POLITECNICO DI TORINO

Master's Degree in Computer Engineering – AI & Data Analytics

Gang Prediction in Graphs for Anti Money Laundering Detection

This thesis was conducted in collaboration with Chalmers University of Technology and Technical University of Munich

Supervisors:

Francesca Pistilli
Javad Aliakbari
Alexandre Graell I Amat
Rawad Bitar

Candidate:

Pasquale Bianco

Academic Year 2024/2025

Abstract

In this thesis, we propose an iterative graph coarsening–learning framework to detect money laundering gangs in bank transaction networks. Instead of classifying individual accounts, our goal is to identify groups of accounts that act together to hide illicit activities.

We begin by providing a mathematical definition of gangs in the context of transaction graphs, modeling them as dense and highly interactive subgraphs that share structural and behavioral similarities, such as transaction patterns. The proposed method progressively reduces the complexity of the transaction graph through embedding–based coarsening, where nodes with similar structural and semantic patterns are merged into super-nodes representing potential groups. This process preserves the most relevant information while reducing noise and computational cost and also helping the model to generalize over different nodes and structures.

After each coarsening step, we apply supervised learning on the reduced graph to detect suspicious groups. To guide the training, we design a coarsening–aware loss function that combines classification objectives with a consistency term linking each super-node to its constituent accounts. This encourages coherent group representations and improves generalization across different graph scales.

We validate our approach on both synthetic and real-world bank transaction datasets and compare it against state-of-the-art graph neural network and semi-supervised embedding methods.

Keywords: Money Laundering, Graph Coarsening, Graph Reduction, Graph Neural Networks, Graph Learning, Semi-Supervised Learning, Anomaly Detection

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The global challenge of money laundering

Money Laundering (ML) is defined as the process of concealing the illicit origin of criminal proceeds so that the funds appear legitimate [1]. This “cleaning" conversion is critical for criminals, enabling them to enjoy profits from crimes such as drug trafficking, corruption, or fraud without exposing their source [2, 3]. It is commonly described in three stages: placement (introducing illicit funds into the financial system), layering (executing complex transactions to obscure the trail), and integration (returning laundered money into the legal economy).

The scale of money laundering is vast. The United Nations Office on Drugs and Crime (UNODC) estimates that between 2-5% of global Gross Domestic Product (GDP) is laundered each year, amounting to trillions of US dollars in illicit flows [4]. Similarly, the Financial Action Task Force (FATF) has emphasized that money laundering erodes the integrity of financial systems worldwide [5]. An International Monetary Fund (IMC) report underlines that illicit finance is a necessary component of organized crime, transnational corruption, and terrorism financing, thereby threatening governance and macroeconomic stability [6].

Illicit flows also deprive governments of essential revenues: it has been estimated that more than $1.3 trillion in illicit capital left sub-Saharan Africa since 1980, funds that could have supported development and poverty alleviation [7]. Beyond fiscal losses, money laundering distorts competition, inflates asset markets such as real estate, and worsens inequality by empowering the corrupt at the expense of law-abiding citizens.

Tackling ML is therefore critical not only to crime prevention but also to financial transparency, fair markets, and sustainable economic growth.

1.2 Traditional AML methods and limitations

Anti-Money Laundering has long been a regulatory and technological priority. Early systems relied heavily on *rule-based* monitoring, flagging transactions above fixed thresholds or involving high-risk jurisdictions [8]; some institutions employ statistical anomaly detection, identifying transactions or accounts that deviate significantly from typical patterns (outliers), as potential laundering signals. In addition, Know Your Customer (KYC) procedures and suspicious activity reports have been mandated by regulators worldwide. Later approaches introduced *statistical anomaly detection* and *supervised learning* models, leveraging transaction-level features and classifiers such as decision trees, logistic regression, or neural networks [9].

However, these methods suffer from serious drawbacks. Rule-based systems are rigid and can be "blind" to novel or sophisticated schemes. Criminals can evade fixed rules by adapting their behavior – for example, breaking down large sums into many small "smurfing" deposits below reporting thresholds, or routing transactions through a web of intermediate accounts to avoid simple pattern triggers. As a result, purely rule-based monitoring often produces an overwhelming number of alerts, the vast majority of which turn out to be false positives. In practice, over 90% of alerts generated by bank Anti-Money Laundering systems are false positives. This imposes large operational costs, as banks employ hundreds of compliance analysts to review alerts manually, yet less than 1% of global illicit flows are ultimately intercepted or seized [10].

Anomaly detection methods, while more flexible, conflate "unusual" with "illicit," producing numerous false alarms from legitimate atypical behavior [9]. Supervised models, on the other hand, face data scarcity and imbalance: confirmed laundering cases are rare and labels are delayed, while criminals continuously adapt patterns. Moreover, all these methods struggle with scalability: modern banks process millions of transactions daily, making real-time monitoring computationally demanding. In summary, current AML systems trade off between efficiency and effectiveness, motivating the exploration of fundamentally new approaches.

1.3 Graph–based approaches in AML

A promising direction in recent research is to leverage graph-based machine learning for money laundering detection. The rationale is that financial transactions do not occur in isolation; instead, they form networks of interactions between entities (individuals, businesses, accounts). We can naturally represent a collection of transactions as a graph, where nodes correspond to accounts (or transactions themselves) and edges represent money flows. This network view provides crucial context: a single transaction that might not raise flags on its own could be part of a suspicious web of many transfers. By analyzing connections and structures in the transaction graph, one can detect complex patterns that traditional methods might miss. Indeed, graph analytics techniques have been shown to reveal “complex webs of money laundering practices that might be overlooked by legacy systems” [11].

Graph-based machine learning, especially Graph Neural Networks (GNNs), has thus gained traction in the AML domain. In a GNN model, the features of each node (e.g. an account’s attributes or transaction statistics) are combined with information from its neighbors in the graph, allowing the model to learn propagating patterns of suspicious activity. A seminal example is the work of Weber et al. (2019), who introduced the large “Elliptic” transaction graph from the Bitcoin cryptocurrency network and applied Graph Convolutional Networks to classify illicit vs. licit transactions [12]. Their results showed that incorporating relational information through graph learning improved detection performance over traditional classifiers. More recently, Cheng et al. (2023) proposed a *group-aware* deep graph learning framework that detects organized laundering by modeling community-level behavior, showing state-of-the-art performance on UnionPay bank card data [13]. Surveys confirm that graph-based methods now dominate AML research, reflecting their ability to reduce false positives by leveraging context and capturing hidden laundering schemes [11].

Despite these advances, challenges remain. Transaction graphs are massive (millions of nodes/edges) and dynamic (streaming transactions). Running GNNs at scale is expensive, and financial data is fragmented across institutions. These issues underscore the need for scalable, generalizable graph-based methods.

1.4 Position of the thesis

This thesis addresses the detection of money laundering in large-scale transaction networks by uniting graph-based learning with a lightweight, scalable preprocessing step. In particular, we employ *graph coarsening*: a family of techniques that compresses large graphs into smaller surrogates while retaining the most important structural information, so that learning can operate efficiently on reduced graphs and still capture the patterns that matter for detection.

In practice, coarsening serves two purposes here:

- it improves computational tractability (time and memory) for graph neural models;
- by abstracting away fine-grained noise, encourages the model to focus on stable, higher-level structures that are characteristic of laundering schemes,

avoiding, then, outliers.

We propose an **iterative framework** that alternates between coarsening and graph learning. Coarsening first yields a compact representation of the transaction network; learning then identifies suspicious activity and propagates those signals back to the original resolution, where the representation can be refined, and the cycle repeated. Iterating across levels allows the method to capture both local motifs and global organization without overburdening computation on the full graph. The remainder of this thesis is structured as follows:

- **Chapter 2** provides an in-depth analysis of money laundering dynamics, examining its behavioral patterns and identifying how these patterns can be effectively exposed through data-driven analysis.
- **Chapter 3** introduces the mathematical formulation of the problem, defining the notion of a *gang* and formalizing the proposed task of *gang prediction* within financial transaction networks.
- **Chapter 4** presents the core research contributions and proposed methodology. It is divided into three main sections:
  - **Section 4.1** introduces the concept of graph coarsening and explains how it can be employed to reduce large-scale transaction networks while preserving their key structural properties.
  - **Section 4.2** extends the coarsening approach to include adaptation to the feature-space, providing a mechanism to merge nodes based on *semantic* similarity.
  - **Section 4.3** describes the proposed graph learning framework, including the coarsening-aware loss function and the iterative refinement process that alternates between learning and coarsening.
- **Chapter 5** reports the experimental results obtained on both the Cora dataset (used for validation) and the AMLGentex synthetic dataset, specifically designed for money laundering detection.

The implementation of all experiments and models described in this thesis is publicly available at: `https://github.com/whiitex/Gang-Prediction`.

Chapter 2

Money Laundering Patterns

The *AMLGentex* simulator and benchmarking framework [14] recently provided a principled basis for studying feature engineering and graph-based detection in a controlled setting. This chapter summarises empirical, structural and temporal properties common to illicit transactions, catalogues the features used by researchers and practitioners to detect them and analyses how criminals deliberately modify these features to evade detection. The discussion draws on AMLGentex and its references, real-world studies of criminal networks and group behaviour [13, 15], synthetic simulators such as PaySim [16], AMLSim and its successors, as well as graph-based feature extraction libraries [17, 11].

The goal is twofold:

1. to provide an empirical and methodological foundation for later chapters that develop graph coarsening and machine learning approaches for Anti-Money Laundering;
2. to offer practical recommendations for computing, windowing and stress-testing features.

The analysis is structured around the stages of the laundering process and the observable consequences in transaction records, followed by a taxonomy of features, evasion tactics, and robustness considerations.

2.1 Money Laundering Stages

Money Laundering typically unfolds in three stages: *placement*, *layering* and *integration* [14], covered in the following subsections. Each stage leaves characteristic footprints in transaction networks. Many simulators explicitly model these phases because they influence the design of features.

2.1.1 Placement

In the placement stage, criminals introduce illicit cash or digital funds into the financial system. Placement often occurs through deposits, purchases of prepaid cards or mobile money transfers. In the PaySim simulator, transaction types such as *Cash-In* (deposit to a mobile money account) and *Debit* (cash-out to a bank account) increase or decrease an account's balance, whereas *Payment* and *Transfer* represent peer-to-peer transactions [16]. Because large deposits draw attention, launderers break funds into smaller amounts ("smurfing") to stay below regulatory reporting thresholds. Placement events therefore manifest as bursts of small incoming transactions spread across multiple accounts. The AMLGentex dataset models these behaviors by generating multiple agents that deposit illicit funds over a time window and by calibrating deposit amounts to mimic real distributions [14].

2.1.2 Layering

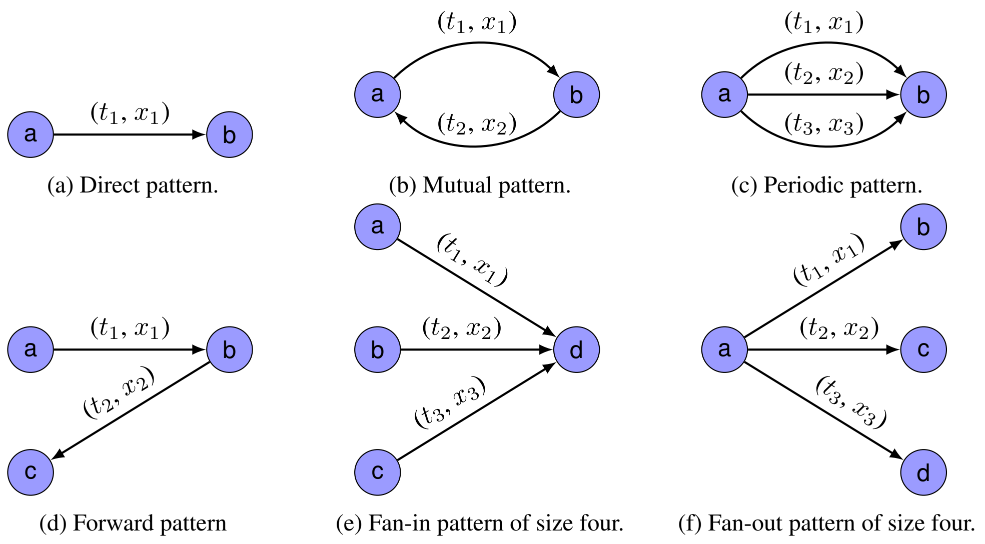
Layering obscures the origin of funds through complex transactions, multiple accounts and intermediaries. Star and chain patterns dominate this phase: the *star pattern* diffuses money from a source node to many recipients, while the *chain pattern* transmits funds through a sequence of bridge nodes. These patterns correspond to the "fan-out" and "fan-in" motifs in earlier simulators (e.g. AMLSim and AMLSim 2.0) and are observed in real networks [13]. Criminals may combine chains and stars into *cyclic* structures where funds flow back and forth among the same accounts, making detection difficult [13]. Layering is thus characterized by many small transfers among tightly connected accounts, high path lengths and temporal interleaving of legitimate transactions to camouflage illicit flows. Further discussion about transaction patterns in Section 2.2.

2.1.3 Integration

Integration reintroduces laundered funds into the legitimate economy through purchases of goods, investments, or business activity. Transfers to merchants, payments for services, and investments appear in transaction logs. In AMLGentex the integration phase is simulated by connecting laundering accounts back into the general transaction network, often involving cross-institution transactions spanning multiple banks or regions [14]. Integration tends to produce transactions with amounts similar to normal economic activities but preceded by suspicious layering behavior.

2.2 Common Illicit Transaction Patterns

Detecting money laundering requires understanding the structural motifs that criminals employ. Table **??** lists common typologies derived from AMLGentex [14], AMLSim [18] and the group-aware [13] study. All patterns are described in Tab. **??**. The normal typologies available are shown in Fig. 2.1 and the money laundering ones are shown in Fig. 2.2. The fan-in, fan-out, cycle, scatter-gather and gather-scatter patterns can be of arbitrary size set by the user. Moreover, each typology is related to a timing schema chosen by the user.

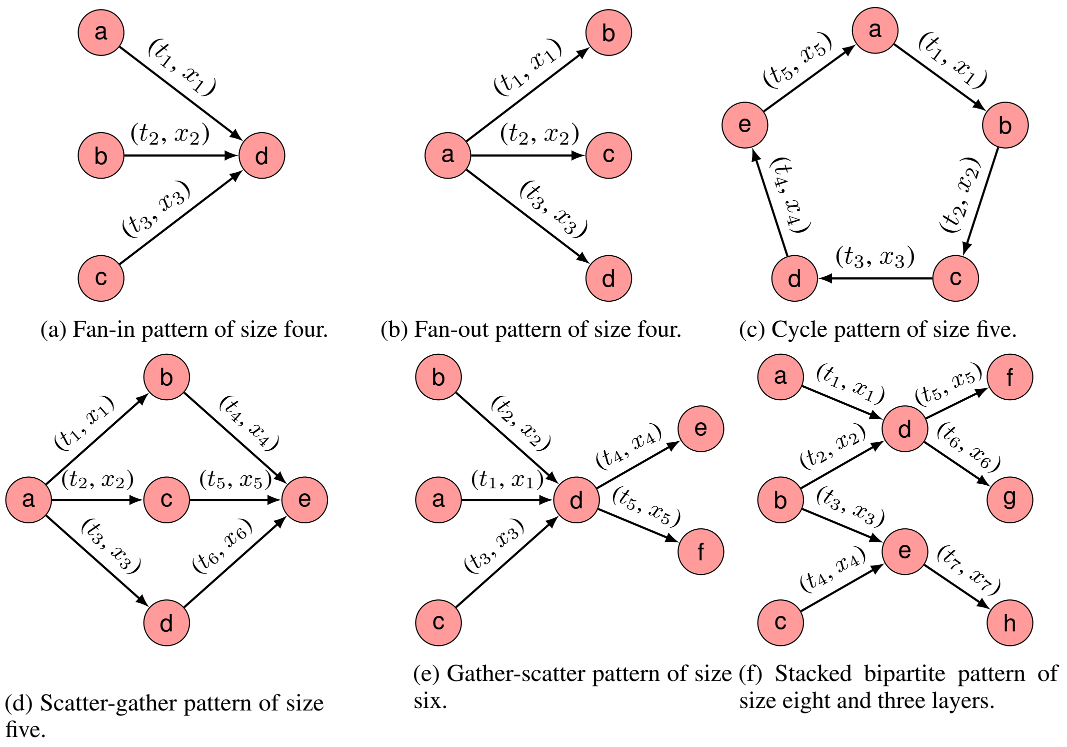


(a) Direct pattern. (b) Mutual pattern. (c) Periodic pattern.
(d) Forward pattern (e) Fan-in pattern of size four. (f) Fan-out pattern of size four.

Figure 2.1: Normal patterns. Each transaction is associated with a time stamp t and an amount x.

The normal patterns and alert patterns reflect typical and suspicious behaviors, respectively. Notably, patterns such as fan-in and fan-out appear in both normal and alert categories, emphasizing how criminals often mimic legitimate activity.

The frequency of each motif varies across simulators and real data. AMLGentex extends earlier models by simulating cycles, scatter–gather, gather–scatter and random walks in addition to basic fan-in/out patterns.



(a) Fan-in pattern of size four. (b) Fan-out pattern of size four. (c) Cycle pattern of size five. (d) Scatter-gather pattern of size five. (e) Gather-scatter pattern of size six. (f) Stacked bipartite pattern of size eight and three layers.

Figure 2.2: Laundering patterns. Each transaction is associated with a time stamp t and an amount x.

2.3 Empirical Statistics of Illicit Transactions

Beyond structural motifs, statistical properties provide clues about illicit activity. The most prominent one is the strong unbalancing of data: the positive class (laundering) is *rare* relative to benign activity. This manifests at multiple granularities:

2.3.1 Transaction/edge level

The share of laundering-related transactions is typically well below 1% of all transactions. Publicly documented synthetic benchmarks mirror this reality for research: for example, SynthAML reports on the order of 10^7 transactions with only $\mathcal{O}(10^4)$ AML *alerts* [20]. Other recent synthetic generators likewise target sub-percent positive rates to emulate production monitoring workloads and, at the same time, allow customizing the amount of positive transactions [21, 14].

Typology	Alias	Description
Fan-out	Star	A single source distributes funds to many downstream accounts; used during placement or layering to disperse proceeds. [13].
Fan-in	Reverse star	Multiple sources send funds to one central account. In layering, criminals gather funds in a single node before further dispersal or cash-out.
Scatter–gather	–	Combined fan-out followed by fan-in: funds scatter from a source to multiple intermediaries and are gathered again into one account.
Gather–scatter	–	Reverse of scatter–gather: multiple sources combine funds then redistribute them among many accounts. Useful for mixing illicit funds with legitimate money.
Cycle	–	Funds circulate among a closed loop of accounts and eventually return to the origin. Cycles often combine multiple chains and stars [13].
Chain	Linear	A sequence of transfers from one account to the next. Chains reflect layering and can be arbitrarily long.
Random walk	Random	Funds move randomly among controlled accounts; used to mimic legitimate activity.
Temporal cycle	–	Cycle where edges occur over an extended period with time lags; emphasizes temporal ordering [17].
Smurfing	Burst	Splitting a large transaction into many small transfers below reporting thresholds. Often implemented as a fan-out into multiple deposit accounts.

Table 2.1: Common illicit transaction typologies and motifs. The descriptions synthesize definitions from AMLGentex and related simulators[14, 17, 13, 19].

2.3.2 Account/node level

Only a small fraction of accounts participate in laundering within a given observation window. Modern generators such as AMLGentex make this fraction explicitly configurable (via typology counts/sizes, reuse probabilities, timing schemes, and label noise) exactly to reproduce the severe skew observed by institutions [14].

Another important statistic is the evolution of account balances. AMLGentex shows that normal accounts exhibit relatively stable balances or gradually increasing trajectories, whereas laundering accounts display sudden spikes and subsequent rapid drops corresponding to placement, layering, and integration events [14].

2.3.3 Why this matters

Such imbalance means that naive thresholds on amounts or volumes are insufficient as a learning method (positive and negative distributions overlap). Therefore, learning objectives and metrics must prioritize performance under skew, and also calibration to operational false-positive is essential. At the macro level, the rarity of confirmed cases is consistent with policy studies and official estimates that indicate that only a tiny fraction of criminal proceeds are ultimately detected or recovered [22].

2.4 Evasion Tactics and Countermeasures

Criminals actively adjust their behavior to evade detection. Understanding evasion tactics informs feature design and model robustness. This section categories common evasion strategies by feature group and proposes countermeasures.

2.4.1 Transaction–level evasion

- **Smurfing and micro–structuring** To avoid triggering amount-based thresholds, launderers divide large sums into many small transactions (smurfing). They exploit different transaction types, such as Cash-In via prepaid cards, peer-to-peer transfers, or merchant payments, to diversify flow channels [16].

  Countermeasure: regulators can aggregate transactions over sub-windows and compute cumulative sums. AMLGentex uses sub-windowing ($m = 4$ by default) so that consecutive small deposits still produce high cumulative amounts and counts.

- **Channel hopping and currency arbitrage** Criminals switch between channels (mobile wallet, online banking, cash) and convert funds across currencies to confuse detection. They may route funds through low-risk merchant categories with small fees.

2.4.2 Account–level and temporal evasion

- **Burst smoothing.** Launderers space out illicit transfers with legitimate activity to smooth transaction counts and avoid peaks. They adjust inter-arrival times to mimic normal patterns.

  Countermeasure: adopt measures such as the coefficient of variation or wavelet-based burst analysis [15]. Use sub-windowing to detect transient spikes and compare patterns relative to account cohorts rather than fixed thresholds.

- **Synthetic identities and account rotation** Criminals open new accounts with clean KYC profiles, then close or abandon them after laundering. They transfer funds through networks of "straw persons" or shell companies to reduce days-in-bank and phone-change indicators.

2.4.3 Structural and graph–based evasion

- **Network fragmentation** To avoid detection of star or chain motifs, criminals split flows across multiple smaller patterns and rotate accounts to keep in-/out-degrees low. They use random walk patterns or scatter–gather motifs across time to hinder motif counts.

  Countermeasure: incorporate multi-hop pattern detection and community detection. Graph motif counting with time windows captures scatter-gather patterns, while approximate centrality identifies subtle hubs [17, 11]. Moreover, graph coarsening that preserves motifs can maintain detection power when scaling to large networks.

- **Cross-institution masking** Patterns spanning multiple FIs may evade detection if each bank observes only partial data.

  Countermeasure: regulators encourage collaborative detection (e.g., federated learning) and adopt cross-FI graph representation. AMLGentex shows that patterns often involve up to seven FIs [14], emphasizing the need for data sharing, but privacy is the main issue here.

2.4.4 Label and supervision evasion

- **Adversarial noise and false positives** Criminals may intentionally file false SARs on benign accounts to mislead models or generate noise to degrade learning.

  Countermeasure: incorporate robust statistics, model uncertainty and adversarial training; evaluate models under simulated label noise as AMLGentex does (class noise, typology noise and neighbor noise) [14]. Use semi-supervised learning to harness unlabeled data.

2.5 Feature Robustness and Evaluation

Selecting time windows and evaluation metrics is critical for robust detection. AMLGentex uses windows of T days (e.g., $T = 30$) divided into m equal sub-windows ($m = 4$ by default) to compute temporal aggregates. Sub-windowing captures bursty behavior and mitigates the effect of concept drift [14], that occurs when the patterns that distinguish normal and suspicious behavior change over time. Rolling windows can further track evolving patterns and adapt to temporal drift. When computing graph features, one must decide whether to include the entire history or only recent transactions; temporal motifs require ordering. Robustness also involves sensitivity analysis. Features should be tested against changes in window length, sub-window count, noise injection and sampling.

Evaluation must account for extreme class imbalance.

Chapter 3

Task

In this chapter, we define the task of gang prediction. Since the current literature has not yet established a clear definition of what constitutes a gang, we also provide our own definition. To this end, we analyzed several papers and examined transaction graphs such as the one shown in Fig. 3.1.

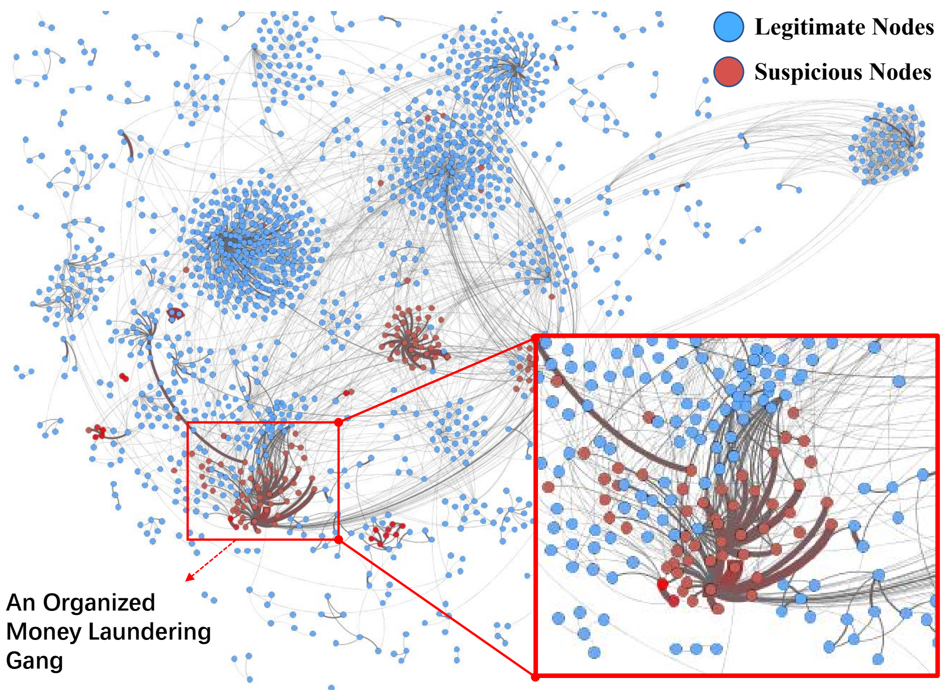


Figure 3.1: The layout of a real-world money laundering user transaction graph. The records being reported as high risk of money laundering are colored red, while the legitimate ones are colored blue.

3.1 Gang Definition

Before introducing the formal task, it is important to clarify what a *gang* is and what it is not. We can state with confidence that a gang:

1. is **not equivalent to a community**: communities are typically defined by dense structural connectivity, whereas gangs can be sparse, overlapping, or even partially disconnected subgraphs;
2. **cannot be identified solely from graph structure**: topological patterns alone are insufficient to discriminate between legitimate and illicit behaviors.

This is illustrated in Fig. 3.1, where the highlighted subgraph contains both suspicious and legitimate nodes within the same community. As discussed in Section 2.2, illicit transaction motifs such as *fan-in/fan-out* often coexist with legitimate flows. Therefore, while a gang may exhibit certain structural regularities, it is essentially defined by its coordinated behavioral intent, that is, by a group of accounts collaborating to disguise the illicit origin of funds. Consequently, detecting gangs requires the joint modeling of both *structural* and *feature-based* information (e.g., transaction attributes, timing, and node features), rather than relying solely on connectivity.

At the same time, we can formulate several reasonable assumptions about the typical properties of a gang:

1. **Densely connected:** members of a gang tend to exchange money frequently, forming local clusters of high transaction intensity;
2. **Small in size:** gangs usually involve a limited number of accounts, as larger structures are more easily detectable and difficult to coordinate;
3. **Multi-account ownership:** a single individual may control multiple accounts, which participate jointly in laundering schemes;
4. **Composed of specific transaction motifs:** typical structural patterns include *fan-in/fan-out*, *cycles*, or *scatter–gather* operations, corresponding to the classical stages of money laundering (placement, layering, integration);
5. **Threshold on total transferred value:** gangs often operate with amounts exceeding a certain limit, to achieve meaningful laundering outcomes .

These assumptions are consistent with empirical evidence from real financial networks and prior AML research. For instance, Cheng et al. [13] observed that organized criminal groups in the UnionPay dataset appear as *locally dense clusters of limited size*, often formed through repeated interactions among the same subset of accounts (this is the reason behind the thicker edges). Similarly, synthetic frameworks such as AMLGENTEX [14] model laundering typologies using recurrent motifs like *fan-in/fan-out*, emphasizing the structured yet small-scale nature of such groups. Therefore, although the precise characteristics of gangs may vary, they generally exhibit strong internal connectivity, repeated cooperative behavior, and distinguishable transactional patterns that differentiate them from random legitimate communities.

3.2 Gang Prediction Task

This section formalizes the *gang prediction* problem addressed in this thesis. We work on a (possibly dynamic) transaction network where vertices are bank accounts and edges are monetary transfers. The goal is to learn a model that detects *organized, coordinated* illicit activity at the level of groups of nodes, going beyond purely structural cues.

3.2.1 Problem setting.

Let $G = (V, E, X)$ be a directed, weighted, attributed graph: V is the set of accounts; $E \subseteq V \times V$ is the set of transactions (optionally with timestamps and amounts); and X collects node and edge features (e.g., degree statistics, country codes, device/IP metadata, KYC-derived attributes). When data are temporal, we assume an aggregation operator $\mathrm{AGG}(w)$ over a time window w that produces a snapshot $G_w = (V_w, E_w, X_w)$.

3.2.2 Prediction targets.

Depending on the operational use case and ground truth availability, we consider two granularities:

- **Node–level (account) prediction.** Given G_w, learn $f_\theta : V_w \to [0,1]$ such that $f_\theta(v)$ is the probability that account v participates in a laundering gang within window w. We call it *fine prediction*, because it is applied at a fine level.
- **Group–level (gang) prediction.** Given a collection of candidate subgraphs $\mathcal{S} = \{S \subseteq V_w\}$ learn $g_\theta : \mathcal{S} \to [0,1]$ such that $g_\theta(S)$ scores the likelihood that S is a gang. We call it *coarse prediction*, because it is applied at a coarse level.

These choices are motivated by the real behavior of financial institutions, which in practice issue alerts on individual accounts or on small groups of related accounts rather than on single transactions. Moreover, in the case of group–level classification, the considered groups correspond to the same entities previously defined as *gangs*, representing sets of accounts that cooperate to disguise illicit money flows.

Chapter 4

Method

In this chapter, we describe the methodology for our graph representation learning pipeline, which crucially incorporates a graph coarsening technique with strong theoretical guarantees. The goal of graph coarsening is to reduce the number of vertices in the graph while preserving its essential structural properties. By working with a smaller graph, we can significantly accelerate computations (e.g., graph embeddings or graph neural network operations) without substantially altering the outcome. Moreover, reducing the graph helps remove outliers and promotes better generalization during the learning phase.

Loukas (2019) [23] introduced a principled coarsening framework based on *Restricted Spectral Approximation* (RSA), which provides theoretical guaranties on the preservation of spectra, cuts and embeddings. However, Loukas' method relies on predefined candidate sets without considering data-driven node representations. To enhance coarsening, we integrate GNNs to learn node embeddings that guide the selection of candidate sets. This leads to an iterative procedure:

- Use GNN embeddings to decide which nodes to merge.
- Perform one-level coarsening based on embedding similarity and local variation.
- Retrain the GNN on the coarsened graph to refine embeddings.
- Jointly optimize classification and coarsening consistency losses.

4.1 Graph Coarsening

In this section we will analyze the coarsening method proposed by Loukas [24, 23, 25] and how it is adapted to our use case.

Graph coarsening refers to the process of producing a smaller graph (with fewer nodes) from an original graph by aggregating or contracting groups of original nodes into super–nodes. Formally, given an original graph $G = (V, E, W)$ with $|V| = N$ number of nodes, $|E| = M$ number of edges and W the edge weight, a coarsened graph $G_c = (V_c, E_c, W_c)$ is constructed where $|V_c| = N_c \ll N$ and each super–node in V_c corresponds to a subset of nodes in V. In general, graph coarsening can be viewed as defining a mapping $f : \mathbb{R}^N \to \mathbb{R}^{N_c}$ that associates each fine–node with a corresponding super–node.

In the rest of this section, we first explain the principles underlying this approach Sec. 4.1.1, then describe the algorithmic procedure to obtain the coarsened graph Sec. 4.1.8, and finally discuss the properties (spectral and cut metrics) that are approximately preserved in the coarsening process.

4.1.1 Graph Reduction

Consider a positive semidefinite (PSD) matrix $L \in \mathbb{R}^{N \times N}$ whose sparsity structure captures the connectivity structure of a connected weighted undirected graph G. In other words, $L(i, j) \neq 0$ only if e_{ij} is a valid edge between vertices v_i and v_j. Moreover, let x be an arbitrary vector of size N. We can study the following generic reduction scheme:

Graph Reduction Scheme: Commence by setting $L_0 = L$ and $x_0 = x$ and proceed according to the following two recursive equations:

$$L_\ell = P_\ell^{\mp} L_{\ell-1} P_\ell^{+}, \quad x_\ell = P_\ell x_{\ell-1},$$

where $P_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ are matrices with more columns than rows, $\ell = 1,2, \dots, c$ is the level of the reduction, symbol $+$ (resp. $\mp$) denotes the pseudoinverse (resp. transposed pseudoinverse), and N_ℓ is the dimensionality at level ℓ such that $N_0 = N$ and $N_c = n \ll N$.Vector x_c is lifted back to $\mathbb{R}^N$ by recursion $\tilde{x}_{\ell-1} = P_\ell^{+} \tilde{x}_\ell$, where $\tilde{x}_c = x_c$.

Graph reduction thus involves a sequence of $c + 1$ graphs:

$$G = G_0 = (V_0, E_0, W_0) \quad G_1 = (V_1, E_1, W_1) \quad \cdots \quad G_c = (V_c, E_c, W_c) \tag{4.1}$$

of decreasing size $N = N_0 > N_1 > \cdots > N_c = n$, where the sparsity structure of L_l matches that of graph G_l and each vertex in G_l represents one of more vertices in G_{l-1}. We can define the reduction ratio simply by:

$$r = 1 - \frac{n}{N} \tag{4.2}$$

and we can express the quantities in a more compact form (avoiding the multi–level setup):

$$x_c = Px, \quad L_c = P^{\mp} L P^{+} \quad \text{and} \quad \tilde{x} = \Pi x, \tag{4.3}$$

where $P = P_c \cdots P_1$, $P^+ = P_1^+ \cdots P_c^+$, $\Pi = P^+ P$. The rationale of this scheme is that vector $\tilde{x}$ should be the best approximation of x given P in an ℓ_2-sense, which is a consequence of the property that Π is a projection matrix.

4.1.2 Properties of Reduced Graphs

Below we list some nice property of the above scheme.

Property 1. Π is a projection matrix.

Property 2. If L is PSD, then so is L_c.

We can further consider the spectrum of the two matrices. Sort the eigenvalues of L as $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$ and denote by $\tilde{\lambda}_k$ the k-th largest eigenvalue of L_c and $\tilde{u}_k$ the associated eigenvector.

It turns out that the eigenvalues $\tilde{\lambda}$ and λ are interlaced.

Theorem 3. For any P with full-row rank and $k = 1, \ldots, n$, we have:

$$\gamma_1 \, \lambda_k \;\leq\; \tilde{\lambda}_k \;\leq\; \gamma_2 \, \lambda_{k+N-n},$$

with $\gamma_1 = \lambda_1\big((PP^\top)^{-1}\big)$ and $\gamma_2 = \lambda_n\big((PP^\top)^{-1}\big)$, respectively the smallest and largest eigenvalue of $(PP^\top)^{-1}$ (Discussion and proof omitted, refer to [23].)

One can also say something about the action of L_c on vectors.

Property 4. For every vector $x \in \text{im}(\Pi)$, one has:

$$x_c^\top L_c x_c = x^\top \Pi L \Pi x = x^\top L x \quad \text{and} \quad \tilde{x} = \Pi x = x.$$

In other words, reduction maintains the action of L of every vector that lies in the image of Π.

4.1.3 Coarsening as a Type of Graph Reduction

Coarsening is a type of graph reduction abiding to a set of constraints that render the graph transformation interpretable. More precisely, in coarsening one selects for each level ℓ a surjective (i.e., many-to-one) map $\varphi_\ell : V_{\ell-1} \to V_\ell$ between the original vertex set $V_{\ell-1}$ and the smaller vertex set V_ℓ. We refer to the set of vertices $V_{\ell-1}^{(r)} \subseteq V_{\ell-1}$ mapped onto the same vertex v'_r of V_ℓ as a *contraction set*:

$$V_{\ell-1}^{(r)} = \{v \in V_{\ell-1} : \varphi_\ell(v) = v'_r\} \tag{4.4}$$

Moreover, we constraint it a little more by requiring that the subgraph of $G_{\ell-1}$ induced by each contraction set $V_{\ell-1}^{(r)}$ is connected.

It is easy to deduce that contraction sets induce a partitioning of $V_{\ell-1}$ into N_ℓ subgraphs, each corresponding to a single vertex of V_ℓ. Every reduced variable thus corresponds to a small set of adjacent vertices in the original graph and coarsening amounts to a scaling operation. An appropriately constructed coarse graph aims to capture the global problem structure, whereas neglected details can be recovered in a local refinement phase.

Coarsening can be placed in the context of Scheme 1 by restricting each P_ℓ to lie in the family of coarsening matrices, defined next:

Definition 5 (Coarsening matrix) Matrix $P_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ is a coarsening matrix w.r.t. graph $G_{\ell-1}$ if and only if it satisfies the following two conditions:

1. It is a surjective mapping of the vertex set, meaning that if $P_\ell(r,i) \neq 0$ then $P_\ell(r',i) = 0$ for every $r' \neq r$. In other words, each vertex can belong to exactly one coarsening set.
2. It is locality preserving, equivalently, the subgraph of $G_{\ell-1}$ induced by the non–zero entries of $P_\ell(r,:)$ is connected for each r.

Proposition 6 (Easy inversion) The pseudo–inverse of a coarsening matrix P_ℓ is given by $P_\ell^+ = P_\ell^\top D^{-2}$, where D is the diagonal matrix with $D(r,r) = \|P_\ell(r,:)\|_2$.

Proposition 6 carries two consequences.

- Coarsening can be done in linear time. Each coarsening level (both in the forward and backward directions) entails multiplication by a sparse matrix.
- Both P_ℓ and P_ℓ^+ have only $N_\ell - 1$ non-zero entries meaning that $O(N)$ and $O(M)$ operations suffice to coarsen respectively a vector and a matrix L whose sparsity structure reflects the graph connectivity.

4.1.4 Laplacian Consistent Coarsening

A further restriction that can be imposed is that coarsening is consistent w.r.t. the Laplacian form. Suppose that L is the combinatorial Laplacian of G defined as:

$$L(i,j) = \begin{cases} d_i & \text{if } i = j \\ -w_{ij} & \text{if } e_{ij} \in E \\ 0 & \text{otw} \end{cases}$$

where w_{ij} is the weight associated with edge e_{ij} and d_i the weighted degree of v_i. For equally weighted graphs (so considering W only contains unit entries), the combinatorial Laplacian matrix can be defined as $L = D - A$, where D is a diagonal matrix built from degrees of vertices and A is the adjacency matrix.

Let's analyze a toy example taken from the graphs in Fig. 4.1.

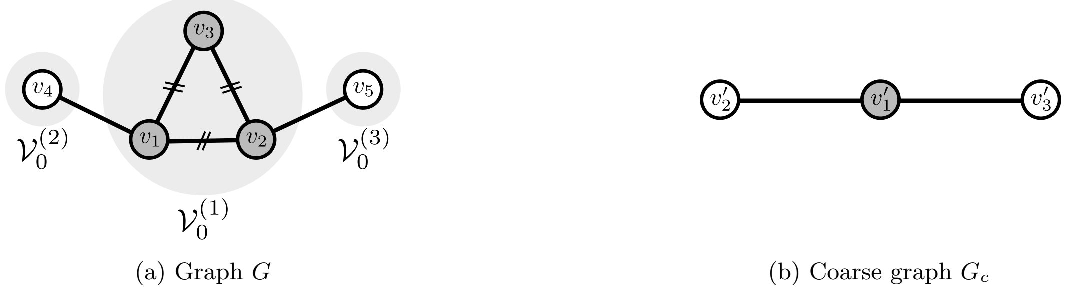


Figure 4.1: Toy coarsening example. Grey discs denote contraction sets. The first three vertices of G forming contraction set V_0^1 are contracted onto vertex v_1'. All other vertices remain unaffected.

Let G_0 be a simple path over five vertices as shown in Fig.1, with combinatorial Laplacian

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 \\ -1 & -1 & 2 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{bmatrix}.$$

Suppose a single level with contraction sets $V_0^{(1)} = \{v_1, v_2, v_3\}$, $V_0^{(2)} = \{v_4\}$,

$V_0^{(3)} = \{v_5\}$. The choice for P_1, P_1^+ and Π is

$$P_1 = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad P_1^+ = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \Pi = P_1^+ P_1 = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

so that coarsening results in

$$L_c = P_1^{\mp} L P_1^+ \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, \qquad x_c = P_1 x \begin{bmatrix} \frac{x(1)+x(2)+x(3)}{3} \\ x(4) \\ x(5) \end{bmatrix}.$$

Finally, when lifted x_c becomes

$$\tilde{x} = P_1^+ x_c = \Pi x = \begin{bmatrix} \frac{x(1)+x(2)+x(3)}{3} \\ \frac{x(1)+x(2)+x(3)}{3} \\ \frac{x(1)+x(2)+x(3)}{3} \\ x(4) \\ x(5) \end{bmatrix}$$

4.1.5 Restricted Spectral Approximation

This section formalizes how should a graph be reduced such that fundamental structural properties (e.g., its spectrum and cuts) are preserved. Inspired by work in graph sparsification, Loukas introduces a measure of approximation that is tailored to graph reduction. The new definition implies strong guarantees about the distance of the original and coarsened spectrum and gives conditions such that the cut structure of a graph is preserved by coarsening.

One way to define how close a PSD matrix L is to its reduced counterpart is to establish an isometry guarantee w.r.t. the following induced semi-norms:

$$\|x\|_L = \sqrt{x^\top L x} \quad \text{and} \quad \|x_c\|_{L_c} = \sqrt{x_c^\top L_c x_c} \tag{4.5}$$

Ideally, one would hope that there exists $\varepsilon > 0$ such that

$$(1-\varepsilon)\|x\|_L \leq \|x_c\|_{L_c} \leq (1+\varepsilon)\|x\|_L \tag{4.6}$$

for all $x \in \mathbb{R}^N$. But since the dimension changes this is impossible except trivially $\varepsilon = 1$. To carry out a meaningful analysis, one needs to consider a subspace of dimension $k \leq n$ and aim to approximate the behavior of L solely within it. Hence we define the Restricted Spectral Approximation:

Definition 7 (Restricted Spectral Approximation) Let $\mathcal{R}$ be a k-dimensional subspace of $\mathbb{R}^N$. Matrices L_c and L are $(\mathcal{R}, \varepsilon)$-similar if there exists $\varepsilon \geq 0$ such that

$$\|x - \tilde{x}\|_L \leq \varepsilon \|x\|_L \tag{4.7}$$

for all $x \in \mathcal{R}$, where $\tilde{x} = P^+ P x = \Pi x$.

Corollary 8. If L_c and L are $(\mathcal{R}, \varepsilon)$-similar, then:

$$(1-\varepsilon)\|x\|_L \leq \|x_c\|_{L_c} \leq (1+\varepsilon)\|x\|_L \ \text{ for all } x \in \mathcal{R} \tag{4.8}$$

To this effect, consider the smallest k eigenvalues and corresponding eigenvectors and define the following matrices

$$U_k \in R^{N \times k} = [u_1, u_2, \ldots, u_k] \qquad \text{and} \qquad \Lambda_k = diag(\lambda_1, \lambda_2, \ldots, \lambda_k)$$

In the next Theorem we show that $\mathcal{R} = span(U_k)$ suffices to guarantee that the first k eigenvalues and eigenvectors of L and L_c are aligned.

Theorem 9 (Eigenvalue approximation) If L_c and L are (U_k, ε_k)-similar, then

$$\gamma_1 \lambda_k \leq \tilde{\lambda}_k \leq \gamma_2 \frac{(1+\varepsilon_k)^2}{1 - \varepsilon_k^2(\lambda_k/\lambda_2)} \tag{4.9}$$

whenever $\varepsilon_k^2 < \lambda^2/\lambda_k$. Crucially, the bound depends on λ_k instead of λ_{k+N-n} and thus can be significantly tighter than the one given by Theorem 3. In Appendix A a sample script in python to empirically prove this theorem is provided.

4.1.6 Decoupling Levels and the Variation Cost

Guaranteeing restricted spectral approximation w.r.t. subspace $\mathcal{R}$ boils down to minimizing at each level ℓ the *variation cost*:

$$\sigma_\ell = \|\Pi_\ell^\perp A_{\ell-1}\|_{L_{\ell-1}} \tag{4.10}$$

The matrix $A_{\ell-1}$ is carrying

1. the information of target subspace $\mathcal{R}$ at the current level
2. accumulated reductions from prior levels.

The precise recursive definition is $A_{\ell-1} = B_{\ell-1}(B_{\ell-1}^\top L_{\ell-1} B_{\ell-1})^{+1/2}$ with $B_{\ell-1} = P_{\ell-1} B_{\ell-2}$ and $B_0 = A_0$ (and $A_0 = VV^\top L^{+1/2}$ when $\mathcal{R} = \text{span}(V)$). Moreover: /

Proposition 10. L_c and L are $(\mathcal{R}, \varepsilon)$-similar with $\varepsilon \leq \prod_{\ell=1}^{c}(1+\sigma_\ell) - 1$. / Crucially, the previous result makes it possible to design a multi-level coarsening greedily, by starting from the first level and optimizing following levels one at a time. Given a total target threshold ε' and an initial cumulative threshold $\varepsilon_0 = 0$, the variation cost at each level ℓ must be chosen carefully to ensure that the overall approximation error does not exceed ε'. Specifically, the cumulative error after level ℓ is given by: /

$$\varepsilon_\ell \leq \prod_{k=1}^{\ell}(1+\sigma_k) - 1 = (1+\sigma_\ell)(1+\varepsilon_{\ell-1}) - 1 \leq \varepsilon' \;\Rightarrow\; \sigma_\ell \leq \frac{1+\varepsilon'}{1+\varepsilon_{\ell-1}} - 1 \quad (4.11)$$

/ From this relation, we can derive an upper bound on the allowable variation cost σ_ℓ for level ℓ, which ensures that the cumulative error remains below the target threshold:

$$\sigma' \leq \frac{1+\varepsilon'}{1+\varepsilon_{\ell-1}} - 1 \quad \text{and} \quad \varepsilon_\ell = (1+\varepsilon_{\ell-1})(1+\sigma_\ell) - 1 \quad (4.12)$$

Thus, every local variation algorithm operates in the following manner:

Algorithm 1 Multi-level coarsening [23]

```
Require: Combinatorial Laplacian $L$, threshold $\varepsilon'$, and target size $n$.
1: Set $\ell \leftarrow 0$, $L_\ell \leftarrow L$, and $\varepsilon_\ell \leftarrow 0$.
2: while $N_\ell \geq n$ and $\varepsilon_\ell < \varepsilon'$ do
3:     $\ell \leftarrow \ell + 1$
4:     Coarsen $L_{\ell-1}$ using Alg. 2 with threshold $\sigma' = \frac{1+\varepsilon'}{1+\varepsilon_{\ell-1}} - 1$ and target size $n$
5:     Let $L_\ell$ be the resulting Laplacian of size $N_\ell$ with variation cost $\sigma_\ell$
6:     $\varepsilon_\ell \leftarrow (1+\varepsilon_{\ell-1})(1+\sigma_\ell) - 1$
7: end while
8: return $L_\ell$
```

Alg. 1 returns a Laplacian matrix L_c that is $(\mathcal{R}, \varepsilon)$-similar to L with $\varepsilon \leq \varepsilon_c \leq \varepsilon'$, where c is the last level ℓ. On the other hand, setting ε' to a large value ensures that the same algorithm always attains the target reduction at the expense of loose restricted approximation guarantees.

4.1.7 Decoupling Contraction Sets and Local Variation

Suppose that $\Pi_C^{\perp}$ is the (complement) projection matrix obtained by contracting solely the vertices in set C, while leaving all other vertices in $V_{\ell-1}$ untouched:

$$\left[\Pi_C^{\perp} x\right](i) = \begin{cases} x(i) - \sum_{v_j \in C} \frac{x(j)}{|C|}, & v_i \in C, \\ 0, & \text{otw} \end{cases} \tag{4.13}$$

Here, for convenience, the level index is suppressed.

Definition 11 (Local Variation Cost) The local cost of contracting the variation set C at level ℓ is defined as:

$$\mathrm{cost}_\ell(C) = \frac{\left\|\Pi_C^{\perp} A_{\ell-1}\right\|_{L_C}^2}{|C|-1} \tag{4.14}$$

with$\|\cdot\|_{L_C}$ the operator norm induced by $\|x\|_{L_C}$. This isolates the interaction to the subgraph induced by $C \cup \mathcal{N}(C)$.

The following proposition shows us how to decouple the contribution of each contraction set to the variation cost.

Proposition 12 (Decoupling bound) The variation cost is bounded by:

$$\sigma_\ell^2 \leq \sum_{C \in \mathcal{P}_\ell} \|\Pi_C^{\perp} A_{\ell-1}\|_{L_C}^2 \;=\; \sum_{C \in P_\ell} \mathrm{cost}_\ell(C)(|C|-1) \tag{4.15}$$

where $\mathcal{P}_\ell = \{\mathcal{V}_{\ell-1}^{(1)}, \ldots, \mathcal{V}_{\ell-1}^{(N_\ell)}\}$ is the family of contraction sets of level ℓ. Hence, one can minimize an upper bound on σ_ℓ by selecting disjoint sets with small local costs.

4.1.8 Local Variation Coarsening Algorithm

Building on the above principles, Loukas [23, 24, 25] develops a family of greedy algorithms for graph coarsening, referred to as local variation coarsening algorithms. The high-level procedure is as follows.

Starting from a *candidate family* $\mathcal{F}_\ell = \{C_1, C_2, C_3, \ldots\}$, that is, an appropriately sized family of candidate contraction sets, the strategy will be to search for a small *contraction family* $\mathcal{P}_\ell = \{\mathcal{V}_{\ell-1}^{(1)}, \ldots, \mathcal{V}_{\ell-1}^{(N_\ell)}\}$ with minimal variation cost σ_ℓ ($\mathcal{P}_\ell$ is valid if it partitions $\mathcal{V}_{\ell-1}$ into N_ℓ contraction sets). Every coarse vertex $v_r' \in \mathcal{V}_\ell$ is then formed by contracting the vertices in $\mathcal{V}_{\ell-1}^{(r)}$. On the other hand, since any

Algorithm 2 Single-level coarsening by local variation [23]

```
Require: Combinatorial Laplacian $L_{\ell-1}$, threshold $\sigma'$, and target size $n$.
 1: Form a family of candidate sets $\mathcal{F}_\ell = \{C_1, C_2, C_3, \ldots\}$
 2: $N_\ell \leftarrow N_{\ell-1}$, $marked \leftarrow \emptyset$, $\sigma_\ell^2 \leftarrow 0$
 3: Sort $\mathcal{F}_\ell$ in terms of increasing $\text{cost}_\ell(C)$
 4: while $|\mathcal{F}_\ell| > 0$ and $N_\ell > n$ and $\sigma_\ell \leq \sigma'$ do
 5:     Pop the candidate set $C$ of minimal cost $s$ from $\mathcal{F}_\ell$
 6:     if all vertices of $C$ are not marked and $\sigma' \geq \sqrt{\sigma_\ell^2 + (|C|-1)s}$ then
 7:         $marked \leftarrow marked \cup C$
 8:         $\mathcal{P}_\ell \leftarrow \mathcal{P}_\ell \cup C$
 9:         $N_\ell \leftarrow N_\ell - |C| + 1$
10:         $\sigma_\ell^2 \leftarrow \sigma_\ell^2 + (|C|-1)s$
11:     end if
12: end while
13: Form the $N_\ell \times N_{\ell-1}$ coarsening matrix $P_\ell$ based on $\mathcal{P}_\ell$
14: return $L_\ell \leftarrow P_\ell^\top L_{\ell-1} P_\ell$ and $\sigma_\ell$
```

permissible contraction family $\mathcal{P}_\ell$ should be a partitioning of $V_{\ell-1}$, choosing C precludes us from selecting any C' with which it intersects.

Two natural choices for the candidate family are:

- *Edge-based candidates:* each candidate set is simply an edge $\{u, v\}$ (a pair of adjacent vertices). Contracting an edge means merging its two endpoints.

- *Neighborhood-based candidates:* each candidate set is a small neighborhood, for example all vertices within distance 1 of a given vertex, including the vertex itself. This allows contracting larger clusters, potentially yielding a higher reduction in one step at the cost of more complex updates.

Alg. 2 sequentially examines candidate sets from $\mathcal{F}_\ell$, starting from those with minimal cost. To decide whether a candidate set C will be added to $\mathcal{P}_\ell$, the algorithm asserts that all vertices in C are unmarked, essentially enforcing that all contraction sets are disjoint. Accordingly, as soon as C is added to $\mathcal{P}_\ell$, all vertices that are in C become marked. The algorithm terminates if either the target reduction is achieved, the error threshold is exceeded, or no candidate sets are left.

In Appendix D, we provide an example of the coarsening evolution on a small graph to build intuition about the process.

4.2 Semantic Variation Cost

The local-variation framework in Secs. 4.1.1–4.1.8 controls the *structural* distortion induced by coarsening through the graph Laplacian L and the target subspace carried by $A_{\ell-1}$. Because L is derived from the adjacency, these guarantees do not explicitly account for the *semantic* information present in node features and representations. To complement structure, we introduce a semantic analogue of the variation cost based on node embeddings computed by any GNN.

Embeddings Let Φ_θ denote a GNN defined on the current graph $G_{\ell-1}$ with feature matrix $X_{\ell-1} \in \mathbb{R}^{N_{\ell-1} \times d_0}$. Its embedding matrix is

$$\mathcal{H}_{\ell-1} \;=\; \Phi_\theta(G_{\ell-1},\, X_{\ell-1}) \in \mathbb{R}^{N_{\ell-1} \times d}$$

where each row $\mathcal{H}_{\ell-1}(v,:)$ summarizes both feature and multi–hop structural context for node v. Further discussion in Section 4.3.

As seen in Chapter 4.1.7, matrix A carries:

1. the information of target subspace $\mathcal{R}$ at the current level
2. accumulated reductions from prior levels.

At the same time, matrix $\mathcal{H}$ carries:

1. information of the subspace X, representing the space of node features
2. accumulated reductions from prior levels, because embeddings are recomputed after each coarsening level.

Hence, we can make matrix A be replaced by $\mathcal{H}$, contributing to the analogous *semantic* variation cost at level ℓ defined as:

$$\sigma_\ell^{\text{sem}} \;=\; \left\| \Pi_\ell^\perp \, \mathcal{H}_{\ell-1} \right\|_{L_{\ell-1}} \tag{4.16}$$

At the same time, we can also look deeper in each coarsening level:

Definition 13 (Semantic Local Variation Cost) The semantic local cost of contracting the variation set C at level ℓ is defined as:

$$\text{cost}_\ell^{\text{sem}}(C) \;=\; \frac{\left\| \Pi_C^\perp \mathcal{H}_{\ell-1} \right\|_{L_C}^2}{|C|-1} \;=\; \frac{\operatorname{tr}\!\left(H^\top \Pi_C^\perp L_C \Pi_C^\perp H \right)}{|C|-1} \tag{4.17}$$

Proposition 14 (Decoupling bound – semantic) Let $\mathcal{P}_\ell = \{\mathcal{V}_{\ell-1}^{(1)}, \dots, \mathcal{V}_{\ell-1}^{(N_\ell)}\}$ be a valid family of disjoint contraction sets at level ℓ. Then the semantic variation cost is bounded by:

$$\left(\sigma_\ell^{\mathrm{sem}}\right)^2 \le \sum_{C \in \mathcal{P}_\ell} \|\Pi_C^\perp \mathcal{H}_{\ell-1}\|_{L_C}^2 \;=\; \sum_{C \in P_\ell} \mathrm{cost}_\ell^{\mathrm{sem}}(C)(|C|-1) \tag{4.18}$$

4.3 Graph Learning

This section gives a compact, self-contained overview of graph neural networks (GNNs) for semi-supervised node classification and explains how learning is integrated with the multi-level coarsening pipeline of Section 4.1 and with the semantic variation cost discussed in Section 4.2.

4.3.1 Problem Setup and Notation

Let $G = (V, E, X)$ with $|V| = N$, adjacency $A \in \mathbb{R}^{N \times N}$, (combinatorial) Laplacian $L = D - A$, node-feature matrix $X \in \mathbb{R}^{N \times d_0}$, and labels $Y \in \mathbb{R}^N$. Only a subset $\mathcal{L} \subset V$ is labeled; the goal is to predict labels for all nodes by exploiting both features and the graph structure.

Throughout the coarsening pipeline we denote by (G_ℓ, L_ℓ, X_ℓ) the graph, Laplacian and features at level ℓ, and by P_ℓ the level-ℓ coarsening matrix with pseudoinverse P_ℓ^+ and projection matrix $\Pi_\ell = P_\ell^+ P_\ell$ (see Section 4.1). We write $N_\ell = |V_\ell|$, with $N = N_0 > N_1 > \cdots > N_c = n$.

4.3.2 Message Passing and Node Embeddings

A GNN layer propagates and transforms information along the edges. In a generic Message Passing Neural Network (MPNN), the hidden representation $h_i^{(k)} \in \mathbb{R}^{d_k}$ of node i at layer k is

$$h_i^{(k)} \;=\; \sigma\Big(W^{(k)} \cdot \mathrm{AGG}\Big(\{\, h_j^{(k-1)} : j \in \mathcal{N}(i) \cup \{i\} \,\}\Big)\Big), \tag{4.19}$$

with learnable weights $W^{(k)}$, nonlinearity σ and a permutation-invariant aggregator AGG (e.g., sum/mean/attention). We set $\mathcal{H}^{(0)} = X$ and stack K layers to obtain $\mathcal{H}^{(K)} \in \mathbb{R}^{N \times d_K}$, whose i-th row $h_i^{(K)}$ is the *embedding* of node i.

A widely used instance of the message-passing framework is the Graph Convolutional Network (GCN) [26]. The GCN layer performs a first-order approximation of spectral graph convolutions by propagating features through a normalized adjacency matrix. With self-loops $\tilde{A} = A + I$ and degree matrix $\tilde{D} = \operatorname{diag}(\tilde{A}\mathbf{1})$, the update rule is

$$\mathcal{H}^{(k+1)} = \sigma\Big(\tilde{D}^{-1/2}\,\tilde{A}\,\tilde{D}^{-1/2}\,\mathcal{H}^{(k)}W^{(k)}\Big), \qquad \mathcal{H}^{(0)} = X, \tag{4.20}$$

where σ denotes a nonlinear activation function and $W^{(k)}$ is a learnable weight matrix. This operation linearly transforms node features, aggregates them from neighboring nodes with symmetric normalization to stabilize training, and applies a nonlinear transformation. Other architectures such as GraphSAGE, GAT, APPNP, and GIN follow the same message-passing template in (4.19), differing mainly in their aggregation and normalization functions.

Embeddings and their role. The matrix $\mathcal{H} = \mathcal{H}^{(K)}$ summarizes feature and multi-hop structural context. In our pipeline, embeddings are: (i) the classifier's input (via a final linear head), and (ii) a semantic signal to guide coarsening (Section 4.3.5), since nodes that are close in $\mathcal{H}$ are good merge candidates provided the local-variation cost remains small.

4.3.3 Prediction Head and Semi-supervised Loss

Given $\mathcal{H}^{(K)}$, we map to logits $Z \in \mathbb{R}^{N\times C}$ and probabilities p_i:

$$Z \;=\; \mathcal{H}^{(K)}W_{\text{out}}, \qquad p_i \;=\; \operatorname{softmax}(Z_i). \tag{4.21}$$

We train with a *masked* cross-entropy on the labeled set $\mathcal{L}$:

$$\mathcal{L}_{\text{cls}} \;=\; \frac{1}{|\mathcal{L}|}\sum_{i\in\mathcal{L}} \operatorname{CE}\Big(Y_i,\, p_i\Big). \tag{4.22}$$

To address class imbalance (typical in AML), one may use class weights α_c in (4.22) or focal loss. In addition, weight decay is applied via the optimizer (AdamW).

4.3.4 Learning on Coarsened Graphs and Lifting

At level ℓ, we train the GNN on (L_ℓ, X_ℓ) to obtain embeddings $\mathcal{H}_\ell = \Phi_\theta(G_\ell, X_\ell)$ and logits $Z_\ell \in \mathbb{R}^{N_\ell\times C}$. When supervision is given at the *fine* level, we lift coarse predictions back using the projector induced by P_ℓ:

$$\tilde{Z}_{\ell-1} \;=\; P_\ell^{+}\, Z_\ell \;\in\; \mathbb{R}^{N_{\ell-1}\times C}, \qquad \mathcal{L}^{(\ell)}_{\text{cls}} \;=\; \frac{1}{|\mathcal{L}|}\sum_{i\in\mathcal{L}} \operatorname{CE}\Big(Y_i,\, \operatorname{softmax}(\tilde{Z}_{\ell-1}[i])\Big). \tag{4.23}$$

This keeps the classifier trained where labels actually live, while benefiting from reduced computation at coarse levels. After training, we can propagate predictions across levels by composing the lifts, using $P = P_c \cdots P_1$ and $P^{+} = P_1^{+} \cdots P_c^{+}$.

4.3.5 Embedding–Driven Coarsening and Similarity

Let $\mathcal{H}_{\ell-1} = \Phi_\theta(G_{\ell-1}, X_{\ell-1})$ be the current embeddings and let $C \subseteq V_{\ell-1}$ be a candidate contraction set. As seen in Section 4.2 We define a *semantic variation cost* analogue that prefers contracting nodes already close in embedding space:

$$\mathrm{cost}_\ell^{\mathrm{sem}}(C) = \frac{\left\|\Pi_C^\perp \mathcal{H}_{\ell-1}\right\|_{L_C}^2}{|C|-1} = \frac{\mathrm{Tr}\Big(\mathcal{H}_{\ell-1}^\top \Pi_C^\perp L_C \Pi_C^\perp \mathcal{H}_{\ell-1}\Big)}{|C|-1}. \tag{4.24}$$

Candidates are accepted if they are disjoint, have small semantic cost, and do not exceed the maximum threshold ϵ', optionally enforcing label consistency when some nodes in C are labeled.

As an additional constraint, we allow a group of nodes to be coarsened together only if they satisfy a predefined similarity threshold. In particular, we use the cosine similarity between node embeddings to quantify how semantically close two nodes are. Given two node representations $h_i, h_j \in \mathbb{R}^d$, their cosine similarity is defined as:

$$\mathrm{sim}_{\cos}(h_i, h_j) = \frac{h_i^\top h_j}{\|h_i\|_2 \, \|h_j\|_2}. \tag{4.25}$$

This metric measures the angle between the two embedding vectors, being independent of their magnitude and sensitive only to their orientation in the feature space. The cosine similarity takes values in $[-1,1]$ and can be interpreted as:

$$\mathrm{sim}_{\cos}(h_i, h_j) = \begin{cases} 1, & \text{if they are perfectly aligned (identical direction)} \\ 0, & \text{if they are orthogonal (no correlation)} \\ -1, & \text{if they point in opposite directions.} \end{cases}$$

By enforcing a minimum cosine similarity threshold, we ensure that only nodes with highly correlated feature representations, thus likely belonging to the same semantic or structural neighborhood, are merged during the coarsening process. In Appendix B a sample script in python to implement the cosine similarity.

4.3.6 Coarsening-Aware Loss Function

The overall training objective combines two complementary components: the standard classification loss, which ensures predictive accuracy on labeled nodes, and a coarsening-aware regularization term, which enforces consistency between the learned embeddings and the hierarchical structure produced by graph coarsening. Formally, the total loss is defined as:

$$\mathcal{L}_{\mathrm{total}} = \mathcal{L}_{\mathrm{cls}} + \lambda \, \mathcal{L}_{\mathrm{coarse}}, \tag{4.26}$$

where λ is a weighting factor controlling the influence of the coarsening term.

Classification loss. The classification objective is implemented as a standard cross-entropy loss over the labeled nodes $\mathcal{L}$:

$$\mathcal{L}_{\text{cls}} = -\frac{1}{|\mathcal{L}|} \sum_{i \in \mathcal{L}} \sum_{c=1}^{C} Y_{ic} \log p_{ic}, \tag{4.27}$$

where Y_{ic} is the one-hot encoded true label for node i and class c, and $p_{ic} = \text{softmax}(Z_i)_c$ is the predicted class probability obtained from the output logits Z in (4.21). This term drives the model to correctly classify labeled nodes based on their learned embeddings.

Coarsening loss. To integrate structural information derived from coarsening, we define a regularization term that aligns the learned embeddings with the coarse-level grouping. Let $\varphi(v)$ denote the supernode into which node v is merged after coarsening (we omit the level index for simplicity), and let $\hat{S}_{ij} = \text{sim}_{\text{cos}}(h_i, h_j)$ be the cosine similarity between the embeddings of nodes i and j. The coarsening loss encourages embeddings of nodes within the same supernode to be similar, while discouraging similarity across different supernodes:

$$\mathcal{L}_{\text{coarse}} = \mathcal{L}_{\text{within}} + \mathcal{L}_{\text{neg}} = \sum_{i,j \,:\, \varphi(v_i) = \varphi(v_j)} (1 - \hat{S}_{ij}) + \sum_{i,j \,:\, \varphi(v_i) \neq \varphi(v_j)} \hat{S}_{ij} \tag{4.28}$$

The first term promotes *intra-supernode compactness*, driving embeddings of merged nodes to align closely in the feature space ($\hat{S}_{ij} \approx 1$), while the second term promotes *inter-supernode separation*, penalizing similarity between nodes belonging to different coarse groups ($\hat{S}_{ij} \approx 0$). This joint objective ensures that the GNN learns representations that are both discriminative for classification and consistent with the structural abstractions induced by the coarsening process.

However, this loss function has a computational complexity of $\mathcal{O}(N^2)$, where N is the number of nodes in the graph. Therefore, we need to improve it to make it usable for large-scale graphs, such as transaction networks.

Let us consider a contraction set $V_{\ell-1}^{(r)}$. For each pair of nodes within this set, we should compute their cosine similarity and add it (with a negative sign) to the coarse loss:

$$\mathcal{L}_{\text{within}}^{(r)} = \sum_{i \,:\, V_{\ell-1}^{(r)}} \sum_{j \,:\, V_{\ell-1}^{(r)}} 1 - \frac{h_i^\top h_j}{\|h_i\|_2 \, \|h_j\|_2} = \sum_{i \,:\, V_{\ell-1}^{(r)}} \sum_{j \,:\, V_{\ell-1}^{(r)}} 1 - h_i^\top h_j \tag{4.29}$$

where the denominator is omitted because the embeddings are already normalized. Moreover, let us consider the embedding of the coarse node v_r, into which the elements of the contraction set $V_{\ell-1}^{(r)}$ are merged. As already shown in the toy example in Sec. 4.1.4, the embedding of node v_r is:

$$h_{v_r} = \frac{1}{|V_{\ell-1}^{(r)}|} \sum_{i\,:\,V_{\ell-1}^{(r)}} h_i$$

and:

$$h_{v_r}^\top h_{v_r} = \frac{1}{|V_{\ell-1}^{(r)}|^2} \sum_{i\,:\,V_{\ell-1}^{(r)}} \sum_{j\,:\,V_{\ell-1}^{(r)}} h_i^\top h_j$$

Since the optimization task is invariant to constant factors, we can ignore them and notice that this is the exact same expression as $\mathcal{L}_{\text{within}}^{(r)}$. Therefore, we can compute it simply by obtaining $\mathcal{H}_C = P\mathcal{H}$ and taking the mean of the trace of $\mathcal{H}_C^\top \mathcal{H}_C$. The complexity of this operation is $\mathcal{O}(K^2)$, where K is the number of nodes that have been coarsened, but since $K \ll N$, it does not significantly slow down the process.

Lastly, we can also consider a way to speed up the computation of $\mathcal{L}_{\text{neg}}$. In this case, we can approximate this term by evaluating it only on a random sample of nodes. This idea is conceptually related to the principle of stochastic optimization introduced by Robbins and Monro [27], where gradients (or, more generally, objective terms) are estimated on random subsets of data to achieve efficient and unbiased approximations of the full objective.

Let $K \ll N$ be the number of coarsened nodes, and let $\mathcal{V} = \{V_\ell^{(r_i)}\}$ denote the family of contraction sets. Then, the probability of sampling two nodes from the same contraction set is:

$$p = \frac{\sum_{V\in\mathcal{V}} \binom{|V|}{2}}{\binom{N}{2}} \leq \frac{\binom{K}{2}}{\binom{N}{2}} \approx \frac{K^2}{N^2}.$$

Since we assumed $K \ll N$, this probability is very low. Hence, we can approximate $\mathcal{L}_{\text{neg}}$ by computing it over a limited sample of nodes. In Appendix C, we provide a sample Python script illustrating how to implement the coarsening-aware loss in PyTorch.

4.3.7 Training Loop per Level

The iterative learning procedure alternates between embedding computation, graph coarsening, and supervised training. At each level, the model refines its representations while progressively simplifying the graph structure. The process can be summarized as follows:

1. **Embedding computation:** the GNN learns node embeddings on the current graph $G_{\ell-1}$ using the message-passing layers defined in Eq. (4.19)–(4.20).
2. **Graph coarsening:** candidate merges are selected based on the learned embeddings and the semantic cost; accepted contractions define the projection matrix P_ℓ and produce the coarse graph G_ℓ.
3. **Training with custom loss:** the model is trained on the coarsened graph using the coarsening–aware loss Eq. (4.28), which balances classification accuracy and embedding consistency.
4. **Iteration:** the process repeats on G_ℓ until the desired reduction ratio or the maximum number of levels is reached.

This hierarchical training loop allows embeddings to guide the coarsening decisions in a task-driven manner, while the coarsened graphs provide a simplified yet semantically meaningful structure for subsequent learning. Algorithm 3 presents the full iterative procedure.

Algorithm 3 GNN Iterative Training with Custom Loss

```
Require: Adj matrix $\mathbf{A}$, Node features $\mathbf{X}$, Labels $\mathbf{Y}$, Number of levels num_levels,
    Number of epochs per level num_epochs_per_lev
 1: // initialization
 2: Create GNN model $f_\theta$
 3: Split nodes into train/val/test indices
 4: // iterative learning
 5: for level = 1 to num_levels do
 6:     // 1. Generate node embeddings
 7:     $\mathbf{Z} \leftarrow f_\theta(\mathbf{A}, \mathbf{X})$
 8:     // 2. Coarsen the graph
 9:     $P$, $G_c \leftarrow$ apply_graph_coarsening($G$, $\mathbf{Z}$, $ratio$, $max\_cost$)
10:     // 3. Train on coarsened graph
11:     Extract features $\mathbf{X}_c$, adjacency $\mathbf{A}_c$, and labels $\mathbf{Y}_c$ from $G_c$
12:     for epoch = 1 to num_epochs_per_lev do
13:         Train $f_\theta$ on $G_c$ using the Coarsening–Aware Loss
14:     end for
15: end for
16: // Evaluate
17: Test on coarse graph $G_c$ (corse–level accuracy)
18: Propagate predictions to original graph and test (fine-level accuracy)
```

Chapter 5

Results

In this chapter we present the experimental evaluation of the proposed iterative learning framework. The experiments were designed with two main objectives. First, we aimed to validate the correctness and stability of the iterative coarsening–learning procedure on a well-known benchmark dataset, *Cora*, where standard graph neural network (GNN) baselines are available for comparison. Second, we evaluated the same pipeline on a synthetic anti–money laundering (AML) dataset generated with the AMLGentex framework [14], in order to assess its behavior on transaction networks that more closely resemble the application domain of our study.

5.1 Validation on the Cora dataset

The *Cora* citation network contains 2,708 scientific publications classified into seven categories, connected by 5,429 citation edges, and described by 1,433 binary attributes corresponding to the presence of specific words in each document. This dataset provides a well-established benchmark for validating graph representation learning methods under controlled conditions.

5.1.1 Experimental Setup

We used *Cora* to evaluate the general behavior of the iterative coarsening–learning pipeline. At each iteration, the graph was reduced using the *local variation* (LV) coarsening algorithms described in Chapter 4.1: one based on **neighborhood variation** and one based on **edge variation**. We tested different reduction ratios $r \in \{0.50, 0.75, 0.85, 0.95\}$, where r indicates the fraction of removed nodes. After each coarsening step, we trained the model using the *coarsening-aware loss* introduced in Chapter 4.3. Model accuracy was computed separately on the coarse graph and on the reconstructed fine graph to measure consistency across hierarchical levels.

Similarity threshold	**Epochs per level**	**# nodes**	**Accuracy coarse**	**Accuracy fine**
0.50	1	256	64.42	46.19
	2	220	52.77	23.43
	5	186	59.31	22.84
	10	187	57.39	22.80
0.75	1	592	63.08	75.99
	2	553	53.20	**84.13**
	5	515	41.43	79.59
	10	370	54.51	61.51
0.85	1	894	66.55	**86.48**
	2	847	64.22	**85.64**
	5	699	54.38	83.02
	10	534	53.51	83.35
0.95	1	1874	**82.16**	**87.96**
	2	1530	**78.64**	**86.04**
	5	1314	**74.21**	**86.92**
	10	1119	67.64	81.92

Table 5.1: Coarse vs fine accuracy in Cora for different similarity thresholds (50%, 75%, 85%, 95%), `num_epochs_per_lev` (1, 2, 5, 10) and `levels` = 100.

Table 5.1 summarizes the main numerical results. As observed, increasing the threshold results in a larger number of preserved nodes and a consistent improvement in both coarse and fine accuracies, because it becomes less likely for nodes to be similar (even if they belong to the same class). At low thresholds (e.g., 0.5), the model produces highly compressed graphs with fewer than ∼200 nodes but exhibits limited fine-level accuracy, below 50%.

In contrast, higher thresholds such as 0.85 and 0.95 yield substantially higher performance, reaching up to 82.16% on the coarse level and 87.96% on the fine level, even if coarsening more than half of the original graph! Interestingly, for these higher thresholds, the fine accuracy remains stable across multiple epoch settings (typically above 85%), suggesting that the coarsening process effectively preserves discriminative features. However, after a certain number of epochs (e.g., 10), both coarse and fine accuracies tend to slightly decrease, indicating potential overfitting or diminishing returns from additional optimization.

Overall, the results highlight a clear trade-off between graph compactness and classification accuracy, with thresholds around 0.85–0.95 and epochs around 1-5 offering the best balance.

5.1.2 Learning Dynamics

To analyze the optimization behavior, we visualize accuracies for different reduction ratios in two complementary views.

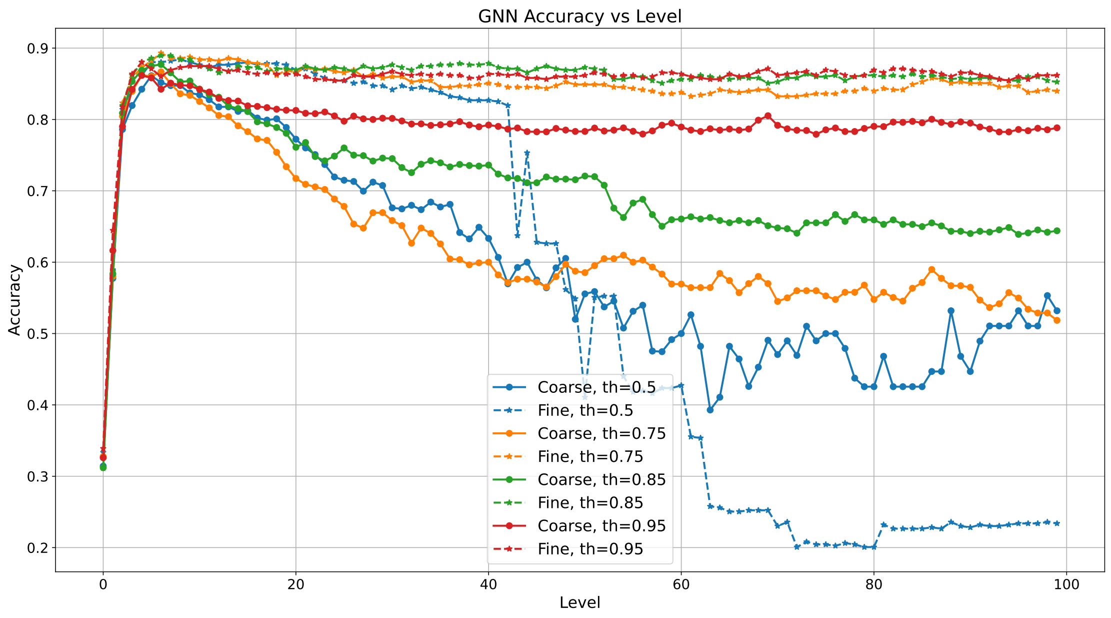


Figure 5.1: Coarse (solid) and fine (dotted) accuracies during training for different similarity thresholds (50%, 75%, 85%, 95%) and `num_epochs_per_lev` = 2.

In Figure 5.1 we can see the evolution of accuracy over iterations. As expected, the learning proceeds promisingly as far as the coarsening is kept low. The coarsening becomes more invasive when the similarity threshold is low, producing smaller graph which lose too much information, hence resulting in low accuracy.

The next figure plots accuracy against the number of nodes in the current (coarsened) graph. Moving to the right corresponds to progressively finer graphs (more nodes), hence the graph should be read from right to left, which is the direction of the learning.

Figure 5.2 shows that the coarsening is not making the learning impossible and the model manages to achieve good results even when the amount of nodes is

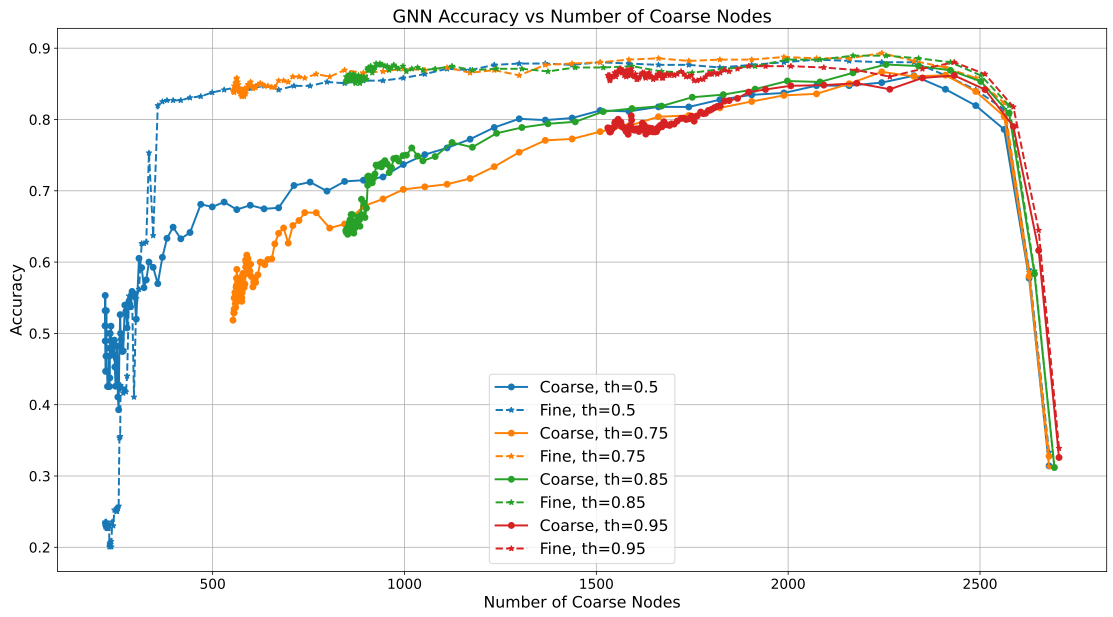


Figure 5.2: Coarse (solid) and fine (dotted) accuracies as a function of the *number of nodes* in the coarsened graph for different similarity thresholds (50%, 75%, 85%, 95%) and `num_epochs_per_lev` = 2.

reduced to about 25% of the original graph. This confirms that the reduced representations still preserve the key structural and feature information required for classification. In practice, this means that the proposed iterative pipeline is able to learn robust embeddings that generalize well across hierarchical levels, maintaining discriminative capacity even under severe dimensionality reduction.

5.2 AMLGentex Dataset Analysis

The AMLGentex dataset [14] is a synthetic benchmark specifically designed for research on anti–money laundering (AML) detection. It was developed to address the scarcity of open, realistic financial transaction data, which in practice is inaccessible due to privacy and regulatory constraints. The dataset aims to emulate the complex characteristics of real banking networks while maintaining full controllability and transparent ground truth.

5.2.1 Generation Process

AMLGentex is generated using an *agent-based simulation* framework that models both legitimate and illicit financial activities. The system represents a population of synthetic bank accounts as agents that interact through monetary transactions forming a dynamic, directed multigraph. Each transaction (u, v, a) corresponds to a money transfer from account u to account v with associated attributes a such as amount, timestamp, and transaction type.

The generation process integrates multiple components:

- **Blueprint network:** a scale-free base graph is created where the in-degree and out-degree distributions follow a Pareto law, reproducing the heterogeneity typically observed in real transaction networks.
- **Typologies:** predefined transaction patterns are injected into the network to simulate normal and suspicious behaviors (e.g., *fan-in*, *fan-out*, *cycle*, *scatter–gather*, *layering*). These typologies correspond to canonical money-laundering schemes described in financial intelligence literature.
- **Agent dynamics:** each node is assigned a demographic profile (age, income, spending behavior) and interacts over time according to probabilistic rules. Money inflow and outflow are controlled by a synthetic income distribution fitted to real salary statistics.
- **Stages of laundering:** the simulator reproduces the three classical AML phases (*placement*, *layering*, and *integration*), so that illicit transactions can be traced along multi-step propagation paths.

5.2.2 Structure and Features

At each discrete time step t, the dataset provides a transaction graph $G_t = (V_t, E_t, X_t, Y_t)$:

- V_t is the set of accounts active at time t;
- E_t is the multiset of transactions among them;
- X_t contains node-level features derived from both static customer information (age, income, type) and aggregated transactional statistics such as total inflow/outflow, number of counterparties, and transaction frequency;
- Y_t is the set of binary labels marking whether an account is involved in any laundering operation within a time window.

Graphs can be aggregated over windows of time to obtain static snapshots suitable for node classification. Each node's label is set to 1 if it participates in at least one suspicious transaction pattern during the aggregation window, otherwise 0.

5.3 Application to Synthetic AML Data

After validating the pipeline on Cora, we applied the same procedure to a synthetic anti–money laundering dataset generated with the AMLGentex framework [14]. This dataset models a realistic banking scenario where nodes correspond to accounts and edges to monetary transactions. Although the simulation captures key challenges of AML detection—such as class imbalance, temporal drift, and adaptive adversarial behaviour—the resulting classification problem is considerably harder than in academic benchmarks like Cora.

In this setting, the model was trained to predict whether each account participates in money-laundering activity, using node features derived from aggregated transaction statistics. Despite extensive hyperparameter tuning and multiple coarsening ratios, the model did not show meaningful convergence: both training and validation accuracies remained close to random guessing. This suggests that, while the iterative learning mechanism works well on graphs with well-separated classes and relatively smooth signals, it struggles in domains with extreme imbalance and weakly correlated features, such as money-laundering networks. In such cases, the difficulty is intrinsic to the data rather than the model, since illicit nodes are intentionally designed to mimic the behavior of normal accounts.

5.4 Discussion

The experiments highlight two main aspects of the proposed framework.

- the iterative coarsening and learning strategy can effectively retain classification accuracy on benchmark datasets even when large portions of the graph are collapsed, confirming the theoretical intuition that coarsening acts as a regularizer and improves generalization [23].
- its performance strongly depends on the signal-to-noise ratio of the underlying node features. In the AML case, where fraudulent nodes are extremely sparse and their features highly camouflaged, no consistent learning signal could be extracted.

These findings emphasize the importance of domain-specific representation learning and the potential need for additional supervision signals or relational priors.

Chapter 6

Conclusion and Future Work

The goal of this thesis was to design and evaluate an iterative learning framework capable of operating on hierarchical representations of graphs, with the long-term aim of applying it to the detection of money-laundering activities in financial transaction networks. The central hypothesis was that combining graph coarsening with a coarsening-aware learning strategy could improve generalization and scalability while preserving the expressive power of graph neural networks (GNNs).

6.1 Summary of Contributions

We proposed a framework that alternates between **graph coarsening** and **learning** phases, progressively reducing the graph size and refining node embeddings across levels. The process is guided by a *coarsening-aware loss*, explicitly enforcing consistency between coarse and fine representations. This allows information to flow between hierarchical levels and helps the model maintain predictive accuracy despite structural simplification.

To support and validate this approach, we:

- Reviewed and analyzed relevant literature on graph reduction and coarsening, including spectral and optimal transport–based methods.
- Implemented local-variation (LV) coarsening algorithms, both edge-based and neighborhood-based, providing a balance between structural fidelity and computational efficiency.
- Developed and tested a custom coarsening-aware loss to align representations across successive graph hierarchies.

- Conducted extensive experiments on the *Cora* dataset to verify the correctness and stability of the iterative framework.
- Applied the method to a synthetic anti–money laundering dataset (AMLGentex) to assess its performance in a more challenging, domain-specific context.

6.2 Future Work

Future developments of this research could focus on exploring more deeply how the coarsening framework proposed by Loukas [23] can be adapted to achieve meaningful results in the context of money-laundering detection.

One promising direction is to incorporate illicit transaction patterns as candidate contraction sets within the coarsening process. In this formulation, nodes that are known or suspected to participate in coordinated activities (e.g., accounts involved in the same laundering scheme) would be assigned a higher probability of being merged during coarsening. This would produce coarse representations that more explicitly encode group-level criminal behavior, rather than relying purely on structural similarity or spectral criteria.

Once such a domain-aware coarsening step is defined, a graph neural network could then be trained on the resulting hierarchical graphs to learn discriminative features not only at the individual node level, but also at the group or community level. This approach could help the model capture higher-order relational dependencies typical of money-laundering operations—such as small, densely connected clusters of coordinated accounts—while maintaining computational efficiency.

Bibliography

[1] United Nations Office on Drugs and Crime. *Money Laundering: Overview.* `https://www.unodc.org/unodc/en/money-laundering/overview.html`. 2025 (cit. on p. 1).

[2] Doug Hopton. *Money laundering: a concise guide for all business.* Routledge, 2020 (cit. on p. 1).

[3] Michael Levi. «Money laundering and its regulation». In: *The Annals of the American Academy of Political and Social Science* 582.1 (2002), pp. 181–194 (cit. on p. 1).

[4] United Nations Office on Drugs and Crime. *Estimating Illicit Financial Flows Resulting from Drug Trafficking and Other Transnational Organized Crimes.* Research Report October 2011. Accessed: 2025-10-04. United Nations Office on Drugs and Crime, 2011. URL: `https://www.unodc.org/documents/data-and-analysis/Studies/Illicit_financial_flows_2011_web.pdf` (cit. on p. 1).

[5] Kern Alexander. «The international anti-money-laundering regime: the role of the financial action task force». In: *Journal of Money Laundering Control* 4.3 (2001), pp. 231–248 (cit. on p. 1).

[6] Martin Jullum, Anders Løland, Ragnar Bang Huseby, Geir Ånonsen, and Johannes Lorentzen. «Detecting money laundering transactions with machine learning». In: *Journal of Money Laundering Control* 23.1 (2020), pp. 173–186 (cit. on p. 1).

[7] Landry Signé, Mariama Sow, and Payce Madden. *Illicit Financial Flows in Africa: Drivers, Destinations, and Policy Options.* Tech. rep. Policy Brief. Accessed: 2025-10-04. Africa Growth Initiative, Brookings Institution, 2020. URL: `https://www.brookings.edu/wp-content/uploads/2020/02/Illicit-financial-flows-in-Africa.pdf` (cit. on p. 1).

[8] Adetoyese Omoseebi, Godwin Ola, and Jackson Tyler. «Rule-Based Systems in AML». In: (2023) (cit. on p. 2).

[9] Zhiyuan Chen, Le Dinh Van Khoa, Ee Na Teoh, Amril Nazir, Ettikan Kandasamy Karuppiah, and Kim Sim Lam. «Machine learning techniques for anti-money laundering (AML) solutions in suspicious transaction detection: a review». In: *Knowledge and Information Systems* 57.2 (2018), pp. 245–285 (cit. on p. 2).

[10] Sarah N Welling. «Smurfs, money laundering, and the federal criminal law: the crime of structuring transactions». In: *Fla. L. Rev.* 41 (1989), p. 287 (cit. on p. 2).

[11] Bruno Deprez, Toon Vanderschueren, Bart Baesens, Tim Verdonck, and Wouter Verbeke. «Network Analytics for Anti-Money Laundering–A Systematic Literature Review and Experimental Evaluation». In: *arXiv preprint arXiv:2405.19383* (2024) (cit. on pp. 3, 5, 11).

[12] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. «Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics». In: *arXiv preprint arXiv:1908.02591* (2019) (cit. on p. 3).

[13] Dawei Cheng, Yujia Ye, Sheng Xiang, Zhenwei Ma, Ying Zhang, and Changjun Jiang. «Anti-money laundering by group-aware deep graph learning». In: *IEEE Transactions on Knowledge and Data Engineering* 35.12 (2023), pp. 12444–12457 (cit. on pp. 3, 5–7, 9, 14).

[14] Johan Östman et al. «AMLgentex: Mobilizing Data-Driven Research to Combat Money Laundering». In: *arXiv preprint arXiv:2506.13989* (2025) (cit. on pp. 5–12, 15, 34, 37, 39).

[15] David Savage, Qingmai Wang, Xiuzhen Zhang, Pauline Chou, and Xinghuo Yu. «Detection of Money Laundering Groups: Supervised Learning on Small Networks.» In: *AAAI Workshops.* 2017 (cit. on pp. 5, 11).

[16] Edgar Lopez-Rojas, Ahmad Elmir, and Stefan Axelsson. «PaySim: A financial mobile money simulator for fraud detection». In: *28th European Modeling and Simulation Symposium, EMSS, Larnaca.* Dime University of Genoa. 2016, pp. 249–255 (cit. on pp. 5, 6, 10).

[17] Jovan Blanuša, Maximo Cravero Baraja, Andreea Anghel, Luc Von Niederhäusern, Erik Altman, Haris Pozidis, and Kubilay Atasu. «Graph Feature Preprocessor: Real-time Subgraph-based Feature Extraction for Financial Crime Detection». In: (2024), pp. 222–230 (cit. on pp. 5, 9, 11).

[18] Matthew Weber, Manuel Gomez-Rodriguez, and Shin Nakajima. *AMLSim: Anti-Money Laundering Simulation.* GitHub repository. `https://github.com/IBM/AMLSim`. 2020 (cit. on p. 7).

[19] Erik Altman, Jovan Blanuša, Luc Von Niederhäusern, Béni Egressy, Andreea Anghel, and Kubilay Atasu. «Realistic synthetic financial transactions for anti-money laundering models». In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 29851–29874 (cit. on p. 9).

[20] Rasmus Ingemann Tuffveson Jensen, Joras Ferwerda, Kristian Sand Jørgensen, Erik Rathje Jensen, Martin Borg, Morten Persson Krogh, Jonas Brunholm Jensen, and Alexandros Iosifidis. «A synthetic data set to benchmark anti-money laundering methods». In: *Scientific Data* 10.661 (2023). DOI: `10.1038/s41597-023-02569-2`. URL: `https://www.nature.com/articles/s41597-023-02569-2` (cit. on p. 8).

[21] Toyotaro Suzumura, Hiroki Kanezashi, and IBM Research. *AMLSim: Anti-Money Laundering Simulator*. `https://github.com/IBM/AMLSim`. Multi-agent simulator with configurable typologies and alert rates. 2021 (cit. on p. 8).

[22] Ronald F Pol. «Anti-money laundering: The world's least effective policy experiment? Together, we can fix it». In: *Policy design and practice* 3.1 (2020), pp. 73–94. DOI: `10.1080/25741292.2020.1725366`. URL: `https://doi.org/10.1080/25741292.2020.1725366` (cit. on p. 10).

[23] Andreas Loukas. «Graph reduction with spectral and cut guarantees». In: *Journal of Machine Learning Research* 20.116 (2019), pp. 1–42 (cit. on pp. 17, 19, 24–26, 39, 41).

[24] Andreas Loukas and Pierre Vandergheynst. «Spectrally approximating large graphs with smaller graphs». In: *International conference on machine learning*. PMLR. 2018, pp. 3237–3246 (cit. on pp. 17, 25).

[25] Yu Jin, Andreas Loukas, and Joseph JaJa. «Graph coarsening with preserved spectral properties». In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 4452–4462 (cit. on pp. 17, 25).

[26] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. «Simple and deep graph convolutional networks». In: *International conference on machine learning*. PMLR. 2020, pp. 1725–1735 (cit. on p. 28).

[27] Herbert Robbins and Sutton Monro. «A stochastic approximation method». In: *The annals of mathematical statistics* (1951), pp. 400–407 (cit. on p. 32).

[28] Wayne W Zachary. «An information flow model for conflict and fission in small groups». In: *Journal of anthropological research* 33.4 (1977), pp. 452–473 (cit. on p. I).

[29] Wayne W Zachary. «An information flow model for conflict and fission in small groups». In: *Journal of anthropological research* 33.4 (1977), pp. 452–473 (cit. on p. VII).

Appendix A

Theorem 9 (Eigenvalue approximation) - Empirical Verification

We want to write a script to verify:

$$\gamma_1 \lambda_k \leq \tilde{\lambda}_k \leq \gamma_2 \frac{(1+\varepsilon_k)^2}{1-\varepsilon_k^2(\lambda_k/\lambda_2)}$$

whenever $\varepsilon_k^2 < \lambda^2/\lambda_k$.

For this purpose, we employ a simple dataset to reduce computational complexity; in particular, we use the `KarateClub` dataset [28]. We initialize the graph by creating an instance of the `torch_geometric.data.Data` class and subsequently apply the coarsening procedure. The process can be summarized as follows:

```
Gall, Call, iCs = coarse_graph(
    # parameters (not of interest here)
)
```

Here, `Gall` is a list that stores the coarsened graphs at each iteration, where `Gall[0]` corresponds to the initial graph, and `Gall[-1]` to the most coarsened graph. The list `Call` contains all the coarsening matrices, such that `Call[i]` maps the initial graph to the graph `Gall[i+1]`. Finally, the list `iCs` stores the *atomic* coarsening matrices, where each `iCs[i]` coarsens `Gall[i]` into `Gall[i+1]`. Eventually, the lengths of the `Call` and `iCs` lists reflect the levels of coarsening.

We can compute γ_1 and γ_2 leveraging a property of Laplacian coarsening:

$$\gamma_1 = \min_{v_i \in V} \left| V_0^{\varphi_1(v_i)} \right| \geq 1 \quad \text{and} \quad \gamma_2 = \max_{v_i \in V} \left| V_0^{\varphi_1(v_i)} \right|$$

which correspond to the minimum and maximum sizes of the contraction sets, respectively. In practice, these quantities can be computed in Python as follows. Since matrix C is stored in a sparse format, it requires specific handling during computation:

```python
def compute_gamma_1_2(C):
    if not C.is_coalesced(): C = C.coalesce()
    gamma1, gamma2 = None, None
    rows, cols = C.indices()
    for r in range(C.size(0)):
        mask = rows == r
        length = len(cols[mask].tolist())
        if gamma1 is None or length < gamma1: gamma1 = length
        if gamma2 is None or length > gamma2: gamma2 = length

    return gamma1, gamma2
```

The calculation of ε_k involves evaluating all possible values of ε derived from the k top eigenvectors of the Laplacian matrix at a given coarsening level ℓ. The final value is then obtained according to Equation 4.7.

```python
def compute_epsilon_k(L, C, k):
    device = C.device

    invC = C.t().coalesce()
    invC.values().fill_(1.0)

    vals, vecs = torch.linalg.eigh(L) # first k eigenvectors

    epsilons = []
    for i in range(k):
        u = torch.tensor(vecs[:, i], dtype=torch.float32, device=device)

        Pi_u = invC @ (C @ u) # projection with C

        num = (u - Pi_u).T @ L @ (u - Pi_u)
        den = torch.max(torch.tensor(0.0, device=device), u.T @ (L @ u))

        ratio = torch.sqrt((num / (den + 1e-12)))
        epsilons.append(ratio.item())

    return np.max([e for e in epsilons if not np.isnan(e)])
```

To verify the theorem at a specific level ℓ, we employ the following procedure:

```python
def verify_th13(G:Data, Gc:Data, C:torch.Tensor, k=5):
    gamma1, gamma2 = get_gamma_1_2(C)

    L = G.L.to_dense().cpu().numpy()
    Lc = Gc.L.to_dense().cpu().numpy()

    # eigenvalues original graph
    vals, _ = eigsh(L, k=k, which="SM")
    lambda2, lambdak = vals[1], vals[k-1]

    # eigenvalues coarsened graph
    vals_c, _ = eigsh(Lc, k=k, which="SM")
    tilde_lambdak = vals_c[k-1]

    eps_k = compute_epsilon_k(torch.tensor(L, device=C.device), C, k)

    # bounds
    lower = gamma1 * lambdak
    upper =
        gamma2 * ((1+eps_k)**2 / (1 - eps_k**2 * (lambdak / lambda2) + 1e
-9)) * lambdak
        if eps_k**2 < (lambda2 / lambdak)
        else float('inf')

    return lower <= tilde_lambdak <= upper
```

Eventually, we extend the verification across all coarsening levels as follows:

```
def get_epsilon_values(Gall, Call, k=5):
    for i in range(len(Call)):
        G_orig = Gall[0]
        G_coarse = Gall[i+1]
        C = Call[i].coalesce()
        Csq = C * C # C contains sqrt values
        assert(verify_th13(G_orig, G_coarse, Csq, k=k))
```

Appendix B

Cosine Similarity Implementation

The following function computes the mean pairwise cosine similarity among a set of node embeddings:

```
def similarity(nodes_features: torch.Tensor):
    nodes_features = F.normalize(nodes_features, p=2, dim=1)
    s = nodes_features @ nodes_features.T
    n = s.shape[0]
    return torch.sum(torch.triu(s, diagonal=1)) / ((n - 1) * n / 2)
```

Given a matrix $X \in \mathbb{R}^{n \times d}$, where each row represents the **normalized** embedding of a node, the function first computes the cosine similarity matrix

$$S = XX^\top,$$

since the dot product between two normalized vectors corresponds to their cosine similarity. It then sums the upper triangular part of S (excluding the diagonal) to consider each node pair only once, and divides by the total number of unique pairs $\frac{n(n-1)}{2}$, obtaining the *average cosine similarity* among all nodes.

This measure provides an indicator of how close, on average, the nodes are in the embedding space, where higher values correspond to more semantically homogeneous groups.

Appendix C

Coarsening-Aware Loss Implementation

The following listing shows the PyTorch implementation of the coarsening–aware loss introduced in Section 4.3.6. This class extends `nn.Module` and combines the standard negative cross-entropy loss with a coarsening regularization term that enforces embedding consistency across merged nodes. To keep the computation efficient on large graphs, the regularizer is approximated using normalized embeddings and random negative sampling.

```python
import torch
import torch.nn as nn

class CoarseningAwareLoss(nn.Module):
    def __init__(self, coarse_weight: float = 1):
        """
        Args:
            coarse_weight: weight for the coarsening loss term.
        """
        super().__init__()
        self.coarse_weight = coarse_weight
        self.class_loss = nn.CrossEntropyLoss()

    def forward(
        self,
        output: torch.Tensor,
        embeddings: torch.Tensor,
        labels: torch.Tensor,
        train_idx: torch.Tensor,
        coarse_loss: bool = True,
    ):
        """
        Args:
            output: [N, C] log-probabilities (log_softmax)
            embeddings: [N, D] node embeddings
            labels: [N] ground-truth class labels
            train_idx: indices used for the classification loss
            coarse_loss: whether to include the coarsening regularizer
        """
        N = embeddings.shape[0]

        # 1. Classification loss
        loss_cls = self.class_loss(output[train_idx], labels[train_idx])
        if not coarse_loss: return loss_cls

        # 2. Coarsening loss: within-cluster compactness
        loss_coarse = -torch.mean(torch.sum(embeddings**2, dim=1))

        # 3. Negative sampling for inter-cluster separation
        n_sample = max(int(N * 0.1), 500)
        sampled_indices1 = torch.randint(0, N, (n_sample,))
        sampled_indices2 = torch.randint(0, N, (n_sample,))
        emb1 = embeddings[sampled_indices1]
        emb2 = embeddings[sampled_indices2]
        loss_coarse += torch.mean(torch.sum(emb1 * emb2, dim=1))

        # 4. Weighted combination
        return loss_cls + self.coarse_weight * loss_coarse
```

The first part of the loss computes the standard classification objective on labeled nodes, while the second introduces a lightweight approximation of the coarsening regularization. The latter penalizes overly similar embeddings between randomly sampled nodes and promotes compactness among embeddings of nodes that have been merged. This implementation achieves good scalability and can be directly integrated into the iterative training loop described in Section 4.3.7.

Appendix D

Coarsening Evolution on a Small Graph

Figure D.1 illustrates an example of the coarsening evolution on KarateClub graph [29] across multiple levels ($L = 8$, ratio $= 0.2$). Each node is colored according to a three-dimensional projection of the eigenvectors of the graph Laplacian matrix, providing an intuitive visualization of the spectral structure preserved during the coarsening process. Specifically, the colors correspond to the first three eigenvectors of the Laplacian, normalized for display as RGB values. This visualization helps build intuition about how spectral properties and structural information are maintained as the graph becomes progressively coarser.

The coloring was generated using the following script:

```
L_dense = Gc.L.to_dense().cpu().numpy()
_, V = np.linalg.eigh(L_dense)
c = V[:, :3]
c = (c - c.mean(axis=0)) / (c.std(axis=0, ddof=0) + 1e-8)
c = (c - c.min(axis=0)) / (c.ptp(axis=0) + 1e-8)
G.colors = torch.tensor(c, device=device)
```

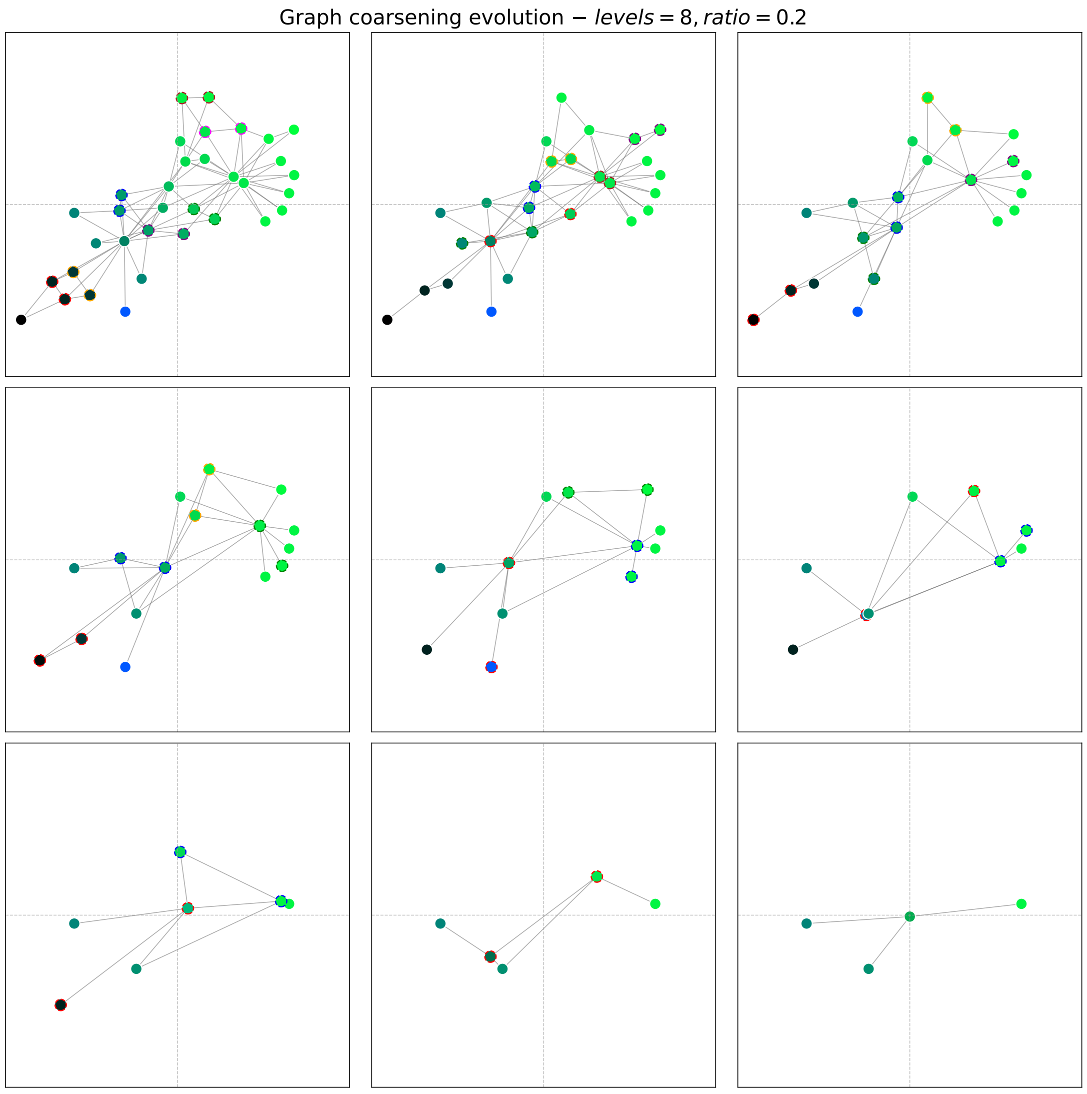


Figure D.1: Graph coarsening evolution for a small graph. Node colors represent the first three eigenvectors of the Laplacian matrix. Nodes sharing the same dotted border are merged into the same super node..