu level,” https://finance.ec.europa.eu/financial-crime/anti-money-laundering-and-countering-financing-terrorism-eu-level_en, accessed: 2025-05-13.

[3] H. Kim, B. S. Lee, W.-Y. Shin, and S. Lim, “Graph anomaly detection with graph neural networks: Current status and challenges,” 2022. [Online]. Available: https://arxiv.org/abs/2209.14930

[4] TU Delft, “Fintech - expertise,” https://www.tudelft.nl/fintech/expertisec1549679, 2024, accessed: 2025-06-08. Archived at: https://archive.ph/GPmmT.

[5] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” 2017. [Online]. Available: https://arxiv.org/abs/1704.01212

[6] J. Nicholls, A. Kuppa, and N.-A. Le-Khac, “Financial cybercrime: A comprehensive survey of deep learning approaches to tackle the evolving financial crime landscape,” *IEEE Access*, vol. PP, pp. 1–1, 12 2021.

[7] B. Egressy, L. von Niederhäusern, J. Blanusa, E. Altman, R. Wattenhofer, and K. Atasu, “Provably powerful graph neural networks for directed multigraphs,” 2024. [Online]. Available: https://arxiv.org/abs/2306.11586

[8] H. Çağrı Bilgi, L. Y. Chen, and K. Atasu, “Multigraph message passing with bi-directional multi-edge aggregations,” 2024. [Online]. Available: https://arxiv.org/abs/2412.00241

[9] E. Altman, J. Blanuša, L. von Niederhäusern, B. Egressy, A. Anghel, and K. Atasu, “Realistic synthetic financial transactions for anti-money laundering models,” 2024. [Online]. Available: https://arxiv.org/abs/2306.16424

[10] J. Lin, X. Guo, Y. Zhu, S. Mitchell, E. Altman, and J. Shun, “Fraudgt: A simple, effective, and efficient graph transformer for financial fraud detection,” 11 2024, pp. 292–300.

[11] Z. Tian, Y. Ding, X. Yu, E. Gong, J. Liu, and K. Ren, “Towards collaborative anti-money laundering among financial institutions,” p. 4722–4733, Apr. 2025. [Online]. Available: http://dx.doi.org/10.1145/3696410.3714576

[12] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[13] J. Ma, S. Ding, and Q. Mei, “Towards more practical adversarial attacks on graph neural networks,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 4756–4766. [Online]. Available: https://arxiv.org/abs/2006.05057

[14] D. Zügner, A. Akbarnejad, and S. Günnemann, “Adversarial attacks on neural networks for graph data,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD ’18. ACM, Jul. 2018, p. 2847–2856. [Online]. Available: http://dx.doi.org/10.1145/3219819.3220078

[15] F. Tramèr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, “The space of transferable adversarial examples,” 2017. [Online]. Available: https://arxiv.org/abs/1704.03453

[16] X. Zhang and M. Zitnik, “Gnnguard: Defending graph neural networks against adversarial attacks,” 2020. [Online]. Available: https://arxiv.org/abs/2006.08149

[17] J. Choi, H. Kim, and J. J. Whang, “Unveiling the threat of fraud gangs to graph neural networks: Multi-target graph injection attacks against gnn-based fraud detectors,” 2025. [Online]. Available: https://arxiv.org/abs/2412.18370

[18] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson, “Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics,” 2019. [Online]. Available: https://arxiv.org/abs/1908.02591

[19] T. Suzumura and H. Kanezashi, “Anti-Money Laundering Datasets: InPlusLab anti-money laundering datadatasets,” http://github.com/IBM/AMLSim/, 2021.

[20] K. Sotiropoulos, L. Zhao, P. J. Liang, and L. Akoglu, “Adamm: Anomaly detection of attributed multi-graphs with metadata: A unified neural network approach,” 2023. [Online]. Available: https://arxiv.org/abs/2311.07355

[21] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, “Principal neighbourhood aggregation for graph nets,” 2020. [Online]. Available: https://arxiv.org/abs/2004.05718

A Full Perturbation Metrics and Extended Results

A.1 Results

Metric	20%	40%	60%	80%
		GIN		
Test F1	69.02% ± 0.42	65.35% ± 0.29	61.49% ± 0.44	57.76% ± 0.34
Test Precision	76.83% ± 0.10	77.89% ± 0.25	78.73% ± 0.18	79.36% ± 0.14
Test Recall	62.66% ± 0.68	56.29% ± 0.39	50.45% ± 0.53	45.41% ± 0.40
		PNA		
Test F1	72.79% ± 0.38	69.99% ± 0.33	67.33% ± 0.29	64.17% ± 0.35
Test Precision	85.57% ± 0.14	86.71% ± 0.13	87.63% ± 0.14	88.17% ± 0.12
Test Recall	63.34% ± 0.53	58.68% ± 0.41	54.66% ± 0.36	50.44% ± 0.41

Table 3: Test performance metrics for Preset 1 (Inject) with GIN and PNA models across percentages.

Metric	20%	40%	60%	80%
		GIN		
Test F1	68.36% ± 0.44	62.89% ± 0.68	55.59% ± 0.43	49.34% ± 0.96
Test Precision	69.28% ± 0.21	64.75% ± 0.58	57.92% ± 0.36	51.68% ± 0.92
Test Recall	67.46% ± 0.76	61.14% ± 0.84	53.45% ± 0.72	47.21% ± 1.02
		PNA		
Test F1	73.35% ± 0.38	68.53% ± 0.36	61.94% ± 0.54	55.63% ± 0.67
Test Precision	81.33% ± 0.27	77.48% ± 0.31	71.61% ± 0.57	65.63% ± 0.40
Test Recall	66.80% ± 0.49	61.43% ± 0.42	54.57% ± 0.62	48.27% ± 0.93

Table 4: Test performance metrics for Preset 2 (Merge) with GIN and PNA models across percentages.

Metric	20%	40%	60%	80%
		GIN		
Test F1	75.75% ± 0.63	77.56% ± 0.90	77.81% ± 0.60	79.22% ± 0.58
Test Precision	80.32% ± 0.57	84.29% ± 0.38	87.31% ± 0.42	89.62% ± 0.12
Test Recall	71.67% ± 0.81	71.84% ± 1.34	70.18% ± 0.88	70.98% ± 0.89
		PNA		
Test F1	77.53% ± 0.28	77.79% ± 0.83	76.26% ± 0.63	76.80% ± 0.46
Test Precision	87.23% ± 0.39	89.79% ± 0.28	91.50% ± 0.22	92.99% ± 0.10
Test Recall	69.78% ± 0.53	68.62% ± 1.18	65.38% ± 0.89	65.41% ± 0.66

Table 5: Test performance metrics for Preset 3 (Split) with GIN and PNA models across percentages.

Metric	**GIN**	**PNA**
Test F1	56.54%	62.07%
Test Precision	83.50%	90.67%
Test Recall	42.74%	47.19%

Table 6: Test performance metrics for Preset 4 (Inject + Split) with GIN and PNA models.

Metric	**GIN**	**PNA**
Test F1	73.32% ± 0.06	76.41% ± 0.09
Test Precision	75.32% ± 0.07	84.15% ± 0.10
Test Recall	71.42% ± 0.08	69.98% ± 0.10

Table 7: Test performance metrics on original unperturbed dataset with GIN and PNA models.

Note that the standard deviations for inference on the original dataset are far lower than what was reported in the MEGA-GNN paper [8] because, for their experiment, a new model was trained each time.

A.2 Perturbation Metrics

Metric	**P1 (Inject)**	**P2 (Merge)**	**P3 (Split)**
Clusters Affected	32.70% ± 2.04	10.97% ± 0.23	32.87% ± 1.95
Laundering Edges Perturbed	33.32% ± 0.00	33.24% ± 1.18	43.56% ± 4.76
%Δ Laundering Transactions	+19.99% ± 0.00	-11.82% ± 0.14	+29.54% ± 4.59

Table 8: Graph Perturbation Statistics at 20%.

Metric	**P1 (Inject)**	**P2 (Merge)**	**P3 (Split)**
Clustetrs Affected	53.38% ± 2.09	13.42% ± 0.24	54.70% ± 1.11
Laundering Edges Perturbed	57.12% ± 0.00	37.54% ± 1.17	73.41% ± 1.39
%Δ Laundering Transactions	+39.98% ± 0.00	-25.48% ± 0.17	+70.03% ± 3.76

Table 9: Graph Perturbation Statistics at 40%.

Metric	P1 (Inject)	P2 (Merge)	P3 (Split)
Clusters Affected	69.32% ± 1.09	17.13% ± 0.20	100.00% ± 0.00
Laundering Edges Perturbed	74.97% ± 0.00	39.78% ± 1.44	93.82% ± 1.22
%Δ Laundering Transactions	+59.96% ± 0.00	-40.42% ± 0.11	+125.56% ± 8.79

Table 10: Graph Perturbation Statistics at 60%.

Metric	P1 (Inject)	P2 (Merge)	P3 (Split)
Clusters Affected	85.15% ± 1.29	17.66% ± 0.55	100.00% ± 0.00
Laundering Edges Perturbed	88.86% ± 0.00	35.71% ± 1.28	99.10% ± 0.14
%Δ Laundering Transactions	+79.95% ± 0.00	-51.50% ± 0.07	+181.46% ± 3.32

Table 11: Graph Perturbation Statistics at 80%.

Metric	Preset 4 (Inject followed by Split)
Clusters Affected	69.32% ± 1.09
Laundering Edges Perturbed	84.55% ± 0.08
%Δ Laundering Transactions	+159.17% ± 1.48

Table 12: Graph Perturbation Statistics for Preset 4 at 60% Inject followed by 40% Split

These tables show the statistics for perturbations introduced by each preset. These perturbations are applied only to the test set, which comprises 20% of the total dataset. Note that for Preset 2 (Merge), the percentage of affected clusters is much lower than the percentage of laundering edges perturbed because many clusters consist of only two nodes, which P2 will not be applied to.
Note that for Preset 1 (Inject), when the percentage is set to 50%, this will cause 66.6% of edges to be modified, since if you select the edge `A` $\rightarrow$ `B`, you end up with `A`$\rightarrow$`X`$\rightarrow$`B`, so two edges are modified for every one that isn't. Additionally, the number of edges perturbed and the change in the number of laundering transactions remain constant because the exact number of edges that will be perturbed is deterministic (two for every one edge that is selected). For merging, it depends on the number of incoming and outgoing nodes attached to each merged node. Splitting is perturbed at the node level, rather than the edge level, so cloning a node with many incoming and outgoing edges significantly increases the number of perturbed edges.

B Configuration Presets

```
{
  "inputfile": "data/
      unformatted_test_split.csv",
  "outputfile": "data/augmented.csv",
  "modification_filter": "any",
  "visualization": 0,
  "seed": 50,
  "actions": [
    {
      "type": "INTERMEDIARY",
      "settings": {
        "percent": 0.2,
        "perturb_by_edges": true
      }
    }
  ]
}
```

(a) Preset1.json

```
{
  "inputfile": "data/
      unformatted_test_split.csv",
  "outputfile": "data/augmented.csv",
  "modification_filter": "any",
  "visualization": 0,
  "seed": 50,
  "actions": [
    {
      "type": "MERGE",
      "settings": {
        "percent": 0.2,
        "perturb_by_edges": true
      }
    }
  ]
}
```

(b) Preset2.json

```
{
  "inputfile": "data/
      unformatted_test_split.csv",
  "outputfile": "data/augmented.csv",
  "modification_filter": "any",
  "visualization": 0,
  "seed": 50,
  "actions": [
    {
      "type": "SPLIT",
      "settings": {
        "percent": 0.2,
        "perturb_by_edges": false,
        "split_depth": 1
      }
    }
  ]
}
```

(c) Preset3.json

```
{
  "inputfile": "data/
      unformatted_test_split.csv",
  "outputfile": "data/augmented.csv",
  "modification_filter": "modified",
  "visualization": 0,
  "seed": 50,
  "actions": [
    {
      "type": "INTERMEDIARY",
      "settings": {
        "percent": 0.6,
        "perturb_by_edges": true,
        "intermediary_depth": 1
      }
    },
    {
      "type": "SPLIT",
      "settings": {
        "percent": 0.4,
        "perturb_by_edges": false,
        "split_depth": 1
      }
    }
  ]
}
```

(d) Preset4.json

Figure 7: Configuration presets used for node perturbation.