Rethinking and Scaling Up Graph Contrastive Learning: An Extremely Efficient Approach with Group Discrimination

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph contrastive learning (GCL) alleviates the heavy reliance on label information for graph representation learning (GRL) via self-supervised learning schemes. The core idea is to learn by maximising mutual information for similar instances, which requires similarity computation between two node instances. However, this operation can be computationally expensive. For example, the time complexity of two commonly adopted contrastive loss functions (i.e., InfoNCE and JSD estimator) for a node are $O(ND)$ and $O(D)$, respectively, where N is the number of nodes, and D is the embedding dimension. Additionally, GCL normally requires a large number of training epochs to be well-trained on large-scale datasets. Inspired by an observation of a technical defect (i.e., inappropriate usage of Sigmoid function) commonly used in two representative GCL works, DGI and MVGRL, we revisit GCL and introduce a new learning paradigm for self-supervised GRL, namely, Group Discrimination (GD), and propose a novel GD-based method called Graph Group Discrimination (GGD). Instead of similarity computation, GGD directly discriminates two groups of node samples with a very simple binary cross-entropy loss. In addition, GGD requires much fewer training epochs to obtain competitive performance compared with GCL methods on large-scale datasets. These two advantages endow GGD with very efficient property. Extensive experiments show that GGD outperforms state-of-the-art self-supervised methods on *eight* datasets. In particular, GGD can be trained in 0.18 seconds (6.44 seconds including data preprocessing) on ogbn-arxiv, which is **orders of magnitude (10,000+) faster than GCL baselines** while consuming much less memory. Trained with 9 hours on ogbn-papers100M with billion edges, GGD outperforms its GCL counterparts in both accuracy and efficiency.

1 Introduction

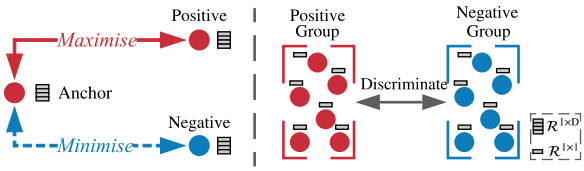
Graph Neural Networks (GNNs) have been widely-adopted in learning representations for graph-structured data. With message-passing to utilise the topology of a graph, GNNs can learn effective low-dimensional node embeddings, which can be used for a variety of downstream tasks such as node classification [1], link prediction [2], and graph classification [3]. GNNs have been further applied in diverse domains, e.g., social networks [4], recommendation system [4], and anomaly detection [5, 6].

However, many GNNs adopt a supervised learning manner to train models with label information, which is expensive and labour-intensive to collect in real-world. To address this issue, a few studies (e.g., DGI [7], MVGRL [3], GMI [8], GRACE [9], and Subg-Con [10]) borrow the idea of contrastive learning from computer vision (CV), and introduce Graph contrastive learning (GCL) methods for self-supervised GRL. The core idea of these methods is to maximise the mutual information

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

(MI) between an anchor node and its positive counterparts, sharing similar semantic information while doing the opposite for negative counterparts as shown in Figure 1(a). Nonetheless, such a scheme relies on expensive similarity calculation in contrastive loss computation. Taking two commonly-adopted contrastive loss functions, InfoNCE and JSD estimator as examples, the time complexity of loss computation for a node is $O(ND)$ and $O(D)$, respectively. Here, N represents the number of nodes, and D is the embedding dimension. Additionally, GCL normally requires a large number of training epochs to be well-trained on large-scale datasets. Thus, when the size of the dataset is large, these methods require a significant amount of time and resources to be well-trained.

Though a few GCL works attempt to alleviate the training load of InfoNCE with specially designed schemes, e.g., BGRL [12] and GBT [13], their proposed contrastive learning losses still have high complexity in training. Inspired by BYOL [14], BGRL [12] adopts a bootstrapping scheme and remove negative node pairs. It only contrasts a node from the online network (i.e., updated with gradient) to its corresponding embedding from the target network (i.e., updated momentumly with stop gradient). This contrastiveness requires $O(D)$ for a node. Based on Barlow-Twins [11], GBT [13] utilises a cross-correlation-based loss to get rid of negative samples. However, it still requires $O(ND)$ for the loss computation of an embedding dimension.



(a)Graph Contrastive Learning (b) Group Discrimination

Figure 1: The left subfigure shows the GCL learning scheme. Red line indicates MI maximisation between two nodes, each of which $\in \mathcal{R}^{1\times D}$, while blue line indicates the opposite operation. The right subfigure presents Group Discrimination. It discriminates positive and negative node samples.

To boost training efficiency of self-supervised GRL, inspired by an observation of a technical defect (i.e., inappropriate application of Sigmoid function) in two representative GCL studies, we introduce a novel learning paradigm, namely, Group Discrimination (GD). Remarkably, with GD, the time complexity of loss computation for a node is only $O(1)$, endowing the scheme with the extremely efficient property. Instead of similarity computation, GD directly discriminates a group of positive nodes from a group of negative nodes, as shown in Figure 1(b). Specifically, GD defines node samples generated with original graph as the positive group, while node samples obtained with corrupted topology are regarded as the negative group. Then, GD trains the model by classifying these node samples into the correct group with a very simple binary cross-entropy loss. By doing so, the model can extract valuable self-supervised signals from learning the edge distribution of a graph. Compared with GCL, GD enjoys numerous merits including extremely fast training, fast convergence (e.g., 1 epoch to be well-trained on large-scale datasets), and high scalability while achieving SOTA or on par performance with existing GCL approaches.

Table 1: Training time in seconds comparison between GGD and GBT [11] (i.e., the most efficient GCL baseline as shown in section 5.1) on ogbn-arxiv. Number in brackets means the hidden size. 'Pre', 'Tr' and 'Epo' indicate preprocessing time, training time per epoch, and the number of epochs for training GNNs. 'Total(E)' and 'Total(T)' are total end-to-end training time (i.e., including preprocessing), which equals to $(Pre+Epo\times Tr)$ and total training time, which is $(Epo \times Tr)$. 'Imp(E)' and 'Imp(T)' indicate how many times GGD improve on 'Total(E)' and 'Total(T)'. 'Acc' is averaged accuracy result on test set over five runs.

Method	Pre	Tr	Epo	Total(E)	Imp(E)	Total(T)	Imp(T)	Acc
GBT(256)	5.52	6.47	300	1,946.52	-	1,941.00	-	70.1
GGD(256)	6.26	0.18	1	6.44	**302.25** ×	0.18	**10,783.33**×	70.3
GGD(1,500)	6.26	0.95	1	7.21	**269.96**×	0.95	**2,043.16**×	71.6

Using GD as backbone, we design a new self-supervised GRL model with the Siamese structure called Graph Group Discrimination (GGD). Firstly, we can optionally augment a given graph with augmentation techniques, e.g., feature and edge dropout. Then, the augmented graph is fed into a GNN encoder and a projector to obtain embeddings for the positive group. After that, the augmented feature is corrupted with node shuffling (i.e., disarranging the order of nodes in the feature matrix) to disrupt the topology of a graph and input to the same network for obtaining embeddings of the opposing group. Finally, the model is trained by discriminating the these two groups of node samples. The contributions of this paper are three-fold: 1) We introduce a novel and efficient self-supervised GRL paradigm, namely, Group Discrimination (GD). Notably, with GD, the time complexity of

loss computation for a node is only $O(1)$. 2) Based on GD, we propose a new self-supervised GRL model, GGD, which is fast in training and convergence, and possess high scalability. 3) We conduct extensive experiments on eight datasets, including an extremely large dataset, ogbn-papers100M with billion edges. The experiment results show that our proposed method **reaches state-of-the-art performance while consuming much less time and memory than baselines, e.g., 10783 $\times$ faster** than the most efficient GCL baseline with its best selected epochs number [13], as shown in Table 1.

2 Rethinking Representative GCL Methods

In this section, we analyse a technical defect observed in two representative GCL methods, DGI [7] and MVGRL [3]. Based on the technical defect, we show the learning method behind these two approaches is not contributed to Contrastive Learning, but a new paradigm, Group Discrimination. Finally, from the analysis, we provide the definition of this new concept.

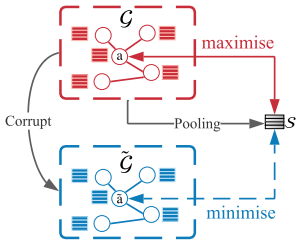


Figure 2: The architecture of DGI. Cubes indicate node embeddings. Red and blue lines represent MI maximisation and minimisation, respectively. $\mathcal{G}$ and $\tilde{\mathcal{G}}$ denote the original graph and the corrupted graph. s is the summary vector.

2.1 Rethinking GCL Methods

DGI [7] is the first work introducing Contrastive Learning into GRL. However, due to a technical defect observed in their official open-source code, we found it is essentially not working as the authors thought (i.e., learning via MI interaction). The backbone that truly makes it work is a new paradigm, Group Discrimination.

Constant Summary Vector. As shown in Figure 2, the original idea of DGI is to maximise the MI (i.e., the red line) between a node a and the summary vector s, which is obtained by averaging all node embeddings in a graph $\mathcal{G}$. Also, to regularise the model training, DGI corrupts $\mathcal{G}$ by shuffling the node order of the input feature matrix to get $\tilde{\mathcal{G}}$. Then, generated embeddings of $\tilde{\mathcal{G}}$ serve as negative samples, which are pulled apart from the summary vector s via MI minimisation.

Nonetheless, in the implementation of DGI, a Sigmoid function is inappropriately applied on the summary vector generated from a GNN whose weight is initialised with Xavier initialisation. As a result, elements in the summary vector are very close to the same value. We have validated this finding on three datasets, Cora, CiteSeer and PubMed. The experiment result is shown in Table 2, which shows that summary vectors in all datasets are approximately a **constant vector** ϵI, where ϵ is a scalar and $I \in \mathcal{R}^D$ is an all-ones vector (i.e., ϵ=0.50 with ReLU/LReLU/PReLU and ϵ=0.62 with Sigmoid as non-linear activation in these datasets).

Table 2: Summary vector statistics on three datasets with different activation functions including ReLU, LeakyReLU (i.e., LReLU shown below), PReLU, and Sigmoid.

Activation	Statistics	Cora	CiteSeer	PubMed
ReLU/LReLU/PReLU	Mean	0.50	0.50	0.50
	Std	1.3e-03	1.0e-04	4.0e-04
	Range	1.4e-03	8.0e-04	1.5e-03
Sigmoid	Mean	0.62	0.62	0.62
	Std	5.4e-05	2.9e-05	6.6e-05
	Range	3.6e-03	3.0e-03	3.2e-03

To theoretically explain this phenomenon, we present the theorem below:

Theorem 1 *Given $\mathcal{G} = (\boldsymbol{X} \in \mathcal{R}^{N\times D}, \boldsymbol{A} \in \mathcal{R}^{N\times N})$, and a GCN encoder $g()$ initialised with Xavier initialisation, we can obtain its embedding $\boldsymbol{H} = g(\mathcal{G})$. Then, when D increases, the value in $\boldsymbol{H}$ converges to 0.5 with ReLU/LReLU/PReLU or 0.62 with Sigmoid as non-linear activation for $g()$.*

Table 3: The experiment result on three datasets with changing value from 0 to 1.0 for the summary vector.

Dataset	0	0.2	0.4	0.6	0.8	1.0
Cora	70.3±0.7	82.4±0.2	82.3±0.3	82.5±0.4	82.3±0.3	82.5±0.1
CiteSeer	61.8±0.8	71.7±0.6	71.9±0.7	71.6±0.9	71.7±1.0	71.6±0.8
PubMed	68.3±1.5	77.8±0.5	77.9±0.8	77.7±0.9	77.4±1.1	77.2±0.9

Based on this theorem, we can see if the feature dimension D is large, these summary vectors can lose variance and become a constant vector. The proof for the theorem is presented in Appendix A.1.

To evaluate the effect of ϵ to constant summary vector, we vary the scalar ϵ (from 0 to 1 increment by 0.2) to change the constant summary vector and report the model performance (i.e., averaged accuracy on five runs) in Table 3.

From this table, we can see, except for 0, the model performance is trivially affected by ϵ for constant summary vector. When the summary vector is set to 0, the model performance plummets because node embeddings become all 0 when multiplying with such vector and the model converges to the trivial solution. As the summary vector only has a trivial effect on model training, the hypothesis of DGI [7] on learning via contrastiveness between anchor nodes and the summary instance does not hold, which raises a question to be investigated: *What truly leads to the success of DGI?*

Simplifying DGI. To answer the question, we predigest the loss proposed in DGI by using an all-ones vector as the summary vector s (i.e., setting $s = \epsilon I = I$) and simplifying the discriminator $\mathcal{D}$ (i.e., removing the learnable weight vector). Then, we rewrite the loss to the following form:

$$\begin{aligned}\mathcal{L}_{DGI} &= \frac{1}{2N}(\sum_{i=1}^{N} \log \mathcal{D}(z_i, s) + \log(1 - \mathcal{D}(\tilde{z}_i, s))),\\ &= \frac{1}{2N}(\sum_{i=1}^{N} \log(z_i \cdot s) + \log(1 - \tilde{z}_i \cdot s))),\\ &= \frac{1}{2N}(\sum_{i=1}^{N} \log(sum(z_i)) + \log(1 - sum(\tilde{z}_i))),\end{aligned} \tag{1}$$

where $\cdot$ is the vector multiplication operation, N is the number of nodes in a graph, $z_i \in \mathcal{R}^{1\times D}$ and $\tilde{z}_i \in \mathcal{R}^{1\times D}$ are the original and corrupted embedding for node i, $sum()$ is the summation function, and $\mathcal{D}$ is a discriminator for bilinear transformation, which can be formulated as follows:

$$\mathcal{D}(z_i, s) = z_i \cdot w \cdot s, \tag{2}$$

where w is a learnable weight vector. Specifically, as shown in Equation 2, by removing the weight vector w, z_i is directly multiplied with s. As s is a vector containing only one, the multiplication of z_i and s is equivalent to summing z_i itself directly. From this form, we can see that the multiplication of z_i and the summary vector only serves as an aggregation function (i.e., summation aggregation) to aggregate z_i. To explore the effect of other aggregation functions, we replace the summation function in Equation 1 with other aggregation methods such as mean-, minimum-, and maximum- pooling, and present the experiment result in Appendix A.3.

Table 4: Comparison of the original DGI and DGI_{BCE} in terms of accuracy (averaged on five runs), memory efficiency (in MB) and training time (in seconds). Number after | shows how many times have DGI_{BCE} improved on top of DGI.

Experiment	Method	Cora	CiteSeer	PubMed
Accuracy	DGI	81.7±0.6	71.5±0.7	77.3±0.6
	DGI_{BCE}	82.5±0.3	71.7±0.6	77.7±0.5
Memory	DGI	4189MB	8199MB	11471MB
	DGI_{BCE}	1475MB|64.8%	1587MB|80.6%	1629MB|85.8%
Time	DGI	0.085s	0.134s	0.158s
	DGI_{BCE}	0.010s|8.5×	0.021s|6.4×	0.015s|10.5×

Based on Equation 1, we can rewrite it to a very simple binary cross entropy loss if we also include corrupted nodes as data samples and setting $h_i = agg(z_i)$, where $agg()$ stands for aggregation:

$$\mathcal{L}_{BCE} = \frac{1}{2N}(\sum_{i=1}^{2N} y_i \log h_i + (1 - y_i)\log(1 - h_i)), \tag{3}$$

where $y_i \in \mathcal{R}^{1\times 1}$ means the indicator for node i (i.e., if node i is corrupted, y_i is 0, otherwise it is 1), and $h_i \in \mathcal{R}^{1\times 1}$ represents the prediction for a node sample i. As we include corrupted nodes as data samples, the size of nodes to be processed is doubled to $2N$ (i.e., the number of corrupted nodes is equal to the number of original nodes). From the equation above, we can easily observe that what DGI truly does is discriminate between a group of nodes generated with correct topology and nodes generated with corrupted topology, as shown in Figure 1. We name this self-supervised learning paradigm "**Group Discrimination**". To validate the effectiveness of this paradigm, we replace the original DGI loss with Equation 3, namely, DGI_{BCE} and compare it with DGI on three datasets in terms of training time, memory efficiency and model performance as shown in Table 4. Here, DGI_{BCE} adopts the same parameter setting as DGI. From this table, we can observe DGI_{BCE} dramatically improves DGI in both memory- and time-efficiency while it slightly enhances the model performance of DGI. This is contributed to the removal of multiplication operations between node pairs, which eases the burden of computation and memory consumption in both forward and backward propagation. In section A.2, we further explore GD and compare it with existing contrastive losses.

Similar to DGI, the same technical defect is observed in MVGRL [3], which makes it become a GD-based method. Extended on DGI, MVGRL [3] incorporates diffusion augmentation to inject additional global information into model training, which enhances the model performance. The detailed analysis for MVGRL is presented in Appendix A.4.

2.2 Definition of Group Discrimination

As mentioned above, Group Discrimination is a self-supervised GRL paradigm, which learns by discriminating different groups of node samples. Specifically, the paradigm assigns different indicators to different groups of node samples. For example, for binary group discrimination, one group is considered as the positive group with class 1 as its indicator, whereas the other group is the negative group, having its indicator assigned as 0. Given a graph $\mathcal{G}$, the positive group usually includes node samples generated with the original graph $\mathcal{G}$ or its augmented views (i.e., similar graph instances of $\mathcal{G}$ created by augmentation). In contrast, the opposing group contains negative samples obtained by corrupting $\mathcal{G}$, e.g., changing its topology structure.

3 Methodology

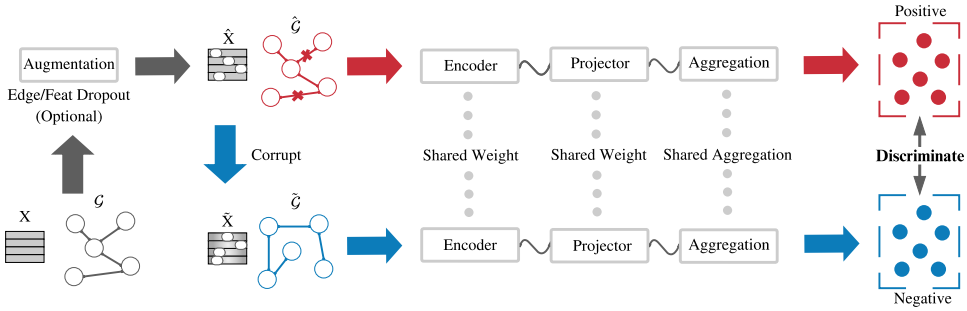


Figure 3: The architecture of GGD. Given a graph $\mathcal{G}$ with a feature matrix $\mathbf{X}$, we can optionally apply augmentation on them to generate $\hat{\mathcal{G}}$ and $\hat{\mathbf{X}}$. Then, we corrupts $\hat{\mathbf{X}}$ and $\hat{\mathcal{G}}$ to obtain $\tilde{\mathbf{X}}$ and $\tilde{\mathcal{G}}$. Taking $\hat{\mathbf{X}}$ and $\hat{\mathcal{G}}$ as input to the encoder and the projector,i.e., multilayer perceptron, positive node samples can be obtained. Similarly, $\tilde{\mathbf{X}}$ and $\tilde{\mathcal{G}}$ are fed to the same encoder and projector to generate negative samples. The generated embeddings are aggregated to get predictions for the group discrimination task. This process will be iteratively conducted until reaching the predefined training epochs.

We first define unsupervised node representation learning and then present the architecture of GGD, which extends DGI_{BCE} with additional augmentation, the projector and embedding reinforcement to reach better model performance. Given a graph $\mathcal{G}$ with attributes $\mathbf{X} \in \mathcal{R}^{N \times D}$, where N is the number of nodes in $\mathcal{G}$, and D is the number of dimensions of $\mathbf{X}$, our aim is to train a GNN encoder without the reliance on labelling information. With the trained encoder, taking $\mathcal{G}$ and $\mathbf{X}$ as input, it can output learned representations $\mathbf{H} \in \mathcal{R}^{N \times \hat{D}}$, where $\hat{D}$ is the predefined output dimension. $\mathbf{H}$ can then be used in many downstream tasks such as node classification.

3.1 Graph Group Discrimination

Based on the proposed self-supervised GRL paradigm, Group Discrimination, we have designed a novel method, namely GGD, to learn node representations using a Siamese network structure and a BCE loss. The architecture of the GGD is presented in Figure 3. The framework mainly consists of four components: augmentation, corruption, a Siamese GNN network, and Group Discrimination.

Augmentation. With a given graph $\mathcal{G}$ and feature matrix $\mathbf{X}$, optionally, we can augment it with augmentation techniques such as edge and feature dropout to create $\hat{\mathcal{G}}$ and $\hat{\mathbf{X}}$. In practice, we follow the augmentation proposed in GraphCL [15]. Specifically, edge dropout removes a predefined fraction of edges, while we use node dropout to mask a predefined proportion of feature dimension, i.e., assigning 0 to replace values in randomly selected dimensions. This step is optional in implementation.

Notably, the motivation of using augmentation in our framework is distinct from contrastive learning methods. In our study, augmentation is used to increase the difficulty of the self-supervised training tasks. With augmentation, $\hat{\mathcal{G}}$ and $\hat{\mathbf{X}}$ change in every training iteration, which forces the model to lessen the dependence on the fixed pattern (i.e., unchanged edge and feature distribution) in a monotonous graph. However, in Contrastive Learning, augmentation creates augmented views sharing similar semantic information for building contrastiveness.

Corruption. $\hat{\mathcal{G}}$ and $\hat{\mathbf{X}}$ are then corrupted to build $\tilde{\mathcal{G}}$ and $\tilde{\mathbf{X}}$ for the generation of node embeddings in the negative group. We adopt the same corruption technique used in DGI [7] and MVGRL [3] (as shown in Figure 5 in Appendix A.2. The corruption technique devastates the topology structure of $\hat{\mathcal{G}}$ by randomly changing the order of nodes in $\hat{\mathbf{X}}$. The corrupted $\tilde{\mathbf{X}}$ and $\tilde{\mathcal{G}}$ can be used for producing node representations with incorrect network connections.

The Siamese GNN. We have designed a Siamese GNN network to output node representations given a graph and its attribute. The Siamese GNN network is made up of two components, which are a GNN encoder and a projector. The backbone GNN encoder is replaceable with a variety of choices of GNNs, e.g., GCN [16] and GAT [17]. In our work, we adopt GCN as the backbone. The projector is a multi-layer perceptron network, whose number of layers can be adjusted. When generating node embeddings of the positive group, the Siamese network takes $\hat{\mathcal{G}}$ and $\hat{\mathbf{X}}$ as input. Using the same encoder and projector, the Siamese network output the negative group with $\tilde{\mathcal{G}}$ and $\tilde{\mathbf{X}}$. These two groups of node embeddings are considered as a collection of data samples with a size of $2N$ for discrimination. Before conducting group discrimination, in the "aggregation" phase, all data samples are aggregated with the same aggregation technique, e.g., sum-, mean-, and linear aggregation.

Group Discrimination. In the group discrimination process, we adopt a very simple binary cross entropy (BCE) loss to discriminate two groups of node samples as shown in Equation 3. In our implementation, y_i is 0 and 1 for node embeddings in negative and positive groups. During model training, the model is optimised by categorising node embeddings in the collection of data samples into their corresponding class correctly. The loss is computed by comparing the prediction of a node i, i.e., a scalar, with its indicator y_i. With the ease of BCE loss computation, the training process of `GGD` is very fast and memory efficient.

3.2 Model Inference

During training, the model is optimised via loss minimisation with Equation 3. The time complexity analysis of `GGD` is provided in Appendix A.5. In the inference phase, we freeze the trained GNN encoder g_θ and obtain node embeddings $\mathbf{H}_\theta$ with the input $\mathcal{G}$.

Inspired by MVGRL [3], which strengthens the output embeddings by including additional global information, we adopt a conceptually similar embedding reinforcement approach. Specifically, they obtain the final embeddings by summing up embeddings from two views: the original view comprising local information and the diffused view with global information. This operation reinforces the final embeddings and leads to model performance improvement. Nonetheless, graph diffusion impairs the scalability of a model [18] and hence cannot be directly applied in our embedding generation. To avoid the diffusion computation, we have come up with a workaround in the virtue of the power of a graph to extract global information. The power of a graph can extend the message passing scope of $\mathbf{H}_\theta$ to n-hop neighbourhood, which encodes global information from distant neighbours. It can be formulated as follows:

$$\mathbf{H}_\theta^{global} = \mathbf{A}^n \cdot \mathbf{H}_\theta, \tag{4}$$

where $\mathbf{H}_\theta^{global}$ is the global embedding, and $\mathbf{A}$ is the adjacency matrix of the graph $\mathcal{G}$. It is notable that this operation can be easily decomposed with the associative property of matrix multiplication and is easy to compute. To show the easiness of such computation, we conduct an experiment showing its time consumption on various datasets in Appendix A.6. Finally, the final embedding can be achieved by $\mathbf{H} = \mathbf{H}_\theta^{global} + \mathbf{H}_\theta$, which can be used for downstream tasks. In our experiment, we conduct node classification tasks. Following the common practice of GCL methods [7, 9, 12, 8, 11], these tasks are performed by using the final embeddings $\mathbf{H}$ to train and test a simple logistic regression classifier.

4 Related Work

Graph Neural Networks (GNNs) are generalised deep neural networks for graph-structured data. GNNs mainly have two categories, spectral-based GNNs and spatial-based GNNs. Spectral GNNs attempt to use eigen-decomposition to obtain the spectral-based representation of graphs, whereas spatial GNNs focus on using spatial neighbours of nodes for message passing. Extending spectral-based methods to the spatial domain, GCN [16] utilises first-order Chebyshev polynomial filters to approximate spectral-based graph convolution. Taking the weight of spatial neighbours in consideration, GAT [17], improves GCN by introducing the attention module in message passing. To decouple message passing from neural networks, SGC [19] simplifies GCN by removing the non-linearity and weight matrices in graph convolution layers. However, these studies cannot handle datasets with limited or no labels. Graph contrastive learning has been recently exploited to address this issue.

Graph Contrastive Learning (GCL) aims to alleviate the reliance on labelling information in model training based on the concept of mutual information (MI). Specifically, GCL approaches maximise MI between instances with similar semantic information, and minimise the MI between dissimilar instances. For example, DGI [7] builds contrastiveness between node embeddings and a summary vector (i.e., a graph level embedding obtained by averaging all node embeddings) with a JSD estimator. To improve DGI, MVGRL [3] and GMI [8] extends the idea of DGI by introducing multi-view contrastiveness with diffusion augmentation, and focusing on a local scope with the first-order neighbourhood, respectively. Adopting InfoNCE loss, GRACE [9] applies augmentation techniques to create two augmented views and inject contrastiveness between them. Though these GCL methods have successfully outperformed some supervised baselines in benchmark datasets, these methods suffer from significant limitations, including time-consuming loss computation, a large number of training epochs and poor scalability. For example, InfoNCE and JSD estimator require $O(ND)$ and $O(D)$ in loss computation of a node, respectively. In contrast, our method `GGD` reduces the time complexity of loss computation for a node to only $O(1)$ and converge fast.

Scalable GNNs. Efficiency is a bottleneck for most existing GNNs to handle large graphs. To address this challenge, there are mainly three categories of approaches: layer-wise sampling (e.g., GraphSage [20]), graph sampling methods such as Cluster-GCN [21] and GraphSAINT [22], and linear models, e.g., SGC [19] and PPRGo [23]. GraphSage [20] introduces a neighbour-sampling approach, which creates fixed-size subgraphs for each node. Underpinned by graph sampling, Cluster-GCN [21] decomposes a large-scale graph into multiple subgraphs based on clustering, while GraphSAINT [22] utilises light-weight graph samplers along with a normalisation technique for biases elimination in mini-batches. Linear models, SGC [19] and PPRGo [23], decouple graph convolution from embedding transformation (i.e., matrix multiplication with weight matrices), and leverage Personalised PageRank to encode multi-hop neighbourhood, respectively. However, all these methods only focus on supervised learning on graphs. For unsupervised/self-supervised learning settings where no labelled supervision signal is available, these frameworks are not applicable. The closest works to ours to handle large scale graph datasets under self-supervised settings are BGRL [12] and GBT [11]. They try to reduce the time complexity of the contrastive losses by removing negative samples. However, BGRL [12] still requires $O(D)$ to calculate the loss for a node, while GBT [11] requires $O(ND)$ for a embedding dimension. In comparison, `GGD` only requires $O(1)$.

5 Experiments

We evaluate the effectiveness of our model using eight benchmark datasets of different sizes. These datasets include five small- and medium-scale datasets: Cora, CiteSeer, PubMed [24], Amazon Computers, and Amazon Photos [25], as well as large-scale datasets ogbn-arxiv, ogbn-products and ogbn-papers100M. Notably, ogbn-papers100M is the largest dataset provided by Open Graph Benchmark[26] for node property prediction tasks. It has over 110 million nodes and 1 billion edges. The statistics of these datasets are summarised in Appendix A.7. To

Table 5: Model performance of node classification on 5 datasets. **X, A** and **Y** represent feature, adjacency matrix, and labels. Best performance for each dataset is in **bold**. Comp and Photo refer to Amazon Computers and Amazon Photos.

Data	Method	Cora	CiteSeer	PubMed	Comp	Photo
X, A, Y	GCN	81.5	70.3	79.0	76.3±0.5	87.3±1.0
X, A, Y	GAT	83.0±0.7	72.5±0.7	79.0±0.3	79.3±1.1	86.2±1.5
X, A, Y	SGC	81.0±0.0	71.9±0.1	78.9±0.0	74.4±0.1	86.4±0.0
X, A, Y	CG3	83.4±0.7	**73.6**±0.8	80.2±0.8	79.9±0.6	89.4±0.5
X, A	DGI	81.7±0.6	71.5±0.7	77.3±0.6	75.9±0.6	83.1±0.5
X, A	GMI	82.7±0.2	73.0±0.3	80.1±0.2	76.8±0.1	85.1±0.1
X, A	MVGRL	82.9±0.7	72.6±0.7	79.4±0.3	79.0±0.6	87.3±0.3
X, A	GRACE	80.0±0.4	71.7±0.6	79.5±1.1	71.8±0.4	81.8±1.0
X, A	GraphCL	82.5±0.2	72.8±0.3	77.5±0.2	OOM	79.5±0.4
X, A	BGRL	80.5±1.0	71.0±1.2	79.5±0.6	89.2±0.9	91.2±0.8
X, A	GBT	81.0±0.5	70.8±0.2	79.0±0.1	88.5±1.0	91.1±0.7
X, A	`GGD`	**83.9**±0.4	73.0±0.6	**81.3**±0.8	**90.1**±0.9	**92.5**±0.6

ensure reproducibility, **the detailed experiment settings and computing infrastructure are summarised in Appendix A.8.**

5.1 Evaluating on Small- and Medium-scale Datasets

We compare GGD with ten baselines including four supervised GNNs (i.e., GCN [16], GAT [17], SGC [19], and CG3 [27]) and six GCL methods (i.e., DGI [7], GMI [8], MVGRL [3], GRACE [9], BGRL [12] and GBT [11]) on five small- and medium scale benchmark datasets. In the experiment, we follow the same data splits as [28] for Cora, CiteSeer and PubMed. For Amazon Computers and Photos, we adopt the semi-supervised experiment setting, which includes 30 randomly-selected nodes per class in the training and validation set. The remaining nodes are used as the test set. The model performance is measured using the averaged classification accuracy with five results along with standard deviations and reported in Table 5.

Table 6: Comparison of training time per epoch in seconds between six GCL-based methods and GGD on five datasets. Improve means how many times are GGD faster than baselines. '-' means the improvement range.

Method	Cora	CiteSeer	PubMed	Comp	Photo
DGI	0.085	0.134	0.158	0.171	0.059
GMI	0.394	0.497	2.285	1.297	0.637
MVGRL	0.123	0.171	0.488	0.663	0.468
GRACE	0.056	0.092	0.893	0.546	0.203
GraphCL	0.073	0.085	0.123	OOM	0.188
BGRL	0.085	0.094	0.147	0.337	0.273
GBT	0.073	0.072	0.103	0.492	0.173
GGD	0.010	0.021	0.015	0.016	0.009
Improve	7.3-39.4×	3.4-23.7×	6.9-152.3×	10.7-15.3×	19.2-70.8×

Accuracy. From Table 5, we can observe that GGD generally outperforms all baselines in all datasets. The only exception is on CiteSeer dataset, where the semi-supervised method, CG3[27], slightly outperforms GGD, which still provides the 2nd best performance. In this experiment, we use the officially released code of GraphCL [15], BGRL [12] and GBT [11] to reproduce the result, while the other results are sourced from previous studies [27, 1].

Table 7: Comparison of memory consumption in MBs of six GCL baselines and GGD on five datasets.

Method	Cora	CiteSeer	PubMed	Comp	Photo
DGI	4,189	8,199	11,471	7,991	4,946
GMI	4,527	5,467	14,697	10,655	5,219
MVGRL	5,381	5,429	6,619	6,645	6,645
GRACE	1,913	2,043	12,597	8,129	4,881
GraphCL	4,163	8,249	11,555	OOM	9,083
BGRL	1,627	1,749	2,299	5,069	3,303
GBT	1,651	1,799	2,461	5,037	2,641
GGD	1,475	1,587	1,629	1,787	1,637
Improve	10.7-72.6%	11.8-80.6%	27.2-85.8%	64.5-83.2%	38.0-75.4%

Efficiency and Memory Consumption. GGD is substantially more efficient than other self-supervised baselines in time and memory consumption as shown in Table 6 and Table 7. Remarkably, GGD is **19.2 times faster** in Amazon Photos for training time per epoch, and consumes **64.5% less memory** in Amazon Computers for memory consumption than the most efficient baseline (i.e., GBT [11]). The dramatic boost of time and memory efficiency of GGD is contributed to the exclusion of similarity computation, which enables model training without multiplication of node embeddings.

5.2 Evaluating on Large-scale datasets

To evaluate the scalability of GGD, we choose three large-scale datasets from Open Graph Benchmark [26], which are ogbn-arxiv, ogbn-products, and ogbn-papers100M. ogbn-papers100M is the most challenging large-scale graph available in Open Graph Benchmark for node property prediction with over 1 billion edges and 110 million nodes. Extending to extremely large graphs (i.e., ogbn-products and ogbn-papers100M), we adopt a Neighbourhood Sampling strategy, which is described in Appendix A.8.

ogbn-arxiv & ogbn-products. For ogbn-arxiv, we compare GGD against four self-supervised baselines (i.e., DGI [7], GRACE [9], BGRL [12]and GBT [11]), whereas BGRL [12] and GBT [11] are selected to be compared for ogbn-products. In addition, we include the performance of MLP, Node2vec [29], and supervised GCN [16] sourced from [26] in Table 8 and Table 9. For memory and training time comparison, we only compare GGD with the two most efficient baselines (i.e., BGRL and GBT according to Tables 6 and 7). In ogbn-arxiv, we

Table 8: Node classification result and efficiency comparison on ogbn-arxiv. 'epo' means epoch. 'Time' means training time per epoch (in seconds). 'Total' is total training time (Number of epochs × 'Time'). OOM indicates out-of-memory on Nvidia A40 (48GB).

Method	Valid	Test	Memory	Time	Total
Supervised GCN	73.0±0.2	71.7±0.3	-	-	-
MLP	57.7±0.4	55.5±0.2	-	-	-
Node2vec	71.3±0.1	70.1±0.1	-	-	-
DGI	71.3±0.1	70.3±0.2	-	-	-
GRACE(10k epos)	72.6±0.2	71.5±0.1	-	-	-
BGRL(10k epos)	72.5±0.1	71.6±0.1	OOM (Full-graph)	/	/
GBT(300 epos)	71.0±0.1	70.1±0.2	14,959MB	6.47	1,941.00
GGD(1 epo)	72.7±0.3	71.6±0.5	4,513MB|69.8%	0.18	0.18|10,783×

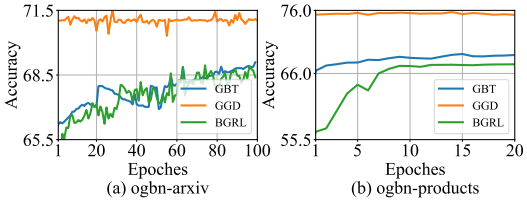
reproduce BGRL [12] and found it fails to process ogbn-arxiv in full batch. Thus, we only compare GGD and GBT in this dataset, which can successfully train in full-graph processing mode.

From Table 8 and Table 9, we can see GGD remarkably achieves the state-of-the-art performance using only one epoch to train. As a result, GGD is 10,783 times faster than the most efficient baseline, i.e., GBT [11], on total training time to reach the desirable performance in ogbn-arxiv. Please be noted that the number of epoches in our experiment is consistent with the optimal choice of this hyperparameter specified in GBT [11]. For ogbn-products, we are 381 $\times$ faster than GBT [11] on total training time. Notably, our performance is significantly higher than GCL baselines using 100 epochs (i.e., 6% and 5.2% improvement on GBT [11] in validation and test set) with only one epoch training in this dataset. In addition, we compare the convergence speed among GGD, BGRL [12] and GBT [11] on ogbn-arxiv and ogbn-products, which are shown in Figure 4. For ogbn-arxiv, BGRL [11] is running using batched processing with neighbour sampling. This figure shows the preeminence of our GGD in convergence speed as GGD can be well-trained with only one epoch (i.e., reaching the peak model performance in the first epoch and staying stable with increased epochs). In contrast, the other two baselines require comparatively much more epochs to gradually improve their performance. Compared with GCL baselines, GGD achieves much faster convergence via Group Discrimination. We conjecture this is because GD-based method focuses on the general edge distribution of graphs instead of node-specific information. Inversely, GCL methods can suffer from convergence inefficiency as they may be easily distracted from too-detailed node-specific information during training.

Table 9: Node classification result and efficiency comparison on ogbn-products.

Method	Valid	Test	Memory	Time	Total
Supervised GCN	92.0±0.0	75.6±0.2	-	-	
MLP	75.5±0.0	61.1±0.0	-	-	
Node2vec	70.0±0.0	68.8±0.0	-	-	
BGRL (100 epos)	78.1±2.1	64.0±1.6	29,303MB	53m16s	5,326m40s
GBT (100 epos)	85.0±0.1	70.5±0.4	20,419MB	48m38s	4,863m20s
GGD(1 epo)	90.9±0.5	**75.7**±0.4	4,391MB|78.5%	12m46s	12m46s|381×

ogbn-papers100M. We further compare GGD with BGRL [12] and GBT [11] on ogbn-papers100M, the largest OGB dataset with billion scale edges. Other self-supervised learning algorithms such as DGI [7] and GMI [8] fail to scale to such a large graph with a reasonable batch size (i.e., 256). We only report the performance of each algorithm after a single epoch of training in Table 10 due to the extreme scale of the dataset and the limitation of our available resources. From the table, we can observe that GGD outperforms the two GCL counterparts, BGRL [12] and GBT [11] in both accuracy and efficiency. Specifically, GGD achieves 60.2 in accuracy while BGRL and GBT reach 59.3 and 58.9 in test set, respectively. With only one epoch, these two algorithms may not be well trained. However, training each epoch of these two requires over 1 day and if we would like to train them for 100 epochs, then we will need 100+ GPU days, which is prohibitively impractical for general practitioners. In contrast, GGD can be trained in about 9 hours to achieve a good result for this dataset, which is more appealing in practice.



(a) ogbn-arxiv (b) ogbn-products

Figure 4: Convergence speed comparison among GGD, BGRL[12] and GBT [11]. X-axis means number of epochs, while Y-axis represents the accuracy on test set.

Table 10: Node classification result and efficiency comparison on ogbn-papers100M.

Method	Validation	Test	Memory	Time
Supervised SGC	63.3±0.2	66.5±0.2	-	-
MLP	47.2±0.3	49.6±0.3	-	-
Node2vec	55.6±0.0	58.1±0.0	-	-
BGRL (1 epoch)	59.3±0.5	62.1±0.3	14,057MB	26h28m
GBT (1 epoch)	58.9±0.4	61.5±0.5	13,185MB	24h38m
GGD(1 epoch)	60.2±0.3	63.5±0.5	4,105MB|68.9%	9h15m|2.7×

6 Future Work

In this paper, we have introduced a new self-supervised GRL paradigm: Group Discrimination, which achieves the same level of performance as GCL methods with much less resource consumption (i.e., training time and memory). Some limitations of this work are we still have not explored some questions for GD. For example, can we extend the current binary Group Discrimination scheme (i.e., classifying nodes generated with different topology) to discrimination among multiple groups? Are there any other corruption technique to create a more difficult negative group for discrimination? More importantly, with the extremely efficient property, GD has the potential to be deployed to various real-world applications, e.g., recommendation systems, which have limited labelling information and desire fast computation with limited resources.

References

[1] Ming Jin, Yizhen Zheng, Yuan-Fang Li, Chen Gong, Chuan Zhou, and Shirui Pan. Multi-scale contrastive siamese networks for self-supervised graph representation learning. *IJCAI*, 2021.

[2] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *NIPS*, 31, 2018.

[3] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *ICML*, pages 4116–4126. PMLR, 2020.

[4] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *WWW*, pages 417–426, 2019.

[5] Yixin Liu, Zhao Li, Shirui Pan, Chen Gong, Chuan Zhou, and George Karypis. Anomaly detection on attributed networks via contrastive self-supervised learning. *TNNLS*, 2021.

[6] Yizhen Zheng, Vincent Lee, Zonghan Wu, and Shirui Pan. Heterogeneous graph attention network for small and medium-sized enterprises bankruptcy prediction. In *PAKDD*, pages 140–151. Springer, 2021.

[7] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *ICLR*, 2019.

[8] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. Graph representation learning via graphical mutual information maximization. In *WWW*, pages 259–270, 2020.

[9] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[10] Yizhu Jiao, Yun Xiong, Jiawei Zhang, Yao Zhang, Tianqi Zhang, and Yangyong Zhu. Sub-graph contrast for scalable self-supervised graph representation learning. In *ICDM*, pages 222–231. IEEE, 2020.

[11] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *ICML*, pages 12310–12320, 2021.

[12] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko. Bootstrapped representation learning on graphs. *ICLR2021*, 2021.

[13] Piotr Bielak, Tomasz Kajdanowicz, and Nitesh V Chawla. Graph barlow twins: A self-supervised representation learning framework for graphs. *arXiv preprint arXiv:2106.02466*, 2021.

[14] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *NIPS*, 2020.

[15] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *NIPS*, 33:5812–5823, 2020.

[16] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

[17] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR*, 2018.

[18] Yizhen Zheng, Ming Jin, Shirui Pan, Yuan-Fang Li, Hao Peng, Ming Li, and Zhao Li. Towards graph self-supervised learning with contrastive adjusted zooming. *arXiv preprint arXiv:2111.10698*, 2021.

[19] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, pages 6861–6871. PMLR, 2019.

[20] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1025–1035, 2017.

[21] 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 *KDD*, pages 257–266, 2019.

[22] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *ICLR*, 2019.

[23] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *KDD*, pages 2464–2473, 2020.

[24] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[25] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[26] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *NIPS*, 2020.

[27] Sheng Wan, Shirui Pan, Jian Yang, and Chen Gong. Contrastive and generative graph convolutional networks for graph-based semi-supervised learning. *AAAI*, 2020.

[28] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, pages 40–48. PMLR, 2016.

[29] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.

[30] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Abstract and section 1.
   (b) Did you describe the limitations of your work? [Yes] See section 6
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Please refer to section 2 and Appendix A.1
   (b) Did you include complete proofs of all theoretical results? [Yes] Please refer to section 2 and Appendix A.1
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Appendix A.8 and the source code in supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See section 5 and Appendix A.8.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See section 5 and Appendix A.8.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See section 5 and Appendix A.8.
   (b) Did you mention the license of the assets? [Yes] See section 5 and Appendix A.8.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] We do not use any new datasets in this paper.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] The datasets used in this paper are publicly available.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix A

A.1 Proof of Theorem 1

Proof. To prove Theorem 1, given a graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$, where $\mathbf{X} \in \mathcal{R}^{N \times D}$ and a GNN encoder $g()$, for simplicity, we consider $g()$ as a one-layer GCN and conduct the following normalisation to $\mathbf{X}$ to make its value to be within the range $[0, 1]$:

$$z_{ij} = \frac{x_{ij} - min(x_j)}{max(x_j) - min(x_j)} \tag{5}$$

where $z_{ij} \in \mathbf{Z}$ is the normalised $x_{ij} \in \mathbf{X}$, i denotes the row-entry of the matrix, j indicates the column-entry of the matrix, $min()$ is the function to get the minimum value, and $max()$ is the function for getting the maximum value.

Then, we input $\mathbf{Z}$ to g whose weight matrix $\mathbf{W}$ is initialised with Xavier initialisation:

$$\mathbf{H} = \sigma(\hat{\mathbf{D}}^{\frac{1}{2}} \hat{\mathbf{A}} \hat{\mathbf{D}}^{\frac{1}{2}} \mathbf{Z}\mathbf{W}), \tag{6}$$

where $\mathbf{H}$ is the output embedding, σ is non-linear activation function, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\hat{\mathbf{D}}$ is the degree matrix for $\hat{\mathbf{A}}$, $\mathbf{W}$ is the learnable weight matrix. As multiplying the normalised adjacency matrix will not change the output data range, the output of $\hat{\mathbf{D}}^{\frac{1}{2}} \hat{\mathbf{A}} \hat{\mathbf{D}}^{\frac{1}{2}} \mathbf{Z}$ is still within the range $[0, 1]$. Then, as the element in Xavier initialised $\mathbf{W}$ is in between $\frac{-1}{\sqrt{D}}$ and $\frac{1}{\sqrt{D}}$, we can easily derive that the output data range of the matrix multiplication of $\hat{\mathbf{D}}^{\frac{1}{2}} \hat{\mathbf{A}} \hat{\mathbf{D}}^{\frac{1}{2}} \mathbf{Z}$ and $\mathbf{W}$ still stay in the same range $[\frac{-1}{\sqrt{D}}, \frac{1}{\sqrt{D}}]$ as $\mathbf{W}$.

After that, we need to apply σ to the multiplication output. Here, we analyse four commonly adopted non-linear activation functions: Sigmoid, ReLu, LeakyReLu and PReLu. We first consider the Sigmoid function $\sigma(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}}}$, as it is a monotonic increasing function, we can derive that the data range of $\mathbf{H} \in [\frac{1}{1+e^{\frac{1}{\sqrt{D}}}}, \frac{1}{1+e^{-\frac{1}{\sqrt{D}}}}]$. Then, we apply the Sigmoid function $S()$ again as DGI[7] did to $\mathbf{H}$ and obtain:

$$S(\mathbf{H}) = \frac{1}{1 + e^{-\mathbf{H}}}. \tag{7}$$

Similarily, as the Sigmoid function is a monotonic increasing function, we can easily derive that the data range for $S(\mathbf{H})$ is in $[\frac{1}{e^{-\frac{1}{e^{\frac{1}{\sqrt{D}}}+1}}+1}, \frac{1}{e^{-\frac{1}{e^{-\frac{1}{\sqrt{D}}}+1}}+1}]$. Then, when $D \to \infty$, for the upper bound

$\frac{1}{e^{-\frac{1}{e^{-\frac{1}{\sqrt{D}}}+1}}+1}$, we can obtain:

$$
\begin{aligned}
\lim_{D\to\infty}\left(\frac{1}{e^{-\frac{1}{e^{-\frac{1}{\sqrt{D}}}+1}}+1}\right) &= \frac{1}{\lim\limits_{D\to\infty}\left(e^{-\frac{1}{e^{-D^{-\frac{1}{2}}}+1}}+1\right)} \\
&= \frac{1}{\exp\left(\lim\limits_{D\to\infty} -\frac{1}{e^{-D^{-\frac{1}{2}}}+1}\right)+1} \\
&= \frac{1}{\exp\left(-\frac{1}{\exp\left(-\lim\limits_{D\to\infty} D^{-\frac{1}{2}}\right)+1}\right)+1} \\
&= \frac{1}{\exp\left(-\frac{1}{\exp\left(-\frac{1}{\sqrt{\lim\limits_{D\to\infty} D}}\right)+1}\right)+1} \\
&= \frac{1}{e^{-\frac{1}{e^{-\frac{1}{\sqrt{\infty}}}+1}}+1} \\
&= \frac{1}{\sqrt{e}^{-1}+1} \\
&\approx 0.62
\end{aligned}
\tag{8}
$$

Also, for the upper bound $\frac{1}{e^{-\frac{1}{e^{\frac{1}{\sqrt{D}}}+1}}+1}$, when $D\to\infty$, we can obtain:

$$
\begin{aligned}
\lim_{D\to\infty}\left(\frac{1}{e^{-\frac{1}{e^{\frac{1}{\sqrt{D}}}+1}}+1}\right) &= \frac{1}{\lim\limits_{D\to\infty}\left(e^{-\frac{1}{e^{D^{-\frac{1}{2}}}+1}}+1\right)} \\
&= \frac{1}{\lim\limits_{D\to\infty} e^{-\frac{1}{e^{D^{-\frac{1}{2}}}+1}}+1} \\
&= \frac{1}{\exp\left(\lim\limits_{D\to\infty} -\frac{1}{e^{D^{-\frac{1}{2}}}+1}\right)+1} \\
&= \frac{1}{\exp\left(-\frac{1}{\exp\left(\lim\limits_{D\to\infty} D^{-\frac{1}{2}}\right)+1}\right)+1} \\
&= \frac{1}{\exp\left(-\frac{1}{\exp\left(\frac{1}{\sqrt{\lim\limits_{D\to\infty} D}}\right)+1}\right)+1} \\
&= \frac{1}{e^{-\frac{1}{e^{\frac{1}{\sqrt{\infty}}}+1}}+1} \\
&= \frac{1}{\sqrt{e}^{-1}+1} \\
&\approx 0.62
\end{aligned}
\tag{9}
$$

Finally, we can easily observe that $\lim_{D\to\infty}(\frac{1}{e^{-\frac{1}{e^{\frac{1}{\sqrt{D}}}+1}}+1}) = \lim_{D\to\infty}(\frac{1}{e^{-\frac{1}{e^{-\frac{1}{\sqrt{D}}}+1}}+1}) \approx 0.62$ and thus prove Theorem 1 when σ is Sigmoid function. For the other three non-linear activation functions ReLu $\sigma(x) = max(0, x)$, Leaky ReLu $\sigma(x) = max(0.01 * x, x)$ and PReLu $\sigma(x) = max(0, x) + a * min(0, x)$, where $a \leq 1$ is a learnable parameter, we can also derive the data range of these functions for $\mathbf{H}$ are $[0, \frac{1}{\sqrt{D}}]$, $[\frac{-0.01}{\sqrt{D}}, \frac{1}{\sqrt{D}}]$ and $[\frac{-a}{\sqrt{D}}, \frac{1}{\sqrt{D}}]$, respectively. Here, we can see these functions share the same upper bound $\frac{1}{\sqrt{D}}$. The only difference for their lower bound is the coefficient $c \in [0, 1]$ for $\frac{-1}{\sqrt{D}}$, i.e., 0 for ReLu, 0.01 for Leaky ReLu and a for PReLu. By inputting $\mathbf{H}$ with these activation functions to the Sigmoid function, we can derive the output data range of the function is $S(\mathbf{H}) \in [\frac{1}{1+e^{\frac{c}{\sqrt{D}}}}, \frac{1}{1+e^{-\frac{1}{\sqrt{D}}}}]$. For the lower bound $\frac{1}{1+e^{\frac{c}{\sqrt{D}}}}$, we can obtain:

$$
\begin{aligned}
\lim_{D\to\infty} \frac{1}{e^{\frac{c}{\sqrt{D}}}+1} &= \frac{1}{\lim\limits_{D\to\infty}\left(e^{cD^{-\frac{1}{2}}}+1\right)} \\
&= \frac{1}{\lim\limits_{D\to\infty} e^{cD^{-\frac{1}{2}}}+1} \\
&= \frac{1}{\exp\left(\lim\limits_{D\to\infty} cD^{-\frac{1}{2}}\right)+1} \\
&= \frac{1}{\exp\left(\frac{c}{\lim\limits_{D\to\infty}\sqrt{D}}\right)+1} \\
&= \frac{1}{\exp\left(\frac{c}{\sqrt{\lim\limits_{D\to\infty} D}}\right)+1} \\
&= \frac{1}{e^{\frac{c}{\sqrt{\infty}}}+1} = 0.5
\end{aligned}
\tag{10}
$$

For the lower bound $\frac{1}{1+e^{-\frac{1}{\sqrt{D}}}}$, we can obtain:

$$
\begin{aligned}
\lim_{D\to\infty} \frac{1}{e^{-\frac{1}{\sqrt{D}}}+1} &= \frac{1}{\lim\limits_{D\to\infty}\left(e^{-D^{-\frac{1}{2}}}+1\right)} \\
&= \frac{1}{\lim\limits_{D\to\infty} e^{-D^{-\frac{1}{2}}}+1} \\
&= \frac{1}{\exp\left(\lim\limits_{D\to\infty} -D^{-\frac{1}{2}}\right)+1} \\
&= \frac{1}{\exp\left(-\lim\limits_{D\to\infty} D^{-\frac{1}{2}}\right)+1} \\
&= \frac{1}{\exp\left(-\frac{1}{\lim\limits_{D\to\infty}\sqrt{D}}\right)+1} \\
&= \frac{1}{\exp\left(-\frac{1}{\sqrt{\lim\limits_{D\to\infty} D}}\right)+1} \\
&= \frac{1}{e^{-\frac{1}{\sqrt{\infty}}}+1} = 0.5
\end{aligned}
\tag{11}
$$

Finally, we can easily observe that $\lim\limits_{D\to\infty} \frac{1}{e^{\frac{c}{\sqrt{D}}}+1} = \lim\limits_{D\to\infty} \frac{1}{e^{-\frac{1}{\sqrt{D}}}+1} = 0.5$ and thus prove Theorem 1 when σ is ReLu, LeakyReLu or PReLu.

A.2 Exploring Group Discrimination

Firstly, we explore the corruption technique used in DGI [7] and MVGRL [3], which is shown in Figure 5. These two studies corrupt the topology of a given graph $\mathcal{G}$ by shuffling the feature matrix $\mathbf{X}$. This is because by changing the node order of $\mathbf{X}$, the neighbouring structure of $\mathcal{G}$ is completely changed, e.g., neighbours of node a become node b neighbours.

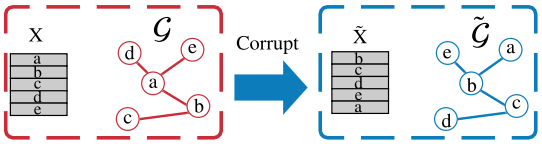


Figure 5: Corruption technique in DGI and MVGRL.

With the corruption technique, negative samples in the negative group are generated with incorrect edges. Thus, by discriminating the positive group (i.e., nodes generated with ground truth edges) and the negative group, we conjecture the model can distil valuable signals by learning how to identify nodes generated with correct topology and output effective node embeddings.

A.2.1 Theoretical Analysis of Group Discrimination

In this section, we first present the theoretical analysis and then provide an intuitive explanation for group discrimination.

Theorem 2 *Given a graph $\mathcal{G}$, a corrupted graph $\tilde{\mathcal{G}}$, and a encoding network $f_\theta()$, we consider the distribution of positive embeddings $f_\theta(\mathcal{G})$ as P_{pos} and negative embeddings $f_\theta(\tilde{\mathcal{G}})$ as P_{neg}. By optimising the group discrimination loss, P_{pos} and P_{neg} tends to separated.*

Proof. The following proof is inspired by the theoretical proof in GAN [30]. During training, we can rewrite the group discrimination loss in Equation 3 into the following form:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{z\sim P_{pos}} log(agg(z)) + \mathbb{E}_{z\sim P_{neg}} log(1 - agg(z)),\\ &= \int_z P_{pos}(z) log(agg(z))dz + \int_z P_{neg}(z) log(1 - agg(z))dz,\end{aligned} \tag{12}$$

where $agg()$ is the aggregation function to turn z into the 1×1 prediction, P_{pos} are the distribution of positive embeddings, P_{neg} are the distribution of negative embeddings. As our objective here is to maximise $\mathcal{L}$, and $P_{pos}(z) > 0; P_{neg}(z) > 0$, we can obtain the optimal solution for $agg(z)$ is $\frac{P_{pos}(z)}{P_{pos}(z)+P_{neg}(z)}$. This is because for any $(a, b) \in \mathbb{R}^2\{0, 0\}$, the maximum of a function $y = alog(x) + blog(1-x)$ is achieved at $\frac{a}{a+b}$ [30]. By replacing $agg(z)$ with $\frac{P_{pos}(z)}{P_{pos}(z)+P_{neg}(z)}$ in Equation 12, we can obtain:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{z\sim P_{pos}} log(\frac{P_{pos}(z)}{P_{pos}(z) + P_{neg}(z)}) + \mathbb{E}_{z\sim P_{neg}} log(1 - \frac{P_{pos}(z)}{P_{pos}(z) + P_{neg}(z)}),\\ &= \mathbb{E}_{z\sim P_{pos}} log(\frac{P_{pos}(z)}{P_{pos}(z) + P_{neg}(z)}) + \mathbb{E}_{z\sim P_{neg}(z)} log(\frac{P_{neg}}{P_{pos}(z) + P_{neg}(z)}).\end{aligned} \tag{13}$$

From the equation above, we can see it looks similar to the Jensen-Shannon divergence between the distribution of positive embeddings and negative embeddings:

$$JS(P_1 \parallel P_2) = \frac{1}{2}\mathbb{E}_{z\sim P_1} log(\frac{\frac{P_1}{P_1+P_2}}{2}) + \frac{1}{2}\mathbb{E}_{z\sim P_2} log(\frac{\frac{P_2}{P_1+P_2}}{2}). \tag{14}$$

Thus, we can rewrite Equation 13 as:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{z\sim P_{pos}} log(\frac{\frac{P_{pos}(z)}{P_{pos}(z)+P_{neg}(z)}}{2}) + \mathbb{E}_{z\sim P_{neg}} log(\frac{\frac{P_{neg}(z)}{P_{pos}(z)+P_{neg}(z)}}{2}) - 2log2,\\ &= 2JS(P_{pos} \parallel P_{neg}) - 2log2,\end{aligned} \tag{15}$$

where we can see maximising $\mathcal{L}$ is the same as maximising $JS(P_{pos} \parallel P_{neg})$. Thus, by optimising $\mathcal{L}$, P_{pos} and P_{neg} tend to be separated. Based on the proof above, we can see group discrimination intuitively is learning to avoid making 'mistakes' (i.e., bias the encoder towards avoiding mistaken samples) as shown in Figure 6. This is because optimising the group discrimination loss can separate P_{pos} and P_{neg}. As $P_{optimal}$ is disjoint with P_{neg}, if the generated embeddings can avoid being similar to out-of-distribution samples, i.e., negative samples, it can be more close to $P_{optimal}$. Therefore, the trained model can improve the quality of generated node embeddings for node samples.

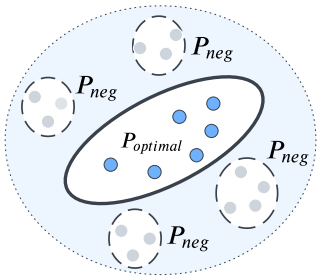


Figure 6: $P_{optimal}$ is the optimal distribution for node embeddings, P_{neg} is the distribution of negative samples, gray nodes represent negative samples, and blue nodes are samples in the optimal distribution.

A.3 Evaluation on aggregation function

To explore the effect of other aggregation functions, we replace the summation function in Equation 1 with other aggregation methods, including mean-, minimum-, maximum- pooling and linear aggregation. We report the experiment results (i.e., averaged accuracy on five runs) in Table 11. The table shows that replacing the summation with other aggregation methods still works, while summation and linear aggregation achieve comparatively better performance.

A.4 Rethinking MVGRL

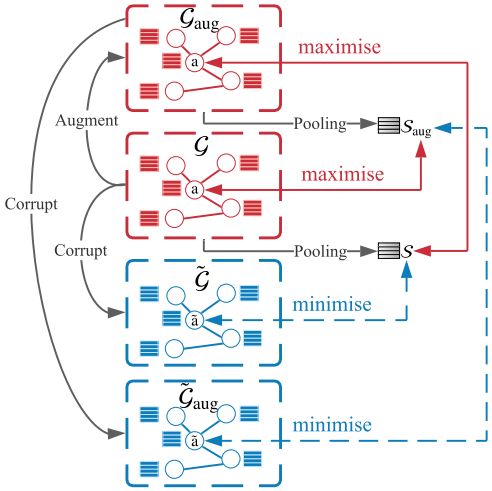


Figure 7: The architecture of MVGRL. Here augment means augmentation. S is the summary vector based on $\mathcal{G}$, and S_{aug} is the summary vector based on the augmented graph $\mathcal{G}_{aug}$.

Extending the architecture of DGI, MVGRL resorts to multi-view contrastiveness via additional augmentation. Specifically, as shown in Figure 7, it first uses the diffusion augmentation to create $\mathcal{G}_{aug}$. Then, it corrupts $\mathcal{G}$ and $\mathcal{G}_{aug}$ to generate negative samples $\tilde{\mathcal{G}}$ and $\tilde{\mathcal{G}}_{aug}$. To build contrastiveness, MVGRL also generates two summary vectors S_{aug} and S by averaging all embeddings in $\mathcal{G}_{aug}$ and $\mathcal{G}$, respectively. Based on the design of MVGRL, the model training is driven by mutual information maximisation between an anchor node embedding and its correspond-

Table 11: The experiment result on three datasets with different aggregation function on node embeddings.

Method	Cora	CiteSeer	PubMed
Sum	82.5 ±0.2	71.7 ±0.6	77.7 ±0.5
Mean	81.8 ±0.5	71.8 ±1.1	76.5 ±1.2
Min	80.4 ±1.3	61.7 ±1.8	70.1 ±1.9
Max	71.4 ±1.2	65.3 ±1.4	70.2 ±2.8
linear	82.2 ±0.4	72.1 ±0.7	77.9 ±0.5

Table 12: The ablation study of MVGRL from the perspective of Group Discrimination.

Method	**Cora**	**CiteSeer**	**PubMed**
$MVGRL$	82.9 $\pm$0.9	72.6 $\pm$0.8	78.8 $\pm$0.6
$MVGRL_{bce}$	83.1 $\pm$0.6	72.8 $\pm$0.5	79.1 $\pm$1.1
$MVGRL_{bce}$ w/o $\mathcal{G}_{aug}$	81.2 $\pm$0.8	52.8 $\pm$3.1	76.6 $\pm$1.3
$MVGRL_{bce}$ w/o $\mathcal{G}$	82.1 $\pm$0.6	71.8 $\pm$1.1	77.1 $\pm$1.2
$MVGRL_{bce}$ w/o $\tilde{\mathcal{G}}_{aug}$	81.1 $\pm$0.8	56.7 $\pm$2.1	74.9 $\pm$1.3
$MVGRL_{bce}$ w/o $\tilde{\mathcal{G}}$	82.7 $\pm$0.9	72.0 $\pm$0.9	78.6 $\pm$0.9
$MVGRL_{bce}$ w/o $\mathcal{G}_{aug}$ and $\tilde{\mathcal{G}}_{aug}$	82.2 $\pm$0.6	71.8 $\pm$1.0	77.0 $\pm$0.8
$MVGRL_{bce}$ w/o $\mathcal{G}$ and $\tilde{\mathcal{G}}$	83.1 $\pm$0.6	72.6 $\pm$0.6	78.5 $\pm$1.4

ing augmented summary vector. However, MVGRL has the same technical error in their official JSD-based implementation as DGI, which makes it also becomes a group-discrimination-based approach.

Similar to Equation 3 of DGI, the proposed loss in MVGRL can also be rewritten as a binary cross entropy loss:

$$\mathcal{L}_{MVGRL} = \frac{1}{4N}(\sum_{i=1}^{4N} y_i \log h_i + (1 - y_i)\log(1 - h_i)), \tag{16}$$

here the number of nodes is increased to $4N$ as we include nodes in $\mathcal{G}_{aug}$ and $\tilde{\mathcal{G}}_{aug}$ as data samples. The indicator y_i for $\mathcal{G}$ and $\mathcal{G}_{aug}$ are 1, while $\tilde{\mathcal{G}}$ and $\tilde{\mathcal{G}}_{aug}$ are considered as negative samples (i.e., the indicator y_i for them are 0). To explore why MVGRL can achieve a better performance than DGI, we replace the original MVGRL loss with Equation 16 and conduct ablation study of $MVGRL_{bce}$ by removing different set of data samples in Equation 16, and report the experiment result in Table 12. From the table, the performance of $MVGRL_{bce}$ is on par with $MVGRL$, which reconfirms the effectiveness of using the BCE loss. Also, we can observe that including $\mathcal{G}_{aug}$ and $\tilde{\mathcal{G}}_{aug}$ is the key of MVGRL surpassing DGI. With $\mathcal{G}_{aug}$ and $\tilde{\mathcal{G}}_{aug}$, the model performance of $MVGRL_{bce} w/o \mathcal{G}_{aug}$ and $\tilde{\mathcal{G}}_{aug}$ is improved from 82.2 to 83.1. We conjecture this is because, with the diffusion augmentation, MVGRL is trained with the additional global information provided by the diffused view $\mathcal{G}_{aug}$. However, the diffusion augmentation involves expensive matrix inversion computation and significantly densifies the given graph, which requires much more memory and time to store and process than the original view. This can hinder the model from extending to large-scale datasets [18].

A.5 Complexity Analysis

The time complexity of our method consists of two components: the *siamese GNN* and the *loss computation*. Existing self-supervised baselines share similar time complexity for the first component. In `GGD`, Given a graph $\mathcal{G} = (\mathbf{X} \in \mathcal{R}^{N \times D}, \mathbf{A} \in \mathcal{R}^{N \times N})$ in the sparse format, taking a L-layer GCN [16] encoder as an example, the time complexity of it is $O(LMD + LND^2)$. As we need to process both the augmented graph $\hat{\mathcal{G}}$ and the corrupted graph $\tilde{\mathcal{G}}$, `GGD` requires the encoder computation twice. Then, the projector network (i.e., MLP) with K linear layers will be applied to the encoder output, which takes $O(ND^2)$ for each layer in computation. Here D is a parameter defining the hidden size. Before group discrimination, we aggregate the generated embedding with aggregation techniques. Here we take the simple summation, which consumes $O(ND)$ as example.

For the loss computation, we use the BCE loss, i.e., Equation 3, to category summarised node embeddings, i.e., scalars. The time complexity of this final step is $O(2N)$ (i.e., processing all data samples from the positive and negative groups). Ignoring the computation cost of the augmentation, the overall time complexity of `GGD` for computing a graph $\mathcal{G}$ is $O(2(LMD + LND^2 + KND^2 + ND + N)) \rightarrow O(LMD + LND^2 + KND^2 + ND + N)$, where we can see the time complexity is mainly contributed by the siamese GNN. More importantly, we compress the self-supervised learning loss computation to $O(N)$ for a whole graph.

A.6 Graph Power

To show the easiness of graph power computation, we conduct an experiment to evaluate the time consumption for graph power computation on eight datasets, whose statistics are shown in Appendix A.7. Specifically, we set the hidden size of $\mathbf{H}_\theta$ to 256, and n is fixed to 10 for all datasets. The experiment results are shown as below:

Table 13: Graph power computation time in seconds on eight benchmark datasets. The experiment is conducted using CPU: Intel Xeon Gold 5320. 'Cite' , 'Comp', 'Photo', 'Arxiv', 'Products', 'Papers' means Citeseer, Amazon Computer, Amazon Photo, ogbn-arxiv, ogbn-products and ogbn-papers100M.

Cora	**Cite**	**PubMed**	**Comp**	**Photo**	**Arxiv**	**Products**	**Papers**
5.4e-3	7.3e-3	9.8e-3	1.2e-2	8.5e-3	2.2e-2	24.5	208.8

This table shows that the computation of graph power is very trivial on small and medium size graphs, e.g., ogbn-arxiv, which has million of edges, consuming only 0.22 seconds. Extending to an extremely large graph, ogbn-papers100M, which has over 1 billion edges and 11 million nodes, the computation only requires 209 seconds (i.e., around three minutes), which is acceptable considering the sheer size of the dataset.

A.7 Dataset Statistics

The following table presents the statistics of eight benchmark datasets including five small to medium -scale datasets and three large-scale datasets from OGB Graph Benchmark[26].

Table 14: The statistics of eight benchmark datasets.

Dataset	**Nodes**	**Edges**	**Features**	**Classes**
Cora	2,708	5,429	1,433	7
CiteSeer	3,327	4,732	3,703	6
PubMed	19,717	44,338	500	3
Amazon Computers	13,752	245,861	767	10
Amazon Photo	7,650	119,081	745	8
ogbn-arxiv	169,343	1,166,243	128	40
ogbn-products	2,449,029	61,859,140	100	47
ogbn-papers-100M	111,059,956	1,615,685,872	100	172

A.8 Experiment Settings & Computing Infrastructure

Extending to Extremely Large Datasets. Extending to extremely large graphs (i.e., ogbn-products and ogbn-papers100M), we adopt a simple Neighbourhood Sampling strategy introduced in GraphSage [20] to decouple model training from the sheer size of graphs. Specifically, we create a fixed size subgraph for each node, which is created by sampling a predefined number of neighbours in each convolution layer for sampled nodes. The same approach is employed in the testing phase to obtain final embeddings.

General Parameter Settings. In our experiment, we mainly tune four parameters for `GGD` ,which are learning rate, hidden size, number of convolution layers in the GNN encoder, and number of linear layers in the projector. For simplicity, we set the n-th graph power for final embedding generation fixed to 5 for all datasets. The parameter setting for each dataset is shown below:

Table 15: Parameter settings on eight datasets. 'lr' means the learning rate for pretraining, 'num-conv' and 'num-proj' represent number of convolution layers in GNNs and number of linear layers in projector, respectively.

Dataset	lr	hidden	num-conv	num-proj
Cora	1e-3	256	1	1
CiteSeer	1e-5	512	1	1
PubMed	1e-3	512	1	1
Amazon Computers	1e-3	256	1	1
Amazon Photo	1e-3	1024	1	1
ogbn-arxiv	5e-5	1500	3	1
ogbn-products	1e-4	1024	4	4
ogbn-papers-100M	1e-3	256	3	1

Large-scale Datasets Parameter Settings. To decouple model training from the scale of graphs, we adopt the neighbouring sampling technique, which has three parameters: batch size, sample size, and number of hops to be sampled. Batch size refers to the number of nodes to be processed in one parameter optimisation step. Sample size means the number of nodes to be sampled in each convolution layer, and the number of hops determines the scope of the neighbourhood for sampling. In GGD implementation, the batch size, sample size, and the number of hops are fixed to 2048, 12 and 3, respectively.

Memory and Training Time Comparison. As memory and training time are very sensitive to hyper-parameters related to the structure of GNNs, including hidden size, number of convolution layers, and batch processing for large-scale datasets, e.g., batch size and number of neighbours sampled in each layer. Thus, in memory and training comparison, to be fair, we set all these parameters to be the same for all baselines and GGD. The specific parameter setting for each dataset is shown below:

Table 16: Parameter settings on eight datasets for memory and training time comparison.

Dataset	hidden	num-conv	batch	num-neigh
Cora	512	1	-	-
CiteSeer	512	1	-	-
PubMed	256	1	-	-
Amazon Computers	256	1	-	-
Amazon Photo	256	1	-	-
ogbn-arxiv	256	3	-	-
ogbn-products	256	3	512	10
ogbn-papers-100M	128	3	512	10

Computing Infrastructure. For experiments in section 3, 5 and 6.1, they are conducted using Nvidia GRID T4 (16GB memory) and Intel Xeon Platinum 8260 with 8 core. For experiments on large-scale datasets (i.e.,ogbn-arxiv, ogbn-products and ogbn-papers100M), we use NVIDIA A40 (48GB memory) and Intel Xeon Gold 5320 with 13 cores.

A.9 Algorithm

We have summarised the overall procedure of GGD in Algorithm 1 as follows:

Specifically, we first conduct augmentation on $\mathcal{G}$ to obtain $\hat{\mathcal{G}}$. Then, $\hat{\mathcal{G}}$ is corrupted to generate the corrupted graph $\tilde{\mathcal{G}}$. In the encoding phase, we feed $\hat{\mathcal{G}}$ and $\tilde{\mathcal{G}}$ to GNN encoder g_θ and projector f_θ to generate embeddings $\hat{\mathbf{H}}_\theta$ for positive samples and $\tilde{\mathbf{H}}_\theta$ for negative samples. After that, we obtain the binary classification result by aggregating the concatenation of $\hat{\mathbf{H}}_\theta$ and $\tilde{\mathbf{H}}_\theta$. The result is a $2N$ dimension vector, which can be used for calculating the loss with a BCE loss. Finally, based on the calculated loss, trainable parameters in g_θ and f_θ can be updated. This training process will continue iteratively until we reach the predefined number of epochs E. When the training process is completed, we freeze the g_θ and f_θ and feed $\mathcal{G}$ to these two encoding components to obtain the local embedding,

Algorithm 1: The Overall Procedure of GGD

Input : Input Graph $\mathcal{G} = (\mathbf{X} \in \mathcal{R}^{N \times D}, \mathbf{A} \in \mathcal{R}^{N \times N})$; GNN encoder g_θ; Projector f_θ; Number of nodes N; Number of feature dimensions D; Number of hidden dimensions D'; Number of training epochs E

Output : Final representation $\mathbf{H}$

//Model Training;
for $e = 1$ *to* E **do**
  //Augmentation(optional);
  Conduct feature dropout and edge dropout on $\mathcal{G}$ to obtain $\hat{\mathcal{G}} = (\hat{\mathbf{X}}, \hat{\mathbf{A}})$;
  //Corruption;
  Corrupt $\hat{\mathcal{G}}$ to obtain $\tilde{\mathcal{G}} = (\tilde{\mathbf{X}}, \tilde{\mathbf{A}})$;
  //Compute Encoding;
  Input $\hat{\mathcal{G}}$ and $\tilde{\mathcal{G}}$ to g_θ and f_θ to obtain graph embedding $\hat{\mathbf{H}}_\theta = f_\theta(g_\theta(\hat{\mathcal{G}}))$ and $\tilde{\mathbf{H}}_\theta = f_\theta(g_\theta(\tilde{\mathcal{G}}))$, respectively;
  //Aggregation;
  Concatenate $\hat{\mathbf{H}}_\theta$ and $\tilde{\mathbf{H}}_\theta$ to obtain $\bar{\mathbf{H}}_\theta$;
  Conduct aggregation on $\bar{\mathbf{H}}_\theta \in \mathcal{R}^{\mathrm{2N} \times \mathrm{D}'}$ to obtain the prediction vector $h_\theta \in \mathcal{R}^{2N}$;
  //Compute Loss;
  Calculate loss $\mathcal{L} = \frac{1}{2N}(\sum_{i=1}^{2N} y_i \log h_i + (1 - y_i)\log(1 - h_i))$, where $h_i \in \mathcal{R}^1$ is the prediction for one node sample and $h_i \in h_\theta$;
  //Update parameters;
  Update trainable parameters in g_θ and f_θ;
//Inference;
Obtain local embedding $\mathbf{H}_\theta = f_\theta(g_\theta(\mathcal{G}))$;
Obtain global embedding $\mathbf{H}_\theta^{global} = \mathbf{A}^n \cdot \mathbf{H}_\theta$;
Obtain final embeddings for downstream tasks $\mathbf{H} = \mathbf{H}_\theta^{global} + \mathbf{H}_\theta$;

$\mathbf{H}_\theta$. Then, we obtain global embedding $\mathbf{H}_\theta^{global}$ with a global information injection operation. By summing $\mathbf{H}_\theta$ and $\mathbf{H}_\theta^{global}$, we can get the final embeddings $\mathbf{H}$.

A.10 Ablation Study

In this section, we conduct an ablation study to evaluate the effectiveness of different components in GGD. Specifically, we evaluate three variants of GGD, including GGD$_{w/o\ aug}$, GGD$_{w/o\ proj}$, GGD$_{w/o\ power}$, which represent GGD without augmentation, the projector and the global information injection process respectively. The experiment results on five small to medium size datasets are presented in Table 17. From the table, we can see without any mentioned component, the performance of GGD degrades, which validates the effectiveness of these components. It is worth noting that even without the global information injection in the inference phase, GGD$_{w/o\ power}$ still achieves the highest performance in 4 out of 5 datasets compared with six self-supervised baselines. This indicates that even without global information injection, GGD is still effective.

Table 17: Ablation Study for GGD.

Method	Cora	CiteSeer	PubMed	Comp	Photo
GGD$_{w/o\ aug}$	83.6±0.3	72.4±0.4	81.2±0.2	89.6±0.4	92.2 ±0.5
GGD$_{w/o\ proj}$	83.0±0.5	72.5±0.4	81.1±0.4	89.4±0.5	91.6 ±0.5
GGD$_{w/o\ power}$	83.0±0.5	72.5±0.4	80.1±0.4	89.9±0.6	91.6±0.4
GGD	**83.9**±0.4	**73.0**±0.6	**81.3**±0.8	**90.1**±0.9	**92.5**±0.6