Data Augmentation for Supervised Graph Outlier Detection via Latent Diffusion Models

Kay Liu[†] **Hengrui Zhang**[†] **Ziqing Hu**[§] **Fangxin Wang**[†] **Philip S. Yu**[†]
[†]University of Illinois Chicago [§]University of Notre Dame
{zliu234, hzhan55, fwang51, psyu}@uic.edu zhu4@nd.edu

Abstract

A fundamental challenge confronting supervised graph outlier detection algorithms is the prevalent problem of *class imbalance*, where the scarcity of outlier instances compared to normal instances often results in suboptimal performance. Recently, generative models, especially diffusion models, have demonstrated their efficacy in synthesizing high-fidelity images. Despite their extraordinary generation quality, their potential in data augmentation for supervised graph outlier detection remains largely underexplored. To bridge this gap, we introduce GODM, a novel data augmentation for mitigating class imbalance in supervised **G**raph **O**utlier detection via latent **D**iffusion **M**odels. Extensive experiments conducted on multiple datasets substantiate the effectiveness and efficiency of GODM. The case study further demonstrated the generation quality of our synthetic data. To foster accessibility and reproducibility, we encapsulate GODM into a plug-and-play package and release it at PyPI: https://pypi.org/project/godm/.

1 Introduction

Graph outlier detection has emerged as a popular and important area of research and practice in graph machine learning [1–3]. Graph outlier detection focuses on detecting outliers within graph-structured data that significantly deviate from standard patterns. It has been proven valuable in domains such as fraud detection [4], fake news detection [5], spam detection [6], anti-money laundering [7], etc. Despite the advancements in graph outlier detection techniques, similar to supervised outlier detection on other data modalities, graph outlier detection suffers a fundamental challenge known as class imbalance. This challenge manifests as a significantly lower number of positive instances (outliers) compared to negative instances (inliers). For example, the ratio of positive to negative is only 1:85 on DGraph dataset [4], reflecting the extreme ratio in real-world financial fraud detection scenarios. This class imbalance problem poses challenges in the training of graph outlier detectors and often results in suboptimal performance. Specifically, as the negative instances dominate the training data, the loss function can be biased towards the majority of the negative class and, hence, exhibit poor generalization capability in identifying true outliers.

Common practices mitigate this imbalance by upsampling or downsampling [8]. Upsampling augments the minority positive class by replicating outliers, whereas downsampling reduces the size of the majority negative class by randomly dropping normal instances. However, these methods often have their own risks, such as overfitting outliers in upsampling or the loss of valuable training data through downsampling. Another common approach to alleviate class imbalance is through instance reweighting in the loss function, assigning greater weights to outliers and less weights to

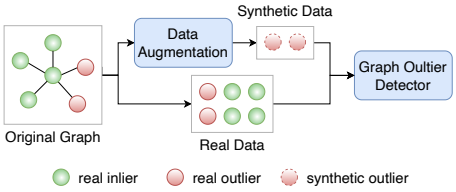

Figure 1: An toy example of data augmentation for class imbalance in graph outlier detection.

K. Liu et al., Data Augmentation for Supervised Graph Outlier Detection via Latent Diffusion Models. *Proceedings of the Third Learning on Graphs Conference (LoG 2024)*, PMLR 269, Virtual Event, November 26–29, 2024.

inliers. This approach is mathematically equivalent to upsampling and downsampling, thereby having a similar problem. For a more comprehensive summary of previous methods, please refer to Appendix A. These challenges underscore the need for more sophisticated data augmentation methods that can generate some synthetic data to balance the class distribution in the training data, thereby improving the performance of graph outlier detection, as Figure 1 shows.

In recent years, generative models, particularly diffusion models, have achieved significant advancements in synthesizing high-fidelity image data [9]. Diffusion models capture intricate data distributions and generate high-quality samples by gradually denoising the samples from a simple prior distribution (e.g., Gaussian distribution). Diffusion models provide more generative power compared to variational autoencoders and offer more stable training than generative adversarial networks. On graph data, although some works have explored the possibility of diffusion on molecular graphs [10, 11], few studies have been conducted to apply diffusion models to tackle the class imbalance problem in the task of graph outlier detection. However, existing diffusion models can hardly be directly adapted to large-scale graph outlier detection applications due to the following non-trivial challenges: (1) **Heterogeneity**: Unlike relatively simple molecular graphs, the information contained in graphs for outlier detection can be highly heterogeneous, including high dimensional features and more than one type of edge, even temporal information. Existing diffusion models are primarily designed for monotypic information, exemplified by the RGB channels in image data. (2) **Efficiency**: In the application of graph outlier detection, e.g., financial fraud detection, the graphs are typically much larger than molecular graphs, up to millions, even billions scale. The prohibitive computational cost of diffusion models hinders their direct application in outlier detection on large graphs. (3) **Condition**: As we only want to generate outliers rather than normal nodes in synthetic graphs, conditional generation is required to mitigate the problem of class imbalance.

To bridge this gap, we apply diffusion models in graph outlier detection and propose a data augmentation for mitigating class imbalance in supervised **G**raph **O**utlier detection via latent **D**iffusion **M**odels (GODM). Our main idea is to generate outliers in graph space while conducting diffusion in latent space. To address heterogeneity, we propose *Variational Encoder* to map the heterogeneous information inherent within the graph data into a unified latent space. In addition, *Graph Generator* synthesizes different types of information back to graph space from the latent embedding. To alleviate the efficiency problem, instead of direct diffusion in graph space, we only conduct diffusion in the latent space crafted by the variational encoder. Furthermore, we use negative sampling and graph clustering to reduce the computational cost. For diffusion models, we also adopt EDM [12] instead of commonly used DDPM [13] to facilitate the generation efficiency of *Latent Diffusion Model*. For the condition, we not only give a class label to the variational encoder to form node embedding with class information but also conduct conditional generation on both Latent Diffusion Model and Graph Generator. Finally, our heterogeneous, efficient, and conditional GODM can generate graphs with outliers that are integrated with the original real graph for the training of the downstream graph outlier detector. Importantly, GODM is model agnostic, providing researchers and practitioners with the flexibility to integrate it across various graph outlier detectors. To foster accessibility, we make our code a plug-and-play package, which can be easily adopted on PyG `Data` object.

2 Preliminary

In this section, we establish the notation adopted in the subsequent sections and rigorously formulate the problem of data augmentation for addressing class imbalance in supervised graph outlier detection.

2.1 Notation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{y}, \mathbf{t}, \mathbf{p})$ denotes a graph with n nodes, where $\mathcal{V} = \{v_i\}_{i=1}^n$ is the set of nodes, and $\mathcal{E} = \{e_{ij}\}$ represents the set of edges. Here, $e_{ij} = (v_i, v_j)$ denotes an edge between node v_i and node v_j. The matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ contains d-dimensional feature vectors $\mathbf{x}_i$ for each node v_i. $\mathbf{y} \in \{0, 1\}^n$ represents the vector of the node label y_i for each node v_i, where 0 denotes an inlier, while 1 denotes an outlier. $\mathbf{t} = \{t_{ij}\} \in \mathbb{N}$ and $\mathbf{p} = \{p_{ij}\} \in \{1, \ldots, P\}$ are the optional non-negative integer edge timestamp and edge type, respectively, where P is the number of edge types.

2.2 Problem Formulation

In this paper, we focus on the task of graph outlier detection, which is formally defined as:

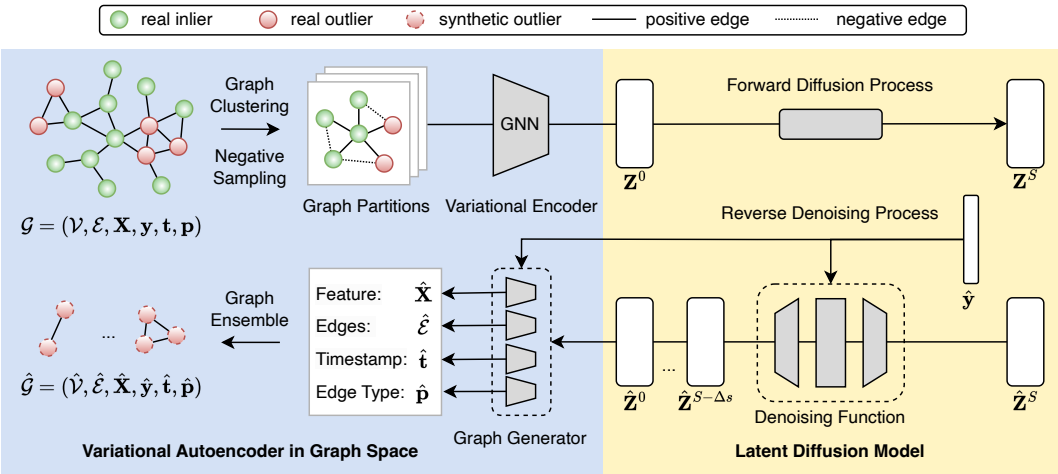

Figure 2: The architecture of proposed data augmentation method GODM.

Definition 1 (Graph Outlier Detection) *In this paper, we focus on node level outlier detection. Given a graph $\mathcal{G}$, graph outlier detection can be regarded as a binary classification task that learns a detector $D : v_i \rightarrow \{0, 1\}$ that classifies every node in $\mathcal{G}$ to an inlier (0) or an outlier (1).*

In the task of graph outlier detection, we aim to mitigate class imbalance by data augmentation:

Definition 2 (Data Augmentation for Class Imbalance) *In outlier detection, the number of inliers is far more than the number of outliers, i.e., $|\{v_i \mid y_i = 0\}| \gg |\{v_i \mid y_i = 1\}|$. We aim to learn a parameterized data augmentation model to generate realistic and diverse synthetic graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}}, \hat{\mathbf{X}}, \hat{\mathbf{y}}, \hat{\mathbf{t}}, \hat{\mathbf{p}})$, where $\hat{y}_i = 1, \forall v_i \in \hat{\mathcal{V}}$. The synthetic graph $\hat{\mathcal{G}}$ is integrated with the original graph $\mathcal{G}$ to alleviate the class imbalance in the training of the graph outlier detector D.*

3 Methodology

In this section, we elaborate on the proposed data augmentation method GODM.

3.1 Overview

Figure 2 shows the architecture of GODM. It starts by partitioning the input graph $\mathcal{G}$ with graph clustering algorithms and conducting negative sampling on edges to reduce memory consumption. Subsequently, each partitioned subgraph is encoded into a latent space representation denoted by $\mathbf{Z}^0$ through Variational Encoder based on graph neural networks (GNNs), followed by a forward diffusion $\mathbf{Z}^0 \rightarrow \mathbf{Z}^S$ in the latent space, where S is the maximum step in the diffusion model. The reverse denoising process $\hat{\mathbf{Z}}^S \rightarrow \hat{\mathbf{Z}}^0$ iteratively estimates the noise and generates the latent embedding $\hat{\mathbf{Z}}^0$ from a predefined prior distribution $p(\hat{\mathbf{Z}}^S)$ using a denoising function, which is a multi-layer perceptron (MLP), conditioned on class labels $\hat{\mathbf{y}}$. Graph Generator employs the latent representations $\hat{\mathbf{Z}}^0$ to conditionally reconstruct the node features and graph structures alongside other information available in the original real graph. By ensembling generated information, we are able to obtain $\hat{\mathcal{G}}$ that contains the nodes that are statistically similar to the real outliers. This synthetic graph $\hat{\mathcal{G}}$, in concert with the original graph $\mathcal{G}$, is leveraged to the training of the downstream graph outlier detector D.

3.2 Variational Encoder

Graph data is inherently complex and heterogeneous, containing both node feature $\mathbf{X}$ and graph structure $\mathcal{E}$, sometimes also partial node labels $\mathbf{y}$, temporal information $\mathbf{t}$, and edge types $\mathbf{p}$. However, current diffusion models are mainly designed for monotypic information (e.g., the magnitude of RGB channels in images). To bridge this gap, we adopt a GNN-based Variational Encoder $E : \mathcal{G} \rightarrow \mathbf{Z}$ to map different types of information in the graph space into a unified latent space.

Node Feature. To encode the node feature into the latent space, we take the feature of each node as the initial embedding for the input of the encoder, i.e., $\mathbf{h}_i^0 = \mathbf{x}_i$.

Class Label. As we only generate outliers by conditional generation, we encode node labels $\mathbf{y} = \{y_i\}$ into the initial node embedding by: $\mathbf{h}_i^0 = \mathbf{x}_i + \mathbf{w}_E^C \cdot y_i$, where $\mathbf{w}_E^C \in \mathbb{R}^d$ is a linear transformation.

Graph Structure. Graph neural networks (GNNs) have emerged as a profound architecture for learning graph data. GNNs efficaciously learn the node representations by encoding the graph topology and node feature simultaneously with message passing. In this paper, we take GraphSAGE [14] as an example. For each layer of GraphSAGE, each node updates its embedding by aggregating the message within the neighborhood:

$$\mathbf{h}_i^l = \text{ACT}(\mathbf{W}^l \cdot \text{CAT}(\mathbf{h}_i^{l-1}, \text{AGG}(\{\mathbf{m}_{ij}^l, \forall e_{ij} \in \mathcal{E}\}), \tag{1}$$

where $\mathbf{h}_i^l$ is the node embedding for node v_i in the $l-$th layer, $\mathbf{m}_{ij}^l$ is the edge-wise message of edge e_{ij}, and ACT is a non-linear activation function. $\mathbf{W}^l \in \mathbb{R}^{d^l \times 2d^{l-1}}$ is the linear transformation, where d^l and d^{l-1} are the hidden dimension of l-th layer and $(l-1)$-th layer, respectively. While CAT represents concatenation, AGG denotes the aggregation function (e.g., mean). In vanilla GraphSAGE, an edge-wise message is the embedding of source node v_j from the last layer $\mathbf{m}_{ij}^l = \mathbf{h}_j^{l-1}$.

Edge Type. However, sometimes, we have different types of edges in graphs, known as heterogeneous graphs or heterogeneous information networks [15, 16]. As different edge types can encapsulate various semantics, encoding edge types can be important for the downstream task. Therefore, we add the edge type information to the edge-wise message $\mathbf{m}_{ij}^l = \mathbf{h}_j^{l-1} + \mathbf{W}^P \cdot \phi(p_{ij})$, where $\mathbf{W}^P \in \mathbb{R}^{d^{l-1} \times P}$ is a linear transformation for edge type, and ϕ is the one-hot encoding.

Edge Time. In addition, temporal information is critical in time series applications [17]. When edge time is available, we are also able to encode the timestamp to the edge-wise message with temporal embedding: $\mathbf{m}_{ij}^l = \mathbf{h}_j^{l-1} + \mathbf{W}^T \cdot TE(t_{ij}, \cdot)$, where $\mathbf{W}^T \in \mathbb{R}^{d^{l-1} \times d^{l-1}}$ is the linear transformation for edge time, and TE is the trigonometric temporal embedding [18].

By stacking multiple layers, GNNs are capable of encoding both node features and neighborhood graph structures into independent and identically distributed (i.i.d.) node embedding $\mathbf{Z}$, which can be further leveraged by Latent Diffusion Model. To ease the generation for Latent Diffusion Model, we use Variational Encoder, which outputs two different matrixes:

$$\boldsymbol{\mu} = \text{GNN}_{\boldsymbol{\mu}}(\text{GNN}_{\text{shared}}(\mathcal{G})), \quad \log\boldsymbol{\sigma} = \text{GNN}_{\boldsymbol{\sigma}}(\text{GNN}_{\text{shared}}(\mathcal{G})), \tag{2}$$

where $\boldsymbol{\mu}$ is the matrix of mean and log $\boldsymbol{\sigma}$ is the matrix of log standard deviation. While $\text{GNN}_{\text{shared}}$ is the shared GNN head, $\text{GNN}_{\boldsymbol{\mu}}$ and $\text{GNN}_{\boldsymbol{\sigma}}$ are specifically for $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, respectively. Then, the latent space embedding $\mathbf{Z}$ can be obtained via the parameterization trick:

$$\mathbf{Z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{3}$$

where $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ refers to a multivariate normal (or Gaussian) distribution with a mean of $\mathbf{0}$ and a covariance of $\boldsymbol{I}$, where $\mathbf{0}$ denotes a zero vector, and $\boldsymbol{I}$ represents a identity matrix.

3.3 Graph Generator

Graph Generator $G : \hat{\mathbf{Z}} \rightarrow \hat{\mathcal{G}}$ take the opposite process of Variational Encoder, generating $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}}, \hat{\mathbf{X}}, \hat{\mathbf{y}}, \hat{\mathbf{t}}, \hat{\mathbf{p}})$ given the latent space embedding $\hat{\mathbf{Z}}$. Each row $\hat{\mathbf{z}}_i$ in the embedding $\hat{\mathbf{Z}}$ is corresponding to a generated node $\hat{v}_i$ in $\hat{\mathcal{V}}$.

Class Label. Recall that our goal is exclusively generating outliers with positive labels. Consequently, rather than generating class labels, we take the desired labels as an input for conditional graph generation. Specifically, the class label is added to the embedding: $\hat{\mathbf{z}}_i^C = \hat{\mathbf{z}}_i + \mathbf{w}_G^C \cdot \hat{y}_i$, where $\hat{\mathbf{z}}_i$ is the i-th row of $\hat{\mathbf{Z}}$ and $\hat{\mathbf{z}}_i^C$ is embedding with class condition. $\mathbf{w}_G^C \in \mathbb{R}^{d^L}$ is the linear transformation for the class label, and d^L is the output dimension of the L-th layer of the encoder, i.e., the latent embedding dimension of $\hat{\mathbf{z}}_i$.

Node Feature. To generate the node feature $\hat{\mathbf{X}}$ from the latent space embedding, for the i-th row of $\hat{\mathbf{X}}$, we take the embedding with class condition $\hat{\mathbf{z}}_i^C$ as input:

$$\hat{\mathbf{x}}_i = \mathbf{W}_G^F \cdot \hat{\mathbf{z}}_i^C, \tag{4}$$

where $\mathbf{W}_G^F \in \mathbb{R}^{d \times d^L}$ is a learnable linear transformation matrix for node feature generation.

Graph Structure. The generation of graph structures denoted by the set of inferred edges $\hat{\mathcal{E}}$ requires link prediction between all pairwise combinations of nodes within the graph. The predicted edge score $\hat{e}_{ij}$ can be formulated as: $\hat{e}_{ij} = \text{sigmoid}(\mathbf{w}_G^E \cdot \text{CAT}(\hat{\mathbf{z}}_i^C, \hat{\mathbf{z}}_j^C)), \forall(\hat{v}_i, \hat{v}_j) \in \hat{\mathcal{V}} \times \hat{\mathcal{V}}$, where $\mathbf{w}_G^E \in \mathbb{R}^{2d^L}$ is the linearn transformation for the edge generation. The generated edges are determined as follows:

$$\hat{\mathcal{E}} = \{(\hat{v}_i, \hat{v}_j) \mid \hat{e}_{ij} \geq 0.5\} \tag{5}$$

Edge Type. With the generated edges in hand, we can predict the type of every generated edge:

$$\hat{p}_{ij} = \text{softmax}(\mathbf{W}_G^P \cdot \text{CAT}(\hat{\mathbf{z}}_i^C, \hat{\mathbf{z}}_j^C)), \forall(\hat{v}_i, \hat{v}_j) \in \hat{\mathcal{E}}, \tag{6}$$

where $\mathbf{W}_G^P \in \mathbb{R}^{P \times 2d^L}$ is the learnable linear transformation weight for the edge type prediction.

Edge Time. Similarly, the timestamp of generated edges can be predicted as follows:

$$\hat{t}_{ij} = \mathbf{w}_G^T \cdot \text{CAT}(\hat{\mathbf{z}}_i^C, \hat{\mathbf{z}}_j^C), \forall(\hat{v}_i, \hat{v}_j) \in \hat{\mathcal{E}}, \tag{7}$$

where $\mathbf{w}_G^T \in \mathbb{R}^{2d^L}$ is the weight for the edge timestamp regression.

3.4 Latent Diffusion Model

Utilizing Variational Encoder and Graph Generator, synthetic graph generation is already feasible. Nonetheless, the inherent complexity and heterogeneity of graph data pose a significant challenge when attempting a one-step estimation from a simple prior distribution, such as a Gaussian distribution, to the intricate target distribution. This challenge often leads to compromised generation quality and results in a marginal effect on the downstream graph outlier detection. Therefore, we integrate Latent Diffusion Model [9] to break down the estimation into a sequence of incremental steps. In each step, Latent Diffusion Model incrementally refines the distribution estimated, bridging the divergence between the simple prior and the intricate target distribution.

Latent Diffusion Model consists of a pair of processes. A fixed forward diffusion process $\{\mathbf{Z}(s)\}_{s=0}^S$, where $s \in [0, S]$ is a continuous diffusion step, which perturbs the original data $\mathbf{Z}(0) = \mathbf{Z}$ by the gradually adding Gaussian noise to obtain $\mathbf{Z}(S) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and a reverse denoising process $\{\hat{\mathbf{Z}}(s)\}_{s=0}^S$ employs a learned denoising function $\boldsymbol{\epsilon}_\theta$ to iteratively denoise the sampled noise from a simple prior distribution $\hat{\mathbf{Z}}(S)$ to obtain $\hat{\mathbf{Z}} = \hat{\mathbf{Z}}(0)$. In order to speed up the generation process, we adopt EDM [12] as our diffusion model. For the detailed design of forward process and reverse process, please refer to Appendix B.1 and B.2, respectively.

3.5 Training

Conventional graph generative models are highly constrained in scalability. Typical graph generative models are working molecular graphs that are at hundreds node scale [19]. However, outlier detection graphs are usually much larger, scaling to millions, even billions. Efficient training of generative models on large graphs requires special designs. We propose to apply negative sampling and graph clustering to improve the scalability of GODM.

Negative Sampling. In the graph structure generation, if the training of the generator includes every pair of nodes, this leads to a computational complexity of $O(n^2)$. This parabolic complexity is catastrophic for large graphs (e.g., millions scale). Furthermore, it will result in a high imbalance in the training of the edge predictor itself. Negative sampling emerges as a crucial technique to reduce computational cost and alleviate the imbalance. To form a concise training set $\bar{\mathcal{E}}$ for edge generator, apart from adding the positive edge, i.e., $\{\bar{e}_{ij} = 1 \mid e_{ij} \in \mathcal{E}\}$, we randomly select a negative edge (a pair of nodes that is not connected) corresponding to every positive edge, $\{\bar{e}_{\tilde{i}\tilde{j}} = 0 \mid e_{\tilde{i}\tilde{j}} \notin \mathcal{E}\}$. By this means, we reduce the complexity from $O(n^2)$ to $O(|\mathcal{E}|)$.

Graph Clustering. When the graph scales to millions of nodes, full-batch training becomes impractical, even with negative sampling. In addition, traditional neighbor sampling methods are well-suited in our case, as we need to reconstruct both the node feature and graph structure (i.e.,

edges). Specifically, node sampling cannot gather complete information for edge prediction, while edge sampling favors the nodes with a high degree. To address these challenges, we resort to graph clustering inspired by [20]. We first apply graph clustering algorithms (e.g., Metis [21]) to divide the large graph into small partitions. Then, we form mini-batches with partitioned subgraphs and train GODM on each mini-batch instead of the whole large graph.

Leveraging these two techniques, we are able to efficiently train GODM in two steps. We first train Variational Encoder and Graph Generator to bridge the graph space and latent space. Then, we train the diffusion model in the latent space.

Variational Encoder and Graph Generator. By integrating Variational Encoder and Graph Generator, we are able to train both of them in a variational autoencoder (VAE) fashion. We reconstruct the node feature with MSE (mean squared error): $\ell_{\mathbf{X}} = \frac{1}{|\mathcal{V}|}\sum_{v_i \in \mathcal{V}} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2$, where $\|\cdot\|_2$ denotes the L2 norm. Then, we predict the edges in the training edge set $\bar{\mathcal{E}}$ with binary cross entropy loss: $\ell_{\mathcal{E}} = -\frac{1}{|\bar{\mathcal{E}}|}\sum_{\bar{e}_{ij} \in \bar{\mathcal{E}}}(\bar{e}_{ij}\log(\hat{e}_{ij}) + (1 - \bar{e}_{ij}\log(1 - \hat{e}_{ij})))$. Similarly, we reconstruct the timestamp with MSE loss: $\ell_{\mathbf{t}} = \frac{1}{|\mathcal{E}|}\sum_{e_{ij} \in \mathcal{E}}(t_{ij} - \hat{t}_{ij})^2$, and edge type with cross-entropy loss: $\ell_{\mathbf{p}} = -\frac{1}{|\mathcal{E}|}\sum_{e_{ij} \in \mathcal{E}} p_{ij}\log(\hat{p}_{ij})$. We can obtain the total loss by the sum up of all reconstruction loss and the KL divergence loss:

$$\mathcal{L}_{\text{VAE}} = \omega_{\mathbf{X}}\ell_{\mathbf{X}} + \omega_{\mathcal{E}}\ell_{\mathcal{E}} + \omega_{\mathbf{t}}\ell_{\mathbf{t}} + \omega_{\mathbf{p}}\ell_{\mathbf{p}} + \beta\ell_{\text{kl}}, \tag{8}$$

where $\omega_{\mathbf{X}}$, $\omega_{\mathcal{E}}$, $\omega_{\mathbf{t}}$, and $\omega_{\mathbf{p}}$ are the weights adjusting the values of each term in the reconstruction loss to a comparable scale. The ℓ_{kl} is the KL divergence loss between the latent space embedding and prior distribution (e.g., Gaussian distribution). However, as we have an additional Latent Diffusion Model, we loosen this regularization on the latent embedding. Consequently, we use a small hyperparameter weight $\beta < 1$ to encourage the model to minimize reconstruction error while ensuring that the resultant embedding shape remains within the desired shape. The training of the VAE, consisting of Variational Encoder and Graph Generator, is presented in Algorithm 1 in Appendix C.

Latent Diffusion Model. Once obtain the well-trained Variational Encoder and Graph Generator, we start to train Latent Diffusion Model via denoising score matching. We train a multi-layer perceptron (MLP) as the denoising function $\boldsymbol{\epsilon}_\theta$ for denoising score matching:

$$\min \mathbb{E}_{\mathbf{Z}^0 \sim p(\mathbf{Z}^0)}\mathbb{E}_{\mathbf{Z}^s \sim p(\mathbf{Z}^s|\mathbf{Z}^0)}\|\boldsymbol{\epsilon}_\theta(\mathbf{Z}_C^s, s) - \boldsymbol{\varepsilon}\|_2^2, \tag{9}$$

where $\mathbf{Z}_C^s = \mathbf{Z}^s + \mathbf{y}\mathbf{W}^C$, $\mathbf{W}^C \in \mathbb{R}^{1 \times d^L}$, encoding class label for conditional generation, and $\boldsymbol{\varepsilon}$ is the noise. More details about the training of Latent Diffusion Model are available in Appendix B.3. The entire training process of Latent Diffusion Model is summarized in Algorithm 2 in Appendix C.

3.6 Inference

The well-trained GODM is utilized in the inference process, augmenting real organic data by generated data. We first sample $\hat{\mathbf{Z}}^S$ from Gaussian distribution. Then, we iteratively denoise $\hat{\mathbf{Z}}^{s_i}$ to obtain $\hat{\mathbf{Z}}^{s_{i-1}}$ conditioned on $\hat{\mathbf{y}} = \mathbf{1}$ via denoising function $\boldsymbol{\epsilon}_\theta$. The estimated $\hat{\mathbf{Z}}^0$ is fed into Graph Generator along with $\hat{\mathbf{y}}$ to generate synthetic graph $\hat{\mathcal{G}}$. Finally, the augmented graph $\mathcal{G}_{\text{sub}}$, which integrates the synthetic graph $\hat{\mathcal{G}}$ and the real organic graph $\mathcal{G}$, is used for training the downstream outlier detector D. Algorithm 3 in Appendix C presents the whole procedure of inference.

3.7 Complexity Analysis

To provide insights for handling large graphs which are common in real-world applications of graph outlier detection, we analyze the time efficiency and memory scalability of GODM in Appendix D.

4 Experiments

In this section, we systematically delineate the experiments to evaluate GODM. The code implementation of GODM is publicly available at: `https://github.com/kayzliu/godm`.

Table 1: Performance in AUC, AP, and Rec (%) on four datasets.

Dataset	**Weibo**			**Tolokers**			**Questions**			**Elliptic**		
Metric	AUC	AP	Rec	AUC	AP	Rec	AUC	AP	Rec	AUC	AP	Rec
GCN [22]	98.11	93.48	89.34	74.69	42.88	42.06	69.81	12.54	16.99	82.68	22.23	27.61
SGC [23]	98.66	92.46	87.90	70.67	38.03	35.98	69.88	10.13	15.62	73.02	11.44	9.14
GIN [24]	97.47	92.67	87.90	74.05	36.57	36.76	67.76	12.30	18.36	84.38	29.66	35.64
GraphSAGE [14]	96.54	89.25	86.17	79.42	48.65	46.42	71.69	17.63	21.10	85.31	37.52	36.20
GAT [25]	94.08	90.25	86.74	77.26	43.14	43.30	70.33	14.51	17.26	84.42	23.43	27.42
GT [26]	97.06	91.44	87.03	79.24	46.22	46.57	70.83	16.14	20.27	87.14	29.91	38.97
GAS [27]	94.88	90.70	86.74	76.91	47.35	45.02	64.50	13.61	17.53	87.81	40.03	44.78
DCI [28]	93.90	87.78	83.86	75.98	39.85	40.19	67.95	14.58	19.18	81.93	27.63	33.15
PCGNN [29]	90.89	84.57	79.83	72.18	37.52	36.76	68.38	14.79	16.99	86.50	42.66	43.77
GATSep [30]	96.72	91.55	89.05	79.63	46.08	46.73	69.96	15.98	19.18	83.89	21.46	21.15
BernNet [31]	93.85	88.00	85.30	76.20	42.20	42.21	70.80	16.04	17.53	82.01	20.52	23.55
AMNet [32]	95.88	89.74	85.59	75.83	42.66	41.90	69.71	17.02	19.18	80.06	16.73	17.17
BWGNN [33]	98.29	92.72	84.73	80.15	49.65	47.35	69.47	16.24	18.63	84.32	22.56	26.50
GHRN [34]	97.21	92.67	88.18	79.80	49.50	48.29	68.24	16.24	18.63	85.36	24.01	30.29
DAGAD [35]	98.54	83.36	90.78	77.69	33.94	44.39	71.21	6.88	20.55	85.62	26.18	40.54
GOVAE (ours)	99.46	96.84	**93.08**	83.42	**53.85**	52.49	75.73	19.13	23.84	83.93	36.66	42.66
GODM (ours)	**99.57**	**97.54**	**93.08**	**83.46**	52.95	**52.96**	**76.84**	**20.48**	**24.66**	**89.77**	**43.92**	**53.92**

4.1 Experimental Setups

To benchmark GODM and contemporary methods, we conduct experiments in a unified environment partially adapted from GADBench [3]. The implementation details are described in Appendix E.

4.1.1 Datasets

We use five different datasets from GADBench. Table 4 in Appendix F provides the statistics of the datasets. Weibo, Elliptic, Tolokers, and Questions are homogeneous static graphs. DGraph is a heterogeneous temporal graph with different types of edges. The graphs vary in scale, ranging from thousands to millions of nodes. Detailed descriptions of datasets are available in Appendix F.

4.1.2 Baselines

In our experiments, we evaluate GODM against different types of baseline methods. We have three types of baseline methods. For **general graph neural networks**, we choose: GCN [22], SGC [23], GIN [24], GraphSAGE [14], GAT [25], and GT [26]. For heterogeneous graphs, despite the rich literature, limited heterogeneous GNNs can cope with temporal information. As one of the representatives, we chose HGT [18] for our baseline. In addition to general GNNs, we also include eight **graph outlier detectors**: GAS [27], DCI [28], PCGNN [29], BernNet [31], GATSep [30], AMNet [32], and GHRN [34]. Because the **data augmentation** for graph outlier detection is a relatively new research topic, there are limited baselines. We compare with DAGAD [35], which is for homogeneous graphs. In addition, we introduce GOVAE, a variational autoencoder-only variant of GODM, that can work on both homogeneous datasets and heterogeneous datasets. We drop Latent Diffusion Model and form a variational autoencoder with Variational Encoder and Graph Generator. The inference process is a direct one-step estimation from prior Gaussian distribution. Detailed descriptions of each baseline are available in Appendix G. Along with GODM, we compare these three data augmentation methods for graph outlier detection in Table 5 in Appendix G.

4.1.3 Metrics

We follow the extensive literature in graph outlier detection [2, 3] to comprehensively evaluate the performance of graph outlier detectors with three metrics robust to class imbalance: Receiver Operating Characteristic-Area Under Curve (AUC), Average Precision (AP), and Recall@k (Rec), where the value of k is set to the number of actual outliers present in the dataset. Appendix H provides more details about the metrics used in the experiments.

4.1.4 Open-Source Package

To enhance accessibility, we encapsulate our data augmentation into user-friendly API and make our code a plug-and-play package. This package is built upon PyTorch and PyTorch Geometric (PyG) frameworks. It accepts a PyG data object as input and returns the augmented graph as output.

Table 2: GODM's performance improvements on graph oultier detection in AUC, AP, and Recall (%) on Tolokers.

	AUC	AP	Rec
GCN [22]	75.45 (+0.76)	44.17 (+1.29)	44.24 (+2.18)
SGC [23]	72.73 (+2.06)	39.75 (+1.72)	38.01 (+2.02)
GIN [24]	74.83 (+0.78)	38.54 (+1.96)	38.32 (+1.56)
GraphSAGE [14]	81.65 (+2.23)	52.53 (+3.87)	50.93 (+4.52)
GAT [25]	82.18 (+4.91)	51.13 (+7.99)	50.78 (+7.48)
GT [26]	82.73 (+3.49)	51.73 (+5.51)	50.93 (+4.36)
GAS [27]	76.96 (+0.05)	45.48 (-1.87)	43.15 (-1.87)
DCI [28]	73.75 (-2.23)	37.52 (-2.33)	37.54 (-2.65)
PCGNN [29]	73.65 (+1.47)	38.42 (+0.90)	38.01 (+1.25)
GATSep [30]	83.46 (+3.83)	52.95 (+6.87)	52.80 (+6.07)
BernNet [31]	76.53 (+0.33)	44.04 (+1.83)	42.21 (+0.00)
AMNet [32]	76.67 (+0.84)	44.31 (+1.65)	43.30 (+1.40)
BWGNN [33]	82.10 (+1.95)	51.94 (+2.29)	51.25 (+3.89)
GHRN [34]	82.05 (+2.25)	51.84 (+2.34)	51.56 (+3.27)

Table 3: Performance in AUC, AP, and Rec (%) on DGraph.

	AUC	AP	Rec
HGT [18]	72.89	3.13	5.20
w/o Feature	75.10	3.64	5.59
w/o Edge	74.36	3.52	5.76
w/o Time	74.23	3.29	4.73
w/o Type	75.09	3.62	5.98
GOVAE	75.36	3.58	5.54
GODM	**75.80**	**3.66**	**6.19**

By inheriting the `transforms.BaseTransform` in PyG, we enable users to apply GODM just like any other transformations supported in PyG, e.g., `ToUndirected`. We are also able to customize hyperparameters during initialization. Code Demo 1 in Appendix I gives an example of API usage. For distribution, we release our package on Python Package Index: `https://pypi.org/project/godm/`. Users can easily install GODM in a command line by executing `$ pip install godm`.

4.2 Performamce on Graph Outlier Detection

We start by examining how effective GODM is in the task of graph outlier detection on homogeneous graphs. We report graph outlier detection performance on four homogeneous datasets in terms of AUC, AP, and Rec of different algorithms in Table 1. In the table, the highest score in each metric is marked in bold, while the second-highest score is underlined. For DAGAD, we report the optimal performance of its two implementations with GCN and GAT. Although DAGAD shows superior performance compared to the standalone GCN and GAT, it fails to surpass more sophisticated detectors. On the other hand, GODM and GOVAE together demonstrate significant enhancements in graph outlier detection performance across all datasets and metrics. Notably, on the Weibo dataset, where the performance metric is already relatively high (AUC of 98.66 for SGC), GODM further elevates the performance, achieving an AUC of 99.57. A particularly noteworthy improvement is observed on Elliptic, where GODM enhances the Rec by over 20% (an increase of 9.14) compared to GAS, which is the second-best performing algorithm. Although GOVAE exhibits competitive performance, its limited generation ability results in generally weaker performance compared to GODM, especially on Elliptic.

Further investigations were carried out to understand the extent of improvement GODM could bring to various graph outlier detection algorithms, specifically on Tolokers. We integrate GODM on the top of all graph outlier detection algorithms and report the performance changes in terms of AUC, AP, and Rec in Table 2. The absolute performance is reported out of brackets, and the relative change is reported in the brackets. We can observe that GODM enhances graph outlier detection performance for most of the algorithms. GODM increases the AP of GAT by 7.99. For DCI and GAS, the only two algorithms that minorly decrease the performance, this could be attributed to suboptimal hyperparameter settings for these specific algorithms or the dataset. We believe the performance can be improved by careful hyperparameter tuning.

Moreover, in Table 3, we benchmark the graph outlier detector performance on DGraph, a large-scale dataset with different types of edges and temporal information. We include HGT along with variants of GODM. For the w/o Feature, we randomly sample the node feature from the Gaussian distribution to replace the generated node feature in the synthetic graph. For the w/o Edge, all generated edges are substituted with random edges. For w/o Time, we give uniformly sampled timestamps to replace the generated timestamps. For the w/o Type, we use a random type sampled from a uniform distribution for all generated edges. According to Table 3, GODM demonstrates enhanced performance across all metrics when compared to HGT. Furthermore, masking any generated information leads to a notable decline in performance, which underscores the significance of each aspect of the generated data. While GOVAE achieves commendable results, its performance still falls short of GODM's, highlighting the superior effectiveness of Latent Diffusion Model.

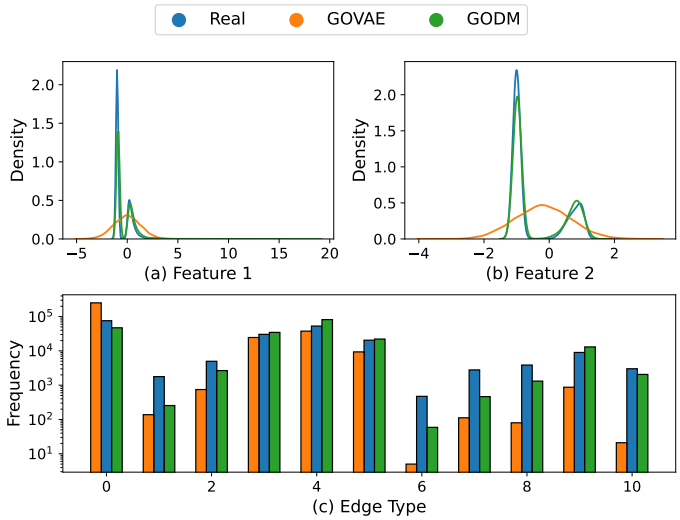

Figure 3: Visualization of single node feature density and edge type frequency of real data and synthetic data on DGraph.

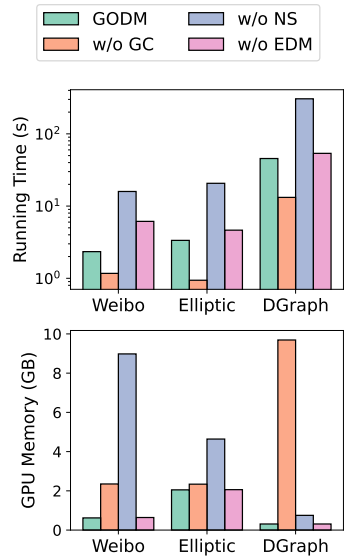

Figure 4: Running time and GPU memory consumption.

4.3 Generation Quality

To assess the quality of generated data, we conduct a case study on DGraph to compare real data and synthetic data from both GOVAE and GODM, in Figure 3. Specifically, Figure 3 (a) and (b) illustrate the distribution density of a single dimension in node features. From the figures, we can see that the distributions of GODM generated node features are close to the complex distributions of real data, while GOVAE can only generate Gaussian distributions, showcasing the generation ability of Latent Diffusion Model. In addition, to evaluate generated edges, Figure 3 (c) shows the frequency across eleven different edge types. The edge type generated by GODM has a more similar distribution to real data compared to GOVAE, particularly on the edge type 10. It further underscores the superiority of GODM over GOVAE.

4.4 Efficiency Study

We further evaluate the efficiency of GODM and its variants in terms of time and memory on graphs of various sizes, including Weibo, Elliptic, and DGraph. In order to demonstrate the efficiency of GODM, we remove graph clustering (w/o GC), negative sampling (w/o NS), and EDM (w/o EDM) for evaluation. To simulate the real-world application, we measure the running time of unit epoch training for VAE and Latent Diffusion Model plus the inference process. For memory, we present the maximum active GPU memory usage within the whole data augmentation, as the GPU memory constraint is usually the bottleneck for machine learning systems. The results are shown in Figure 4.

From Figure 4, we can observe that employing a graph clustering algorithm significantly diminishes GPU memory utilization. Specifically, on DGraph, w/o GC results in a memory usage that is over $30\times$ higher than GODM. On the other hand, graph clustering turns full-batch training into mini-batch training, leading to an anticipated increase in running time. Additionally, negative sampling reduces both the running time and the memory usage, as the number of edges in the training set reduces from $O(n^2)$ to $O(|\mathcal{E}|)$. While EDM maintains consistent memory consumption, it saves running time by minimizing the number of sampling steps.

5 Conclusion

In this paper, we introduce GODM, a novel method for data augmentation in supervised graph outlier detection. GODM is designed to address the significant challenge of class imbalance, a pervasive issue in graph outlier detection. GODM leverages the power of latent diffusion models to synthesize high-fidelity graph data that is statistically similar to real outliers. Note that GODM is model agnostic, which means it can be flexibly integrated with different downstream graph outlier detectors. We encapsulate GODM into a plug-and-play package, making it accessible to the community. For future research, we list several promising directions in Appendix J.

Acknowledgement

This work is supported in part by NSF under grants III-2106758, and POSE-2346158.

References

[1] Kay Liu, Yingtong Dou, Xueying Ding, Xiyang Hu, Ruitong Zhang, Hao Peng, Lichao Sun, and S Yu Philip. PyGOD: A Python library for graph outlier detection. *Journal of Machine Learning Research*, 25(141):1–9, 2024. 1, 13, 17

[2] Kay Liu, Yingtong Dou, Yue Zhao, Xueying Ding, Xiyang Hu, Ruitong Zhang, Kaize Ding, Canyu Chen, et al. BOND: Benchmarking unsupervised outlier node detection on static attributed graphs. *Advances in Neural Information Processing Systems*, 35:27021–27035, 2022. 7

[3] Jianheng Tang, Fengrui Hua, Ziqi Gao, Peilin Zhao, and Jia Li. GADBench: Revisiting and benchmarking supervised graph anomaly detection. *Advances in Neural Information Processing Systems*, 36:29628–29653, 2023. 1, 7, 14, 16

[4] Xuanwen Huang, Yang Yang, Yang Wang, Chunping Wang, Zhisheng Zhang, Jiarong Xu, Lei Chen, and Michalis Vazirgiannis. Dgraph: A large-scale financial dataset for graph anomaly detection. *Advances in Neural Information Processing Systems*, 35:22765–22777, 2022. 1, 18

[5] Yingtong Dou, Kai Shu, Congying Xia, Philip S Yu, and Lichao Sun. User preference-aware fake news detection. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2051–2055, 2021. 1

[6] Yingtong Dou, Guixiang Ma, Philip S Yu, and Sihong Xie. Robust spammer detection by nash reinforcement learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 924–933, 2020. 1

[7] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. 1, 18

[8] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 315–324, 2020. 1, 13, 14

[9] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. 2, 5

[10] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International Conference on Machine Learning*, pages 10362–10383. PMLR, 2022. 2, 14

[11] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations*, 2022. 2, 14

[12] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems*, 35: 26565–26577, 2022. 2, 5, 15

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 2, 15

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017. 4, 7, 8, 18

[15] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1): 17–37, 2016. 4

[16] Jianan Zhao, Xiao Wang, Chuan Shi, Zekuan Liu, and Yanfang Ye. Network schema preserving heterogeneous information network embedding. In *International Joint Conference on Artificial Intelligence*, 2020. 4

[17] Leyan Deng, Defu Lian, Zhenya Huang, and Enhong Chen. Graph convolutional adversarial networks for spatiotemporal anomaly detection. *IEEE Transactions on Neural Networks and Learning Systems*, 33(6):2416–2428, 2022. 4

[18] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of the Web Conference 2020*, pages 2704–2710, 2020. 4, 7, 8, 18

[19] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018. 5

[20] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 257–266, 2019. 6

[21] George Karypis and Vipin Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *http://glaros. dtc. umn. edu/gkhome/metis/metis/download*, 1997. 6

[22] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016. 7, 8, 18

[23] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pages 6861–6871. PMLR, 2019. 7, 8, 18

[24] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018. 7, 8, 18

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. 7, 8, 18

[26] Yunsheng Shi, Huang Zhengjie, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In *International Joint Conference on Artificial Intelligence*, pages 1548–1554, 08 2021. doi: 10.24963/ijcai.2021/214. 7, 8, 18

[27] Ao Li, Zhou Qin, Runshi Liu, Yiqun Yang, and Dong Li. Spam review detection with graph convolutional networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2703–2711, 2019. 7, 8, 13, 18

[28] Yanling Wang, Jing Zhang, Shasha Guo, Hongzhi Yin, Cuiping Li, and Hong Chen. Decoupling representation learning and classification for gnn-based anomaly detection. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1239–1248, 2021. 7, 8, 13, 18

[29] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. Pick and choose: a gnn-based imbalanced learning approach for fraud detection. In *Proceedings of the Web Conference 2021*, pages 3168–3177, 2021. 7, 8, 13, 14, 18

[30] Arthur Zimek, Ricardo JGB Campello, and Jörg Sander. Ensembles for unsupervised outlier detection: challenges and research questions a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1):11–22, 2014. 7, 8, 19

[31] Mingguo He, Zhewei Wei, Hongteng Xu, et al. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. *Advances in Neural Information Processing Systems*, 34: 14239–14251, 2021. 7, 8, 13, 19

[32] Ziwei Chai, Siqi You, Yang Yang, Shiliang Pu, Jiarong Xu, Haoyang Cai, and Weihao Jiang. Can abnormality be detected by graph neural networks. In *International Joint Conference on Artificial Intelligence*, pages 23–29, 2022. 7, 8, 13, 19

[33] Jianheng Tang, Jiajin Li, Ziqi Gao, and Jia Li. Rethinking graph neural networks for anomaly detection. In *International Conference on Machine Learning*, pages 21076–21089. PMLR, 2022. 7, 8, 13, 19

[34] Yuan Gao, Xiang Wang, Xiangnan He, Zhenguang Liu, Huamin Feng, and Yongdong Zhang. Addressing heterophily in graph anomaly detection: A perspective of graph spectrum. In *Proceedings of the ACM Web Conference 2023*, pages 1528–1538, 2023. 7, 8, 13, 19

[35] Fanzhen Liu, Xiaoxiao Ma, Jia Wu, Jian Yang, Shan Xue, Amin Beheshti, Chuan Zhou, Hao Peng, Quan Z Sheng, and Charu C Aggarwal. Dagad: Data augmentation for graph anomaly detection. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 259–268, 2022. 7, 14, 19

[36] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 594–602. SIAM, 2019. 13

[37] Zhiming Xu, Xiao Huang, Yue Zhao, Yushun Dong, and Jundong Li. Contrastive attributed network anomaly detection with data augmentation. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 444–457. Springer, 2022.

[38] Yixin Liu, Zhao Li, Shirui Pan, Chen Gong, Chuan Zhou, and George Karypis. Anomaly detection on attributed networks via contrastive self-supervised learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(6):2378–2392, 2021. 13

[39] Zhiwei Liu, Yingtong Dou, Philip S Yu, Yutong Deng, and Hao Peng. Alleviating the inconsistency problem of applying graph neural network to fraud detection. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1569–1572, 2020. 13

[40] Di Jin, Zhizhi Yu, Cuiying Huo, Rui Wang, Xiao Wang, Dongxiao He, and Jiawei Han. Universal graph convolutional networks. *Advances in Neural Information Processing Systems*, 34:10654–10664, 2021. 13

[41] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020. 13

[42] Wenjing Chang, Kay Liu, Kaize Ding, Philip S Yu, and Jianjun Yu. Multitask active learning for graph anomaly detection. *arXiv preprint arXiv:2401.13210*, 2024. 13

[43] Haoyan Xu, Kay Liu, Zhengtao Yao, Philip S Yu, Kaize Ding, and Yue Zhao. Lego-learn: Label-efficient graph open-set learning. *arXiv preprint arXiv:2410.16386*, 2024. 13

[44] Fangxin Wang, Yuqing Liu, Kay Liu, Yibo Wang, Sourav Medya, and Philip S. Yu. Uncertainty in graph neural networks: A survey. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=0e1Kn76HM1. 13

[45] Wenjing Chang, Kay Liu, Philip S Yu, and Jianjun Yu. Enhancing fairness in unsupervised graph anomaly detection through disentanglement. *arXiv preprint arXiv:2406.00987*, 2024. 13

[46] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 833–841, 2021. 14

[47] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016. 14

[48] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5708–5717. PMLR, 2018. 14

[49] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *International Conference on Machine Learning*, pages 610–619. PMLR, 2018. 14

[50] Youzhi Luo, Keqiang Yan, and Shuiwang Ji. Graphdf: A discrete flow model for molecular graph generation. In *International Conference on Machine Learning*, pages 7192–7203. PMLR, 2021. 14

[51] Hengrui Zhang, Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. In *The Twelfth International Conference on Learning Representations*, 2024. 14, 17

[52] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2020. 14, 15

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 2019. 17

[54] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019. 17

[55] Minjie Wang, Lingfan Yu, Quan Gan Da Zheng, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, et al. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 17

[56] Tong Zhao, Chuchen Deng, Kaifeng Yu, Tianwen Jiang, Daheng Wang, and Meng Jiang. Error-bounded graph anomaly loss for gnns. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1873–1882, 2020. 17, 20

[57] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of gnns under heterophily: Are we really making progress? In *The International Conference on Learning Representations*, 2022. 17

[58] Mufei Li, Eleonora Kreačić, Vamsi K Potluru, and Pan Li. Graphmaker: Can diffusion models generate large attributed graphs? *arXiv preprint arXiv:2310.13833*, 2023. 20

[59] Hongrui Liu, Binbin Hu, Xiao Wang, Chuan Shi, Zhiqiang Zhang, and Jun Zhou. Confidence may cheat: Self-training on graph neural networks under distribution shift. In *Proceedings of the ACM Web Conference 2022*, pages 1248–1258, 2022. 20

[60] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the International Conference on Computer Vision*, pages 4195–4205, 2023. 20

[61] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016. 20

[62] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015. 20

A Detailed Related Work

This section summarizes the previous related works in three key areas, including graph outlier detection, class imbalance, and graph generative models.

A.1 Graph Outlier Detection

Graph outlier detection is a vital task in data mining and machine learning, aiming to identify anomalous structures (e.g., nodes, edges, subgraphs) in graphs. Due to its structural simplicity and broader applications, most research focuses on node-level graph outlier detection and can easily convert to edge level and graph level task [1]. Many studies focus on unsupervised settings, detecting outliers in the graph only based on the graph data itself without any ground truth labels [36–38]. However, such unsupervised approaches may not align well with scenarios necessitating the detection of specific outlier types containing domain-specific knowledge. In this case, (semi-)supervised graph outlier detectors, which can learn ground truth labels, are better options. [27] apply attention mechanism to detect spam review. [8, 39] alleviate the camouflage issue in fraud detection by message passing in selected neighborhoods. [29] further improves the neighbor selection in fraud detection. [40] builds the outlier detection graph via the KNN algorithm. [28] decouples the representation learning and anomaly detection, and [41] focuses on heterophily in graph outlier detection. Some other works detect outliers from a spectral perspective. [31] learns arbitrary graph spectral filters via Bernstein approximation, and [32] is capable of discerning both low-frequency and high-frequency signals, thereby adaptively integrating signals across a spectrum of frequencies. In [33], the Beta kernel is employed to detect anomalies at higher frequencies using flexible and localized band-pass filters. [34] addresses heterophily in graph outlier detection from a graph spectrum perspective. [42, 43] alleviates the problem of limited supervision. [44] summarizes uncertainty-based detection methods. [45] focuses on fair graph outlier detection. Despite the fruitful literature on graph outlier detection, most methods underrate a pivotal challenge of class imbalance.

A.2 Class Imbalance

Conventional methods typically mitigate class imbalance by primitive techniques. GADBench uses reweighting [3], assigning a higher weight to outliers in the loss function. Some works adopt upsampling to replicate outliers, which is mathematically equivalent to reweighting. [8] applies downsampling, reducing the number of normal instances and losing a large amount of valuable supervision information from normal instances. However, these tricks may not sufficiently address the underlying complexities associated with imbalanced datasets. Consequently, there is a compelling need for more sophisticated techniques that can effectively address this class imbalance to ensure the quality of graph outlier detection. [29] balances the neighborhood distribution by reducing edges between different classes and adding edges between the same classes but does not change the overall number of positive and negative nodes. [35] adopts random perturbation for data augmentation to generate more samples. Other general graph machine learning studies apply interpolation-based data augmentation methods in latent space [46]. However, these methods are not specifically designed for graph outlier detection, and interpolation is too naive to generate helpful instances. This paper seeks to build upon these foundational works, proposing a novel graph data augmentation algorithm via latent diffusion models that not only mitigate class imbalance effectively but are also scalable and efficient on large graphs, ensuring broader applicability in graph outlier detection.

A.3 Graph Generative Models

Recent advances in graph generative models have catalyzed a significant body of research, focusing primarily on the synthesis of realistic and structurally coherent graphs. [47] proposes VGAE, leveraging variational autoencoder frameworks to learn latent representations of graph structures for generation purposes. [48] introduces GraphRNN, generating nodes and edges autoregressively with two recurrent neural networks. [49] applies a generative adversarial networks framework to generate graphs via random walks. [50] introduces the normalizing flow method for molecular graph generation. With the recent proliferation of works in image generation with diffusion models, some efforts have extended the generation power of diffusion models to graph-structured data. [10] proposes a score-based generative modeling of graphs via the system of stochastic differential equations, while [11] uses a discrete diffusion model to generate graphs. However, the potential of graph generative models to address the class imbalance in graph outlier detection remains largely underexplored. We propose to generate some synthetic outliers to mitigate this class imbalance.

B Detailed Design of Latent Diffusion Model

We follow [51] for the detailed design of Latent Diffusion Model.

B.1 Forward Diffusion Process

We construct a forward diffusion process $\{\mathbf{Z}(s)\}_{s=0}^{S}$, where $s \in [0, S]$ is a continuous diffusion step. In the diffusion process, $\mathbf{Z}(0) = \mathbf{Z}$ is the embedding from Variational Encoder, while $\mathbf{Z}(S) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is sampled from the prior distribution. According to [52], the forward diffusion process can be written in stochastic differential equation (SDE) as:

$$\mathrm{d}\mathbf{Z} = \boldsymbol{f}(\mathbf{Z}, s)\mathrm{d}s + g(s)\mathrm{d}\boldsymbol{\omega}_t, \tag{10}$$

where $\boldsymbol{\omega}_t$ is the standard Wiener process (i.e., Brownian motion). $\boldsymbol{f}(\cdot, s)$ and $g(\cdot)$ are the drift coefficient and the diffusion coefficient, respectively. The selection of $\boldsymbol{f}(\cdot, s)$ and $g(\cdot)$ vary between different diffusion models. $f(\cdot)$ is usually of the form $\boldsymbol{f}(\mathbf{Z}, s) = f(s)\mathbf{Z}$. Thus, Equation 10 can be written as:

$$\mathrm{d}\mathbf{Z} = f(s)\mathbf{Z}\mathrm{d}s + g(s)\mathrm{d}\boldsymbol{\omega}_t. \tag{11}$$

Let $\mathbf{Z}$ be a function of diffusion step s, i.e., $\mathbf{Z}^s = \mathbf{Z}(s)$. The diffusion kernel of Equation 11 can be represented in the conditional distribution of $\mathbf{Z}^s$ given $\mathbf{Z}^0$:

$$p(\mathbf{Z}^s|\mathbf{Z}^0) = \mathcal{N}(a(s)\mathbf{Z}^0, a^2(s)\sigma^2(s)\boldsymbol{I}), \tag{12}$$

where $a(s)$ and $\sigma(s)$ can be derived as:

$$a(s) = \exp\int_0^s f(\xi)\mathrm{d}\xi, \quad \sigma(s) = \sqrt{\int_0^s g^2(\xi)/a^2(\xi)\mathrm{d}\xi}. \tag{13}$$

Consequently, the formulation of the forward diffusion process is equivalent to the definition of diffusion kernels characterized by both $a(s)$ and $\sigma(s)$.

Denoising diffusion probabilistic models (DDPM) [13] can be seen as discretizations of the variance preserving SDE with $a(s) = \sqrt{1-\beta(s)}$ and $\sigma(s) = \sqrt{\beta(s)/(1-\beta(s))}$, as $a^2(s)+a^2(s)\sigma^2(s) = 1$. However, in order to achieve more efficient generation in GODM, we adopt EDM [12], which belongs to variance exploding SDE. Variance exploding SDE set $a(s) = 1$, which implies that noise is directly added to the data instead of being blended through weighting. In this case, the variance of the noise (the noise level) is exclusively determined by $\sigma(s)$. In EDM, a linear noise level $\sigma(s) = s$ is applied. Therefore, the diffusion kernel can be written as:

$$p(\mathbf{Z}^s|\mathbf{Z}^0) = \mathcal{N}(\mathbf{0}, \sigma^2(s)\boldsymbol{I}), \tag{14}$$

and the forward diffusion process can be formulated as:

$$\mathbf{Z}^s = \mathbf{Z}^0 + \sigma(s)\boldsymbol{\varepsilon},\ \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{15}$$

B.2 Reverse Denoising Process

As derived in [52], the reverse denoising process is formulated as the reverse SDE of Equation 10:

$$\mathrm{d}\mathbf{Z} = [\boldsymbol{f}(\mathbf{Z}, s) - g^2(s)\nabla_{\mathbf{Z}} \log p_s(\mathbf{Z})]\mathrm{d}s + g(s)\mathrm{d}\boldsymbol{\omega}_s, \tag{16}$$

where $\nabla_{\mathbf{Z}} \log p_s(\mathbf{Z})$ is the score function of $\mathbf{Z}$. In EDM, given $a(s) = 1$ and Equation 13,

$$\boldsymbol{f}(\mathbf{Z}, s) = f(s)\mathbf{Z} = \mathbf{0}, \quad g(s) = \sqrt{2\sigma(s)\dot{\sigma}(s)}, \tag{17}$$

where $\dot{\sigma}$ denotes the first order derivative of σ. With $\boldsymbol{f}(\mathbf{Z}, s)$ and $g(s)$, we are able to obtain:

$$\mathrm{d}\mathbf{Z} = -2\dot{\sigma}(s)\sigma(s)\nabla_{\mathbf{Z}} \log p_s(\mathbf{Z})\mathrm{d}s + \sqrt{2\dot{\sigma}(s)\sigma(s)}\mathrm{d}\boldsymbol{\omega}_s, \tag{18}$$

where the noise level $\sigma(s) = s$.

B.3 Training of Latent Diffusion Models

To solve the SDE in Equation 18, we need to obtain the score function $\nabla_{\mathbf{Z}} \log p_s(\mathbf{Z})$, which is as intractable as $p_s(\mathbf{Z}) = p(\mathbf{Z}^s)$. However, the conditional distribution $p(\mathbf{Z}^s|\mathbf{Z}^0)$ is tractable. From Equation 12, we obtain its analytical solution:

$$\nabla_{\mathbf{Z}} \log p(\mathbf{Z}^s|\mathbf{Z}^0) = -\frac{\boldsymbol{\varepsilon}}{a(s)\sigma(s)}. \tag{19}$$

C Algorithms

In this section, we provide algorithms used in GODM. Algorithm 1 gives the training process of VAE. Algorithm 2 shows the training process of Latent DIffusion Model. Algorithm 3 presents the inference process.

Algorithm 1 GODM: Training of VAE

Input: Input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{y}, \mathbf{t}, \mathbf{p})$
Output: Node representation $\mathbf{Z}$, well-trained Variational Encoder E and Graph Generator G

```
Partition graph G into a set of subgraphs {G_sub}
for each G_sub do
    Get μ and logσ via Equation 2
    Reparameterization of Z via Equation 3
    Generate X̂, Ê, t̂, p̂ via Equation 4, 5, 7, 6
    Calculate loss via Equation 8
    Update Variational Encoder and Graph Generator parameters via the Adam optimizer
end for
```

Algorithm 2 GODM: Training of Latent Diffusion Model

Input: Node representation $\mathbf{Z}$
Output: Well-trained denoising function $\boldsymbol{\epsilon}_\theta$
1: Sample the embedding $\mathbf{Z}^0$ from $p(\mathbf{Z}) = p(\boldsymbol{\mu})$
2: Sample diffusion steps s from $p(s)$ then get $\sigma(s)$
3: Sample noise vectors $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2(s)\boldsymbol{I})$
4: Get perturbed data $\mathbf{Z}^s = \mathbf{Z}^0 + \boldsymbol{\varepsilon}$
5: Calculate loss $\ell(\theta) = \|\boldsymbol{\epsilon}_\theta(\mathbf{Z}_C^s, s) - \boldsymbol{\varepsilon}\|_2^2$
6: Update the network parameter θ via Adam optimizer

Algorithm 3 GODM: Inference

Input: Graph generator G and denoising function $\boldsymbol{\epsilon}_\theta$
Output: Augmented graph $\mathcal{G}_{\text{aug}}$
1: Sample $\hat{\mathbf{Z}}^S \sim \mathcal{N}(\mathbf{0}, \sigma^2(S)\boldsymbol{I})$
2: **for** $i = \max, \cdots, 1$ **do**
3: $\quad \nabla_{\hat{\mathbf{Z}}^{s_i}} \log p(\hat{\mathbf{Z}}^{s_i}) = -\boldsymbol{\epsilon}_\theta(\hat{\mathbf{Z}}_C^{s_i}, s_i)/\sigma(s_i)$
4: $\quad$ get $\hat{\mathbf{Z}}^{s_{i-1}}$ via solving the SDE in Equation 18
5: **end for**
6: Generate $\hat{\mathcal{G}} = G(\hat{\mathbf{Z}}^0, \hat{\mathbf{y}})$
7: $\mathcal{G}_{\text{aug}} = \text{batch}(\mathcal{G}, \hat{\mathcal{G}})$

D Complexity Analysis

In this section, we meticulously analyze and elucidate the time efficiency and memory scalability of GODM in training and inference for both VAE (Variational Encoder and Graph Generator) and Latent Diffusion Model.

D.1 Time Efficiency

In the training of VAE, the major bottleneck comes from the graph structure generation in Graph Generator. This generation originally requires link prediction for every pair of nodes in the graphs, which is $O(n^2)$. By graph clustering, we divide the large graph into relatively small partitions for mini-batch training. We denote the average partition size (i.e., batch size) by b, so the number of partitions is $\frac{n}{b}$. Then, the complexity is reduced to $O(\frac{n}{b}b^2) = O(nb)$, as only node pairs within each partition are predicted. We apply the negative sampling to further reduce the complexity to $O(|\mathcal{E}|)$, as we only sample a negative edge for each positive edge. For Latent Diffusion Model, we only sample a fixed number of diffusion steps to train the denoising function. As a result, assuming the number of latent dimensions is a constant, the complexity is $O(n)$. Thus, the total training complexity is $O(n + |\mathcal{E}|)$. For the inference, we first take S steps sampling in Latent Diffusion Model, which is $O(nS)$, and predict edges between node pairs in each partition, which is $O(nb)$. Therefore, the total inference complexity is $O(nS + nb)$.

D.2 Memory Scalability

The vanilla VAE requires memory consumption of $O(n^2)$, which is infeasible for large graphs. To reduce memory consumption, we adopt mini-batch training on graph partitions and negative sampling for link prediction. The complexity of VAE training can be obtained by the number of edges divided by the number of partitions $O(|\mathcal{E}|/\frac{n}{b}) = O(\frac{|\mathcal{E}|b}{n})$. Latent Diffusion Model only requires a memory of $O(b)$. In the inference process, the VAE performs link prediction for all node pairs in a batch, leading to a memory complexity of $O(b^2)$, and Latent Diffusion Model only demands $O(b)$ for each batch. Therefore, the total inference process needs $O(b^2)$ of memory.

E Implementation Details

We modified GADBench [3] to benchmark graph outlier detection performance.

Environment. The key libraries and their versions used in experiments are as follows: Python 3.9, CUDA 11.8, PyTorch 2.0.1 [53], PyG 2.4.0 [54], DGL 1.1.2 [55], and PyGOD 1.0.0 [1].

Hardware. All of our experiments were performed on a Linux server with an AMD EPYC 7763 64-core CPU, 192GB RAM, and an NVIDIA RTX A40 GPU with 48GB memory.

Hyperparameters. GODM is implemented with default hyperparameters. For Variational Encoder, we use one layer GraphSAGE for $\text{GNN}_{\text{shared}}$, $\text{GNN}_{\boldsymbol{\mu}}$, and $\text{GNN}_{\boldsymbol{\sigma}}$, respectively. We set the hidden dimension to the largest power of 2 that is no greater than the feature dimension of the dataset divided by 2. To avoid extensive hyperparameter search, we set the weights of reconstruction loss $\omega_{\mathbf{X}}$, $\omega_{\mathcal{E}}$, $\omega_{\mathbf{t}}$, and $\omega_{\mathbf{p}}$ to roughly balance their absolute value only, which are 1, 0.5, 1, and 0.3, respectively. The weight for KL-divergence β is 0.001. For the diffusion model, we use the dimension twice as the hidden dimension in VAE. We use five layers of MLP as the denoising function following the detailed design in [51]. For training, we adopt the Adam optimizer with a learning rate of 0.001 without weight decay to train the VAE and the diffusion model for 100 epochs, respectively, and apply early stopping with a patience of 50. The negative sampling ratio is 1, and the approximate graph partition size is 2048. For inference, we use 50 steps of diffusion and generate the same amount of synthetic outliers as the number of real organic outliers in the training set. For all the hyperparameters in graph outlier detection, we apply the default setting in GADBench. We report the graph outlier detection performance of GODM with the optimal downstream graph outlier detector.

F Description of Datasets

Table 4: Statistics of datasets (*: with multiple edge types and temporal infomation).

Dataset	#Nodes	#Edges	#Features	Outlier
Weibo	8,405	407,963	400	10.3%
Tolokers	11,758	519,000	10	21.8%
Questions	48,921	153,540	301	3.0%
Elliptic	203,769	234,355	166	9.8%
DGraph*	3,700,550	4,300,999	17	1.3%

This section describes the datasets used in the experiments. Table 4 provides the statistics of the datasets. In the table, #Nodes stands for the number of nodes, and #Edges stands for the number of edges. #Feature denotes the raw feature dimension, i.e., the number of node attributes. The Outlier column represents the outlier ratio in the label, indicating the extent of class imbalance. The detailed descriptions for each dataset used in the experiments are as follows:

Weibo [56]: This dataset involves a graph delineating the associations between users and their corresponding hashtags from the Tecent-Weibo platform, consisting of 8,405 users and a collection of 61,964 hashtags. Activities within this dataset are regarded as suspicious (i.e., outliers) if they consist of a pair of posts occurring within narrowly defined temporal intervals (e.g., 60 seconds). Users engaging in at least five incidents of such behavior are classified under the suspicious category, in contrast to the remainder designated as benign. Following this classification criterion, the dataset contains 868 users identified as suspicious and 7,537 as benign. The primary feature vector includes geolocation data for each micro-blog entry and a representation utilizing the bag-of-words model.

Tolokers [57]: The dataset is obtained from the Toloka crowdsourcing platform. It is composed of nodes corresponding to individual workers who have engaged in at least one out of thirteen specified projects. Edges are established between pairs of workers who have concurrently contributed to an identical task. The primary objective is to predict the likelihood of a worker having received a ban in any one of the projects. The features attributed to each node are constructed utilizing the worker's personal profile information in conjunction with their task-related metrics.

Questions [57]: The dataset is derived from Yandex Q, a question-answering platform. The nodes represent users, and an edge is established between two nodes to denote the scenario wherein one user has responded to another's inquiry within the temporal bounds of one year, stretching from September 2021 to August 2022. This dataset is specifically curated to encapsulate the engagement of users who exhibit interest in the medical topic. The target for this dataset is to predict the likelihood of users' continued activity on the platform by the conclusion of the observed period. The average FastText embeddings of the lexical content present in the users' descriptions are used for node features.

Elliptic [7]: The dataset comprises a graph of 203,769 nodes representing Bitcoin transactions, connected through 234,355 edges representing payment flows, along with 166 distinctive node attributes. It correlates Bitcoin transactional data with corresponding real-world entities that are categorized as lawful, including exchanges, wallet services, mining operations, and legitimate services, as well as unlawful categories, including scams, malicious software, terrorist-related organizations, ransomware operations, and fraudulent investment activities known as Ponzi schemes.

DGraph [4]: DGraph is a large-scale graph with different edge types and temporal information supplied by Finvolution Group. It includes around 3 million nodes, 4 million dynamic edges, and 1 million node labels. The nodes represent user accounts within a financial organization offering personal loan services, while an edge between two nodes indicates that one account has designated the other as an emergency contact. Nodes classified as fraud correspond to users displaying delinquent financial conduct. For those accounts with borrowing records, outliers are identified as accounts with a history of overdue payments, while inliers are those without such a history. Additionally, the dataset includes 17 node features derived from user profile information.

G Description of Baselines

This section provides detailed descriptions of the baselines in the experiments.

G.1 General Graph Neural Network

GCN (Graph Convolutional Networks) [22]: GCN is the seminal work that applies convolution operation on graph data. It propagates the information of a node to its neighbors, thus enabling the network to develop a representation for each node that reflects its local neighborhood context.

SGC (Simplified Graph Convolution) [23]: This variant of GCN leverages Chebyshev polynomials to approximate the spectral graph convolution operator. This strategy allows the model to encompass both local and global graph structures, enhancing its scalability for handling larger graphs.

GIN (Graph Isomorphism Network) [24]: GIN is a form of GNN that effectively captures graph structures while maintaining graph isomorphism. It achieves this by generating consistent embeddings for structurally identical graphs, which is permutation invariant.

GraphSAGE (Graph Sample and Aggregate) [14]: GraphSAGE presents a general inductive learning approach where node embeddings are generated through the sampling and aggregation of features from a node's immediate neighborhood.

GAT (Graph Attention Networks) [25]: GAT incorporates the attention mechanism within the GNN framework. It dynamically assigns varying importance to different nodes during the information aggregation process, focusing the model's learning on the most relevant parts of the neighborhood.

GT (Graph Transformer) [26]: Drawing inspiration from the Transformer model in neural networks, GT adapts these principles for graph-structured data. It utilizes masks in the self-attention mechanism to capitalize on the inherent structure of graphs, thus boosting the model's efficiency.

HGT (Heterogeneous Graph Transformer) [18]: HGT is designed to address the challenges of modeling heterogeneous graphs. It introduces node- and edge-type dependent parameters for heterogeneous attention mechanisms for each type of edge.

G.2 Graph Outlier Detector

GAS (GCN-based Anti-Spam) [27]: GAS is an attention-based spam review detector, extending the capabilities of GCN to process heterogeneous and heterophilic graphs. It employs the KNN algorithm to align with the structure of each graph.

DCI (Deep Cluster Infomax) [28]: DCI is a self-supervised learning strategy that separates the learning of node representations from outlier detection. It addresses discrepancies between node behavioral patterns and their label semantics by clustering, thus capturing intrinsic graph properties in focused feature spaces.

PCGNN (Pick and Choose Graph Neural Network) [29]: Tailored for imbalanced GNN learning in fraud detection scenarios, PCGNN uses a label-balanced sampler for node and edge selection during training. This results in a more balanced label distribution within the induced subgraph.

BernNet [31]: BernNet is a GNN variant offering a robust approach to designing and learning arbitrary graph spectral filters. It utilizes an order-K Bernstein polynomial approximation for estimating filters over the normalized Laplacian spectrum, catering to a variety of graph structures.

GATSep [30]: Designed to optimize learning on heterophily graphs, GAT-sep merges key design elements like ego- and neighbor-embedding separation, higher-order neighborhood processing, and combinations of intermediate representations.

AMNet (Adaptive Multi-frequency GNN) [32]: AMNet is structured to capture signals across both low and high frequencies by stacking multiple BernNets and adaptively integrating signals from different frequency bands.

BWGNN (Beta Wavelet Graph Neural Network) [33]: BWGNN addresses the "right-shift" phenomenon on outliers. It uses the Beta kernel to address higher frequency anomalies through spatially/spectrally localized band-pass filters.

GHRN (Graph Heterophily Reduction Network) [34]: GHRN tackles the issue of heterophily in the spectral domain for graph outlier detection. This approach focuses on pruning inter-class edges to enhance the representation of high-frequency components in the graph spectrum.

G.3 Data Augmentation

DAGAD (Data Augmentation for Graph Anomaly Detection) [35]: DAGAD incorporates three modules to augment the graph, including an information fusion module for representation learning, a data augmentation module to enrich the training set with synthetic samples, and an imbalance-tailored learning module to distinguish between minority anomalous class and majority normal class.

GOVAE: In this variant, we drop Latent Diffusion Model and form a variational autoencoder with Variational Encoder and Graph Generator. The inference process is a direct one-step estimation from prior Gaussian distribution.

Along with **GODM**, we compare three data augmentation methods in Table 5, where the flexibility represents whether the data augmentation is model agnostic. Heterogeneous and time indicate whether the method supports these two types of information. The diffusion column indicates whether the method adopts diffusion models.

Table 5: Comparison of data augmentation methods.

	Flexibility	Heterogeneous	Time	Diffusion
DAGAD	✗	✗	✗	✗
GOVAE	✓	✓	✓	✗
GODM	✓	✓	✓	✓

H Description of Metrics

AUC (Receiver Operating Characteristic-Area Under Curve): AUC quantifies the area beneath the Receiver Operating Characteristic Curve, which is constructed by plotting the true positive rate against the false positive rate across varied determined threshold levels. An AUC of 1 indicates flawless predictive accuracy, whereas an AUC of 0.5 indicates an absence of discriminative power, equivalent to random guessing. This metric is favored over accuracy for evaluating outlier detection models due to its robustness against class imbalance prevalent within the class distributions.

AP (Average Precision): AP offers a comprehensive summary of the precision-recall curve, represented as the weighted average of precision values attained at each threshold, utilizing the increment in recall from the preceding threshold as the weight. This metric offers a balance between recall and precision, with a higher AP signifying a lower rate of both false-positive rate (FPR) and false-negative rate (FNR). For outlier detection applications, e.g., fraud detection, both FPR and FNR are critical, as misclassification could either cause financial loss or harm normal user experience.

Rec (Recall@k): Considering the minority of outliers relative to the abundance of normal instances in datasets, Recall@k is proposed as a measure of how well the detectors rank outliers over the normal samples. The value of k is set to the number of actual outliers present in the dataset. Recall@k is

then determined by the ratio of the true outliers within the top k-ranked samples to k. A maximal Recall@k value of 1 means a model perfectly ranks all outliers over normal samples, while minimal value of 0 means none of the outliers is ranked at top k.

I API Demo

```
from pygod.utils import load_data        # import data
data = load_data('weibo')                # load weibo data

from godm import GODM                    # import GODM
godm = GODM(lr=0.004)                    # initialize GODM
aug_data = godm(data)                    # augment data

detector(aug_data)                       # train on data
```

Code Demo 1: Using GODM on Weibo dataset [56].

J Future Directions

For future research, here are several promising directions we may explore to further enhance GODM and expand its applicability in the domain of graph outlier detection:

- **Diffusion in graph sapce**. Considering the heterogeneity and scalability of the graphs for outlier detection applications, GODM currently employs a diffusion model in latent space. [58] starts to generate large graphs directly in graph space. Future research could explore the application of diffusion models on heterogeneous information directly in graph space, potentially enhancing the model's ability to capture complex graph structures.
- **Collaboration with outlier detection**. Currently, GODM is model agnostic, meaning the training of GODM and downstream outlier detection tasks are asynchronous. On the other hand, integrating the supervision signals from both the generation task and the downstream outlier detection in a unified training framework could be mutually beneficial. Self-training offers a promising paradigm for this integration of the supervision signals [59].
- **Enhancing generative models**. Although the current diffusion model in GODM has shown impressive capabilities, there are opportunities for further improvement. This could be achieved by employing more expressive denoising functions, such as transformers [60], or by exploring alternative generative models like energy-based models [61] and or normalizing flows [62], which might offer different advantages over diffusion models.