MEGAD: A Memory-Efficient Framework for Large-Scale Attributed Graph Anomaly Detection

Yifan Zhang[1] , **Haolong Xiang**[1*] , **Xiaolong Xu**[1] , **Zishun Rui**[1] , **Xiaoyong Li**[2] , **Lianyong Qi**[3] , **Fei Dai**[4]

[1]Nanjing University of Information Science and Technology
[2]National University of Defense Technology
[3]China University of Petroleum (East China)
[4]Southwest Forestry University
{yifanzhang20011231, njuxlxu, zishunrui, lianyongqi, flydai.cn}@gmail.com, hlxiang@nuist.edu.cn, sayingxmu@nudt.edu.cn,

Abstract

Graph anomaly detection (GAD), with its ability to accurately identify anomalous patterns in graph data, plays a vital role in areas such as network security, social media platforms, and fraud detection. Graph autoencoder-based methods are widely used for GAD due to their efficiency and effectiveness in capturing complex patterns and learning meaningful representations. However, the above methods are constrained by hardware memory, hindering the detection for large-scale graph data. In this paper, we propose a Memory-Efficient framework for large-scale attributed Graph Anomaly Detection (MEGAD). Specifically, MEGAD first generates node embeddings and then refines them through a lightweight joint optimization model, ensuring minimal memory overhead. The optimized embeddings are subsequently fed into a detector to compute anomaly scores. Extensive experiments demonstrate that our framework achieves comparable accuracy to state-of-the-art methods across multiple datasets while significantly reducing memory consumption on large-scale graphs.

1 Introduction

Graph Anomaly Detection (GAD) aims to identify unusual patterns in graph data that deviate from the majority, with diverse applications in social network analysis, vulnerability detection, and biological network analysis [Pazho *et al.*, 2023]. Although anomaly points constitute a small proportion of the graph data and occur infrequently, failing to detect these anomalies promptly can lead to serious consequences, such as engineering accidents in manufacturing or fatalities in healthcare. To address these issues, a lot of methods have been proposed, such as matrix factorization-based methods [Liu *et al.*, 2017] and random walk-based methods [Wang and Davidson, 2009]. These methods provide intuitive results and high computational efficiency [Tang *et al.*, 2023]. While the aforementioned methods are capable of handling simple graph structures, they often fail to capture the nonlinear relationships and intricate patterns present in attributed graph data. To comprehensively analyze graph data with node attributes, isolation forest-based models have been introduced for the GAD task [Xiang *et al.*, 2024; Zhang *et al.*, 2024]. A significant advantage of tree ensemble models is their ability to efficiently train models on large-scale graph data while maintaining strong interpretability [Zhang *et al.*, 2017; Xiang *et al.*, 2023]. Unfortunately, tree ensemble-based models focus on node attributes while neglecting the graph structure information, resulting in poor robustness and low accuracy. In summary, traditional models struggle to account for both graph structure information and node attribute information when addressing GAD issues.

Recently, deep learning-based GAD methods have exhibited superior performance, as they can fully exploit both the graph structure and node attribute information. For instance, Graph Attention Networks (GAN) [Veličković *et al.*, 2018] and Graph Autoencoders (GAE) [He *et al.*, 2024] have achieved remarkable success in capturing complex graph structures and node relationships. However, these methods rely heavily on relatively complete labeled data, and their accuracy drops significantly when labeled data is scarce[Tang *et al.*, 2023]. To mitigate the impact of missing labels, unsupervised learning GAD methods are employed to prevent accuracy degradation. A widely used framework is the combination of Graph Neural Networks (GNN) and Autoencoders(AE) [Kipf and Welling, 2022; Zhu *et al.*, 2023], which achieves unsupervised graph anomaly detection through graph reconstruction. Nevertheless, the memory consumption of graph reconstruction technology increases exponentially with the size of the graph, causing most devices to fall short of the memory requirements when processing large-scale graphs. To reduce memory consumption, numerous methods based on subgraph sampling or neighborhood reconstruction have been proposed, such as GADNR [Roy *et al.*, 2024]. These methods alleviated the issue of insufficient memory, but their accuracy is generally lower than that of full-graph reconstruction methods due to information loss caused by sampling [Tang *et al.*, 2023]. An interesting question arises naturally: How to design a GAD method for large-

scale attributed graphs that optimizes precision while ensuring low memory consumption?

To address the issue of memory constraints without sacrificing accuracy, we propose a novel GAD method, called MEGAD. This method achieves efficient anomaly detection with low memory consumption by employing node-level reconstruction, making it suitable for large-scale datasets. Specifically, we first utilize a pre-training module to generate graph embeddings. Subsequently, we design a lightweight model to optimize these embeddings. The optimized embeddings are then passed to a detector to generate anomaly scores. We conducted extensive experiments on multiple datasets, including ablation experiments and comparative studies. The results demonstrate that our proposed method achieves accuracy comparable to state-of-the-art GAD methods while maintaining the lowest memory consumption on large-scale graph datasets.

The main contributions of our work are summarised as follows:

- We propose a GAD method that combines pre-training with a lightweight joint optimization model, successfully achieving high accuracy and robustness in large-scale graph data.
- Our MEGAD enables dynamic trade-offs between computational efficiency and memory usage, allowing flexible parameter adjustments based on the requirements of specific application scenarios.
- Extensive experiments show that our proposed method achieves state-of-the-art accuracy, especially on large-scale graph datasets where other models struggle to perform effectively. The source code is available at https://github.com/GraphAnomalyDetection/MEGAD.

2 Related Work

2.1 Graph Anomaly Detection

The early works [Xu *et al.*, 2007; Li *et al.*, 2017] typically used non-deep learning methods, such as node clustering, to detect anomalies. However, these methods failed to fully leverage the deep structural information in graphs, making it difficult to improve detection accuracy. The rapid advancement of deep learning technologies has led to the proliferation of AE-based GAD algorithms, achieving high accuracy. DONE [Bandyopadhyay *et al.*, 2020] trains separate autoencoders for attributes and structure, linking them through a joint loss function. Building upon this, its successor AdONE replaces the rigid L2 alignment constraint with an adversarial discriminator to flexibly align the two embeddings. Similarly, AnomalyDAE [Fan *et al.*, 2020] employs a dual AE to model attribute-structure interactions for anomaly scoring jointly. Similarly, DOMINANT follows the reconstruction-based paradigm but refines anomaly scoring by explicitly decoupling and fusing feature and structural reconstruction errors, enhancing node-level anomaly detection. Additionally, other non-AE-based methods have also shown strong performance. Among them, CONAD [Xu *et al.*, 2022a] innovatively incorporates contrastive learning, significantly enhancing the model's discriminative capability for anomalous samples; while GAAN employs a generative adversarial framework, where a generator synthesizes anomalous nodes and an encoder contrasts structural features between real and generated nodes in the latent space, ultimately performing anomaly detection by combining reconstruction error and real-sample discrimination confidence.

2.2 Graph Autoencoder

GAE employs GCN as the encoder to learn low-dimensional node embeddings and reconstructs the adjacency matrix through a decoder [Kipf and Welling, 2022]. This architecture effectively captures structural information and has demonstrated strong performance in graph embedding tasks [Yin *et al.*, 2024; Tao *et al.*, 2024]. However, GAE-based methods exhibit significant scalability bottlenecks. These approaches require processing the complete adjacency matrix, resulting in memory complexity that scales quadratically with the number of nodes, thereby hindering their application to large-scale graph data. Although approximation techniques such as neighborhood sampling [Hamilton *et al.*, 2017] and subgraph processing [Chiang *et al.*, 2019] can alleviate computational burdens, they typically come at the cost of compromised anomaly detection accuracy. More critically, directly applying generic graph learning methods to graph anomaly detection tasks often yields suboptimal detection performance. Consequently, the design of GAD methods that simultaneously achieve scalability and high precision remains a crucial research challenge.

3 Problem Statement

We are committed to identifying outlier nodes in a static attributed graph. Specifically, a graph can be represented as $\mathcal{G} = \{\mathcal{V}, \mathcal{A}, \mathcal{X}\}$, where $\mathcal{V} = \{v_1, v_2 \cdots v_N\}$ represents the node set of $\mathcal{G}$, $\mathcal{A} \in \mathbb{R}^{N \times N}$ is a adjacency matrix. $\mathcal{A}_{i,j} = 1$ if there exist an edge between v_i and v_j, otherwise $\mathcal{A}_{i,j} = 0$. $\mathcal{X} \in \mathbb{R}^{N \times M}$ is a feature matrix of $\mathcal{G}$. The i-th row of $\mathcal{X}$ represent the M-dimensional feature of v_i.

Due to the difficulty in obtaining ground truth for large-scale graph datasets, we hope to use unsupervised learning to complete the graph anomaly detection task. To address this problem, a commonly employed approach is to train the model for anomaly detection by leveraging reconstruction-based loss functions:

$$\begin{aligned} \mathcal{Z} &= \text{forward}(\mathcal{X}, \mathcal{A}), \\ \mathcal{L} &= ||\mathcal{A} - \mathcal{Z}\mathcal{Z}^T||_2, \end{aligned} \tag{1}$$

where common "forward" methods include GCN, GNN. $\mathcal{Z} \in \mathbb{R}^{N \times D}$ is the embedding of the graph $\mathcal{G}$ with dimension D.

However, the aforementioned methods struggle to process large-scale graph datasets due to excessive memory consumption. To address these issues, we are committed to designing an innovative memory-efficient GAD framework ($\mathcal{F}$) that aims to improve detection accuracy on large-scale graph datasets while minimizing memory usage. The prediction result of $\mathcal{F}$ is represented as:

$$\begin{aligned} P &= \mathcal{F}(\mathcal{G}), \\ \mathcal{G} &= \{\mathcal{V}, \mathcal{E}, \mathcal{X}\}, \end{aligned} \tag{2}$$

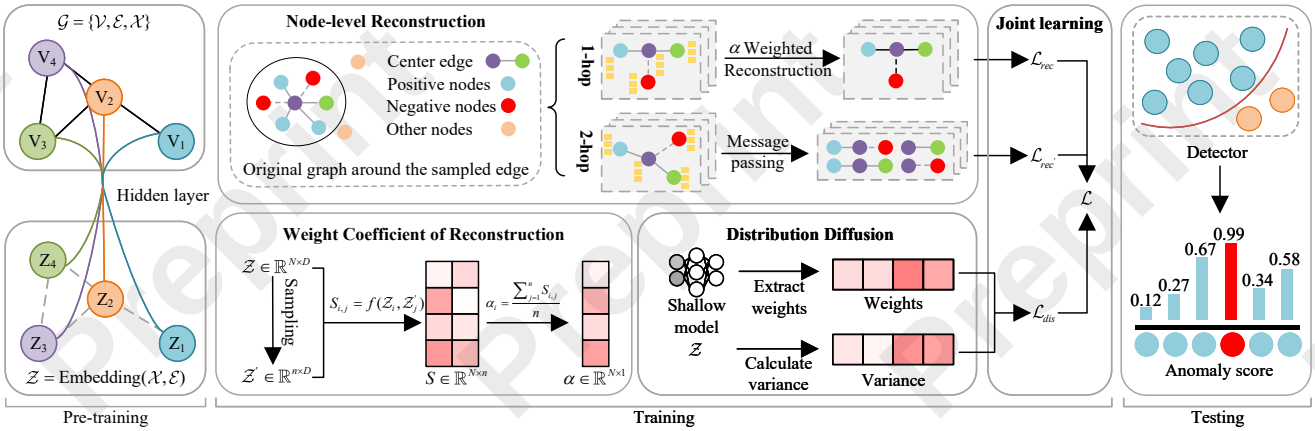


Figure 1: Framework of MEGAD.

where P denotes the prediction result of $\mathcal{F}$. $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents the edge set of $\mathcal{G}$, $\mathcal{E}_{i,j} = 1$ represents there is an edge between v_i and v_j. It is worth noting that we emphasize the use of $\mathcal{E}$ rather than $\mathcal{A}$ in the definition of $\mathcal{G}$ because storing $\mathcal{A}$ or performing calculations based on $\mathcal{A}$ in large-scale graph data results in unacceptable resource consumption. In the following, unless otherwise specified, our MEGAD uniformly adopts the definition $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{X}\}$, which has no access to $\mathcal{A}$. Additionally, our objective function can be expressed as:

$$\begin{aligned} &\text{Minimize} \left(\sum_{i=1}^{N} \left(GT_i - P_i \right)^2 \right), \\ &\text{s.t.} \quad \text{Memory} \leq \text{Memory}_{\max}, \end{aligned} \tag{3}$$

where GT_i represents the ground truth of each node in $\mathcal{G}$, Memory represents the memory usage of MEGAD, $\text{Memory}_{\max}$ donates the memory limitation of the hardware.

4 Methodology

MEGAD is an unsupervised attributed graph anomaly detection method comprising three core modules: Pre-training, Training, and Testing, as illustrated in Fig. 1. First, the input graph data is processed to generate low-dimensional embeddings $\mathcal{Z} \in \mathbb{R}^{N \times D}$. Then, weight coefficients are computed to determine reconstruction weights between nodes, providing a foundation for node-level reconstruction. In the reconstruction process, positive sampling and negative sampling are employed for each node, and the reconstruction loss $\mathcal{L}_{rec}$ and $\mathcal{L}_{rec'}$ are calculated. Simultaneously, the embeddings are used to extract node weights and variance distributions via a shallow model, generating node distribution features and calculating the distribution loss $\mathcal{L}_{dis}$. By combining the reconstruction loss and distribution loss, the embedding representations are jointly optimized to make them more suitable for anomaly detection tasks. Finally, the optimized embeddings are used by the detector to generate anomaly scores for each node, enabling precise detection of anomalous nodes in graph data.

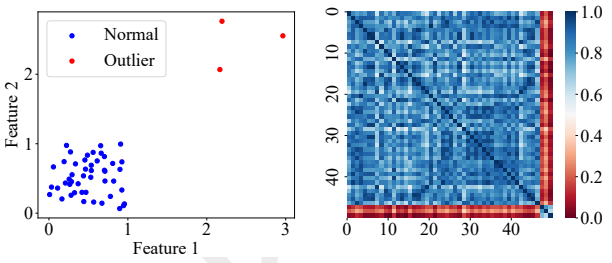
4.1 Pre-training

Graph autoencoders are commonly used for graph anomaly detection tasks. The encoder-decoder framework works by mapping input data into a lower-dimensional latent space, reconstructing the graph topology, and optimizing reconstruction accuracy to ultimately identify anomalous patterns. MEGAD adopts the encoder-decoder framework and designs innovative modules to optimize its performance. Specifically, MEGAD first generates node embeddings to represent the input graph: $\mathcal{Z} = \text{Embedding}(\mathcal{G})$, where $\mathcal{Z} \in \mathbb{R}^{N \times D}$ donates the embedding result of $\mathcal{G}$. The embedding method is flexible and can be implemented using any suitable approach. In this work, we use node2vec [Grover and Leskovec, 2016] for demonstration. The initial embeddings are then optimized through a lightweight training module to enhance their expressiveness. Finally, the optimized embeddings are passed to the detector for anomaly detection.

4.2 Node-level Reconstruction

One-hop reconstruction. The computational process for training an encoder-decoder framework with graph reconstruction typically involves the following steps: First, the model computes the node embeddings $\mathcal{Z}$ through a forward pass, denoted as $\mathcal{Z} = \text{forward}(\mathcal{A}, \mathcal{X})$. These embeddings are then used to reconstruct the adjacency matrix by computing the dot product $\mathcal{A}' = \mathcal{Z} \cdot \mathcal{Z}^T$. The model is optimized by minimizing the reconstruction loss $\mathcal{L} = \sum_i \sum_j |\mathcal{A}_{i,j} - \mathcal{A}'_{i,j}|$, which measures the discrepancy between the original and reconstructed adjacency matrices. However, this approach inherently depends on the adjacency matrix $\mathcal{A}$, resulting in substantial memory consumption and computational overhead during training.

To address these issues, we introduce an edge sampling strategy to optimize the graph reconstruction process. Specifically, we begin by selecting an edge $\mathcal{E}_{l,r}$ connecting nodes v_l and v_r. We then sample a set of neighboring nodes (positive samples) $v_x \in \text{nei}$ of v_l and non-neighboring nodes (negative samples) $v_y \in \text{neg}$ of v_l. Since v_l and v_x are connected in the



(a) Node distribution (b) Similarity matrix

Figure 2: Similarity distribution between outlier points and normal points.

original graph, the reconstructed graph should preserve this edge. Conversely, no edge should be reconstructed between v_l and v_y as they are not neighbors. Based on the above analysis, $\mathcal{L}_{rec}$ can be calculated as follow:

$$\mathcal{L}_{rec} = \sum_{i \in \mathcal{E}.l} \alpha_i \left(\sum_{j \in nei} (1 - \mathcal{E}^*_{i,j})^2 + \sum_{k \in neg} (0 - \mathcal{E}^*_{i,k})^2 \right), \quad (4)$$

where α_i denotes the weight coefficient associated with v_i, which will be explained in the next section. $\mathcal{E}.l$ represents the set of left nodes of all edges. $\mathcal{E}^*_{i,j}$ represents the probability of an edge existing between v_i and v_j in the reconstructed graph, which is calculated as follows:

$$\mathcal{E}^*_{i,j} = \frac{cos(z_i, z_j) + 1}{2}, \quad (5)$$

where $\cos(z_i, z_j)$ is the cosine similarity between v_i and v_j.

Two-hop message passing. To acquire more comprehensive neighborhood information, we aim to gather 2-hop neighborhood details to optimize $\mathcal{Z}$. However, under the premise of discarding the adjacency matrix, obtaining higher-order neighborhood information proves to be quite challenging. To overcome this challenge, we continue the 1-hop approach, enumerating edges (v_l, v_r), positive samples $x \in nei$, and negative samples $y \in neg$. Then, we calculate $\mathcal{L}_p$ to measure the loss of the positive samples:

$$\mathcal{L}_p = -\left\| z_l - z_r \right\|^2 - \sum_{j \in nei} att_{j,l} \left\| z_j - z_r \right\|^2, \quad (6)$$

where $att_{j,l} = softmax_{j \in nei}(-\left\| z_j - z_l \right\|^2)$, $\left\| z_i - z_j \right\|^2 = \sum_k (z_{i,k} - z_{j,k})^2$. Similarly, we use Eq. 7 to measure the loss for negative samples.

$$\mathcal{L}_n = -\sum_{i \in neg} (\left\| z_i - z_l \right\|^2 + \sum_{j \in nei} att_{j,l} \left\| z_i - z_j \right\|^2), \quad (7)$$

Combining Eq. 6 and Eq. 7, we obtain the final loss function:

$$\mathcal{L}_{rec'} = -\log(\sigma(L_p)) - \log(\sigma(-L_n)), \quad (8)$$

where σ represents the sigmoid function.

4.3 Weight Coefficient of Reconstruction

To eliminate noise and interference from outlier nodes in the model, we introduce a weighting coefficient α. Fig. 2 is an example of anomaly detection, which contains 47 normal points and 3 anomaly points. In the figure, Fig. 2(a) shows a two-dimensional distribution, and Fig. 2(b) shows the corresponding similarity matrix. The calculation method for similar matrices is as follows:

$$S_{i,j} = f(\mathcal{Z}_i, \mathcal{Z}'_j) = 2 \cdot (1 - \text{sigmoid}(\text{dis}(v_i, v_j))), \quad (9)$$

$$\text{dis}(v_i, v_j) = \frac{\sum_{k=1}^{D} abs(z_{i,k} - z_{j,k})}{D}, \quad (10)$$

where $S_{i,j}$ donates the similarity between v_i and v_j, the calculation method of $\text{dis}(v_i, v_j)$ is not unique and does not affect the conclusion, this paper adopts the Manhattan distance. Notably, $\mathcal{Z}'_j$ is used here rather than $\mathcal{Z}_j$ because we sampled $\mathcal{Z}$ to simplify the computation.

By analyzing the similarity matrix, it becomes apparent that a point exhibiting high similarity with the majority of other points is likely to be classified as normal. In contrast, if the similarity is relatively low, the point in question may be considered an outlier or noise. Based on this observation, we can deduce the method for calculating the Weight Coefficient of Reconstruction:

$$\alpha_i = \frac{\sum_{j=1}^{N} S_{i,j}}{N} \approx \frac{\sum_{j=1}^{n} S_{i,j}}{n}. \quad (11)$$

Given the large size of N, computing the full similarity matrix becomes computationally prohibitive. To address this issue, we leverage the law of large numbers by sampling n points, allowing for an efficient approximation of α.

4.4 Distribution diffusion

The embedding results cannot achieve optimal performance solely by minimizing the node-level reconstruction loss (i.e., $\mathcal{L}_{rec}$ and $\mathcal{L}_{rec'}$). Embeddings trained exclusively with node-level reconstruction loss typically have low discriminative power, making them unsuitable for practical applications when directly applied to the detector.

To enhance the discriminative power of the embeddings, we designed the following loss function:

$$\mathcal{L}_{dis} = e^{-\sum_{j=1}^{D} \beta_i \cdot \frac{\sum_{i=1}^{N} (z_{i,j} - \overline{z_j})^2}{N}}, \quad (12)$$

where β denotes the weight assigned to each dimension in the embedding. Given that different dimensions of the features exert varying influences on the results, we quantified these impact weights using the following approach:

$$\beta = \mathcal{H}(\mathcal{Z}, P), \quad (13)$$

where $\mathcal{H}$ donates a shallow model. In this paper, we adopted a simple regression model to measure this weight.

4.5 Joint learning and Anomaly Detection

Joint learning. Building on the methods introduced in Sections 4.2 and 4.4, we derive the final loss function:

$$\mathcal{L} = \eta(\mathcal{L}_{rec} + \mathcal{L}_{rec'}) + (1 - \eta)\mathcal{L}_{dis}, \quad (14)$$

where $\mathcal{L}_{rec}$ and $\mathcal{L}_{rec'}$ represent the reconstruction losses, $\mathcal{L}_{dis}$ denotes the distribution loss, and η is a hyperparameter used to balance these two components. By jointly optimizing

module	Time complexity
Weight Coefficient	$\theta(NnD)$
Node-level Reconstruction	$\theta(\mathcal{E}DT\|nei\|\|neg\|)$
Distribution diffusion	$\theta(ND)$
Detector_train	$\theta(t\phi log_v \phi D)$
Detector_predection	$\theta(tNlog_v \phi D)$

Table 1: Time complexity. All uppercase variables have large values, while lowercase variables have small values.

Dataset	Node	Edges	Feat.	Degree	Rate
Weibo	8,405	407,963	400	48.5	10.3%
Facebook	1081	27552	576	25.5	2.4%
Disney	124	335	28	2.7	4.8%
Books	1,418	3,695	21	2.6	2.0%
DGraph	3,700,550	4,300,999	17	1.2	0.4%
Flickr	89,250	933,804	500	10.5	4.9%

Table 2: Summary of dataset information. Here, "Feat." represents the number of features, and "Rate" represents the percentage of anomalies.

these objectives, the model can better capture the latent structural features of the data, thereby improving its representation capability and generalization performance.

Anomaly detection. During the anomaly detection phase, the embedding vectors $\mathcal{Z}$ obtained from training are used as input, and anomalies are identified by the detector:

$$P = \text{Detector}(\mathcal{Z}), \tag{15}$$

where P represents the anomaly score (i.e., detection result). The embedding quality is crucial for the accuracy of anomaly detection results. An appropriate detector can fully leverage the information within the embeddings, improving detection performance. Therefore, selecting the ideal detector is key to optimizing detection outcomes. Our method allows for flexible detector selection based on task requirements. After testing several detectors, such as ECOD, iForest, we found that their accuracy and efficiency are generally comparable. In this paper, we chose LSHiForest as the detector because it strikes a good balance between computational resource consumption and detection accuracy.

4.6 Time complexity analysis

The time consumption of MEGAD primarily stems from the training and prediction modules. Hence, this section focuses on analyzing the time complexity of these two components. The time complexity of calculating the weight coefficient matrix, which involves multiplying an $N \times D$ matrix with a $D \times n$ matrix, is $\theta(NnD)$. During the Node-level Reconstruction phase, the process begins by enumerating each edge. For each edge, we sample $|nei|$ neighboring nodes and $|neg|$ negative sample nodes. The loss is then computed based on these samples. Consequently, the overall time complexity for this phase is $\theta(\mathcal{E}DT|nei||neg|)$. The time complexity associated with the distribution diffusion process is mainly determined by the computation of multiple regressions, which depends solely on the number of nodes (N) and the feature dimensions (D). As such, the time complexity for this stage is $\theta(ND)$. Finally, the testing process consists of two components: training and prediction. The time complexities for these components are $\theta(t\phi \log_v \phi D)$ and $\theta(tN \log_v \phi D)$, respectively [Zhang *et al.*, 2017]. Here, v and ϕ are two key parameters of LSHiforest. Specifically, v represents the branching factor of the Isolation Tree, and ϕ denotes the sampling size.

The MEGAD is specifically designed to address large-scale GAD problems. In this context, N and $\mathcal{E}$ are considered large parameters, while n, $|nei|$ and $|neg|$ are treated as small constants. Therefore, we have:

$$\begin{aligned} &\theta(ND) < \theta(NnD) < \theta(\mathcal{E}DT|nei||neg|), \\ &\theta(t\phi log_v \phi D) < \theta(tNlog_v \phi D). \end{aligned} \tag{16}$$

In summary, the time complexity of training is $\theta(\mathcal{E}DT|nei||neg|)$, while the time complexity of prediction is $\theta(tNlog_v \phi D)$.

5 Experiment

In this section, we conduct extensive experiments on six datasets to answer the following research questions:

- $\mathcal{Q}$**1:** How does our MEGAD perform compared to the SOTA methods in terms of precision?
- $\mathcal{Q}$**2:** Is the memory consumption of MEGAD really low?
- $\mathcal{Q}$**3:** Have all modules of MEGAD functioned as expected?
- $\mathcal{Q}$**4:** How do hyperparameters affect the experimental results?

5.1 Experiment Setting

To ensure the fairness of the experiments, all experiments were conducted using Python 3.9.20 in VScode and were run on a computer equipped with an Intel(R) Core(TM) i7-14700KF 3.40 GHz processor, 32.0 GB of memory, and NVIDIA GeForce RTX 4070 SUPER GPU.

Baselines. We compare our method with several SOTA methods [Liu *et al.*, 2022], including deep learning-based approaches such as GADNR [Roy *et al.*, 2024] and CONAD [Xu *et al.*, 2022b], as well as ensemble-based techniques like LSHiForest [Zhang *et al.*, 2017]. Among the deep learning-based methods, GAAN [Chen *et al.*, 2020] utilizes Generative Adversarial Networks, while AdONE combines Multi-Layer Perceptron (MLP) and AE. Other methods, including GAE [Kipf and Welling, 2022], DOMINANT [Ding *et al.*, 2019], AnomalyDAE [Fan *et al.*, 2020], CONAD [Xu *et al.*, 2022b], and GADNR [Roy *et al.*, 2024], all leverage a combination of GNN and AE.

Datasets. To validate the robustness of our method, we selected six publicly available datasets, including five real-world datasets (Weibo [Zhao *et al.*, 2020], Facebook [Xu *et al.*, 2022a], Disney [Sánchez *et al.*, 2013], Books [Sánchez *et al.*, 2013], and DGraph [Huang *et al.*, 2022]) and one real dataset with artificially injected anomalies (Flickr [Zeng *et*

Algorithm	SCAN	LSHiForest	GCNAE	DOMINANT	AdONE	AnomalyDAE	GAAN	CONAD	GADNR	MEGAD
Year	2007	2017	2016	2019	2020	2020	2020	2022	2024	ours
Graph	✓	×	✓	✓	✓	✓	✓	✓	✓	✓
Deep	×	×	✓	✓	✓	✓	✓	✓	✓	✓
Weibo	63.9	92.2	91.0	85.7	83.0	91.2	91.9	72.7	87.7	**95.0**
Facebook	50.0	57.3	77.1	75.6	**96.1**	69.1	96.0	50.0	44.8	94.6
Disney	50.8	44.8	42.0	47.0	48.2	48.6	48.1	55.4	**76.4**	72.5
Books	49.6	38.1	50.1	50.3	53.7	62.0	54.3	45.6	**65.7**	62.1
DGraph	TLE	46.3	40.8	OOM	OOM	OOM	OOM	OOM	TLE	**58.9**
Flickr	62.3	71.2	64.6	OOM	OOM	OOM	OOM	OOM	73.2	**73.7**

Table 3: AUC-ROC(%) performance of all methods. "TLE" stands for running over 24 hours, and "OOM" stands for out-of-memory errors. Our MEGAD achieves comparable performance to state-of-the-art methods.

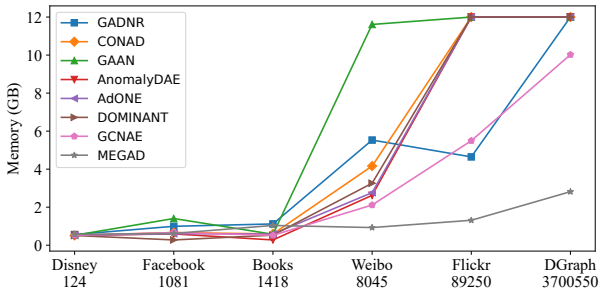


Figure 3: Memory consumption of different datasets.

Combination	Weibo	Facebook	Disney	Books	DGraph	Flickr
MEGAD	**95.0**	**94.6**	**72.5**	**62.1**	**58.9**	**73.7**
Without TR	67.8	73.5	63.1	53.0	44.3	64.8
Without PRE	92.3	63.1	49.1	30.4	47.4	66.5
Without PRE& TR	92.2	57.3	44.8	38.1	46.3	71.2

Table 4: AUC-ROC(%) results for different module combinations. Here, "PRE" represents the pre-training module, and "TR" represents the training module.

al., 2019]), as shown in Table 2. These datasets are publicly available and widely used in the field of anomaly detection. The dataset selection covers different numbers of nodes, sparsity levels, and feature dimensions. It is worth noting that existing works rarely use datasets with more than 40,000 points. In our experiments, we include a large-scale dataset, Flickr (89,250 nodes), and an ultra-large-scale dataset, DGraph (3,700,550 nodes). The inclusion of these datasets allows for a thorough validation of our method across datasets of varying scales.

Metrics. To evaluate the performance of the proposed framework, we use the Area Under Receiver Operating Characteristic Curve (AUC-ROC) as performance evaluation metric, following [Liu *et al.*, 2022; Tang *et al.*, 2023]. The AUC-ROC values range from 0 to 1, with larger values indicating better performance.

5.2 Comparison Results and Discussion

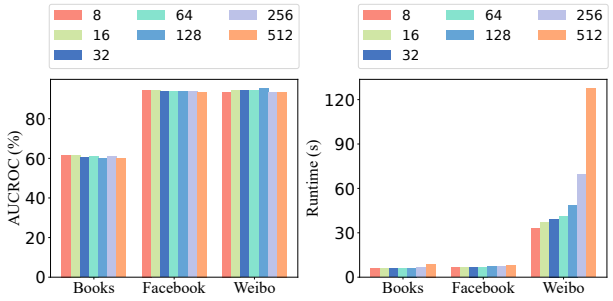
$\mathcal{Q}$1: How does our MEGAD perform compared to the SOTA methods in terms of precision?

Table 3 compares the AUC-ROC performance of various methods across six datasets. Although MEGAD may not achieve optimal results on all datasets, its overall performance is the most outstanding. It is particularly noteworthy that existing methods commonly face Out-of-Memory (OOM) issues when processing large-scale graph data, primarily due to the infeasibility of adjacency matrix-based computations caused by excessive node counts. Although techniques like subgraph segmentation can mitigate this issue, it should be especially noted that MEGAD fundamentally avoids OOM risks through its batched training mechanism of node-level reconstruction, making it more flexible in handling large-scale graph data. In contrast, when employing the full-sample learning strategy on four medium- and small-scale datasets, the GPU memory consumption of the GADNR method remained within controllable limits. However, when applied to large-scale datasets like Flickr and DGraph, the method must switch to mini-batch learning mode to avoid out-of-memory errors. Unfortunately, this compromise solution still leads to training time limit exceeded issues on the DGraph dataset, highlighting its lack of robustness.

$\mathcal{Q}$2: Is the memory consumption of MEGAD really low? As shown in Fig. 3, we compared the maximum memory usage of various deep learning methods across different datasets. The results indicate that the memory usage of most methods increases significantly with larger datasets, eventually leading to "OOM" errors. GCNAE is the only method capable of completing GAD tasks on the DGraph dataset without encountering "OOM". However, GCNAE's memory usage reaches nearly 10GB, which is close to the hardware limit. In contrast, MEGAD demonstrates exceptional stability in memory usage, consistently maintaining around 3GB. These results demonstrate that MEGAD successfully achieves high precision and low memory consumption on large-scale graphs, aligning well with our expectations.

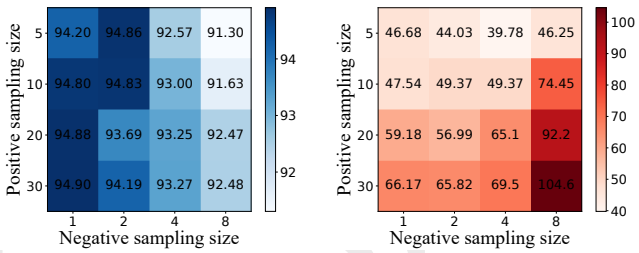
5.3 Ablation Studies and Discussion

$\mathcal{Q}$3: Have all modules of MEGAD functioned as expected? We divided MEGAD into three ablation experiment modules: pre-training, training, and testing. The testing module is indispensable, as without it, no output would be generated. Four distinct experimental setups were conducted, as presented in Table 4. The complete MEGAD demonstrated the highest accuracy, aligning with our expectations. The suboptimal accuracy observed in MEGAD without the PRE & TR modules can be attributed to the absence of graph structure information, which is crucial for performance. Although omitting the PRE module improved accuracy by incorporating the training module, the system still failed to leverage



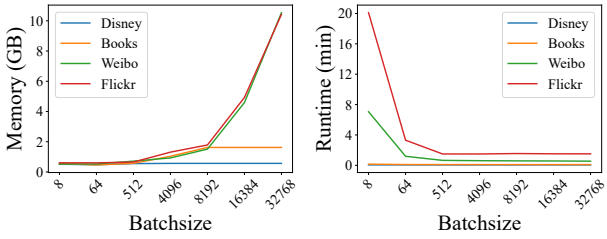
(a) AUC-ROC performance(%) (b) Time consumption(s)

Figure 4: The impact of emb_size on the accuracy and runtime of MEGAD.



(a) AUC-ROC(%) (b) Time consumption(s)

Figure 5: The performance of MEGAD on the Weibo dataset in terms of AUC-ROC and time consumption under different sampling sizes.



(a) Memory consumption(GB) (b) Time consumption(min)

Figure 6: The relationship between time consumption and memory consumption of MEGAD.

the graph structure information. Furthermore, when only the training module was omitted, the lack of a training module for embedding optimization still resulted in suboptimal accuracy.

***Q*4: How do hyperparameters affect the experimental results?** This section discusses the impact of three parameters on the accuracy, efficiency, and memory consumption of MEGAD.

Emb_size. To explore the impact of Emb_size on the efficiency and accuracy of MEGAD, we performed experiments as shown in Fig. 4. Existing studies typically set Emb_size between 32 and 128. Considering that our dataset includes examples with dimensions smaller than 32 (e.g., Books) as well as examples with dimensions larger than 128 (e.g., Weibo), we expanded the range of Emb_size to include [8, 16, 32, 64, 128, 256, 512]. As shown in Fig. 4(a), the Emb_size only marginally affects MEGAD's accuracy. However, as depicted in Fig. 4(b), increasing Emb_size leads to a significant rise in runtime. Unless otherwise specified, all time measurements in this paper represent the time required for a single epoch.

Positive sampling size and Negative sampling size. As shown in Fig. 5, the positive sampling size and negative sampling size have an impact on the accuracy and efficiency of MEGAD. Overall, as positive sampling size and negative sampling size increase, the running time also increases. The accuracy improves with the increase of positive sampling size, but it is not positively correlated with negative sampling size. Specifically, when the negative sampling size equates to 2, the overall accuracy is the best. This may be because too many negative samples may cause the model to deviate from normal, while too few negative samples cannot effectively distinguish. Therefore, we set the negative sampling size to 2. Considering that too many positive samplings will increase the computational cost without significantly improving accuracy, we set the positive sampling size to 10.

Batch size. As shown in Fig. 6, GPU memory usage shows a continuous increase as the batch size grows, while the training time decreases. For the Disney dataset, which includes 670 edges since each undirected edge is counted as two directed edges in the code, GPU memory usage stabilizes when the batch size reaches 4096. Similarly, for the Books dataset, which contains 7390 edges under the same counting method, GPU memory usage stabilizes at a batch size of 8192. This stabilization occurs because, at these points, the sampling size exceeds the number of edges in the dataset, and any further increase in batch size will not result in additional memory demand. In summary, MEGAD achieves efficient memory usage while maintaining strong scalability, enabling deployment across diverse hardware configurations with relaxed memory requirements. As evidenced in Fig. 6, the framework provides configurable optimization strategies that effectively manage the time-space trade-off - allowing users to prioritize either memory efficiency or computational speed based on their specific needs.

6 Conclusion

Existing graph reconstruction-based algorithms achieve good accuracy in Graph Anomaly Detection problems. However, they face issues such as high memory requirements and poor scalability, which make their application to large-scale graph data challenging. In this paper, we propose a memory-efficient framework for large-scale attributed graph anomaly detection. First, graph information is embedded into low-dimensional representations; then, the embeddings are optimized by a lightweight model; finally, an appropriate detector is selected to calculate anomaly scores. Extensive experiments demonstrate that our proposed method achieves accuracy comparable to state-of-the-art methods while being more robust, suitable for large-scale datasets, and requiring less memory. In the future, we plan to introduce large language models into MEGAD to further enhance its performance and applicability.

Acknowledgments

This work was supported in part by Jiangsu Provincial Major Project on Basic Research of Cutting-edge and Leading Technologies, under grant no. BK20232032.

References

[Bandyopadhyay *et al.*, 2020] Sambaran Bandyopadhyay, Lokesh N, Saley Vishal Vivek, and M Narasimha Murty. Outlier resistant unsupervised deep architectures for attributed network embedding. In *Proceedings of the 13th international conference on web search and data mining*, pages 25–33, 2020.

[Chen *et al.*, 2020] Zhenxing Chen, Bo Liu, Meiqing Wang, Peng Dai, Jun Lv, and Liefeng Bo. Generative adversarial attributed network anomaly detection. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1989–1992, 2020.

[Chiang *et al.*, 2019] 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.

[Ding *et al.*, 2019] 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.

[Fan *et al.*, 2020] Haoyi Fan, Fengbin Zhang, and Zuoyong Li. Anomalydae: Dual autoencoder for anomaly detection on attributed networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5685–5689. IEEE, 2020.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[He *et al.*, 2024] Junwei He, Qianqian Xu, Yangbangyan Jiang, Zitai Wang, and Qingming Huang. Ada-gad: Anomaly-denoised autoencoders for graph anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 8481–8489, 2024.

[Huang *et al.*, 2022] 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.

[Kipf and Welling, 2022] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2022.

[Li *et al.*, 2017] Jundong Li, Harsh Dani, Xia Hu, and Huan Liu. Radar: Residual analysis for anomaly detection in attributed networks. In *IJCAI*, volume 17, pages 2152–2158, 2017.

[Liu *et al.*, 2017] Ninghao Liu, Xiao Huang, and Xia Hu. Accelerated local anomaly detection via resolving attributed networks. In *IJCAI*, pages 2337–2343, 2017.

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

[Pazho *et al.*, 2023] Armin Danesh Pazho, Ghazal Alinezhad Noghre, Arnab A Purkayastha, Jagannadh Vempati, Otto Martin, and Hamed Tabkhi. A survey of graph-based deep learning for anomaly detection in distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 36(1):1–20, 2023.

[Roy *et al.*, 2024] Amit Roy, Juan Shu, Jia Li, Carl Yang, Olivier Elshocht, Jeroen Smeets, and Pan Li. Gad-nr: Graph anomaly detection via neighborhood reconstruction. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 576–585, 2024.

[Sánchez *et al.*, 2013] Patricia Iglesias Sánchez, Emmanuel Müller, Fabian Laforet, Fabian Keller, and Klemens Böhm. Statistical selection of congruent subspaces for mining attributed graphs. In *2013 IEEE 13th international conference on data mining*, pages 647–656. IEEE, 2013.

[Tang *et al.*, 2023] 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.

[Tao *et al.*, 2024] Xiang Tao, Liang Wang, Qiang Liu, Shu Wu, and Liang Wang. Semantic evolvement enhanced graph autoencoder for rumor detection. In *Proceedings of the ACM Web Conference 2024*, pages 4150–4159, 2024.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. accepted as poster.

[Wang and Davidson, 2009] Xiang Wang and Ian Davidson. Discovering contexts and contextual outliers using random walks in graphs. In *2009 Ninth IEEE International Conference on Data Mining*, pages 1034–1039. IEEE, 2009.

[Xiang *et al.*, 2023] Haolong Xiang, Xuyun Zhang, Mark Dras, Amin Beheshti, Wanchun Dou, and Xiaolong Xu. Deep optimal isolation forest with genetic algorithm for anomaly detection. In *2023 IEEE International Conference on Data Mining (ICDM)*, pages 678–687. IEEE, 2023.

[Xiang *et al.*, 2024] Haolong Xiang, Xuyun Zhang, Xiaolong Xu, Amin Beheshti, Lianyong Qi, Yujie Hong, and Wanchun Dou. Federated learning-based anomaly detection with isolation forest in the iot-edge continuum. *ACM Transactions on Multimedia Computing, Communications and Applications*, 2024.

[Xu *et al.*, 2007] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833, 2007.

[Xu *et al.*, 2022a] 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.

[Xu *et al.*, 2022b] Zhiming Xu, Xiao Huang, Yue Zhao, Yushun Dong, and Jundong Li. Contrastive attributed network anomaly detection with data augmentation. In *Advances in Knowledge Discovery and Data Mining: 26th Pacific-Asia Conference, PAKDD 2022, Chengdu, China, May 16–19, 2022, Proceedings, Part II*, page 444–457, Berlin, Heidelberg, 2022. Springer-Verlag.

[Yin *et al.*, 2024] Shu Yin, Peican Zhu, Lianwei Wu, Chao Gao, and Zhen Wang. Gamc: an unsupervised method for fake news detection using graph autoencoder with masking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 347–355, 2024.

[Zeng *et al.*, 2019] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[Zhang *et al.*, 2017] Xuyun Zhang, Wanchun Dou, Qiang He, Rui Zhou, Christopher Leckie, Ramamohanarao Kotagiri, and Zoran Salcic. Lshiforest: A generic framework for fast tree isolation based ensemble anomaly analysis. In *2017 IEEE 33rd international conference on data engineering (ICDE)*, pages 983–994. IEEE, 2017.

[Zhang *et al.*, 2024] Yifan Zhang, Haolong Xiang, Xuyun Zhang, Xiaolong Xu, Wei Fan, Qin Zhang, and Lianyong Qi. Eeif: efficient isolated forest with e branches for anomaly detection. In *2024 IEEE International Conference on Data Mining*, 2024.

[Zhao *et al.*, 2020] 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.

[Zhu *et al.*, 2023] Yifan Zhu, Fangpeng Cong, Dan Zhang, Wenwen Gong, Qika Lin, Wenzheng Feng, Yuxiao Dong, and Jie Tang. Wingnn: dynamic graph neural networks with random gradient aggregation window. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pages 3650–3662, 2023.