Rethinking Reconstruction-based Graph-Level Anomaly Detection: Limitations and a Simple Remedy

Sunwoo Kim[1], **Soo Yong Lee**[1], **Fanchen Bu**[2], **Shinhwan Kang**[1], **Kyungho Kim**[1], **Jaemin Yoo**[2], **Kijung Shin**[1,2]
[1]Kim Jaechul Graduate School of AI, [2]School of Electrical Engineering
Korea Advanced Institute of Science and Technology (KAIST)
`{kswoo97, syleetolow, boqvezen97, shinhwan.kang, kkyungho, jaemin, kijungs}@kaist.ac.kr`

Abstract

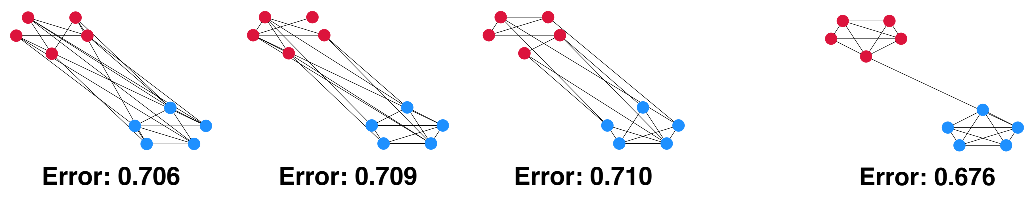
Graph autoencoders (Graph-AEs) learn representations of given graphs by aiming to accurately reconstruct them. A notable application of Graph-AEs is graph-level anomaly detection (GLAD), whose objective is to identify graphs with anomalous topological structures and/or node features compared to the majority of the graph population. Graph-AEs for GLAD regard a graph with a high mean reconstruction error (i.e. mean of errors from all node pairs and/or nodes) as anomalies. Namely, the methods rest on the assumption that they would better reconstruct graphs with similar characteristics to the majority. We, however, report non-trivial counter-examples, a phenomenon we call *reconstruction flip*, and highlight the limitations of the existing Graph-AE-based GLAD methods. Specifically, we empirically and theoretically investigate when this assumption holds and when it fails. Through our analyses, we further argue that, while the reconstruction errors for a given graph are effective features for GLAD, leveraging the multifaceted summaries of the reconstruction errors, beyond just mean, can further strengthen the features. Thus, we propose a novel and simple GLAD method, named MUSE. The key innovation of MUSE involves taking multifaceted summaries of reconstruction errors as graph features for GLAD. This surprisingly simple method obtains SOTA performance in GLAD, performing best overall among 14 methods across 10 datasets.

1 Introduction

Graph autoencoders (Graph-AEs) are a family of graph neural networks (GNNs) that learn latent representations of given graphs by aiming to accurately reconstruct them. Representative examples of Graph-AEs include GAE [22] and GraphMAE [15], which respectively aim to accurately reconstruct graph structure and node features. Graph-AEs have a broad spectrum of applications, such as anomaly detection [38, 33, 6] and link prediction [13, 22, 24].

Graph-AEs are trained to capture the structural information of graphs used for training (i.e., training graphs). Thus, intuitively, Graph-AEs are expected to better reconstruct graphs that are similar to the training graphs. However, to our surprise, this expectation is not always true. Given Graph-AEs trained to reconstruct the graphs in Figure 1(a), which share common structural characteristics, one would expect that the Graph-AEs would reconstruct the training graphs with smaller reconstruction errors than a dissimilar counterpart (e.g., in Figure 1 (b)). We, however, report the opposite. The reconstruction error is lower for the graph in Figure 1(b) than the training graphs. While similar observations were reported in computer vision application [31], which we elaborate on in Section 2.1, the existing literature does not clearly explain the aforementioned phenomenon.

This counterintuitive phenomenon guides our analysis, implication, and the proposed method. First, we aim to understand this counterintuitive phenomenon, which we refer to as the *reconstruction flip*.



(a) Training graphs (b) Unseen graph

Figure 1: The training graphs in (a) and the unseen graph in (b) exhibit different structural characteristics, but a Graph-AE model reconstructs the graph in (b) more accurately than those in (a).

We argue that the phenomenon can occur when (1) the training graphs and unseen graphs share a primary structural pattern and (2) the pattern is more pronounced in unseen graphs than in training graphs. We corroborate our claim with our theoretical and empirical analysis.

Second, our finding has strong implications for graph-level anomaly detection (GLAD). GLAD aims to identify graphs of different characteristics (e.g., structures and/or features) from the majority [38, 33, 39, 32]. Many existing GLAD methods leverage Graph-AEs [33, 38]. They regard a graph with a high mean reconstruction error (i.e. mean of errors from all node pairs and/or nodes) as anomalies, assuming trained Graph-AEs would struggle to reconstruct the graphs with structural patterns (and/or node attributes) that deviate from most training graphs. However, our finding suggests that this assumption does not always hold. Specifically, anomalous graphs with structural characteristics distinct from the training graphs may exhibit similar or even lower mean reconstruction errors. In such cases, the existing methods would struggle to detect anomalous graphs. We argue that this issue arises from how the existing methods leverage reconstruction errors, yet these errors remain valuable graph features for GLAD. We propose using multifaceted summaries of the reconstruction errors, rather than relying solely on their mean, to further enhance the features.

Third, based on the analysis and implication, we propose a simple and novel GLAD method, **MUSE** (**Mu**ltifacted **S**ummarization of Reconstruction **E**rrors). MUSE, like other Graph-AE-based GLAD methods, reconstructs given graphs. However, MUSE employs a new way of using reconstruction errors: representing a graph with multifaceted summaries (e.g., mean, standard deviation, etc.) of the graph's reconstruction errors for GLAD. This simple innovation allows MUSE to obtain SOTA performance in the GLAD tasks. Through our extensive experiments including 13 baseline methods and 10 benchmark datasets, MUSE performs overall best, achieving up to 28.1% performance gain (in terms of AUROC) compared to the best competitor. We summarize our contribution as follows:

- **Analysis (Section 3) with implications (Section 4):** We investigate the reconstruction flip phenomenon both theoretically and empirically, yielding practical implications for GLAD.
- **Effective method (Section 5):** Motivated by our analysis, we propose MUSE, a simple yet effective GLAD method that represents a graph as multifaceted summaries of its reconstruction errors.
- **Extensive experiments (Section 6):** Our experiments on 10 datasets demonstrate the superiority of MUSE over prior GLAD methods. Our code and datasets are available at `https://github.com/kswoo97/GLAD_MUSE`.

2 Related Work and Preliminaries

In this section, we provide a brief literature review on graph-level anomaly detection (GLAD) and peculiar observations in GLAD. We then present the preliminaries of our work.

2.1 Related work

Graph-level anomaly detection. Graph-level anomaly detection (GLAD) aims to identify graphs whose characteristics deviate from those of the majority of the graph population [38, 33, 39, 32]. GLAD has various applications, including brain diagnosis [25] and drug discovery [46]. While several GLAD methods require anomaly label supervision to train a detector [54, 35], we focus on the methods that do not require anomaly labels [38, 33, 39, 32], since anomaly labels are often not available in real-world scenarios. A GLAD method typically trains a model to perform certain pretext tasks on given graphs, and the graphs are regarded as anomalies if the trained model shows poor pretext task performance on them. Notable pretext tasks include graph reconstruction [38, 33], graph embedding hypersphere minimization [39, 57], and cross-view mutual information maximization [34, 32].

Peculiar observations in graph-level anomaly detection. The most relevant literature with our work is a study of Zhao and Akoglu [56]. In graph classification datasets, they regarded graphs belonging to a particular class as normal graphs and otherwise as anomalies to benchmark several GLAD methods. In this setting, they observed that several GLAD methods, especially kernel-based ones, occasionally perform worse than random guessing. However, their analysis focused on graph kernels [37, 41], without covering graph autoencoders, which are our focus. In computer vision, Liu et al. [31] suggested that certain anomalous images can be easier to reconstruct than normal ones, presenting a method that can overcome these undesirable circumstances. However, neither their analysis nor the method can be trivially extended to the graph domain, as elaborated in Appendix F.1.

2.2 Preliminary

Graphs. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a node set $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ and an edge set $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{E}|}\}$, where each edge $e_i \in \mathcal{E}$ is defined by a node pair. We assume that each node $v_i \in \mathcal{V}$ is equipped with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$, which forms a node feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ where each i-th row $\mathbf{X}_{i,:} = \mathbf{x}_i$. Moreover, $\mathcal{E}$ can be represented by an adjacency matrix $\mathbf{A} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$, where $\mathbf{A}_{i,j} = 1$ if and only if $\{v_i, v_j\} \in \mathcal{E}$. Therefore, the graph can also be defined as $\mathcal{G} = (\mathbf{X}, \mathbf{A})$.

Graph neural networks. Graph neural networks (GNNs) are a category of neural networks designed for processing graph data. GNNs primarily leverage message passing schemes [49, 14, 45, 11, 28, 30]. In this work, we consider GNNs as parameterized functions f_θ that generate node embeddings $\mathbf{Z} = f_\theta(\mathbf{X}, \mathbf{A}) \in \mathbb{R}^{|\mathcal{V}| \times d'}$ for each graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$. [1]

Graph autoencoders. Graph autoencoders (Graph-AEs) are a family of GNNs that learn graph latent representations by graph reconstruction. Specifically, Graph-AEs reconstruct a given graph's structure [22, 43] and/or node features [15, 16]. For example, GAE [22] reconstructs the adjacency matrix of a given graph. Specifically, GAE first uses an encoder GNN f_θ to generate node embeddings $\mathbf{Z} = f_\theta(\mathbf{X}, \mathbf{A})$ of a given graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$. Then, GAE obtains a reconstructed adjacency matrix $\hat{\mathbf{A}} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ through the inner-product of node embeddings and entry-wise sigmoid activation σ (i.e., $\hat{\mathbf{A}} = \sigma(\mathbf{Z}\mathbf{Z}^T)$). Lastly, it minimizes the difference between $\mathbf{A}$ and $\hat{\mathbf{A}}$, for which one can use the binary cross-entropy (BCE) loss $\mathcal{L}_{\text{BCE}}(\mathcal{G} \mid f_\theta) \coloneqq -\sum_{i,j=1}^{|\mathcal{V}|} \left(\mathbf{A}_{i,j} \log \hat{\mathbf{A}}_{i,j} + (1 - \mathbf{A}_{i,j}) \log (1 - \hat{\mathbf{A}}_{i,j}) \right)$ or the squared Frobenius-norm loss $\mathcal{L}_{\text{SFN}}(\mathcal{G} \mid f_\theta) \coloneqq \|\mathbf{A} - \hat{\mathbf{A}}\|_F^2 = \sum_{i,j=1}^{|\mathcal{V}|} \left(\mathbf{A}_{i,j} - \hat{\mathbf{A}}_{i,j} \right)^2$. Given a set of training graphs $\mathbb{G} = \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_{|\mathbb{G}|}\}$, GAE is trained (i.e., the encoder parameter θ is updated) to minimize $\sum_{\mathcal{G} \in \mathbb{G}} \mathcal{L}(\mathcal{G} \mid f_\theta)$ (or equivalently, $\mathbb{E}_{\mathcal{G} \in \mathbb{G}} \mathcal{L}(\mathcal{G} \mid f_\theta)$), where $\mathcal{L}$ is $\mathcal{L}_{\text{BCE}}$ or $\mathcal{L}_{\text{SFN}}$.

In our analysis below, we focus on methods that reconstruct graph structures, while we describe and investigate node-feature-reconstruction methods in Appendix E.1.

3 Analysis of Graph Autoencoders

In this section, we explore *reconstruction flip* phenomena (e.g., the phenomenon described in Figure 1), where graph autoencoders (Graph-AEs) reconstruct some unseen graphs that are dissimilar to training graphs better than training graphs.

Our analysis focuses on evidencing the claims below. We assume GAE [22] trained on graphs sharing a primary pattern $\mathcal{P}$ of strength $\mathcal{S}$.

Claims

- *Reconstruction flip* ***tends to occur*** *when unseen (test) graphs have the same primary pattern $\mathcal{P}$ but with a greater strength $\mathcal{S}' > \mathcal{S}$.*
- *Reconstruction flip* ***tends not to occur*** *when unseen (test) graphs have a different primary pattern $\mathcal{P}' \neq \mathcal{P}$.*

[1]Some GNNs are designed to produce edge embeddings [58, 58] and/or graph embeddings [52, 10].

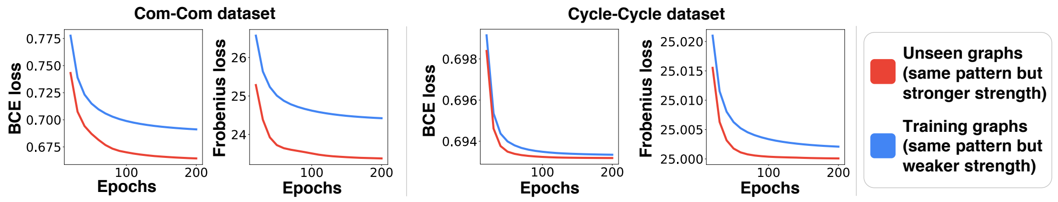


Figure 2: **Reconstruction flip occurs.** When Graph-AEs are trained on graphs sharing a primary pattern of weak strength, the trained Graph-AEs exhibit lower reconstruction errors for graphs with the same pattern but with higher strength (red lines) than those with weaker strength (blue lines).

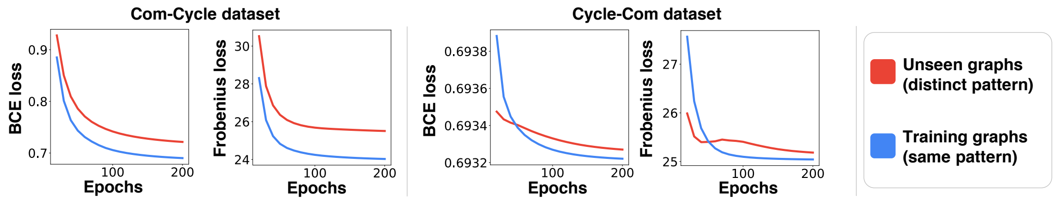


Figure 3: **Reconstruction flip does NOT occur.** When Graph-AEs are trained on graphs sharing a primary pattern, the trained Graph-AEs exhibit higher reconstruction errors for graphs with a different pattern (red lines) than those with the same pattern (blue lines).

That is, when Graph-AEs are trained on graphs with a primary pattern $\mathcal{P}$ of strength $\mathcal{S}$, the trained Graph-AEs tend to exhibit lower errors in reconstructing graphs with $\mathcal{P}$ of a greater strength $\mathcal{S}' > \mathcal{S}$. However, the trained Graph-AEs tend to exhibit higher errors in reconstructing graphs with a different pattern $\mathcal{P}' \neq \mathcal{P}$. For demonstration, we elaborate on $\mathcal{P}$ and $\mathcal{S}$ via synthetic datasets (Section 3.1) and present both empirical (Section 3.2) and theoretical (Section 3.3) investigations.

3.1 Synthetic graphs

To elaborate on the concepts of a primary pattern $\mathcal{P}$ and a pattern strength $\mathcal{S}$, we employ two types of synthetic graphs: (1) graphs with community structures and (2) graphs with cycles.

Community type. Syn-Com graphs have community structures, a pervasive pattern in real-world graphs [12], as the primary pattern $\mathcal{P}$. Graphs in Figure 1 are instances of such graphs. We control the strength S of community structures through a parameter $\tau \in [0, 1]$. The 10 nodes in each graph are evenly divided into two communities. Each intra- and inter-community edge is formed with probability $(1 + \tau)/2$ and $(1 - \tau)/2$, respectively. In our empirical analysis, we consider two graph classes with different τ's. Graphs in one class $\mathbb{G}^{com}_{\tau=0.4}$ are generated with $\tau = 0.4$ (weaker community structure; the graphs in Figure 1(a)) and those in the other class $\mathbb{G}^{com}_{\tau=0.8}$ are generated with $\tau = 0.8$ (stronger community structure; the graph in Figure 1(b)).

Cycle type. Syn-Cycle graphs contain a cycle of nodes (i.e., a closed chain of nodes), which is commonly observed in real-world graphs [8]. In Syn-Cycle, the primary pattern $\mathcal{P}$ corresponds to the node cycle. Graphs in Figure 4 are instances of Syn-Cycle. In Syn-Cycle graphs, the pattern strength $\mathcal{S}$ is stronger in the clean-cycle graph consisting only of a cycle with n nodes (the left graph in Figure 4), and it is weaker in noisy-cycle graphs, where a node within the cycle of $(n - 1)$ nodes is linked to an extra node (the right graph in Figure 4). The number of nodes n is fixed to 10.

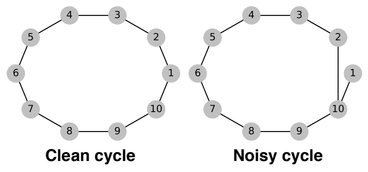


Figure 4: A clean-cycle graph and a noisy-cycle graph.

We provide detailed descriptions of both graph types in Appendix C.

3.2 Empirical analysis

Setting. We consider two scenarios. In Scenario 1, we aim to validate our first claim. To this end, we leverage (1) the Com-Com dataset, where the training graphs are sampled from $\mathbb{G}^{com}_{\tau=0.4}$ and the unseen graphs are sampled from $\mathbb{G}^{com}_{\tau=0.8}$, and (2) the Cycle-Cycle dataset, where the training graphs are sampled from the noisy-cycle graphs and the unseen graph is the clean-cycle graph.

In Scenario 2, we aim to validate our second claim. To this end, we leverage (3) the Com-Cycle dataset, where the training graphs are sampled from $\mathbb{G}^{com}_{\tau=0.4}$ and the unseen graphs are sampled from the noisy-cycle graphs, and (4) the Cycle-Com dataset, where the training graphs are sampled from the noisy-cycle graphs and the unseen graphs are sampled from $\mathbb{G}^{com}_{\tau=0.4}$.

For each dataset, we train GAE [22] equipped with GIN [49] as the graph encoder. We use either the (1) binary cross-entropy (BCE) loss or the (2) squared Frobenius-norm (Frobenius) loss, which are formalized in Section 2.2, for the reconstruction loss. We compare the reconstruction losses for the training graphs and unseen graphs. Further details are provided in Appendix D.2.

Note. *While we focus on adjacency reconstruction in this section, we provide an analysis of feature-reconstruction methods in Appendix E.1, where the results are consistent with those below.*

Empirical results. In Scenario 1, for both Com-Com and Cycle-Cycle datasets, reconstruction losses are lower for the graphs with stronger pattern strengths, which are unseen graphs (see Figure 2). That is, all graphs share a primary pattern $\mathcal{P}$ (either node community or cycle) with two different strengths $\mathcal{S} < \mathcal{S}'$, and GAE trained on graphs of pattern strength $\mathcal{S}$ reconstructs those of pattern strength $\mathcal{S}'$ with lower losses, demonstrating our first claim. In Scenario 2, for both Com-Cycle and Cycle-Com datasets, reconstruction losses become lower for the training graphs after some training epochs (see Figure 3). That is, given training graphs with a primary pattern $\mathcal{P}$ (either node communities or a cycle), the trained GAE reconstructs graphs with the same pattern $\mathcal{P}$ with lower losses than those with a different pattern $\mathcal{P}'$, demonstrating our second claim.

3.3 Theoretical analysis

In this subsection, we theoretically analyze the empirical observations in Section 3.2, focusing on the occurrence of reconstruction flip. Informally speaking, we shall show that when GAE is trained on graphs with a primary pattern $\mathcal{P}$ of strength $\mathcal{S}$, the following holds:

(**A1**) Reconstruction losses decrease for graphs with the same $\mathcal{P}$ of various strengths.

(**A2**) Reconstruction losses decrease more rapidly for graphs with the same $\mathcal{P}$ of greater strengths.

Setting. For theoretical analysis, we use a single-layer linear GAE. Formally, for a given graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$, the linear GAE reconstructs the given graph's adjacency matrix as follows: $\hat{\mathbf{A}} = \mathbf{AXW}(\mathbf{AXW})^T$, where $\mathbf{W} \in \mathbb{R}^{n\times n}$ is a learnable weight matrix and n is the number of nodes. We use the squared Frobenius norm as the reconstruction loss $\mathcal{L}(\mathcal{G}, \mathbf{W}) = \|\mathbf{A} - \hat{\mathbf{A}}\|_F^2$. We consider Syn-Com graphs $\mathbb{G}^{com}_{\tau}$ (Section 3.1). To formalize the training of linear GAE, we define the expected gradient descent update of $\mathbf{W}$ as follows:

$$\mathbb{W}(\tau, \mathbf{W}, \gamma) = \mathbf{W} - \gamma \mathbb{E}_{\mathcal{G}}\left[\frac{\partial \mathcal{L}(\mathcal{G}, \mathbf{W})}{\partial \mathbf{W}} \mid \tau\right] \in \mathbb{R}^{n\times n}, \tag{1}$$

where $\gamma > 0$ is a learning rate and $\mathbb{E}_{\mathcal{G}}[\cdot|\tau]$ takes the expectation over all the graphs $\mathcal{G} \in \mathbb{G}^{com}_{\tau}$. We assume graphs of strength τ_1 are used as the training graphs and graphs of strength τ_2 are used as the unseen (test) graphs.

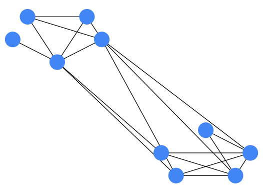
Theoretical results. We now present our theoretical results, with proof of each theorem in Appendix A. We first demonstrate that (**A1**) holds by expectation.

Theorem 1 (Generalization across τ)**.** *For any $n \geq 12$ and $0 < \tau_1 \leq 1$, there exists $\epsilon > 0$ such that for any $0 < \gamma \leq \epsilon$ and any $0 < \tau_2 \leq 1$ (recall that 0 and 1 are the lower and upper bounds of τ, respectively) with the initial weight matrix $\mathbf{W}^{(0)} = \mathbb{I}_n$,*

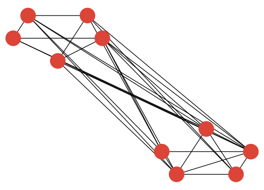
$$\underbrace{\mathbb{E}_{\mathcal{G}}[\mathcal{L}(\mathcal{G}, \mathbb{W}(\tau_1, \mathbf{W}^{(0)}, \gamma))|\tau_2]}_{\textit{the loss on graphs of strength } \tau_2 \textit{ \textbf{after} update}} < \underbrace{\mathbb{E}_{\mathcal{G}}[\mathcal{L}(\mathcal{G}, \mathbf{W}^{(0)})|\tau_2]}_{\textit{the loss on graphs of strength } \tau_2 \textit{ \textbf{before} update}}.$$

Roughly, Theorem 1 states that regardless of the values of τ_1 and τ_2, updating $\mathbf{W}$ with graphs of strength τ_1 reduces the reconstruction losses for graphs of strength τ_2. Intuitively, this suggests that GAE learns the primary pattern $\mathcal{P}$, regardless of the strength $\mathcal{S}$ (here, τ_1) of the training graphs.

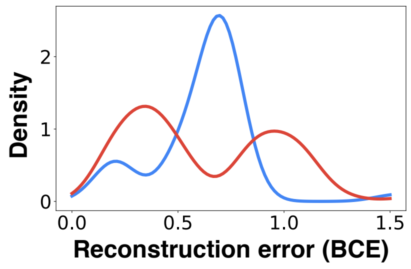
Now, we demonstrate that (**A2**) holds by expectation.



(a) Graph $\mathcal{G}_1$



(b) Graph $\mathcal{G}_2$



(c) Gaussian KDE visualization of error distributions of $\mathcal{G}_1$ (blue) and $\mathcal{G}_2$ (red).

Figure 5: A case of Graph-AEs (specifically, GAE [22]) having similar mean reconstruction errors for two dissimilar graphs (specifically, $\mathcal{G}_1$ **has 0.6622** and $\mathcal{G}_2$ **has 0.6627**), while their error distributions differ significantly.

Theorem 2 (Correlation with τ)**.** *For any $n \geq 34$ and $0 < \tau_1 \leq 0.98$, there exists $\epsilon > 0$ such that for any $0 < \gamma \leq \epsilon$ and any $\tau_1 + 0.02 \leq \tau_2 \leq 1$ with the initial weight $\mathbf{W}^{(0)} = \mathbb{I}_n$,*

$$\underbrace{\mathbb{E}_{\mathcal{G}}\left[\mathcal{L}(\mathcal{G}, \mathbf{W}^{(0)}) - \mathcal{L}(\mathcal{G}, \mathbb{W}(\tau_1, \mathbf{W}^{(0)}, \gamma)) \mid \tau_1\right]}_{\textit{the decrease in loss on graphs of strength } \tau_1 \textit{ (training graphs)}} < \underbrace{\mathbb{E}_{\mathcal{G}}\left[\mathcal{L}(\mathcal{G}, \mathbf{W}^{(0)}) - \mathcal{L}(\mathcal{G}, \mathbb{W}(\tau_1, \mathbf{W}^{(0)}, \gamma)) \mid \tau_2\right]}_{\textit{the decrease in loss on graphs of strength } \tau_2 \textit{ (unseen graphs)}}.$$

Roughly, Theorem 2 states that updating $\mathbf{W}$ using graphs of pattern strength τ_1 results in a greater reduction of reconstruction losses for graphs of higher strength $\tau_2 > \tau_1$ than for the graphs used in $\mathbf{W}$ update (i.e., graphs of strength τ_1). Intuitively, this result suggests that even if GAE is trained on graphs with a primary pattern $\mathcal{P}$ of strength $\mathcal{S}$ (here, τ_1), the trained GAE tends to reconstruct graphs of a greater strength $\mathcal{S}'$ (here, τ_2) with smaller losses.

4 Implications in Graph-Level Anomaly Detection

In this section, we investigate the practical implications of our analysis in Section 3 for graph-level anomaly detection (GLAD). Specifically, we discuss the limitations of existing graph-autoencoder-based GLAD methods and propose an improved approach to using graph reconstruction errors.

4.1 Limitations of existing Graph-AEs in GLAD

Anomalous graphs may have low mean reconstruction errors. As outlined in Section 2.1, many methods for graph-level anomaly detection (GLAD) use graph autoencoders (Graph-AEs), aiming to identify graphs whose characteristics deviate from the majority. Typically, such GLAD methods [33, 38] consider graphs with higher reconstruction losses (i.e. mean (or sum) of reconstruction errors from all node pairs and/or nodes) as anomalies. This is based on the intuition that Graph-AEs would struggle to reconstruct anomalous graphs that are dissimilar to the majority of training graphs. However, our empirical (Section 3.2) and theoretical (Section 3.3) analyses reveal that Graph-AEs can exhibit lower mean reconstruction errors for graphs that are less similar to training graphs. In the context of GLAD, when anomalous graphs have the same primary pattern as training graphs but have stronger strengths, Graph-AEs tend to exhibit lower mean reconstruction errors, compared to the training graphs. Thus, such anomalies tend not to be detected by existing Graph-AE-based GLAD methods, revealing their severe limitations.

Mean is not all you need. In addition in practice, we find that in some cases, dissimilar graphs can have similar mean reconstruction errors. As shown in Figure 5, although the graph $\mathcal{G}_1$ in (a) and the graph $\mathcal{G}_2$ in (b) have significantly different numbers of edges, their mean reconstruction errors are similar, making it difficult to distinguish between the two graphs (refer to the caption of Figure 5). On the other hand, Figure 5(c) shows that the reconstruction error distributions over individual node pairs [2] of $\mathcal{G}_1$ (blue distribution) and $\mathcal{G}_2$ (red distribution) differ significantly in shape. Other descriptive statistics, such as standard deviation, can effectively distinguish the two graphs in such cases. Thus, incorporating additional statistical measures beyond the mean is beneficial for Graph-AEs in GLAD.

[2] An example error distribution is the distribution of $\mathbf{A}_{i,j} \log \hat{\mathbf{A}}_{i,j} + (1 - \mathbf{A}_{i,j}) \log (1 - \hat{\mathbf{A}}_{i,j}), \forall i, j \in [n]$.

4.2 Enhancing the use of reconstruction errors

Based on the implications discussed in Section 4.1, we argue that *a graph's reconstruction errors aggregated in various ways* can provide features that effectively distinguish anomalous graphs from normal graphs. We provide detailed descriptions of this claim below.

Reconstruction errors serve as good features for GLAD. Consider the following scenarios where Graph-AEs are trained on graphs sharing a pattern $\mathcal{P}$ of strength $\mathcal{S}$. First, for an unseen graph of a pattern $\mathcal{P}$ with higher strength $\mathcal{S}' > \mathcal{S}$, Graph-AEs tend to return lower mean reconstruction errors compared to those of the training graphs (Figure 2). Second, for an unseen graph of a different pattern $\mathcal{P}' \neq \mathcal{P}$, Graph-AEs tend to give higher mean reconstruction error than those of the training graphs (Figure 3). In both cases, the error distributions of anomalous graphs differ from those of normal graphs, serving as effective features for distinguishing them.

Using multifaceted summaries of errors is important for GLAD. In practice, we need a fixed-size representation of the error distribution to be used with anomaly classifiers, which generally require a fixed-size input vector. To achieve this, we propose using various summary statistics (e.g., mean and standard deviation) to represent the given error distribution. Specifically, rather than relying on a single statistic, we use multifaceted summaries that capture various aspects of the error distributions. This is because distinct error distributions may share certain similar summary statistics, as shown in Figure 5. Therefore, multifaceted summaries of the error distribution can act as an effective feature vector for distinguishing anomalous graphs from normal graphs.

5 Proposed Method: MuSE

Based on the discussions provided in Section 4.2, we present **MuSE** (**Mu**ltifacted **S**ummarization of Reconstruction **E**rrors), a simple yet effective graph-level anomaly detection (GLAD) method. MuSE aggregates a graph's reconstruction errors into multiple summary statistics and uses a vector of these aggregated errors as the graph's representation for GLAD. We describe how we train the reconstruction model in Section 5.1 and how we obtain the error representation (i.e., a multifaceted summary of reconstruction errors) in Section 5.2.

5.1 Step 1: Training a reconstruction model

We first train MuSE to reconstruct graphs in training data. Step 1 is composed of three parts: (1) augmentation, (2) encoding/decoding, and (3) reconstruction loss minimization. For an input graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$, MuSE first augments the input graph by randomly dropping some edges, mitigating potential overfitting problems. Specifically, MuSE randomly samples $\lceil p|\mathcal{E}|\rceil$ edges from $\mathcal{E}$ for some $0 \leq p < 1$. Let $\mathcal{E}'$ denote the set of sampled edges. We mask the corresponding entries in $\mathbf{A}$ by setting them to zero to generate the augmented adjacency matrix $\mathbf{A}' \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ (i.e., $\mathbf{A}'_{i,j} = 0$ if $\{v_i, v_j\} \in \mathcal{E}'$, otherwise $\mathbf{A}'_{i,j} = \mathbf{A}_{i,j}, \forall i,j \in [|\mathcal{V}|]$).

MuSE then encodes the augmented graph $\mathcal{G}' = (\mathbf{X}, \mathbf{A}')$ to obtain node embeddings $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}|\times d'}$ by using a graph neural network (GNN) encoder f_θ (i.e., $\mathbf{Z} = f_\theta(\mathbf{X}, \mathbf{A}')$). Subsequently, MuSE decodes the node embeddings $\mathbf{Z}$ to obtain reconstructed node features $\hat{\mathbf{X}} \in \mathbb{R}^{|\mathcal{V}|\times d}$ and reconstructed adjacency matrix $\hat{\mathbf{A}} \in (0,1)^{|\mathcal{V}|\times|\mathcal{V}|}$. For node feature reconstruction, MuSE uses a node feature decoder $g_\psi : \mathbb{R}^{d'} \mapsto \mathbb{R}^{d}$ (i.e., $\hat{\mathbf{X}} = g_\psi(\mathbf{Z})$). Similarly, for adjacency matrix reconstruction, MuSE uses an adjacency matrix decoder $h_\phi : \mathbb{R}^{d'} \mapsto \mathbb{R}^{d''}$ and the inner product of its output (i.e., $\hat{\mathbf{A}} = \sigma(\mathbf{Z}'\mathbf{Z}'^{T})$, where σ is a sigmoid function and $\mathbf{Z}' = h_\phi(\mathbf{Z})$).

Lastly, for training, we leverage reconstruction losses regarding the node features and the adjacency matrix. For node feature reconstruction, we use the cosine-similarity loss as in GraphMAE [15]. Formally, the loss is defined as follows:

$$\mathcal{L}_{\mathbf{X}}(\mathcal{G}) = \frac{1}{|\mathcal{V}|}\sum_{i=1}^{|\mathcal{V}|}\left(1 - \frac{\mathbf{X}_{i,:}^{T}\hat{\mathbf{X}}_{i,:}}{\|\mathbf{X}_{i,:}\|_2 \cdot \|\hat{\mathbf{X}}_{i,:}\|_2}\right), \text{ where } \|\cdot\|_2 \text{ is the } \ell_2\text{-norm.} \quad (2)$$

For adjacency-matrix reconstruction, we use the binary cross-entropy (BCE) loss as in GAE [22]. Since real-world graphs are often sparse (i.e., the adjacency matrix contains significantly more zero entries than ones), we weight the loss terms associated with non-zero entries to stabilize model training in the presence of this imbalance. Formally, the loss is defined as follows:

$$\mathcal{L}_{\mathbf{A}}(\mathcal{G}) = -\left(\frac{1}{|\mathcal{V}|}\right)^2 \sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|}\left(\omega \mathbf{A}_{i,j}\log \hat{\mathbf{A}}_{i,j} + (1-\mathbf{A}_{i,j})\log(1-\hat{\mathbf{A}}_{i,j})\right), \tag{3}$$

where $\omega = (\frac{|\mathcal{V}|^2}{\sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|}\mathbf{A}_{i,j}} - 1)^{\tau}$ and τ is a scale hyperparameter. Then, the total reconstruction loss $\mathcal{L}(\mathcal{G})$ is defined as the mean of $\mathcal{L}_{\mathbf{X}}(\mathcal{G})$ (Eq. (2)) and $\mathcal{L}_{\mathbf{A}}(\mathcal{G})$ (Eq. (3)) (i.e., $\mathcal{L}(\mathcal{G}) = (\mathcal{L}_{\mathbf{X}}(\mathcal{G}) + \mathcal{L}_{\mathbf{A}}(\mathcal{G}))/2$). For a set of training graphs $\mathbb{G} = \{\mathcal{G}_1, \cdots, \mathcal{G}_K\}$, we update the parameters (i.e., θ, ψ, and ϕ) by using gradient descent aiming to minimize $\frac{1}{K}\sum_{t\in[K]}\mathcal{L}(\mathcal{G}_t)$.

5.2 Step 2: Generating error representation

After training the reconstruction model, we represent each graph using its reconstruction errors, specifically those from all node pairs and/or nodes. Consider obtaining the reconstruction errors for a graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$ with the trained reconstruction model (i.e., f_θ, g_ψ, and h_ϕ). To this end, we first obtain reconstructed features $\hat{\mathbf{X}}$ and a reconstructed adjacency matrix $\hat{\mathbf{A}}$ by using the encoding and decoding scheme described in Section 5.1. Note that we do not perform augmentation for Step 2 (i.e., the original graph $(\mathbf{X}, \mathbf{A})$ is the input of the encoder f_θ).

After that, we obtain node-feature reconstruction errors $\mathbb{L}_{\mathbf{X}}(\mathcal{G} \mid f_\theta, g_\psi, h_\phi)$ on each node and adjacency-matrix reconstruction errors $\mathbb{L}_{\mathbf{A}}(\mathcal{G} \mid f_\theta, g_\psi, h_\phi)$ on each node pair. Formally,

$$\mathbb{L}_{\mathbf{X}}(\mathcal{G}) = \mathbb{L}_{\mathbf{X}}(\mathcal{G} \mid f_\theta, g_\psi, h_\phi) := \left[1 - \frac{\mathbf{X}_{i,:}^T \hat{\mathbf{X}}_{i,:}}{\|\mathbf{X}_{i,:}\|_2 \cdot \|\hat{\mathbf{X}_{i,:}}\|_2}\right]_{i\in[|\mathcal{V}|]} \in \mathbb{R}^{|\mathcal{V}|}, \text{ and} \tag{4}$$

$$\mathbb{L}_{\mathbf{A}}(\mathcal{G}) = \mathbb{L}_{\mathbf{A}}(\mathcal{G} \mid f_\theta, g_\psi, h_\phi) := [\mathbf{A}_{i,j}\log \hat{\mathbf{A}}_{i,j} + (1-\mathbf{A}_{i,j})\log(1-\hat{\mathbf{A}}_{i,j})]_{i,j\in[|\mathcal{V}|]} \in \mathbb{R}^{|\mathcal{V}|^2}. \tag{5}$$

Lastly, based on the motivation described in Section 4, we aggregate reconstruction errors (i.e., "*summarize*" vectors into scalars) using T different aggregation functions $\texttt{Agg}_t$'s as follows:

$$\texttt{Err}(\mathcal{G}) := [\underbrace{\texttt{Agg}_1(\mathbb{L}_{\mathbf{X}}(\mathcal{G})), \cdots, \texttt{Agg}_T(\mathbb{L}_{\mathbf{X}}(\mathcal{G}))}_{\text{different aggregations of } \mathbb{L}_{\mathbf{X}}(\mathcal{G})}, \underbrace{\texttt{Agg}_1(\mathbb{L}_{\mathbf{A}}(\mathcal{G})), \cdots, \texttt{Agg}_T(\mathbb{L}_{\mathbf{A}}(\mathcal{G}))}_{\text{different aggregations of } \mathbb{L}_{\mathbf{A}}(\mathcal{G})}] \in \mathbb{R}^{2T}. \tag{6}$$

Any aggregation functions that provides a representative summary of $\mathbb{L}_{\mathbf{X}}(\mathcal{G})$ and $\mathbb{L}_{\mathbf{A}}(\mathcal{G})$ are applicable, such as mean and standard deviation. As a result, we represent a graph $\mathcal{G}$ as a $2T$-dimensional vector $\texttt{Err}(\mathcal{G})$ (Eq. (6)), which we refer to as the final *error representation* of $\mathcal{G}$. We present a time complexity analysis of MUSE in Appendix F.3.

After obtaining error representations of graphs, we leverage one-class classifiers on these representations to finally perform anomaly detection. Note that Steps 1 and 2 are decoupled from the application of one-class classifiers (i.e., one-class classifier is trained after completing Steps 1 and 2).

6 Experiments

In this section, we evaluate the effectiveness of MUSE in graph-level anomaly detection (GLAD) by addressing four key research questions.

- **RQ1.** How accurately does MUSE detect anomalous graphs?
- **RQ2.** How robust is MUSE against contamination of the training set?
- **RQ3.** Can the error representations of MUSE distinguish anomalies from normal graphs in the representation space?
- **RQ4.** Are the key components of MUSE essential for its performance?

6.1 Experimental settings

Datasets. Given the absence of benchmark datasets with ground-truth graph-level anomalies, following existing GLAD studies [39, 32, 33, 55, 34, 38], we use graph classification benchmark datasets

Table 1: **GLAD performance**: Mean and standard deviation of test AUROC values ($\times 100$) in the GLAD task are reported. The best and second-best performances are highlighted in green and yellow. A.R. denotes average ranking. MUSE obtains the best average ranking among 18 methods.

	Method	DD	Protein	NCI1	AIDS	Reddit	IMDB	MUTAG	DHFR	BZR	ER	AR
GLAD methods	DOMINANT-G [6]	64.3 (4.4)	55.9 (9.7)	65.5 (6.1)	80.6 (4.0)	58.6 (5.3)	60.8 (6.7)	65.0 (4.2)	56.6 (9.2)	76.2 (7.8)	58.7 (5.5)	10.7
	OCGTL [39]	74.5 (5.1)	71.0 (8.7)	61.2 (5.5)	95.3 (3.7)	69.0 (4.0)	65.8 (5.8)	64.9 (4.9)	66.5 (9.9)	71.3 (17.1)	63.0 (3.6)	6.9
	GLocalKD [34]	47.8 (8.5)	50.7 (8.5)	51.6 (5.6)	51.2 (1.2)	49.8 (4.2)	58.5 (6.7)	55.1 (4.4)	54.1 (8.1)	55.8 (16.7)	54.4 (4.4)	17.0
	GLADC [33]	52.1 (5.2)	50.7 (5.6)	51.4 (3.6)	51.4 (1.0)	52.2 (2.6)	57.7 (5.2)	53.3 (4.5)	55.8 (4.1)	59.0 (14.5)	52.8 (4.2)	16.8
	GLAM [57]	61.6 (5.2)	60.3 (5.6)	58.1 (1.9)	93.6 (2.6)	75.6 (4.0)	65.1 (3.5)	63.0 (2.0)	57.2 (2.7)	72.6 (8.9)	55.2 (2.9)	9.8
	HIMNET [38]	52.1 (3.7)	56.9 (5.8)	53.6 (4.6)	64.3 (3.2)	65.7 (2.4)	61.8 (4.3)	57.5 (2.9)	63.6 (6.7)	72.0 (9.9)	55.7 (2.8)	12.3
	SIGNET [32]	64.2 (9.3)	56.4 (6.4)	63.1 (4.0)	97.2 (1.6)	78.0 (4.4)	48.2 (4.8)	67.5 (1.6)	40.2 (5.8)	66.6 (9.5)	56.2 (4.3)	10.4
SSL- based	GraphCL-1 [53]	64.5 (3.9)	60.7 (4.2)	55.8 (3.1)	71.2 (6.6)	57.7 (5.5)	54.2 (6.2)	53.6 (2.3)	57.8 (6.7)	60.5 (9.3)	55.5 (4.1)	14.2
	GAE-1 [22]	64.7 (5.2)	61.3 (7.0)	62.5 (2.2)	86.2 (1.4)	74.8 (3.2)	63.8 (7.4)	63.2 (3.3)	56.5 (9.6)	68.5 (13.7)	60.0 (3.9)	10.3
	GraphMAE-1 [15]	56.7 (7.3)	60.5 (4.9)	53.4 (3.2)	91.8 (5.3)	72.7 (3.2)	67.0 (5.0)	62.3 (2.6)	62.2 (9.6)	70.1 (7.6)	52.2 (3.6)	10.6
	GraphCL-2 [53]	66.1 (3.0)	59.1 (5.2)	60.3 (4.4)	91.8 (3.5)	77.3 (4.1)	66.3 (5.6)	67.4 (3.3)	59.1 (4.6)	71.9 10.4	67.3 (3.4)	7.2
	GAE-2 [22]	67.2 (3.4)	62.3 (5.0)	62.4 (3.9)	85.8 (1.6)	75.3 (5.7)	66.6 (7.6)	67.3 (3.3)	60.8 (5.6)	72.0 (8.8)	65.7 (2.0)	7.0
	GraphMAE-2 [15]	68.0 (4.3)	61.2 (4.0)	68.3 (3.6)	90.8 (3.6)	75.8 (4.8)	66.7 (5.8)	68.1 (2.4)	61.4 (6.0)	72.8 (6.4)	66.2 (6.4)	5.1
Variants	MUSE w/o $\mathbb{L}_{\mathbf{X}}$	79.4 (3.7)	75.6 (3.7)	69.2 (3.7)	99.6 (0.5)	72.2 (4.0)	65.8 (5.7)	65.8 (3.1)	60.4 (6.6)	65.6 (19.4)	66.3 (3.6)	5.8
	MUSE w/o $\mathbb{L}_{\mathbf{A}}$	61.8 (7.6)	64.7 (7.1)	63.1 (3.3)	89.3 (2.8)	72.0 (4.8)	56.9 (7.1)	57.0 (3.5)	58.1 (3.1)	68.7 (14.2)	60.7 (4.0)	11.0
	MUSE w/o `AVG`	78.6 (4.0)	68.1 (5.5)	68.0 (2.0)	95.0 (2.6)	73.2 (6.6)	66.2 (6.5)	60.9 (3.9)	60.1 (2.4)	66.3 (13.0)	62.0 (3.5)	7.7
	MUSE w/o `STD`	74.3 (5.4)	74.4 (5.2)	65.2 (3.6)	98.7 (0.5)	70.5 (4.3)	70.7 (3.7)	62.0 (2.4)	62.9 (6.4)	71.3 (11.5)	66.7 (2.4)	5.6
	MUSE	80.5 (2.3)	78.4 (2.2)	71.1 (2.0)	99.7 (0.5)	78.4 (5.7)	69.2 (3.5)	67.5 (3.4)	63.8 (8.6)	69.5 (12.6)	67.9 (3.6)	2.2

for evaluation. Specifically, we use 10 datasets from diverse domains, such as chemical molecules, bioinformatics, and social networks. Detailed descriptions of the datasets are in Appendix B.

Training and evaluation. Following Qiu et al. [39], for a dataset with C graph classes, we use C experimental configurations. In each configuration, graphs with one class are treated as normal, while graphs with the other classes are considered anomalies. For each configuration, the normal graphs are split into training, validation, and test sets in an 80%/10%/10% ratio. Additionally, 5% of anomalies are sampled for the validation set (only for hyperparameter tuning) and 5% for the test set. We conduct five trials for each configuration, with each trial using different data splits and model initializations. Each model's performance on each dataset is evaluated by averaging the test mean AUROC across all configurations.

Baseline methods and MUSE. We compare MUSE against 13 baseline methods, including 7 GLAD methods [39, 32, 33, 55, 34, 38, 6] and 6 graph self-supervised learning (SSL) methods [15, 22, 53]. Among SSL methods, those with '-1' in their names are one-stage SSL methods that use SSL losses as anomaly scores. Those with '-2' are two-stage SSL methods that (1) obtain graph embeddings the models learn via SSL and (2) employ a one-class classifier. For aggregation functions of MUSE (Eq. (6)), we use two aggregation functions: mean and standard deviation. Therefore, we represent each graph with a 4-dimensional vector, which is formalized in Appendix D.4. All methods use GIN [49] as a graph encoder. All two-stage approaches use an MLP autoencoder as a one-class classifier. Details of two-stage methods and the MLP autoencoder are in Appendix D.3. In addition, we provide further experimental details, including hyperparameter configurations, in Appendix D.

6.2 RQ1. Performance in graph-level anomaly detection

As shown in Table 1, MUSE outperforms all baseline methods in terms of average ranking. Two points stand out. First, in 8 out of 10 datasets, MUSE outperforms all other GLAD methods. Compared to the second-best GLAD method (OCGTL), MUSE achieves up to 16.2% performance gain (in the NCI1 dataset). Second, in 8 out of 10 datasets, MUSE outperforms all the SSL-based two-stage baselines (GraphCL-2, GAE-2, and GraphMAE-2). Given that they use the same one-class classifier with MUSE, this result demonstrates the effectiveness of the proposed error representation (Eq. (6)).

6.3 RQ2. Robustness against the training set contamination

In real-world scenarios, training data may also contain anomalous graphs, making the robustness of GLAD methods against training set contamination crucial for their practical effectiveness. To assess this robustness, we inject anomalous graphs into the training set at rates of 10%, 20%, and 30%. [3] As shown in Figure 6, MUSE exhibits the least performance drop, among the three strongest GLAD methods, demonstrating its robustness.

[3]Specifically, for a training set containing n graphs, we sample $n \times (k/100)$ anomalous graphs and inject.

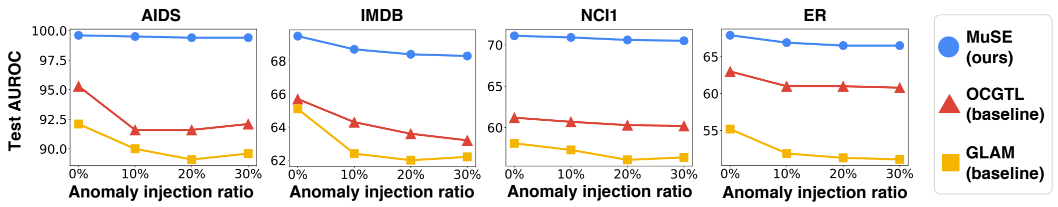


Figure 6: Comparison of the three strongest GLAD methods' robustness against training set contamination. MUSE undergoes the least performance drop among the three methods.

6.4 RQ3. Error representation visualization

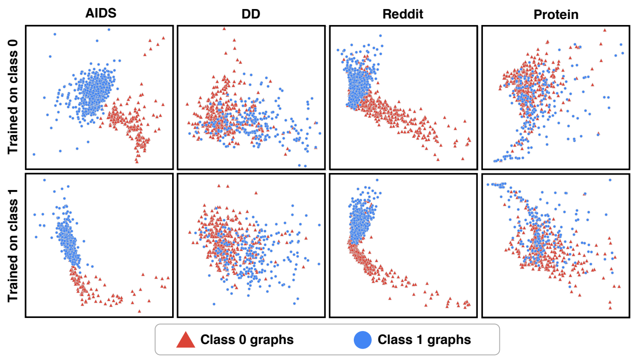


Figure 7: PCA visualization of MUSE's error representations of graphs. Graphs belonging to different classes are well-separated in the representation space.

We evaluate whether the error representation offered by MUSE can effectively distinguish graphs of different classes. We sample half of the graphs belonging to the same class to train MUSE. Then, we visualize the error representations generated by the trained MUSE for the rest half and graphs sampled from another class. As shown in Figure 7, the error representations produced by MUSE are distinguishable across different classes.

6.5 RQ4. Ablation study

Lastly, we provide an ablation study of MUSE. We aim to justify the two key components of MUSE: (1) the reconstruction errors $\mathbb{L}_{\mathbf{X}}$ and $\mathbb{L}_{\mathbf{A}}$, and (2) multiple aggregation functions (specifically, mean and standard deviation). To this end, we leverage the following four variants of MUSE:

(V1) MUSE w/o $\mathbb{L}_{\mathbf{X}}$: A variant that only leverages adjacency matrix reconstruction,
(V2) MUSE w/o $\mathbb{L}_{\mathbf{A}}$: A variant that only leverages node feature reconstruction,
(V3) MUSE w/o `STD`: A variant that only leverages mean aggregation,
(V4) MUSE w/o `AVG`: A variant that only leverages standard-deviation aggregation.

As shown in Table 1, MUSE with all components outperforms all of its variants in 8 out of 10 datasets, justifying the design choice of MUSE.

7 Discussions

In this work, we report and analyze an intriguing phenomenon, *reconstruction flip* of graph autoencoders (Graph-AEs). We investigate the phenomenon theoretically and empirically (Section 3) and further claim their implication in graph-level anomaly detection (GLAD; Section 4). Based on our analysis, we propose a novel GLAD method, MUSE (Section 5), and report its superior performance in GLAD via extensive experiments (Section 6). Below, we conclude the paper by discussing some potential limitations of our research, which could be addressed in future work.

More diverse and general primary patterns. In our analysis, we focus on the two patterns: *community structures* and *a node cycle* (Section 3). Although our claims hold for these patterns, a wide range of graph patterns remains unexplored in this work. Furthermore, a unified framework that can incorporate various graph patterns into a single pattern may exist. Further investigation into these patterns may improve our analysis, leading to new potential applications.

Scalability improvement. Since MUSE reconstructs the entire adjacency matrix of a graph (Eq. (3)), its time complexity for reconstruction is $O(n^2)$, where n is the number of nodes in the graph. While this complexity is manageable for many real-world graph-level datasets (refer to Table 2), leveraging MUSE for large-scale graphs can be challenging. A simple technique that reconstructs only a sampled subset of an adjacency matrix results in a performance drop (refer to Appendix E.5). Therefore, improving the scalability of MUSE while preserving its detection performance is a non-trivial task, making it a promising direction for future research.

Acknowledgements. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00406985) (90%). This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2019-II190075, Artificial Intelligence Graduate School Program (KAIST)) (10%).

References

[1] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):i47–i56, 2005.

[2] Z. Chen, Y. Fu, Y.-X. Wang, L. Ma, W. Liu, and M. Hebert. Image deformation meta-networks for one-shot learning. In *CVPR*, 2019.

[3] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, 2019.

[4] M. Choe, S. Kim, J. Yoo, and K. Shin. Classification of edge-dependent labels of nodes in hypergraphs. In *KDD*, 2023.

[5] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. In *ICCV*, 2017.

[6] K. Ding, J. Li, R. Bhanushali, and H. Liu. Deep anomaly detection on attributed networks. In *SDM*, 2019.

[7] P. D. Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

[8] I. J. Farkas, I. Derényi, A.-L. Barabási, and T. Vicsek. Spectra of "real-world" graphs: Beyond the semicircle law. *Physical Review E*, 64(2):026704, 2001.

[9] S. Freitas, Y. Dong, J. Neil, and D. H. Chau. A large-scale database for graph representation learning. In *NeurIPS*, 2021.

[10] H. Gao and S. Ji. Graph u-nets. In *ICML*, 2019.

[11] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[12] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[13] Z. Guo, F. Wang, K. Yao, J. Liang, and Z. Wang. Multi-scale variational graph autoencoder for link prediction. In *WSDM*, 2022.

[14] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[15] Z. Hou, X. Liu, Y. Cen, Y. Dong, H. Yang, C. Wang, and J. Tang. Graphmae: Self-supervised masked graph autoencoders. In *KDD*, 2022.

[16] Z. Hou, Y. He, Y. Cen, X. Liu, Y. Dong, E. Kharlamov, and J. Tang. Graphmae2: A decoding-enhanced masked self-supervised graph learner. In *WWW*, 2023.

[17] J. Kazius, R. McGuire, and R. Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of medicinal chemistry*, 48(1):312–320, 2005.

[18] S. Kim, D. Lee, Y. Kim, J. Park, T. Hwang, and K. Shin. Datasets, tasks, and training methods for large-scale hypergraph learning. *Data Mining and Knowledge Discovery*, 37(6):2216–2254, 2023.

[19] S. Kim, S. Kang, F. Bu, S. Y. Lee, J. Yoo, and K. Shin. Hypeboy: Generative self-supervised representation learning on hypergraphs. In *ICLR*, 2024.

[20] S. Kim, S. Y. Lee, Y. Gao, A. Antelmi, M. Polato, and K. Shin. A survey on hypergraph neural networks: an in-depth and step-by-step guide. In *KDD*, 2024.

[21] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[22] T. N. Kipf and M. Welling. Variational graph auto-encoders. In *NeurIPS Worksop of Bayesian Deep Learning*, 2016.

[23] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[24] G. Kollias, V. Kalantzis, T. Idé, A. Lozano, and N. Abe. Directed graph auto-encoders. In *AAAI*, 2022.

[25] T. Lanciano, F. Bonchi, and A. Gionis. Explainable classification of brain networks via contrast subgraphs. In *KDD*, 2020.

[26] S. Laue, M. Mitterreiter, and J. Giesen. Computing higher order derivatives of matrix and tensor expressions. In *NeurIPS*, 2018.

[27] J. Lee, S. Kim, and K. Shin. Slade: Detecting dynamic anomalies in edge streams without labels via self-supervised learning. In *KDD*, 2024.

[28] S. Y. Lee, F. Bu, J. Yoo, and K. Shin. Towards deep attention in graph neural networks: Problems and remedies. In *ICML*, 2023.

[29] S. Y. Lee, S. Kim, F. Bu, J. Yoo, J. Tang, and K. Shin. Feature distribution on graph topology mediates the effect of graph convolution: Homophily perspective. In *ICML*, 2024.

[30] L. Liang, S. Kim, K. Shin, Z. Xu, S. Pan, and Y. Qi. Sign is not a remedy: Multiset-to-multiset message passing for learning on heterophilic graphs. In *ICML*, 2024.

[31] W. Liu, H. Chang, B. Ma, S. Shan, and X. Chen. Diversity-measurable anomaly detection. In *CVPR*, 2023.

[32] Y. Liu, K. Ding, Q. Lu, F. Li, L. Y. Zhang, and S. Pan. Towards self-interpretable graph-level anomaly detection. In *NeurIPS*, 2023.

[33] X. Luo, J. Wu, J. Yang, S. Xue, H. Peng, C. Zhou, H. Chen, Z. Li, and Q. Z. Sheng. Deep graph level anomaly detection with contrastive learning. *Scientific Reports*, 12(1):19867, 2022.

[34] R. Ma, G. Pang, L. Chen, and A. van den Hengel. Deep graph-level anomaly detection by glocal knowledge distillation. In *WSDM*, 2022.

[35] X. Ma, J. Wu, J. Yang, and Q. Z. Sheng. Towards graph-level anomaly detection via deep evolutionary mapping. In *KDD*, 2023.

[36] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *Workshop on Graph Representation Learning and Beyond (GRL+)*, 2020.

[37] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. Propagation kernels: efficient graph kernels from propagated information. *Machine learning*, 102:209–245, 2016.

[38] C. Niu, G. Pang, and L. Chen. Graph-level anomaly detection via hierarchical memory networks. In *ECML-PKDD*, 2023.

[39] C. Qiu, M. Kloft, S. Mandt, and M. Rudolph. Raising the bar in graph-level anomaly detection. In *IJCAI*, 2022.

[40] K. Riesen and H. Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop*, pages 287–297. Springer, 2008.

[41] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[42] J. J. Sutherland, L. A. O'brien, and D. F. Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences*, 43(6):1906–1915, 2003.

[43] Q. Tan, N. Liu, X. Huang, S.-H. Choi, L. Li, R. Chen, and X. Hu. S2gae: self-supervised graph autoencoders are generalizable learners with graph masking. In *WSDM*, 2023.

[44] W. Tu, Q. Liao, S. Zhou, X. Peng, C. Ma, Z. Liu, X. Liu, Z. Cai, and K. He. Rare: Robust masked graph autoencoder. *IEEE Transactions on Knowledge and Data Engineering*, 36(10): 5340–5353, 2024.

[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. In *ICLR*, 2017.

[46] H. G. Vogel. *Drug discovery and evaluation: pharmacological assays*. Springer Science & Business Media, 2002.

[47] N. Wale, I. A. Watson, and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14:347–375, 2008.

[48] J. Xia, L. Wu, J. Chen, B. Hu, and S. Z. Li. Simgrace: A simple framework for graph contrastive learning without data augmentation. In *WWW*, 2022.

[49] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[50] X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining significant graph patterns by leap search. In *SIGMOD*, 2008.

[51] P. Yanardag and S. Vishwanathan. Deep graph kernels. In *KDD*, 2015.

[52] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[53] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph contrastive learning with augmentations. In *NeurIPS*, 2020.

[54] G. Zhang, Z. Yang, J. Wu, J. Yang, S. Xue, H. Peng, J. Su, C. Zhou, Q. Z. Sheng, L. Akoglu, et al. Dual-discriminative graph neural network for imbalanced graph-level anomaly detection. In *NeurIPS*, 2022.

[55] Z. Zhang and L. Zhao. Unsupervised deep subgraph anomaly detection. In *ICDM*, 2022.

[56] L. Zhao and L. Akoglu. On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights. *Big Data*, 11(3):151–180, 2023.

[57] L. Zhao, S. Sawlani, A. Srinivasan, and L. Akoglu. Graph anomaly detection with unsupervised gnns. In *ICDM*, 2022.

[58] Y. Zhou, H. Huo, Z. Hou, L. Bu, J. Mao, Y. Wang, X. Lv, and F. Bu. Co-embedding of edges and nodes with deep graph convolutional neural networks. *Scientific Reports*, 13(1):16966, 2023.

A Proof

In this section, we provide proof of each theorem presented in the paper.

A.1 Proof of Theorem 1

Theorem 1 (Suspicious generalization across τ)**.** *For any $n \geq 12$ and $0 < \tau_1 \leq 1$, there exists $\epsilon > 0$ such that for any $0 < \gamma \leq \epsilon$ and any $0 < \tau_2 \leq 1$ with the initial weight matrix $\mathbf{W}^{(0)} = \mathbb{I}_n$,*

$$\underbrace{\mathbb{E}_{\mathcal{G}}[\mathcal{L}(\mathcal{G}, \mathbb{W}(\tau_1, \mathbf{W}^{(0)}, \gamma))|\tau_2]}_{\text{the error on graphs of strength } \tau_2 \text{ after update}} < \underbrace{\mathbb{E}_{\mathcal{G}}[\mathcal{L}(\mathcal{G}, \mathbf{W}^{(0)})|\tau_2]}_{\text{the error on graphs of strength } \tau_2 \text{ before update}} .$$

Proof. Note that the reconstruction loss is defined as follows:

$$\mathcal{L} \coloneqq \|\mathbf{A} - \mathbf{AXW}(\mathbf{AXW})^T\|_F^2. \tag{7}$$

Here, since $\mathbf{X}$ is identity, Eq. (7) is equal to $\mathcal{L} \coloneqq \|\mathbf{A} - \mathbf{AWWA}\|_F^2$. The derivative of Eq. (7) and its expectation are as follows [26]:

$$\left.\frac{\partial \mathcal{L}}{\partial \mathbf{W}}\right|_{\mathbf{W}=\mathbb{I}_n} = 2(\mathbf{A}^4 - \mathbf{A}^3),$$
$$\mathbb{E}_{\mathbf{A}}\left[\left.\frac{\partial \mathcal{L}}{\partial \mathbf{W}}\right|_{\mathbf{W}=\mathbb{I}_n}\right] = 2(\mathbb{E}[\mathbf{A}^4] - \mathbb{E}[\mathbf{A}^3]).$$

Given n (the total number of nodes) and τ (the community strength), let $N = \frac{n}{2}$ (the number of nodes in each community) and $p = \frac{1+\tau}{2}$ (the inner-community edge probability), we have

(1) If $i = j$:

$$\begin{aligned}\mathbb{E}[\mathbf{A}_{ij}^3] &= (N-1)(N-2)p^3 + 2N(N-1)p(1-p)^2 + N(N-1)p(1-p)^2,\\ &= (N-1)(N-2)p^3 + 3N(N-1)p(1-p)^2,\end{aligned}$$

and

$$\begin{aligned}\mathbb{E}[\mathbf{A}_{ij}^4] &= (N-1)(N-2)(N-3)p^4 + (N-1)(N-2)p^2 + (N-1)(N-2)p^2 + (N-1)p\\ &\quad + 2(N-1)(N-2)Np^2(1-p)^2 + 2N(N-1)p(1-p)^2\\ &\quad + (N-1)(N-2)Np^2(1-p)^2 + (N-1)Np(1-p)\\ &\quad + 2N^2(N-1)p^2(1-p)^2 + N(N-1)^2(1-p)^4 + N(N-1)(1-p)^2 + N(N-1)(1-p)^2N(1-p)\\ &\quad + N(N-1)(N-2)p^2(1-p)^2 + N(N-1)p(1-p),\\ &= (-6 + 11N - 6N^2 + N^3)p^4 + 2(-1+N)p^2(-2 + N - 4N(1-p)^2 + 3N^2(1-p)^2)\\ &\quad + N(1-p)(1 + 2(-1+N)(1-p) + (-1+N)^2(1-p)^3)\\ &\quad + p(-1 + 2N^2(1-p)(1 + (1-p)) + N(1 - 2(1-p) - 2(1-p)^2)),\end{aligned}$$

(2) If i and j are in the same group:

$$\begin{aligned}\mathbb{E}[\mathbf{A}_{ij}^3] &= (N-2)(N-3)p^3 + p + 2N(N-2)p(1-p)^2 + 2Np(1-p) + N(N-1)p(1-p)^2,\\ &= (N-2)(N-3)p^3 + N(3N-5)p(1-p)^2 + p + 2Np(1-p),\end{aligned}$$

and

$$
\begin{aligned}
\mathbb{E}[\mathbf{A}_{ij}^4] = {} & (N-2)(N-3)(N-4)p^4 + (N-2)(N-3)p^3 + 2(N-2)(N-3)p^3 + 2(N-2)p^3 \\
& + (N-2)p^3 + 2(N-2)(N-3)p^4 + 2(N-2)p^2 \\
& + 2(N-2)(N-3)Np^2(1-p)^2 + 2(N-2)Np(1-p)^2 + 2Np(1-p)^2 + 2(N-2)Np^2(1-p)^2 \\
& + (N-2)(N-3)Np^2(1-p)^2 + 2(N-2)Np^2(1-p)^2 + Np(1-p)^2 + 2N(N-1)(N-2)p^2(1-p)^2 \\
& + 2N(N-1)p^2(1-p)^2 + N(N-1)(N-2)(1-p)^4 + N(N-2)(1-p)^3 + 2N(N-1)(1-p)^4 \\
& + 2N(1-p)^2 + N(N-1)(N-2)p^2(1-p)^2 + N(N-1)p(1-p)^2, \\
= {} & 3(-2+N)^2p^3 + (-3+N)(-2+N)^2p^4 + N(-2+3N)p(1-p)^2 \\
& + N(1-p)^2(2 + (-2+N)(1-p) + (-1+N)N(1-p)^2) \\
& + 2p^2(-2+N+7N(1-p)^2 - 9N^2(1-p)^2 + 3N^3(1-p)^2).
\end{aligned}
$$

(3) If i and j are in different groups

$$
\begin{aligned}
\mathbb{E}[\mathbf{A}_{ij}^3] = {} & 2(N-1)(N-2)p^2(1-p) + 2(N-1)p(1-p) \\
& + (N-1)^2p^2(1-p) + (N-1)^2(1-p)^3 + (1-p), \\
= {} & (N-1)(3N-5)p^2(1-p) + (N-1)^2(1-p)^3 + 2(N-1)p(1-p) + (1-p),
\end{aligned}
$$

and

$$
\begin{aligned}
\mathbb{E}[\mathbf{A}_{ij}^4] = {} & (N-1)(N-2)(N-3)p^3(1-p) + (N-1)(N-2)p^3(1-p) + (N-1)(N-2)p^2(1-p) \\
& + (N-1)(N-2)p^2(1-p) + (N-1)p(1-p) + (N-1)^2(N-2)p^3(1-p) \\
& + (N-1)(N-2)p^2(1-p) + (N-1)^2(N-2)p(1-p)^3 + (N-1)^2p(1-p)^3 \\
& + (N-1)(N-2)p(1-p)^2 + (N-1)p(1-p)^2 + (N-1)^2p(1-p)^2 + (N-1)p(1-p) \\
& + (N-1)^2(N-2)p^3(1-p) + (N-1)^2p^2(1-p) + (N-1)^2(N-2)p(1-p)^3 \\
& + (N-1)^2p(1-p)^2 + (N-1)(N-2)p(1-p)^3 + (N-1)^2p(1-p)^3 + (N-1)p(1-p)^2 \\
& + (N-1)p(1-p)^2 + (N-1)^2(N-2)p(1-p)^3 + (N-1)(N-2)p(1-p)^2 \\
& + (N-1)^2p(1-p)^3 + (N-1)^2p(1-p)^2 + (N-1)p(1-p)^2 + (N-1)p(1-p) \\
& + (N-1)^2(N-2)p(1-p)^3 + (N-1)^2p(1-p)^3 + (N-1)^2p(1-p)^3 + (N-1)(N-2)p(1-p)^3 \\
& + (N-1)p(1-p)^2 + (N-1)p(1-p)^2 + (N-1)(N-2)(N-3)p^3(1-p) \\
& + (N-1)(N-2)p^2(1-p) + (N-1)(N-2)p^2(1-p) + (N-1)p(1-p) \\
= {} & (-1+N)p(1-p)\triangle,
\end{aligned}
$$

where

$$\triangle = 4 + (-11+6N)p + (14-15N+4N^2)p^2 + (-1+5N)(1-p) + (-1-5N+4N^2)(1-p)^2.$$

Hence, the entries in

$$\mathbb{E}_A[\mathbf{A}^4 - \mathbf{A}^3] = \frac{1}{2}\mathbb{E}_{\mathbf{A}}\left[\left.\frac{\partial\mathcal{L}}{\partial\mathbf{W}}\right|_{\mathbf{W}=\mathbb{I}_n}\right]$$

have three different values: (1) i and j are equal, (2) $i \neq j$ and v_i and v_j belong to the same group, and (3) $i \neq j$ and v_i and v_j belong to the different groups. Thus, we can express $\mathbb{E}_A[\mathbf{A}^4 - \mathbf{A}^3]$ as $a\mathbf{I} + b\mathbf{P} + c\mathbf{U}$, where $a, b, c \in \mathbb{R}$ and $\mathbf{P}$ and $\mathbf{U}$ are defined as:

$$\mathbf{P} = \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}, \text{ and } \mathbf{U} = \begin{pmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} \end{pmatrix}, \tag{8}$$

where $\mathbf{1} \in \{1\}^{N\times N}$ and $\mathbf{0} \in \{0\}^{N\times N}$.

Specifically,

$$
\begin{aligned}
a &= 3(N-2)(N-3)p^4 + 2N(4N-6)p(1-p)^2 + N(N-1)(1-p)^4 \\
b &= (N-2)(N-3)p^3((N-4)p-1) + N(3N-5)p(1-p)^2(2(N-2)p-1) \\
&\quad + N(N-1)(N-2)(1-p)^4, \\
&= N^3(1-4p+12p^2-16p^3+8p^4) + N^2(-3+9p-34p^2+52p^3-34p^4) \\
&\quad + N(2-3p+22p^2-38p^3+48p^4) - 6p^3(1+4p) \\
c &= (N-1)p^2(1-p)(4(N-2)^2p-3N+5) + (N-1)(1-p)^3(4(N-2)^2p-N+1)
\end{aligned}
$$

For any training graphs with $0 < \tau_1 \leq 1$, i.e., $0.5 < p \leq 1$, we have

- For any $N \geq 4$, $a > 0$,
- For any $N \geq 6$, $b > 0$,
- For any $N \geq 4$, $c > 0$.

Thus, $a, b, c > 0$ for any $N \geq 6$ (i.e., $n \geq 12$) hold.

Let $W' = \mathbb{W}(\tau_1, \mathbf{W}^{(0)}, \gamma)$ be the updated W after one step of gradient descent with learning rate γ trained on graphs with τ_1 (and n). Now, we derive the reconstruction loss on test graphs computed with W'. Formally, reconstruction loss by using W' is defined as follows:

$$\mathcal{L} = \|\mathbf{A} - \mathbf{A}W'(\mathbf{A}W')^T\|_2^2 = \|\mathbf{A} - \mathbf{A}(W')^2\mathbf{A}\|_2^2. \tag{9}$$

We further simplify $(W')^2$ as follows:

$$(W')^2 = \mathbf{I} + 2(\epsilon_1\mathbf{I} + \epsilon_2\mathbf{P} + \epsilon_3\mathbf{U}) + (\epsilon_1\mathbf{I} + \epsilon_2\mathbf{P} + \epsilon_3\mathbf{U})^2, \tag{10}$$

where $\epsilon_1 = -\frac{1}{2}\gamma a$, $\epsilon_2 = -\frac{1}{2}\gamma b$, and $\epsilon_3 = -\frac{1}{2}\gamma c$.

To simplify Eq. (10), we use the property that $\mathbf{P}^2 = N\mathbf{P}$, $\mathbf{U}^2 = N\mathbf{U}$, and $\mathbf{PU} = \mathbf{UP} = N\mathbf{U}$. Using these facts, we rewrite Eq. (10) as below:

$$(W')^2 = \mathbf{I} + 2(\epsilon_1\mathbf{I} + \epsilon_2\mathbf{P} + \epsilon_3\mathbf{U}) + (\epsilon_1^2\mathbf{I} + \epsilon_2^2 N\mathbf{P} + \epsilon_3^2 N\mathbf{U}) + 2\epsilon_1\epsilon_2\mathbf{P} + 2\epsilon_1\epsilon_3\mathbf{U} + 2\epsilon_2\epsilon_3 N\mathbf{U}, \tag{11}$$

where

$$
\begin{aligned}
\alpha_1 &= 2\epsilon_1 + \epsilon_1^2 \\
\alpha_2 &= 2\epsilon_2 + N\epsilon_2^2 + 2\epsilon_1\epsilon_2 \\
\alpha_3 &= 2\epsilon_3 + N\epsilon_3^2 + 2\epsilon_1\epsilon_3 + 2N\epsilon_2\epsilon_3
\end{aligned}
$$

Now, by using the derived results, we rewrite the reconstruction loss term Eq. (9) as below:

$$\mathcal{L} = \|\mathbf{A} - \mathbf{A}(\mathbf{I} + \alpha_1\mathbf{I} + \alpha_2\mathbf{P} + \alpha_3\mathbf{P})\mathbf{A}\|_F^2 = \|(\mathbf{A} - \mathbf{A}^2) - (\alpha_1\mathbf{A}^2 + \alpha_2\mathbf{APA} + \alpha_3\mathbf{AUA})\|_F^2, \tag{12}$$

where

$$
\begin{aligned}
(\mathbf{APA})_{ij} &= \sum_{k,\ \ell \text{ in the same group}} \mathbf{A}_{ik}\mathbf{A}_{jl} \\
(\mathbf{AUA})_{ij} &= \sum_{k,\ \ell \text{ in different groups}} \mathbf{A}_{ik}\mathbf{A}_{jl} \\
(\mathbf{A}^2)_{ij} &= \sum_k \mathbf{A}_{ik}\mathbf{A}_{jk}
\end{aligned}
$$

We can decompose the loss by

$$\mathcal{L} = \sum_{i,j} \big(\mathbf{A}_{ij} - (1+\alpha_1)(\mathbf{A}^2)_{ij} - \alpha_2(\mathbf{APA})_{ij} - \alpha_3(\mathbf{AUA})_{ij}\big)^2.$$

Given i and j, let $x = A_{ij}$, $y = (\mathbf{A}^2)_{ij}$, $z = (\mathbf{APA})_{ij}$, and $w = (\mathbf{AUA})_{ij}$, we have

$$\begin{aligned}\mathcal{L}_{ij} &= (x - (1+\alpha_1)y - \alpha_2 z - \alpha_3 w)^2 \\ &= x^2 + -2(1+\alpha_1)xy - 2\alpha_2 xz - 2\alpha_3 xw + (1+\alpha_1)^2 y^2 \\ &\quad + 2(1+\alpha_1)\alpha_2 yz + 2(1+\alpha_1)\alpha_3 yw + \alpha_2^2 z^2 + 2\alpha_2\alpha_3 zw + \alpha_3^2 w^2.\end{aligned}$$

Let $\bar{\mathcal{L}}_{ij} = (\mathbf{A}_{ij} - (\mathbf{A}^2)_{ij})^2 = (x-y)^2$. By using this notation, we derive the following result:

$$\begin{aligned}\mathcal{L}_{ij} - \bar{\mathcal{L}}_{ij} &= -2\alpha_1 xy - 2\alpha_2 xz - 2\alpha_3 xw + \alpha_1(\alpha_1+2)y^2 \\ &\quad + 2(1+\alpha_1)\alpha_2 yz + 2(1+\alpha_1)\alpha_3 yw + \alpha_2^2 z^2 + 2\alpha_2\alpha_3 zw + \alpha_3^2 w^2.\end{aligned}$$

We shall show that

$$\mathbb{E}[\sum_{i,j}(\mathcal{L}_{ij} - \bar{\mathcal{L}}_{ij})] < 0.$$

As $\gamma \to 0^+$, ignoring higher-order terms and keeping the term linear w.r.t. γ, it suffices to show that

$$\sum_{i,j} a(\mathbb{E}[xy] - \mathbb{E}[y^2]) + b(\mathbb{E}[xz] - \mathbb{E}[yz]) + c(\mathbb{E}[xw] - \mathbb{E}[yw]) < 0.$$

Note: In the following derivation, p is from test graphs (i.e., $p = \frac{\tau_2+1}{2}$).

(1) If $i = j$

$$\begin{aligned}\mathbb{E}[xy] &= 0 \\ \mathbb{E}[xz] &= 0 \\ \mathbb{E}[xw] &= 0 \\ \mathbb{E}[y^2] &= (N-1)p + (N-1)(N-2)p^2 + N(1-p) + N(N-1)(1-p)^2 + 2N(N-1)p(1-p) \\ &= N^2 + N(-2p^2) + (2p^2 - p) \\ \mathbb{E}[yz] &= (N-1)(N-2)(N-3)p^3 + 3(N-1)(N-2)p^2 + (N-1)p \\ &\quad + N(N-1)^2 p(1-p)^2 + N(N-1)p(1-p) \\ &\quad + N(N-1)(N-2)p^2(1-p) + N(N-1)p(1-p) \\ &\quad + N(N-1)(N-2)(1-p)^3 + 3N(N-1)(1-p)^2 + N(1-p) \\ &= N^3(1-2p+2p^2) + N^2(3p-4p^2-2p^3) \\ &\quad + N(-p-4p^2+8p^3) + (-p+6p^2-6p^3) \\ \mathbb{E}[yw] &= 2N(N-1)(N-2)p^2(1-p) + 2N(N-1)p(1-p) \\ &\quad + 2N(N-1)^2 p(1-p)^2 + 2N(N-1)p(1-p) \\ &= N^3(2p-2p^2) + N^2(2p^3-2p^2) + N(4p^2-2p-2p^3)\end{aligned}$$

Hence,

$$\begin{aligned}d_{11} &:= \mathbb{E}[xy] - \mathbb{E}[y^2] = -N^2 + N(2p^2) + (p - 2p^2) \\ d_{12} &:= \mathbb{E}[xz] - \mathbb{E}[yz] = N^3(-1+2p-2p^2) + N^2(-3p+4p^2+2p^3) \\ &\qquad\qquad\qquad\qquad + N(p+4p^2-8p^3) + (p-6p^2+6p^3) \\ d_{13} &:= \mathbb{E}[xw] - \mathbb{E}[yw] = N^3(-2p+2p^2) + N^2(-2p^3+2p^2) + N(-4p^2+2p+2p^3)\end{aligned}$$

(2) Else if i and j are in the same group

$$\begin{aligned}
\mathbb{E}[xy] &= p\mathbb{E}[y] = Np(1-2p+2p^2) - 2p^3 \\
\mathbb{E}[xz] &= p\mathbb{E}[z] = N^2p(1-2p+2p^2) - 2Np^3 + p^3 \\
\mathbb{E}[xw] &= p\mathbb{E}[w] = 2N^2(p^2-p^3) + 4N(p^3-p^2) \\
\mathbb{E}[y^2] &= (N-2)p^2 + (N-2)(N-3)p^4 + N(1-p)^2 + N(N-1)(1-p)^4 + 2N(N-2)p^2(1-p)^2 \\
&= N^2(1-2p+2p^2)^2 + 2Np(1-4p+6p^2-5p^3) + 6p^4 - 2p^2 \\
\mathbb{E}[yz] &= (N-2)(N-3)(N-4)p^4 + 2(N-2)(N-3)p^4 + (N-2)p^3 \\
&\quad + (N-2)(N-3)p^4 + 2(N-2)(N-3)p^3 + (N-2)p^2 \\
&\quad + N(N-1)(N-2)p^2(1-p)^2 + N(N-2)p^2(1-p) \\
&\quad + N(N-2)(N-3)p^2(1-p)^2 + N(N-2)p(1-p)^2 \\
&\quad + N(N-1)(N-2)(1-p)^4 + N(N-1)(1-p)^4 + 2N(N-1)(1-p)^3 + N(1-p)^2 \\
&= N^3(1-2p+2p^2)^2 - 2p^2(1-5p+3p^2) \\
&\quad + N^2p(3-15p+24p^2-16p^3) + Np(-2+12p-27p^2+20p^3) \\
\mathbb{E}[yw] &= 2N(N-2)(N-3)p^3(1-p) + 2N(N-2)p^3(1-p) + 2N(N-2)p^2(1-p) \\
&\quad + 2N(N-1)(N-2)p(1-p)^3 + 2N(N-2)p(1-p)^3 + 2N(N-2)p(1-p)^2 \\
&= N^3(2p-6p^2+8p^3-4p^4) + N^2(-2p+10p^2-20p^3+12p^4) + N(-4p+4p^2+8p^3-8p^4)
\end{aligned}$$

Hence,

$$\begin{aligned}
d_{21} &:= \mathbb{E}[xy] - \mathbb{E}[y^2] = -N^2(1-2p+2p^2)^2 + Np(-1+6p-10p^2+10p^3) + 2p^2(1-p-3p^2) \\
d_{22} &:= \mathbb{E}[xz] - \mathbb{E}[yz] = -N^3(1-2p+2p^2)^2 + p^2(2-9p+6p^2) \\
&\qquad + Np(2-12p+25p^2-20p^3) + N^2p(-2+13p-22p^2+16p^3) \\
d_{23} &:= \mathbb{E}[xw] - \mathbb{E}[yw] = 2N(-1+p)p(N(-1+3p-6p^2) + N^2(1-2p+2p^2) + 2(-1+p+2p^2))
\end{aligned}$$

(3) Else if i and j are in different groups

$$\begin{aligned}
\mathbb{E}[xy] &= (1-p)\mathbb{E}[y] = 2(N-1)p(1-p)^2 \\
\mathbb{E}[xz] &= (1-p)\mathbb{E}[z] = 2(N-1)^2p(1-p)^2 \\
\mathbb{E}[xw] &= (1-p)\mathbb{E}[w] = (N-1)^2p^2(1-p) + N^2(1-p)^3 \\
\mathbb{E}[y^2] &= 2(N-1)p(1-p) + 2(N-1)(N-2)p^2(1-p)^2 + 2(N-1)^2p^2(1-p)^2 \\
&= 2(N-1)p(1-p) + 2(N-1)(2N-3)p^2(1-p)^2 \\
\mathbb{E}[yz] &= 2(N-1)(N-2)(N-3)p^2(1-p)^2 + 2(N-1)(N-2)p^2(1-p)^2 \\
&\quad + 2(N-1)(N-2)p^2(1-p)^2 + 2(N-1)(N-2)p(1-p)^2 \\
&\quad + 2(N-1)(N-2)p^2(1-p) + 2(N-1)p(1-p) \\
&\quad + 2(N-1)^2(N-2)p^2(1-p)^2 + 2(N-1)^2p^2(1-p)^2 \\
&= N^3(4p^2-8p^3+4p^4) + N^2(2p-16p^2+28p^3-14p^4) \\
&\quad + N(-4p+20p^2-32p^3+16p^4) + 2p-8p^2+12p^3-6p^4 \\
\mathbb{E}[yw] &= 2(N-1)^2(N-2)p^3(1-p) + 2(N-1)^2p^2(1-p) \\
&\quad + 2(N-1)^2(N-2)p(1-p)^3 + 2(N-1)^2p(1-p)^3 \\
&\quad + (N-1)(N-2)p(1-p)^3 + (N-1)p(1-p)^2 \\
&\quad + (N-1)^2p(1-p)^2 + (N-1)p(1-p)^2 \\
&= N^3(2p-6p^2+8p^3-4p^4) + N^2(-4p+15p^2-24p^3+13p^4) \\
&\quad + N(3p-13p^2+23p^3-13p^4) + (-p+4p^2-7p^3+4p^4)
\end{aligned}$$

Hence,

$$\begin{aligned}
d_{31} &:= \mathbb{E}[xy] - \mathbb{E}[y^2] = -2(-1+N)(2+2N(-1+p)-3p)(-1+p)p^2 \\
d_{32} &:= \mathbb{E}[xz] - \mathbb{E}[yz] = -2(-1+N)^2(2+2N(-1+p)-3p)(-1+p)p^2 \\
d_{33} &:= \mathbb{E}[xw] - \mathbb{E}[yw] = (-1+p)(p(-1+2p-4p^2)+2N^3p(1-2p+2p^2) \\
&\qquad + Np(3-8p+13p^2) - N^2(1+2p-9p^2+13p^3))
\end{aligned}$$

Conclusion.

Recall that $a, b, c > 0$ for any $n \geq 12$ and $0 < \tau_1 \leq 1$ and we aim to show that

$$\sum_{i,j} a(\mathbb{E}[xy] - \mathbb{E}[y^2]) + b(\mathbb{E}[xz] - \mathbb{E}[yz]) + c(\mathbb{E}[xw] - \mathbb{E}[yw]) < 0.$$

It suffices to show that, for any $0.5 < p \leq 1$,

$$\sum_{i,j}(\mathbb{E}[xy] - \mathbb{E}[y^2]) \leq 0 \sum_{i,j}(\mathbb{E}[xz] - \mathbb{E}[yz]) \leq 0 \sum_{i,j}(\mathbb{E}[xw] - \mathbb{E}[yw]) \leq 0$$

with at least one inequality being strict. Since there are

- $2N$ pairs of (i, j) such that $i = j$,
- $2N(N-1)$ pairs of (i, j) such that $i \neq j$ and i and j are in the same group, and
- $2N^2$ pairs of (i, j) such that $i \neq j$ and i and j are in different groups,

it is equivalent to showing that

$$d_{1i} + (N-1)d_{2i} + Nd_{3i} \leq 0, \forall i \in \{1, 2, 3\},$$

and at least for one i value, the inequality is strict.

Indeed,

$$\begin{aligned}
d_{11} + (N-1)d_{21} + Nd_{31} &= -N^4(1-2p+2p^2)^2 + 2Np(-1+6p-12p^2+10p^3) \\
&\quad + N^2(-1+4p-17p^2+29p^3-26p^4) + N^3(1-6p+17p^2-22p^3+16p^4) \\
&\quad + (p-4p^2+9p^3-6p^4) < 0
\end{aligned}$$

for any $N \geq 4$;

$$\begin{aligned}
d_{12} + (N-1)d_{22} + Nd_{32} &= N^4(-1+4p-12p^2+16p^3-8p^4) + N^3p(-4+31p-56p^2+34p^3) \\
&\quad + N^2(p-33p^2+77p^3-52p^4) + Np(-1+22p-52p^2+32p^3) \\
&\quad + p - 8p^2 + 15p^3 - 6p^4 < 0
\end{aligned}$$

for any $N \geq 3$;

$$\begin{aligned}
d_{13} + (N-1)d_{23} + Nd_{33} &= N(-1+p)(p-12p^3+4N^3p(1-2p+2p^2) \\
&\qquad + Np(1-12p+33p^2) - N^2(1+4p-19p^2+29p^3)) < 0,
\end{aligned}$$

for any $N \geq 2$.

In conclusion, when trained with any $0.5 < p_{train} \leq 1$ (i.e., $0 < \tau_1 \leq 1$) and $N_{train} \geq 6$ (i.e., $n \geq 12$), as γ approaches 0^+, for any $0.5 < p_{test} \leq 1$ (i.e., $0 < \tau_2 \leq 1$),

$$\begin{aligned}
d_1 &:= d_{11} + (N-1)d_{21} + Nd_{31} < 0 \\
d_2 &:= d_{12} + (N-1)d_{22} + Nd_{32} < 0 \\
d_3 &:= d_{13} + (N-1)d_{23} + Nd_{33} \leq 0,
\end{aligned}$$

completing the proof.

□

A.2 Proof of Theorem 2

Theorem 2 (Correlation with τ). *For any $n \geq 34$ and $0 < \tau_1 \leq 0.98$, there exists $\epsilon > 0$ such that for any $0 < \gamma \leq \epsilon$ and any $\tau_1 + 0.02 \leq \tau_2 \leq 1$ with the initial weight $\mathbf{W}^{(0)} = \mathbb{I}_n$,*

$$\underbrace{\mathbb{E}_{\mathcal{G}}\left[\mathcal{L}(\mathcal{G}, \mathbf{W}^{(0)}) - \mathcal{L}(\mathcal{G}, \mathbb{W}(\tau_1, \mathbf{W}^{(0)}, \gamma)) \mid \tau_1\right]}_{\textit{the decrease in error on graphs of strength } \tau_1 \textit{ (training graphs)}} < \underbrace{\mathbb{E}_{\mathcal{G}}\left[\mathcal{L}(\mathcal{G}, \mathbf{W}^{(0)}) - \mathcal{L}(\mathcal{G}, \mathbb{W}(\tau_1, \mathbf{W}^{(0)}, \gamma)) \mid \tau_2\right]}_{\textit{the decrease in error on graphs of strength } \tau_2}$$

Proof. Following the notations in the proof of Theorem 1 above, regarding the "speed" of gradient,

$$\begin{aligned}
\frac{\partial d_1}{\partial p} &= 4N^4(1 - 4p + 6p^2 - 4p^3) + N^3(-6 + 34p - 66p^2 + 64p^3) \\
&\quad + N^2(4 - 34p + 87p^2 - 104p^3) + N(-2 + 24p - 72p^2 + 80p^3) \\
&\quad + (1 - 8p + 27p^2 - 24p^3) \\
\frac{\partial d_2}{\partial p} &= 4N^4(1 - 2p)^3 + 2N^3(-2 + 31p - 84p^2 + 68p^3) \\
&\quad + N^2(1 - 66p + 231p^2 - 208p^3) + N(-1 + 44p - 156p^2 + 128p^3) \\
&\quad + (1 - 16p + 45p^2 - 24p^3) \\
\frac{\partial d_3}{\partial p} &= 4N^4(-1 + 2p)^3 + N^3(3 - 46p + 144p^2 - 116p^3) \\
&\quad + N^2(-1 + 26p - 135p^2 + 132p^3) + N(-1 + 2p + 36p^2 - 48p^3).
\end{aligned}$$

Trained with $p' = \frac{\tau_1+1}{2}$ and testing with $p = \frac{\tau_2+1}{2}$, we have

$$\begin{aligned}
f(p, p', N) &:= \frac{\partial(ad_1 + bd_2 + cd_2)}{\partial p} \\
&= \underbrace{c_0(N)}_{\text{a positive constant determined by } N} \lim_{\gamma \to 0^+} -\frac{1}{\gamma}\mathbb{E}_{\mathcal{G}}\left[\mathcal{L}(\mathcal{G}, \mathbf{W}^{(0)}) - \mathcal{L}(\mathcal{G}, \mathbb{W}(\tau_1, \mathbf{W}^{(0)}, \gamma)) \mid \tau_2\right] \\
&= a(N, p')\frac{\partial d_1(N, p)}{\partial p} + b(N, p')\frac{\partial d_2(N, p)}{\partial p} + c(N, p')\frac{\partial d_3(N, p)}{\partial p} \\
&= -((-1 + N)N(-1 + 2p + 36p^2 - 48p^3 + 4N^3(-1 + 2p)^3 + N^2(3 - 46p + 144p^2 - 116p^3) \\
&\quad + N(-1 + 26p - 135p^2 + 132p^3))(-1 + p')(1 + 14p' - 26p'^2 + 32p'^3 + 4N^2p'(1 - 2p' + 2p'^2) \\
&\quad - N(1 + 14p' - 28p'^2 + 32p'^3))) - (-1 + 8p - 27p^2 + 24p^3 + N(2 - 24p + 72p^2 - 80p^3) \\
&\quad + N^3(6 - 34p + 66p^2 - 64p^3) + 4N^4(-1 + 4p - 6p^2 + 4p^3) \\
&\quad + N^2(-4 + 34p - 87p^2 + 104p^3))(18p'^4 + N^2(1 + 4p' - 10p'^2 + 4p'^3 + 4p'^4) \\
&\quad - N(1 + 8p' - 18p'^2 + 8p'^3 + 16p'^4)) - (-1 + 16p - 45p^2 + 24p^3 + 4N^4(-1 + 2p)^3 \\
&\quad + N(1 - 44p + 156p^2 - 128p^3) \\
&\quad - 2N^3(-2 + 31p - 84p^2 + 68p^3) + N^2(-1 + 66p - 231p^2 + 208p^3))(-6p'^3(1 + 4p') \\
&\quad + N^2(-3 + 9p' - 34p'^2 + 52p'^3 - 34p'^4) + N^3(1 - 4p' + 12p'^2 - 16p'^3 + 8p'^4) \\
&\quad + N(2 - 3p' + 22p'^2 - 38p'^3 + 48p'^4)).
\end{aligned}$$

Given $p_1 = \frac{\tau_1+2}{2}$ and $p_2 = \frac{\tau_2+2}{2}$,

$$
\begin{aligned}
&f(p_2, p_1, N) - f(p_1, p_1, N) \\
&= (-1+N)N(-1+p_1)(1+14p_1-26p_1^2+32p_1^3+4N^2p_1(1-2p_1+2p_1^2) \\
&\quad - N(1+14p_1-28p_1^2+32p_1^3))(-1+2p_1+36p_1^2-48p_1^3+4N^3(-1+2p_1)^3 \\
&\quad + N^2(3-46p_1+144p_1^2-116p_1^3)+N(-1+26p_1-135p_1^2+132p_1^3)) \\
&\quad + (-1+8p_1-27p_1^2+24p_1^3+N(2-24p_1+72p_1^2-80p_1^3)+N^3(6-34p_1+66p_1^2-64p_1^3) \\
&\quad + 4N^4(-1+4p_1-6p_1^2+4p_1^3) \\
&\quad + N^2(-4+34p_1-87p_1^2+104p_1^3))(18p_1^4+N^2(1+4p_1-10p_1^2+4p_1^3+4p_1^4) \\
&\quad - N(1+8p_1-18p_1^2+8p_1^3+16p_1^4))+(-1+16p_1-45p_1^2+24p_1^3+4N^4(-1+2p_1)^3 \\
&\quad + N(1-44p_1+156p_1^2-128p_1^3)-2N^3(-2+31p_1-84p_1^2+68p_1^3) \\
&\quad + N^2(-1+66p_1-231p_1^2+208p_1^3))(-6p_1^3(1+4p_1)+N^2(-3+9p_1-34p_1^2+52p_1^3-34p_1^4) \\
&\quad + N^3(1-4p_1+12p_1^2-16p_1^3+8p_1^4)+N(2-3p_1+22p_1^2-38p_1^3+48p_1^4)) \\
&\quad - (18p_1^4+N^2(1+4p_1-10p_1^2+4p_1^3+4p_1^4) \\
&\quad - N(1+8p_1-18p_1^2+8p_1^3+16p_1^4))(-1+8p_2-27p_2^2+24p_2^3+N(2-24p_2+72p_2^2-80p_2^3) \\
&\quad + N^3(6-34p_2+66p_2^2-64p_2^3)+4N^4(-1+4p_2-6p_2^2+4p_2^3) \\
&\quad + N^2(-4+34p_2-87p_2^2+104p_2^3))-(-1+N)N(-1+p_1)(1+14p_1-26p_1^2+32p_1^3 \\
&\quad + 4N^2p_1(1-2p_1+2p_1^2)-N(1+14p_1-28p_1^2+32p_1^3))(-1+2p_2+36p_2^2 \\
&\quad - 48p_2^3+4N^3(-1+2p_2)^3 \\
&\quad + N^2(3-46p_2+144p_2^2-116p_2^3)+N(-1+26p_2-135p_2^2+132p_2^3))-(-6p_1^3(1+4p_1) \\
&\quad + N^2(-3+9p_1-34p_1^2+52p_1^3-34p_1^4) \\
&\quad + N^3(1-4p_1+12p_1^2-16p_1^3+8p_1^4)+N(2-3p_1+22p_1^2-38p_1^3+48p_1^4))(-1+16p_2 \\
&\quad - 45p_2^2+24p_2^3+4N^4(-1+2p_2)^3 \\
&\quad + N(1-44p_2+156p_2^2-128p_2^3)-2N^3(-2+31p_2-84p_2^2+68p_2^3) \\
&\quad + N^2(-1+66p_2-231p_2^2+208p_2^3))
\end{aligned}
$$

By analyzing the above equation, we can find that when $N \geq 17$ (i.e., $n \geq 34$), $0.5 < p_1 \leq 0.99$ (i.e., $0 < \tau_1 \leq 0.98$), and $p_1 + 0.01 \leq p_2 \leq 1$ (i.e., $\tau_1 + 0.02 \leq \tau_2 \leq 1$), the above equation is negative, i.e., $f(p_2, p_1, N) - f(p_1, p_1, N) < 0$ or equivalently

$$
\begin{aligned}
\lim_{\gamma \to 0^+} \frac{1}{\gamma}(\mathbb{E}_{\mathcal{G}}\Big[\mathcal{L}(\mathcal{G}, \mathbf{W}^{(0)}) - \mathcal{L}(\mathcal{G}, \mathbb{W}(\tau_1, \mathbf{W}^{(0)}, \gamma)) \mid \tau_1\Big] \\
-\mathbb{E}_{\mathcal{G}}\Big[\mathcal{L}(\mathcal{G}, \mathbf{W}^{(0)}) - \mathcal{L}(\mathcal{G}, \mathbb{W}(\tau_1, \mathbf{W}^{(0)}, \gamma)) \mid \tau_2\Big]) < 0,
\end{aligned}
$$

completing the proof. □

B Datasets

In this section, we describe the details of the benchmark graph classification datasets utilized in our experiments. For experiments, we use datasets that are preprocessed by Morris et al. [36]. We provide descriptive statistics of each dataset in Table 2.

The **DD** dataset [7] is a bioinformatics dataset that contains protein graphs. Each graph represents a particular protein. Each node represents an individual amino acid. Each edge represents two nodes that satisfy certain spatial proximity. Each node feature represents a type of the corresponding node's amino acid type. Each graph binary label represents whether the corresponding compound is an enzyme or not.

The **Protein** dataset [1] is a bioinformatics dataset that contains protein graphs. Each graph represents a particular protein. Each node represents a secondary structure element of a protein. Each edge

Table 2: Descriptive statistics of the utilized graph classification benchmark datasets.

Methods	DD	Protein	NCI1	AIDS	Reddit	IMDB	MUTAG	DHFR	BZR	ER
# of graphs	1,178	1,113	4,110	2,000	2,000	1,000	4,337	756	405	7,697
Avg. $\lvert\mathcal{V}\rvert$	284.32	39.06	29.87	15.69	429.63	19.77	30.32	42.43	35.75	17.58
Avg. $\lvert\mathcal{E}\rvert$	715.66	72.82	32.30	16.20	497.75	96.53	30.77	44.54	38.36	17.94
# of features	89	4	37	42	-	-	14	56	56	50
# of classes	2	2	2	2	2	2	2	2	2	2

represents two nodes that are neighbors along the amino acid sequence or are top-K neighbors (K=3) at the protein space. Each node feature represents a type of the corresponding node's secondary structure element type. Each graph binary label represents whether the corresponding protein is an enzyme or not.

The **NCI1** dataset [47] is a chemical dataset that contains chemical molecular compound graphs. Each graph represents a particular chemical molecular compound. Each node represents an atom. Each edge represents a chemical bond between two atoms. Each node feature represents a type of atom (e.g., carbon, and hydrogen). Each graph binary label represents whether the corresponding compound is an anti-cancer chemical or not.

The **AIDS** dataset [40] is a chemical dataset that contains chemical molecular compound graphs. Each graph represents a particular chemical molecular compound. Each node represents an atom. Each edge represents a chemical bond between two atoms. Each node feature represents a type of atom (e.g., carbon, and hydrogen). Each graph binary label represents whether the corresponding compound shows an anti-HIV activity or not.

The **Reddit** dataset [51] is a social-network dataset that contains social network graphs. Among Reddit-Binary, Reddit-Multi-5K, and Reddit-Multi-12K, we utilized Reddit-Binary. Each graph represents a thread on Reddit. Each node represents a user. Each edge represents a comment between two users. There are no available node features, thus, we utilize the one-hot encoded degree of the node as its feature, following existing literature in GLAD [39]. Each graph binary label represents whether the corresponding thread is a discussion-based thread or question-answer-based thread.

The **IMDB** dataset [51] is an actor-ego-network dataset that contains social network graphs. Each graph represents the ego network of a particular actor. Each node represents an actor. Each edge represents a co-appearance of two actors in (a) movies. There are no available node features, thus, we utilize the one-hot encoded degree of the node as its feature, following existing literature in GLAD [39]. Each graph binary label represents whether the corresponding ego-user particularly acts in the romance genre or action genre.

The **MUTAG** (formally, mutagenicity) dataset [17] is a chemical dataset that contains chemical molecular compound graphs. Each graph represents a particular chemical molecular compound. Each node represents an atom. Each edge represents a chemical bond between two atoms. Each node feature represents a type of atom (e.g., carbon, and hydrogen). Each graph binary label represents whether the corresponding compound is mutagenic or not.

The **DHFR** dataset [42] is a chemical dataset that contains chemical molecular compound graphs. Each graph represents a particular chemical molecular compound. Each node represents an atom. Each edge represents a chemical bond between two atoms. Each node feature represents a type of atom (e.g., carbon, and hydrogen). Each graph binary label represents whether the corresponding compound works as a dihydrofolate reductase inhibitor or not.

The **BZR** [42] dataset is a chemical dataset that contains chemical molecular compound graphs. Each graph represents a particular chemical molecular compound. Each node represents an atom. Each edge represents a chemical bond between two atoms. Each node feature represents a type of atom (e.g., carbon, and hydrogen). Each graph binary label represents whether the corresponding compound is active against the benzodiazepine receptor.

The **ER** dataset is a chemical dataset that contains chemical molecular compound graphs. Specifically, it contains compounds that are NR-estrogen receptor (ER)-LBD. Each graph represents a particular chemical molecular compound. Each node represents an atom. Each edge represents a chemical bond between two atoms. Each node feature represents a type of atom (e.g., carbon, and hydrogen). Each graph binary label represents whether the corresponding compound has toxicity or not.

The source of the ER dataset is https://tripod.nih.gov/tox21/challenge/data.jsp.

C Formal Expression of Synthetic Datasets

In this section, we formally describe the leveraged synthetic graphs: Syn-Com and Syn-Cycle. We elaborate on how each graph is generated.

C.1 Syn-Com

We formalize the data generation process of Syn-Com. For a given even number of nodes $n > 2$, assume two disjoint node communities $\mathcal{V}_a$ and $\mathcal{V}_b$ (WLOG, $\mathcal{V}_a = \{v_i : 1 \leq i \leq n/2\}$ and $\mathcal{V}_b = \{v_j : n/2 + 1 \leq j \leq n\}$). For a given community parameter $\tau \in [0, 1]$, a graph of Syn-Com $\mathcal{G} = (\mathbf{X}, \mathbf{A})$ satisfies the following: $\mathbf{A}_{i,j} = \mathbf{A}_{j,i} \sim b(1, (\tau + 1)/2)$ for $\{v_i, v_j\} \in \binom{\mathcal{V}_a}{2} \cup \binom{\mathcal{V}_b}{2}$, $\mathbf{A}_{i,j} = \mathbf{A}_{j,i} \sim b(1, (1 - \tau)/2)$ for $v_i \in \mathcal{V}_a, v_j \in \mathcal{V}_b$, and $\mathbf{A}_{i,i} = 0, \forall i \in [n]$, where $b(1, p)$ is a Bernoulli sampling with a parameter p. We let $\mathbf{X} = \mathbb{I}_n$, where $\mathbb{I}_n$ is a n-by-n identity matrix.

C.2 Syn-Cycle

We formalize the data generation process of Syn-Cycle. For a given number of nodes $n > 2$, we consider a clean-cycle graph $\mathcal{G}_*^{cyc} = (\mathcal{V}, \mathcal{E}^*) \equiv (\mathbf{X}, \mathbf{A}^*)$ where $\mathcal{E}^* = \{v_i, v_{i+1} : i \in [n]\} \cup \{v_1, v_n\}$ and $\mathbf{A}^*$ is the adjacency matrix that represents $\mathcal{E}^*$.

We also consider a set of noisy-cycle graphs (i.e., pan graphs) $\mathbb{G}_{\prime}^{cyc}$, constructed by randomly relocating an edge from $\mathcal{G}_*^{cyc}$ to form a $(n-1)$-size cycle, (i.e., $(\mathbf{X}, \{v_i, v_{j|i}\} \cup \mathcal{E}^* \setminus \{e_k\}), (\mathbf{X}, \{v_j, v_{i|j}\} \cup \mathcal{E}^* \setminus \{e_k\}) \in \mathbb{G}_{\prime}^{cyc}, \forall e_k = \{v_i, v_j\} \in \mathcal{E}^*$, where $v_{j|i}$ is a v_j's neighbor except for v_i). We let $\mathbf{X} = \mathbb{I}_n$. Note that for n nodes, there exist $2n$ noisy-cycle graphs in $\mathbb{G}_{\prime}^{cyc}$ and a single clean-cycle graph.

D Experimental Details

In this section, we provide experimental details of the conducted experiments in our work.

D.1 Machines

All experiments of this work are performed on a machine with NVIDIA RTX 8000 D6 GPUs (48GB VRAM) and two Intel Xeon Silver 4214R processors.

D.2 Details of empirical analysis

For the Com-Com dataset, we sample 500 graphs from $\mathbb{G}_{\tau=0.4}^{com}$ for training graphs and sample 500 graphs from $\mathbb{G}_{\tau=0.8}^{com}$ for unseen graphs.

For the Cycle-Cycle dataset, we sample half of $\mathbb{G}_{\prime}^{cyc}$ for training graphs and employ the clean-cycle graph as the unseen graph.

For the Com-Cycle dataset, we sample 500 graphs from $\mathbb{G}_{\tau=0.4}^{com}$ for training graphs and sample half of $\mathbb{G}_{\prime}^{cyc}$ for unseen graphs.

For the Cycle-Com dataset, we sample half of $\mathbb{G}_{\prime}^{cyc}$ for training graphs and sample 500 graphs from $\mathbb{G}_{\tau=0.4}^{com}$ for unseen graphs.

Training and evaluation We train GAEs for 200 epochs. The results are similar for 500 epochs, but we present the 200-epoch results for better visualization since the error lines flatten after 200 epochs. For every 10 epochs of training, we measure the mean reconstruction errors for each graph and report the mean of these values for graphs within the same class.

D.3 Details of two-stage GLAD methods

We provide an overview of two-stage anomaly detection methods. First, by performing certain pretext tasks, they learn representations of data. Then, upon the obtained representations, they utilize a one-class classifier to detect anomalies. Below, (1) we describe graph representations of utilized GLAD methods, including MUSE, and (2) describe the utilized one-class classifier, an MLP autoencoder.

Note. MUSE *is a two-stage method.*

Graph representation. As described in Section 5, MUSE obtains a representation of each graph with the proposed error representation Eq (6). On the other hand, utilized baseline two-stage approaches, which are GraphCL [53], GAE [23], and GraphMAE [15], obtain graph representations from the graph or node embeddings. Specifically, they first train graph neural network encoders with their pretext tasks. Specifically, GraphCL performs graph contrastive learning, GAE performs adjacency matrix reconstruction, and GraphMAE performs node feature reconstruction. After finalizing self-supervised learning, we utilize the trained encoder to obtain embeddings of graphs or nodes. Details of each method are as follows:

- GraphCL: It learns graph embeddings to perform its contrastive learning task. Thus, we directly utilize the graph embeddings as representations of graphs.
- GAE: It only learns node embeddings. Thus, we pool node embeddings by using the elementwise-mean readout function.
- GraphMAE: It only learns node embeddings. Thus, we pool node embeddings by using the elementwise-mean readout function.

One-class classifier. In our experiments, every two-stage method leverages an MLP autoencoder as its one-class classifier. In a nutshell, the MLP autoencoder detects anomalies by using the reconstruction loss of a graph. We first elaborate on how we train the MLP autoencoder and then describe how the trained MLP autoencoder is leveraged to detect anomalies.

We start with the training of the MLP autoencoder. Formally, for a given data representation vector $\mathbf{z} \in \mathbb{R}^d$, the MLP autoencoder $\texttt{MLP}_\xi : \mathbb{R}^d \mapsto \mathbb{R}^d$ generates a reconstructed representation $\hat{\mathbf{z}} \in \mathbb{R}^d$ (i.e., $\hat{\mathbf{z}} = \texttt{MLP}_\xi(\mathbf{z})$). Then, we compute a L2-norm reconstruction loss $\mathcal{L}$ as follows: $\mathcal{L} := \|\mathbf{z} - \hat{\mathbf{z}}\|_2$. We update the parameters of the MLP autoencoder ξ by using gradient descent to minimize $\mathcal{L}$.

After training the MLP autoencoder $\texttt{MLP}_\xi$, we detect anomalies by using a reconstruction loss as an anomaly score. However, since the scale of each representation dimension significantly varies, the reconstruction loss is often dominated by dimensions with larger scales, overshadowing those with smaller values. To mitigate this undesirable phenomenon, in the anomaly inference step, we leverage dimensionwise-weighted L2 scores. Importantly, we apply different weights to different dimensions $w_l \in \mathbb{R}, \forall l \in [d]$ (note that d is the dimension of an input vector). Specifically, we utilize the dimensionwise standard deviation of train data's representations, which is defined as follows:

$$w_\ell = \sqrt{\frac{1}{|\mathcal{D}_{train}|} \sum_{\mathcal{G}_k \in \mathcal{D}_{train}} \left(\mathbf{z}_{k,\ell} - \left(\frac{1}{|\mathcal{D}_{train}|} \sum_{\mathcal{G}_t \in \mathcal{D}_{train}} \mathbf{z}_{t,\ell} \right) \right)^2}, \forall \ell \in [d]. \tag{13}$$

where $\mathbf{z}_{i,l}$ is a l-th element of i-th graph's representation vector and $\mathcal{D}_{train}$ is a set of training graphs. By using the normalization weight of Eq (13), we compute the anomaly score. Formally, the anomaly detection score of $\mathcal{G}_t$, denoted by s_T, is defined as follows:

$$s_t := \exp\left(-\sqrt{\sum_{\ell=1}^{d} \left(\frac{\mathbf{z}'_{t,\ell} - \texttt{MLP}_\xi(\mathbf{z}'_t)_\ell}{w_\ell} \right)^2} \right) \in [0,1]. \tag{14}$$

Note that the higher s_t indicates the t-th data point is normal.

D.4 Details of MUSE

In this subsection, we provide details of MUSE.

Reconstruction models. As described in Section 6, we use GIN [49] as our backbone encoder, which is the same as other leveraged GLAD baseline methods. For the node feature decoder and adjacency matrix decoder, we use a 2-layer MLP with a ReLU activation function.

Error representation. For aggregation functions of error representation, we leverage a mean function and a standard deviation function. Formally, for a set $\mathcal{A}$ where $\mathcal{A} \subset \mathbb{R}$, a mean aggregation function $\texttt{Agg}_{\texttt{mean}} : 2^{\mathbb{R}} \mapsto \mathbb{R}$ and a standard deviation aggregation function $\texttt{Agg}_{\texttt{std}} : 2^{\mathbb{R}} \mapsto \mathbb{R}_{\geq 0}$ are defined as follows:

$$\texttt{Agg}_{\texttt{mean}}(\mathcal{A}) := \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} a, \quad \texttt{Agg}_{\texttt{std}}(\mathcal{A}) := \sqrt{\frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} (a - \texttt{Agg}_{\texttt{mean}}(\mathcal{A}))^2}. \tag{15}$$

Thus, we represent each graph $\mathcal{G}$ by using a $\mathbb{R}^4$ vector as follows:

$$\texttt{Err}(\mathcal{G}) = [\texttt{Agg}_{\texttt{mean}}(\mathbb{L}_{\mathbf{X}}(\mathcal{G})),\ \texttt{Agg}_{\texttt{std}}(\mathbb{L}_{\mathbf{X}}(\mathcal{G})),\ \texttt{Agg}_{\texttt{mean}}(\mathbb{L}_{\mathbf{A}}(\mathcal{G})),\ \texttt{Agg}_{\texttt{std}}(\mathbb{L}_{\mathbf{A}}(\mathcal{G}))] \in \mathbb{R}^4. \tag{16}$$

D.5 Hyperparameters and Details of Baselines

For every method, we tune the hyperparameters and choose the hyperparameter configuration that gives the best validation AUROC. We provide details of the search space of each model category and its hyperparameters.

Fixed settings. For all the methods, we fix the dropout probability and weight decay as 0.3 and $1e-6$, respectively. In addition, all the methods are trained with Adam optimizer [21].

One-stage learning-based GLAD methods. We tune the following hyperparameters for one-stage learning-based GLAD methods, which are DOMINANT, OCGTL, GLADC, GLocalKD, GLAM, HIMNET, and SIGNET:

- Training learning rate $\gamma \in \{10^{-3}, 10^{-4}\}$
- Models hidden dimension $d' \in \{16, 32, 64, 128, 256\}$
- Number of GNN layers $K \in \{3, 4, 5\}$
- Number of model training epochs $L \in \{30, 60, \cdots 300\}$

For other hyperparameters, we follow the default configuration provided in their official Github.

Two-stage learning-based GLAD methods. We tune the following hyperparameters for two-stage learning-based GLAD methods, which are GraphCL, GAE, GraphMAE, and MuSE.

Note. Our proposed method MuSE is also included in this category and tuned hyperparameters according to the below

- Pretext task learning rate $\gamma \in \{10^{-3}, 10^{-4}\}$
- Models hidden dimension $d' \in \{16, 32, 64, 128, 256\}$
- Number of GNN layers $K \in \{3, 4, 5\}$
- Number of pretext task training epochs $L \in \{20, 40, \cdots, 200\}$

In addition to these hyperparameters, we also tune the positive weight coefficient for our adjacency matrix reconstruction, which is denoted as τ in Section 5, within $\{0.0, 1.0, 2.0\}$.

Here, models indicate both the encoder model and decoder model that are utilized to perform pretext tasks. Note that as described in Appendix D.3, all two-stage methods utilize MLP autoencoder for their one-class classifier. We tune the hyperparameter of the MLP autoencoder as follows:

- MLP hidden dimension $d' \in \{32, 64, 128\}$
- MLP learning rate $\gamma \in \{10^{-2}.10^{-3}, 10^{-4}\}$

We fix the MLP auteoncoder training epochs and the number of MLP layers as 500 and 3, respectively.

Two-stage baselines details. For GraphCL, we obtain graph views by randomly dropping 50% of edges and adding Gaussian noise to node features sampled from $\mathcal{N}(0, 0.1)$ and use a two-layer MLP projection head. For GraphMAE, we mask 50% of node features with the zero mask, which is empirically demonstrated to be more effective than a learnable mask in our preliminary study. For GAE, we utilize a binary cross-entropy loss and a two-layer MLP decoder. [4]

[4]While the original GAE [22] does not use a projection head, we found that projection head enhances the performance of GAE, through our preliminary study.

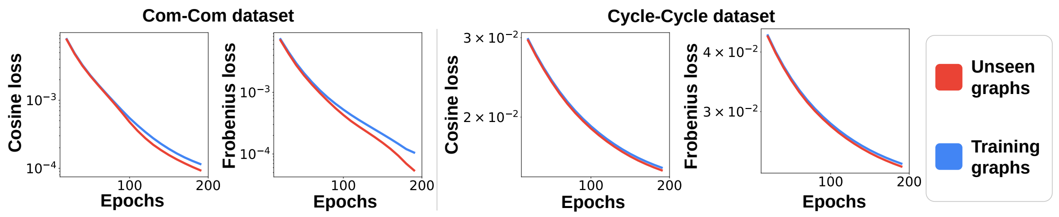


Figure 8: **Reconstruction flip occurs on feature reconstruction method.** When Graph-AEs are trained on graphs sharing a primary pattern of weaker strength, the trained Graph-AEs give smaller reconstruction losses for graphs having the same pattern of a stronger strength (red lines) than those of a weaker strength (blue lines).

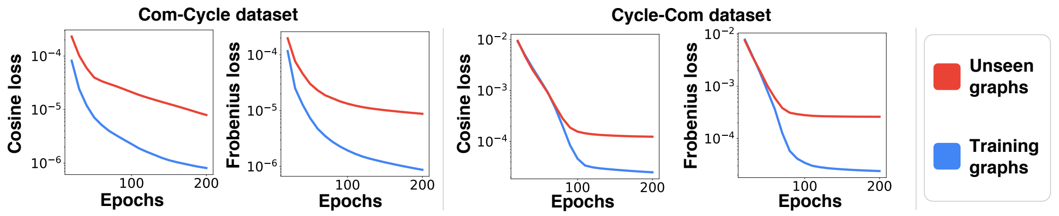


Figure 9: **Reconstruction flip does NOT occur on feature reconstruction method.** When Graph-AEs are trained on graphs sharing a primary pattern, the trained Graph-AEs give larger reconstruction for graphs having a different pattern (red lines) than those of the same pattern (blue lines).

E Additional Experimental Results

E.1 Node feature reconstruction methods

Note that in Section 3, we investigate Graph-AEs that reconstruct the adjacency matrix of a given graph. In this subsection, we analyze another large branch of Graph-AEs: node feature reconstruction methods [15, 16, 44].

Feature reconstruction model. We first formalize the node feature reconstruction method. Consider a graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$, where $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$. First, a GNN encoder f_θ is utilized to generates node embeddings $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}| \times d'}$ (i.e., $f_\theta(\mathbf{X}, \mathbf{A}) = \mathbf{Z}$). Then, by using the feature decoder $g_\phi : \mathbb{R}^{d'} \mapsto \mathbb{R}^{d}$, the reconstructed node features $\hat{\mathbf{X}} \in \mathbb{R}^{|\mathcal{V}| \times d}$ are obtained. Lastly, we compute the feature reconstruction loss, which is either (1) squared Frobenius norm loss [38] (i.e., $\mathcal{L} \coloneqq \|\mathbf{X} - \hat{\mathbf{X}}\|_F^2$) or (2) cosine similarity loss [15] (i.e., $\mathcal{L} \coloneqq \frac{1}{|\mathcal{V}|}\sum_{i=1}^{|\mathcal{V}|}(1 - \frac{\mathbf{X}_i^T \hat{\mathbf{X}}_i}{\|\mathbf{X}_i\|_2 \cdot \|\hat{\mathbf{X}}_i\|_2})$). Parameters of the GNN encoder f_θ and the node feature decoder g_ϕ are updated by gradient descent to minimize $\mathcal{L}$.

Setting. The dataset setting is equal to that of empirical analysis provided in Section 3.2. Regarding the model setting, we utilize 3-layer GIN as the GNN encoder f_θ and 2−layer MLP as the feature decoder g_ϕ. We utilize either squared Frobenius norm or cosine similarity loss for the reconstruction loss. We train each reconstruction model for 200 epochs.

Results. In a nutshell, we can observe the same results as those of GAE. In Scenario 1, as shown in Figure 8, one can observe that the reconstruction flip occurs, which is also observed in GAE. In Scenario 2, as shown in Figure 9, one can observe that the reconstruction flip does not occur. Thus, we verify that our findings in Section 3.2 are not only restricted to the adjacency matrix reconstruction methods but also hold for the node feature reconstruction methods.

E.2 Experimental results on each setting.

We present experimental results in each setting of our main experiment. Specifically, experimental results for the case where class 0 graphs are anomalies and class 1 graphs are normal are provided in Table 3. In addition, experimental results for the case where class 1 graphs are anomalies and class 0 graphs are normal are provided in Table 4. Notably, in both cases, MUSE obtains the best average ranking among 18 methods (refer to Table 3 and Table 4).

Table 3: Average and standard deviation of the test AUROC when graphs belonging to class 0 are anomalies and graphs belonging to class 1 are normal. A.R. denotes the average ranking. MUSE outperforms other GLAD methods in terms of average ranking.

Methods	DD	Protein	NCI1	AIDS	Reddit	IMDB	MUTAG	DHFR	BZR	ER	A.R.
DOMINANT [6]	44.5 (3.0)	43.5 (11.6)	66.1 (6.0)	71.1 (5.3)	34.8 (5.5)	59.9 (3.7)	59.8 (5.7)	49.6 (9.2)	72.3 (10.3)	66.8 (4.4)	12.7
OCGTL [39]	77.9 (2.6)	67.8 (10.6)	64.1 (4.0)	99.1 (1.5)	75.3 (3.5)	69.1 (3.3)	63.1 (6.6)	69.7 (5.9)	75.2 (11.3)	65.8 (4.7)	6.0
GLocalKD [34]	69.6 (7.1)	62.5 (3.3)	45.9 (6.6)	97.5 (0.8)	79.0 (2.9)	68.1 (5.5)	55.8 (2.7)	61.8 (9.6)	62.8 (12.8)	41.0 (5.0)	11.2
GLADC [33]	74.7 (3.6)	60.5 (2.7)	43.0 (4.5)	98.0 (0.4)	83.3 (3.2)	68.6 (3.5)	55.5 (4.0)	62.0 (2.9)	63.5 (13.0)	39.8 (6.3)	11.2
GLAM [57]	46.6 (5.8)	62.6 (3.9)	60.7 (1.9)	94.2 (3.0)	72.2 (4.5)	63.7 (2.4)	60.6 (1.5)	48.3 (6.2)	62.1 (8.2)	66.6 (2.6)	12.6
HIMNET [38]	70.2 (2.7)	66.0 (2.9)	41.1 (5.1)	97.9 (0.5)	72.6 (1.9)	68.8 (3.8)	58.6 (2.4)	60.6 (6.9)	69.5 (7.1)	52.5 (3.4)	10.8
SIGNET [32]	63.7 (9.1)	58.1 (6.0)	62.8 (5.1)	98.1 (2.1)	84.8 (4.8)	54.3 (4.0)	67.1 (1.5)	18.1 (7.1)	74.6 (6.3)	56.5 (3.3)	10.3
GraphCL-1 [53]	49.0 (5.5)	51.3 (3.3)	66.4 (3.9)	44.8 (12.7)	58.8 (5.3)	45.8 (6.7)	53.6 (1.5)	51.4 (8.3)	56.5 (7.9)	63.2 (3.4)	15.4
GAE-1 [22]	67.8 (6.1)	76.7 (4.9)	59.0 (1.8)	99.9 (0.1)	86.3 (2.1)	60.6 (8.2)	55.1 (3.1)	59.0 (11.6)	61.4 (8.8)	60.2 (3.5)	10.3
GraphMAE-1 [15]	50.4 (4.9)	62.5 (3.5)	51.7 (3.7)	90.2 (9.5)	74.0 (3.0)	71.8 (5.9)	62.6 (2.4)	52.3 (9.2)	64.1 (5.2)	65.4 (4.6)	11.3
GraphCL-2 [53]	59.0 (3.3)	51.8 (1.8)	62.3 (4.3)	97.2 (0.9)	77.8 (3.4)	70.0 (4.0)	68.8 (2.7)	57.0 (2.8)	77.6 8.2	66.0 (3.9)	8.5
GAE-2 [22]	58.3 (5.1)	52.9 (3.6)	64.0 (3.1)	99.3 (0.4)	85.4 (4.0)	73.7 (4.2)	68.2 (3.3)	57.3 (4.1)	75.5 (14.7)	64.2 (1.8)	8.9
GraphMAE-2 [15]	61.5 (3.3)	52.4 (1.8)	62.1 (4.3)	98.5 (0.9)	83.0 (3.4)	69.6 (4.0)	67.2 (2.7)	59.3 (2.1)	77.1 (14.2)	64.3 (1.2)	8.3
MUSE w/o $\mathbb{L}_{\mathbf{X}}$	80.4 (2.2)	77.6 (1.7)	70.1 (0.8)	100.0 (0.0)	83.6 (2.5)	65.9 (7.1)	66.0 (3.3)	62.4 (8.4)	67.3 (15.2)	65.1 (3.4)	5.4
MUSE w/o $\mathbb{L}_{\mathbf{A}}$	56.9 (6.9)	61.3 (5.8)	63.2 (1.2)	86.4 (3.5)	73.4 (4.6)	47.9 (1.4)	56.3 (4.6)	53.0 (8.5)	65.8 (12.9)	63.7 (3.5)	12.9
MUSE w/o `AVG`	77.9 (2.8)	70.4 (4.9)	70.1 (0.7)	92.1 (2.8)	82.5 (4.2)	67.4 (6.4)	63.2 (3.4)	60.6 (7.6)	59.6 (11.5)	65.3 (3.7)	8.4
MUSE w/o `STD`	74.8 (4.0)	77.8 (1.8)	62.9 (2.9)	100.0 (0.1)	84.0 (4.0)	72.1 (3.8)	64.5 (1.8)	64.1 (6.0)	64.3 (10.2)	66.2 (2.6)	4.9
MUSE	81.0 (2.0)	78.7 (2.2)	72.5 (0.3)	100.0 (0.0)	84.7 (4.4)	69.6 (3.7)	68.1 (3.0)	64.3 (8.0)	68.2 (11.3)	67.6 (4.0)	2.7

Table 4: Average and standard deviation of the test AUROC when graphs belonging to class 1 are anomalies and graphs belonging to class 0 are normal. A.R. denotes average ranking. MUSE outperforms other GLAD methods in terms of average ranking.

Methods	DD	Protein	NCI1	AIDS	Reddit	IMDB	MUTAG	DHFR	BZR	ER	A.R.
DOMINANT [6]	84.0 (5.8)	68.3 (7.7)	64.8 (6.1)	90.1 (2.6)	82.3 (5.1)	61.6 (9.6)	70.2 (2.6)	63.6 (9.2)	80.0 (5.3)	50.5 (6.5)	6.7
OCGTL [39]	71.0 (7.6)	74.2 (6.9)	58.2 (7.0)	91.4 (5.9)	62.6 (4.4)	62.2 (8.2)	66.6 (3.1)	63.2 (14.0)	67.3 (22.9)	60.2 (2.5)	9.5
GLocalKD [34]	25.9 (5.6)	38.8 (13.6)	57.2 (4.5)	4.9 (1.6)	20.5 (5.5)	48.9 (7.9)	54.3 (6.1)	46.3 (6.6)	48.8 (20.6)	67.8 (3.8)	15.8
GLADC [33]	29.5 (6.7)	40.9 (8.5)	59.7 (2.7)	4.7 (1.6)	21.1 (1.9)	46.7 (6.9)	51.0 (4.9)	49.6 (5.3)	54.4 (15.9)	65.7 (2.1)	15.8
GLAM [57]	74.0 (5.3)	60.6 (6.4)	55.4 (1.9)	93.0 (2.1)	78.9 (3.5)	66.5 (4.5)	65.3 (2.4)	66.1 (5.4)	83.0 (9.5)	43.7 (3.1)	8.1
HIMNET [38]	33.9 (4.6)	47.8 (8.6)	66.1 (4.0)	30.6 (5.9)	58.8 (2.9)	54.8 (4.8)	56.4 (3.3)	66.6 (6.4)	74.4 (12.7)	58.9 (2.1)	11.5
SIGNET [32]	64.7 (9.5)	54.6 (6.8)	63.4 (2.8)	96.2 (1.0)	71.2 (3.9)	42.1 (5.6)	67.8 (1.7)	62.2 (4.5)	58.5 (12.7)	55.8 (5.3)	11.1
GraphCL-1 [53]	79.9 (2.2)	70.0 (5.0)	45.2 (2.3)	97.6 (0.5)	56.6 (5.7)	62.5 (5.7)	53.6 (3.1)	64.1 (4.8)	64.4 (10.7)	47.7 (4.7)	10.7
GAE-1 [22]	61.6 (4.3)	45.8 (9.1)	66.0 (2.6)	72.2 (2.7)	63.2 (4.3)	66.9 (6.6)	71.3 (3.5)	54.0 (7.6)	75.5 (18.6)	59.8 (4.2)	9.8
GraphMAE-1 [15]	63.0 (9.6)	58.5 (6.3)	55.0 (2.6)	93.3 (1.0)	71.3 (3.3)	62.1 (4.0)	62.5 (2.8)	72.1 (9.9)	77.0 (10.0)	39.0 (2.6)	10.2
GraphCL-2 [53]	73.2 (2.7)	66.3 (8.5)	58.3 (4.5)	86.3 (6.0)	76.8 (4.7)	62.5 (7.1)	66.0 (3.8)	61.2 (6.3)	66.1 (12.5)	68.5 (2.8)	9.3
GAE-2 [22]	76.0 (1.6)	71.7 (6.4)	60.8 (4.6)	89.0 (5.8)	65.2 (7.3)	59.5 (11.0)	66.3 (3.2)	64.3 (7.0)	68.5 (2.9)	67.2 (2.1)	8.3
GraphMAE-2 [15]	74.4 (5.3)	70.0 (6.1)	74.4 (2.8)	83.0 (6.3)	68.5 (6.1)	63.8 (7.6)	69.0 (2.1)	63.5 (9.8)	68.5 (6.4)	68.1 (3.0)	6.7
MUSE w/o $\mathbb{L}_{\mathbf{X}}$	78.4 (5.2)	73.5 (5.7)	68.3 (3.8)	99.3 (0.9)	60.8 (5.4)	65.6 (4.3)	65.5 (2.8)	58.3 (4.8)	63.9 (23.6)	67.4 (3.7)	7.5
MUSE w/o $\mathbb{L}_{\mathbf{A}}$	66.6 (8.3)	68.1 (8.3)	63.0 (5.3)	92.1 (2.1)	69.9 (4.9)	65.9 (12.8)	57.7 (2.3)	63.1 (8.5)	71.5 (15.4)	57.7 (4.5)	9.7
MUSE w/o `AVG`	79.3 (5.1)	65.7 (6.0)	65.8 (3.3)	97.8 (2.3)	63.9 (8.9)	64.9 (6.6)	58.6 (4.4)	59.6 (12.5)	73.0 (14.4)	58.6 (3.3)	8.8
MUSE w/o `STD`	73.7 (6.8)	71.0 (8.6)	67.4 (4.2)	97.3 (0.9)	57.0 (4.5)	69.3 (3.5)	59.5 (3.0)	61.7 (6.7)	78.2 (12.7)	67.1 (2.2)	7.3
MUSE	79.9 (3.5)	78.0 (2.1)	69.6 (3.7)	99.3 (1.0)	72.1 (6.9)	68.7 (3.3)	66.9 (3.7)	63.2 (9.2)	70.8 (13.9)	68.2 (3.2)	3.6

E.3 Experimental results on additional evaluation metrics

In this section, we compare the GLAD performance of MUSE against those of two strongest competitors, OCGTL and GLAM. To this end, we leverage two metrics: average precision score (AP score) and precision@K. As shown in Table 5 and Table 6, MUSE outperforms the competitors in 7/10 and 8/10 in terms of AP score and precision@K, respectively. These results demonstrate the effectiveness of MUSE is not limited to a particular evaluation metric.

E.4 Experimental results on additional ablation study

In this section, we demonstrate the effectiveness of the following two components of MUSE: graph augmentation and cosine-similarity feature reconstruction loss. To this end, we use two variants of MUSE. **MUSE w/o Aug.** is a variant where the input graph augmentation is removed. **MUSE w/o Cos.** is a variant that uses the Frobenious norm loss for the feature reconstruction loss, instead of the

Table 5: **GLAD performance**: Average and standard deviation of test **AP score** values ($\times$100) in the GLAD task are reported. The best performances are highlighted in green.

Method	DD	Protein	NCI1	AIDS	Reddit	IMDB	MUTAG	DHFR	BZR	ER
OCGTL [39]	84.9 (3.1)	78.1 (1.8)	73.6 (1.6)	95.3 (3.0)	88.0 (1.8)	81.0 (2.2)	72.7 (1.9)	87.6 (4.1)	88.6 (0.9)	60.3 (1.1)
GLAM [57]	74.7 (1.7)	71.1 (2.0)	73.6 (9.0)	95.4 (2.4)	83.2 (1.6)	78.7 (3.0)	75.9 (1.5)	75.4 (3.2)	76.9 (1.2)	65.0 (0.6)
MUSE	88.1 (1.3)	86.5 (1.3)	81.8 (1.2)	99.7 (0.6)	81.7 (2.0)	81.7 (2.7)	79.6 (1.4)	78.9 (2.8)	79.1 (2.1)	70.1 (0.4)

Table 6: **GLAD performance**: Average and standard deviation of test **Precision@10 score** values ($\times$10) in the GLAD task are reported. The best performances are highlighted in green.

Method	DD	Protein	NCI1	AIDS	Reddit	IMDB	MUTAG	DHFR	BZR	ER
OCGTL [39]	5.2 (0.7)	3.5 (0.7)	4.4 (0.8)	9.8 (0.2)	3.5 (0.6)	3.9 (0.6)	5.1 (1.2)	3.4 (0.9)	4.0 (0.6)	5.1 (0.4)
GLAM [57]	5.1 (0.9)	6.0 (0.7)	5.1 (1.0)	9.9 (0.4)	8.4 (0.7)	5.9 (0.8)	4.9 (0.8)	4.1 (0.6)	5.5 (0.6)	4.7 (0.5)
MUSE	7.7 (1.3)	7.9 (1.3)	7.8 (1.3)	10.0 (0.0)	5.9 (1.7)	6.6 (1.6)	8.5 (1.1)	5.2 (1.1)	4.8 (1.3)	5.3 (0.9)

cosine similarity loss. Note that MUSE uses both input graph augmentation and cosine similarity loss. As shown in Table 7, MUSE outperforms its variants in 8 out of 10 datasets, demonstrating the effectiveness of our graph augmentation and cosine-similarity loss.

E.5 Experimental results from the scalability analysis

In this section, we study the scalability and effectiveness of GLAD methods in large-scale graphs.

Datasets. We use two large-scale real-world graphs, which are MalNetTiny [9] and OVCAR-8 [50]. Specifically, the MalNetTiny dataset consists of graphs, each containing a large number of nodes and edges. In contrast, the OVCAR-8 dataset consists of a large number of graphs.

Scalable MUSE. Note that MUSE may not be scalable for graphs with a large number of nodes, since MUSE reconstructs all the entries of an adjacency matrix, which results in the complexity of $O(|\mathcal{V}|^2)$. Thus, we present a scalable version of MUSE, which we call **MUSE-Sample**. Specifically, MUSE-Sample samples K number of entries from the adjacency matrix, and reconstructs only the sampled entries. Therefore, the time complexity of computing reconstruction loss becomes

$$O(|\mathcal{V}|K) \equiv O(|\mathcal{V}|), \because K \text{ is a constant, which is the same across all graphs.} \tag{17}$$

Experimental results. We compare the (1) GLAD performance and (2) inference runtime of our proposed methods (i.e., MUSE and MUSE-Sample) against the two strongest baseline methods: OCGTL and GLAM. Regarding the performance comparison, as shown in Table 8, MUSE and MUSE-Sample outperform the baseline methods in both datasets. Regarding the runtime analysis, as shown in Table 9, MUSE is the second-fastest method among the four methods.

F Further Analysis and Discussion

In this section, we provide our further analyses and discussions.

F.1 Reconstruction flip in computer vision

Analogous to our observations of the reconstruction flips in graphs, certain images are more easily reconstructed. For instance, Liu et al. [31] noted that "*anomalies with colors close to the background may yield unreliable reconstruction errors*". Moreover, in the MNIST dataset, a reconstruction model trained on class-7 images sometimes reconstructs class-4 images better than class-7 images [31].

To mitigate this undesirable phenomenon, Liu et al. [31] proposed a deformation-based anomaly detection method. In this method, an input image is first deformed in a way that makes a reconstruction model easy to reconstruct. Here, a deformation method that maintains the semantics of the input image is employed [5]. To identify whether an image is an anomaly, they first deform the image and then let a reconstruction model reconstruct it. Their underlying intuition is that the deformation model would drastically change the image dissimilar to the majority of the training images, and this would

Table 7: **Additional ablation study**: Average and standard deviation of test AUROC values ($\times 10$) in the GLAD task are reported. The best performances are highlighted in green.

Method	DD	Protein	NCI1	AIDS	Reddit	IMDB	MUTAG	DHFR	BZR	ER
MUSE w/o Aug.	78.9 (3.0)	77.1 (3.2)	68.8 (2.3)	99.6 (0.1)	74.2 (3.3)	69.1 (3.6)	66.0 (2.3)	64.4 (5.4)	66.4 (12.9)	69.9 (3.0)
MUSE w/o Cos.	79.5 (2.4)	72.8 (3.4)	68.4 (3.7)	96.5 (2.8)	72.7 (6.7)	63.3 (3.8)	65.1 (3.1)	60.1 (4.1)	59.9 (9.7)	68.3 (2.5)
MUSE	80.5 (2.3)	78.4 (2.2)	71.1 (2.0)	99.7 (0.5)	78.4 (5.7)	69.2 (3.5)	67.5 (3.4)	63.8 (8.6)	67.5 (12.6)	67.9 (3.6)

Table 8: **Large-scale graphs GLAD performance**: The average and standard deviation of test **AUROC** values ($\times 10$) in the GLAD task on large-scale graphs are reported. The best performances are highlighted in green.

Method	OCGTL [39]	GLAM [57]	MUSE-Sample	MUSE
MalNetTiny	60.5 (4.3)	56.8 (4.3)	66.0 (2.5)	68.1 (2.7)
OVCAR-8	69.8 (7.7)	66.0 (3.5)	72.2 (8.6)	72.8 (8.2)

make the deformed anomalous image different from its original image. Finally, when the deformed image is reconstructed by the reconstruction model, the reconstructed anomalous image would be significantly different from the original anomalous image, resulting in a large reconstruction error.

While the method is intuitive in computer vision, the extension of the suggested method to the graph domain is non-trivial. First, the graph deformation that maintains semantics is non-trivial. It is hard to know the semantics of a graph without sufficient domain knowledge, especially without any external graph labels. Furthermore, naive graph augmentations, such as node dropping and edge perturbation, may harm the semantics of a graph [48]. Second, while the deformation techniques are widely studied in computer vision [5, 2], those of a graph is an underexplored region. Therefore, employing an adequate technique for graph deformation is also practically infeasible.

Considering these challenges, we consider that the method proposed in Liu et al. [31] is hard to be trivially extended to the graph domain.

F.2 Reconstruction flip in real-world graph datasets

We provide our in-depth analysis result of reconstruction flip in Section 3 by using the *synthetic datasets*. In this section, we analyze whether the reconstruction flip phenomena occur in *real-world graph datasets*.

Setting. We focus on the anomaly detection task performance of GAE [22], which reconstructs the adjacency matrix of a given graph. GAE uses reconstruction loss as an anomaly score, where an anomalous graph is likely to have high reconstruction errors. We measure the AUROC score on test graphs, based on the setting described in Section 6.1. Here, the AUROC score lower than 0.5 indicates the corresponding model performs worse than random guessing, implying the reconstruction flip has occurred.

Results. As shown in Table 3, in the protein dataset, GAE [22] tends to perform worse than random guessing (spec., average AUROC is 0.458), which indicates that the reconstruction flip occurs. Thus, we demonstrate that the reconstruction flip is not limited to synthetic scenarios but also occurs in real-world graph datasets.

F.3 Complexity analysis of MUSE

In this section, we provide a time complexity analysis for the forward pass of MUSE, concerning the size of a graph (i.e., number of nodes $|\mathcal{V}|$ and edges $|\mathcal{E}|$).

Encoding. We first encode a graph $\mathcal{G} = (\mathbf{X} \in \mathbb{R}^{|\mathcal{V}|\times d}, \mathbf{A} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|})$ with a GNN encoder. The time complexity of encoding $\mathcal{G}$ with $L-$layer GCN [23] is $O(L|\mathcal{V}|d^2 + L|\mathcal{E}|d)$ [3]. Given that d and L are constants, the complexity is equivalent to $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$. [5]

[5]Note that the forward pass time complexity of GIN [49] is equivalent to that of GCN concerning graph size. This is because the difference between GCN and GIN lies in the feature transformation, which is independent of the graph size.

Table 9: **Runtime analysis:** Average runtime (secs) per each graph of each method on each dataset. Overall, MUSE is the second-fasted method among the four methods.

Method	DD	Protein	NCI1	AIDS	Reddit	IMDB	MUTAG	DHFR	BZR	ER	MalNetTiny	OVCAR-8
GLAM [57]	0.003	0.002	0.002	0.002	0.005	0.002	0.002	0.003	0.002	0.002	0.01	0.003
OCGTL [39]	0.005	0.004	0.004	0.004	0.007	0.004	0.004	0.004	0.004	0.004	0.01	0.006
MUSE-Sample	0.006	0.004	0.004	0.004	0.012	0.004	0.004	0.004	0.004	0.004	0.05	0.004
MUSE	0.005	0.003	0.003	0.003	0.007	0.003	0.003	0.004	0.003	0.003	0.03	0.004

Decoding. We decode node embeddings by using a node decoder and edge decoder, which are MLPs. Since the hidden dimensions and the number of layers of MLPs are all constants, the complexity of decoding is equivalent to $O(|\mathcal{V}|)$.

Reconstruction loss. We compute adjacency matrix reconstruction loss and node feature reconstruction loss. Each loss computation has the time complexity of $O(|\mathcal{V}|^2)$ and $O(|\mathcal{V}|)$, respectively. Thus, the overall time complexity of the reconstruction loss computation becomes $O(|\mathcal{V}|^2)$.

Error representation. We utilize average pooling and standard deviation pooling, which both have linear time complexity. Thus, the time complexity is equivalent to $O(|\mathcal{V}|^2)$.

Overall. Thus, the time complexity of the forward pass of MUSE is as follows:

$$O(|\mathcal{V}| + |\mathcal{E}|) + O(|\mathcal{V}|) + O(|\mathcal{V}|^2) + O(|\mathcal{V}|^2) = O(|\mathcal{V}|^2). \tag{18}$$

While MUSE has a quadratic complexity to the number of nodes, our task is a graph-level task where real-world graphs typically have an affordable number of nodes (as shown in Table 2, the dataset that has the largest average number of nodes has smaller than 430 average number of nodes).

F.4 Broader impacts of our work

In this section, we discuss the broader impacts of our research. While our focus lies in the graph-level task, similar observations can be found in node-level tasks within various types of graphs, such as dynamic graphs [27], heterophilic graphs [29, 30], and hypergraphs [19, 20, 4, 18]. We anticipate that our findings and the proposed MUSE will be widely utilized in applications requiring graph-level anomaly detection, such as drug discovery and brain diagnosis. In addition, as described in Section 2.1, the phenomenon of the reconstruction flip is not limited to the graph domain, and it is also observed in other domains such as computer vision. To our knowledge, we are the first to utilize summarized reconstruction errors as a representation feature of data. This approach can be directly applied to other fields, such as computer vision.

F.5 Generalized categorization of primary patterns

Expectation. In this work, as a primary pattern, we use community structure and node cycles. As a future direction, more general and formal categorization of graph patterns can be considered. This categorization would enable our analysis (Section 3) to be more systematic and generalized.

Challenge. However, real-world datasets may exhibit a wider variety of patterns, including those related to node attributes. For example, in the protein dataset, both normal and anomalous graphs exhibit homophilic patterns (i.e., edges tend to join nodes with the same attribute), with anomalies displaying a stronger pattern than normal graphs. This diversity makes the categorization not straightforward. Thus, adequately considering real-world graph patterns would play a crucial role in this categorization.

G Checklist

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We clarify our goal, analyzing reconstruction flip, and provide analysis results in Section 3. We present a method (Section 5) and demonstrate its superiority (Section 6).

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We provide limitations of the proposed method MUSE in Section 7.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: We provide the full statements for all theorems in Section 3.3 and provide full proofs in Appendix A.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide architectures of MUSE in Section 5, details regarding hyperparameters in Appendix D, and model code in `https://github.com/kswoo97/GLAD_MUSE`.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide all the utilized datasets, full code for models, and instructions to run the code in `https://github.com/kswoo97/GLAD_MUSE`.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide the key setting in Section 6, and further details in Appendix D.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We describe the number of experimental runs and standard deviation of the performance in Section 6.

8. **Experiments Compute Resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We describe machines utilized for the experiments in Appendix D.1.

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We strictly follow the NeurIPS Code of Ethics, particularly in terms of anonymity that we do not reveal our nationality and institutes, etc.

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the broader impact of our work in Appendix F.4

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We believe we do not have risks of misusing.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly add citations for all the mentioned existing literature or methods.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We do not release any new assets.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: We do not utilize any crowdsourcing or human-related tasks.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA] .

    Justification: We do not utilize any crowdsourcing or human-related tasks.