Towards Effective and Robust Graph Contrastive Learning with Graph Autoencoding

Wen-Zhi Li, Chang-Dong Wang, *Member, IEEE,* Jian-Huang Lai, *Senior Member, IEEE,* Philip S. Yu, *Fellow, IEEE*

Abstract—Graph contrastive learning (GCL) has become the de-facto approach to conducting self-supervised learning on graphs for its superior performance. However, non-semantic graph augmentation methods prevent it from achieving better performance, and it suffers from vulnerability to graph attacks. To deal with these problems, we propose AEGCL to leverage graph AutoEncoder in Graph Contrastive Learning which directly targets graph property reconstruction to boost GCL effectiveness and robustness. Specifically, AEGCL has two distinctive characteristics, (1) a novel adaptive augmentation strategy based on *motif* centrality is proposed, which leverages semantic significant higher-order graph property; (2) the original attributed graph is decoupled into feature graph and topology graph to extract their dedicated information, and a simple *AttnFuse* is proposed to combine the two augmented graphs and the two decoupled graphs. Graph autoencoder can thus be applied to the topology domain and raw attribute domain. Empirically, extensive experiments on benchmark graph datasets show that AEGCL outperforms existing baseline methods in terms of classification accuracy and robustness.

Index Terms—Graph Neural Networks, Self-supervised Learning, Contrastive Learning, Graph Representation Learning, Robustness.

1 Introduction

Graph Neural Networks (GNNs) has become the yardstick to handle graph-structured data in the past few years, with the maturing of deep neural network. Its application involves all aspects of graph-related representation learning from protein discovery to traffic prediction to name a few [1], [2], [3], [4], [5], [6].

Nowadays, the main obstacle of GNNs lies in that the methods are mostly devised for semi-supervised downstream tasks, which means adequate labeled data are required. However, it is hard to satisfy as annotated labels are always scarce in real-world scenarios, and manual annotation is too costly to implement. To tackle this problem, self-supervised network embedding approaches emerge, which leverage the supervision signal from the unlabeled data itself. They can be roughly categorized into two classes, i.e., graph contrastive learning (GCL) [7], where two graph augmentations are generated and the mutual information between positive pairs is maximized while the mutual information between negative pairs is minimized; and graph autoencoder [8], where reconstruction of the original graph is directly leveraged.

Among them, GCL approaches dominate the community for their superior performance on downstream tasks. However, recent studies [7], [9], [10], [11] point out that they might be vulnerable to graph attacks for that graph properties are distorted during augmentation. Although [9] integrates explicit adversarial transformation to graph augmentations to remit this issue, it brings intolerant training time overhead, which heavily reduces model efficiency. Therefore, an efficient method is required to improve GCL robustness. Graph autoencoder, on the other hand, directly targets at graph property reconstruction, which is expected to boost the model robustness effectively [12].

As the most crucial part of GCL, graph augmentation is determinant to model performance. However, unlike images which can be augmented via *semantic* operations like clipping or rotating [13], [14], [15], prevalent data augmentation methods for graphs are either operations far less intuitive like randomly dropping nodes and randomly masking attributes [7], [11], or operations specified for a certain domain which requires domain knowledge [16]. To remit this issue, adaptive augmentations are proposed [17], [18] to drop less important edges and node attributes based on semantic edge/node centrality. Though semantic information is implied, it might lead to sub-optimal results. For one thing, all adaptive augmentations are based on node-level centrality, while higher-order characteristics in graphs which contain rich semantic information is not leveraged. For another, as shown in Fig. 1, node degree usually follows a long-tailed distribution in real-world graph scenarios. Dropping less important nodes will force GNNs mainly focus on scarce high-degree nodes, which exacerbates the long-tailed effect and therefore prevents GCL from getting better performance.

In view of these issues, we are naturally motivated to

- *Wen-Zhi Li and Chang-Dong Wang are with School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China, Guangdong Province Key Laboratory of Computational Science, Guangzhou, China, and Key Laboratory of Machine Intelligence and Advanced Computing, Ministry of Education, China. Corresponding author: Chang-Dong Wang.*
  E-mail: liwzh63@mail2.sysu.edu.cn, changdongwang@hotmail.com.
- *Jian-Huang Lai is with School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China, Guangdong Key Laboratory of Information Security Technology, Guangzhou, China, and Key Laboratory of Machine Intelligence and Advanced Computing, Ministry of Education, China.*
  E-mail: stsljh@mail.sysu.edu.cn.
- *Philip S. Yu is with the University of Illinois at Chicago, Chicago, IL 60607, and also with the Institute for Data Science, Tsinghua University, Beijing 100084, China.*
  E-mail: psyu@cs.uic.edu.

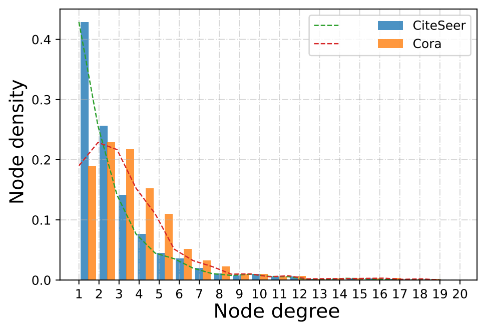


Fig. 1. Node degree ususally follows long-tailed distribution on real-world datasets.

ask the following question:

> *How to leverage graph autoencoder and semantic higher-order graph characteristics to boost the effectiveness and robustness of GCL?*

Inspired by some recent works [19], [20], [21], we propose AEGCL, which adopts motifs in graphs for adaptive augmentation, and decouples node features and topology structures then reconstructing them to leverage graph AutoEncoder in Graph Ccontrastive Learning. Specifically, to force augmentation in GCL to gain semantic higher-order graph characteristics, we first propose an intuitive and effective adaptive augmentation based on motif centrality. Then, we find that a typical input graph of GNN is attributed graph [1], [2], which means there is two-fold information, i.e., node features derived from traditional feature engineering and topology structures derived from the intrinsic graph property. So graph reconstruction mainly lies in two classes, i.e., node feature reconstruction and graph structure recovery [8], [22], [23], [24]. Unlike traditional GNNs which consider both simultaneously by propagating features along with graph structures, we propose to decouple the original graph into feature graph and topology graph, and extract their specific embeddings. However, how to fuse the two decoupled graphs into the GCL structure remains a challenge. We notice that GCL also has two augmentation graph views, which can be conjoined with the two decoupled graphs naturally. Inspired by this, we propose a learnable *AttnFuse* module to fuse information from the two augmentation graphs and the two decoupled graphs automatically. The reconstruction objectives are defined as two fine-grained reconstruction losses on topology and feature, which are combined with the GCL objective in a multi-task learning manner. In this way, AEGCL is expected to be robust to link or feature attacks for that it leverages the two-fold ground-truth information explicitly.

To sum up, we highlight our contributions as follows:

- We propose an effective and robust self-supervised learning framework on graphs, AEGCL, by leveraging AutoEncoder in Graph Contrastive Learning.
- We propose a novel adaptive augmentation method based on *motif centrality* to capture higher-order graph characteristics for graph contrastive learning, which is semantically significant and can thus boost GCL performance.
- Empirically, extensive experiments demonstrate the effectiveness and robustness of AEGCL compared with both self-supervised and semi-supervised graph representation learning baseline methods.

The rest parts of this paper are organized as follows. In Section 2, we review some related work. In Section 3, we give the problem definition, main notations, and some basic preliminaries about the model. In Section 4, we give the details of AEGCL. In Section 5, we conduct experiments to evaluate the effectiveness and robustness of AEGCL. In Section 6, we conclude this paper with future work.

2 Related Work

In this section, We briefly review three classes of related work: self-supervised learning on graphs, graph attack, and higher-order structure on graphs.

2.1 Self-Supervised Learning on Graphs

Labeled data are often scarce in real-world scenarios. To tackle this challenge, one mainstream solution is to apply self-supervised learning to vast unlabeled data. This idea is popular from NLP [25], [26], [27] to CV [28], [29], [30], [31] in recent years. Some previous works also apply self-supervised learning to graph representation learning. They are generally in two categories, namely, graph autoencoder and graph contrastive learning.

Autoencoder [32], [33] refers to the paradigm of "encoding-decoding" to recover graph properties. Compared with GCL, graph autoencoder is easy to implement and faithful to ground-true graph properties. Traditional random-walk-based approaches like Deepwalk [34], LINE [35] and Node2vec [36] can be regarded as shallow graph autoencoders, as indicated in [37]. The goal of these approaches is to learn node embeddings solely from graph structure information, and the learned node embeddings can be used for the downstream tasks directly. These methods, however, can not be customized for specific downstream tasks, which prevents them from better performance. Recently, with the fast development of deep neural networks, graph autoencoder approaches are often transferred from other deep learning domains like computer vision or natural language processing. Kipf and Welling [12] apply a VAE [38] structure where a simple 2-layer GCN is leveraged as the encoder, and dot-product is leveraged as the decoder to get a reconstructed topology graph as the self-supervised task. Inspired by Transformer [39], Zhang et al. [22] take a bold step by ignoring the graph's topological structure and propose a model solely based on Bert [25] with node feature and graph structure reconstruction tasks. Likewise, Hu et al. [23] propose a model imitating GPT [26] by considering the attributed graph generation task step-by-step. For specific domains, Hu et al. [8] propose a model for biology graphs with two recovery tasks including neighborhood prediction and node attributes prediction. However, some recent studies [37], [40] point out that graph autoencoder methods fall behind graph contrastive learning methods by a large margin.

Contrastive learning [7], [41], [42] is another line of approaches for self-supervised learning. It has been dominating self-supervised learning in the literature for its expressivity and simplicity. It refers to the paradigm of making pair-view representations to agree with each other under proper data augmentations, which is also adapted from CV to GNNs. Though a recent study [43] argues that complicated graph augmentations are not that necessary in downstream fields such as recommendation as long as the uniformity and alignment of contrastive learning could be well preserved, graph augmentation is still a research hotspot for graph representation learning, as proper augmentation can boost the GCL performance in practice [17]. Specifically, Velickovic et al. propose DGI [44] which contrasts node-local patches against global graph representations. Peng et al. propose GMI [45] which maximizes graphical mutual information for a more fine-grained contrastive loss. Hassani et al. propose MVGRL [46] which maximizes information between the cross-view representations of nodes and graphs. Zhu et al. propose GRACE [47] which adopts the popular contrastive learning paradigm SimCLR [15] in the computer vision domain to network embedding. Furthermore, Zhu et al. propose GCA [17] which improves GRACE with adaptive data augmentation to leverage graph semantics. Thakoor et al. propose BGRL [48] which firstly explores bootstrapping learning to graphs to reduce memory space. Different from these approaches, to the best of our knowledge, we are the first to introduce graph autoencoder to GCL. In our proposed AEGCL, we leverage attribute recovery and topology reconstruction as two auxiliary tasks to improve GCL robustness and effectiveness.

2.2 Graph Robustness

As the extensions of deep neural networks to graphs, GNNs inherit the vulnerability of easily getting fooled by small input perturbation, i.e., graph attacks [49], [50]. From another perspective, graph attacks can be used to test the robustness of GNN models. Attacks can be categorized into white-box attacks, black-box attacks, and gray-box attacks based on their knowledge over the entire graph [51], [52]. For the white-box attack, the GNN parameters, node labels together with graph information are all given to the attacker [53], [54], [55], while for the black-box attack, only graph information is visible [49], [50], [56] and for the gray-box attack, the attacker only has limited knowledge about the victim model [57], [58]. In this paper, instead of specializing in network attacks on graphs, we only apply it deliberately to test the robustness of different models. For a fair comparison, we apply the gray-box attack *nettack* proposed in [49] to test the GNN robustness for all reported experiments. A recent work ARIEL [11] also considers robustness in the context of GCL. However, it differs from our method as an adversarial view is generated explicitly via PGD attack [59], which brings heavy computational overhead, thus reducing model efficiency. The proposed AEGCL, on the other hand, directly targets lightweight graph property reconstruction, which is expected to boost the model robustness effectively.

2.3 Higher-Order Structure on Graphs

Besides node-level and edge-level interaction on graphs, researchers find that higher-order structure contains rich semantic information which can boost model performance [60]. Among them, the most widely studied higher-order structure is motif, which is a subnetwork appearing frequently in the original network [61], [62]. Motifs appear in a diversity of graph genres from biochemistry to sociology, and the research of such patterns often results in discovering new graph properties. There exist some studies which apply motif to graph analysis in the literature. In [63], Lee et al. try to capture higher-order neighborhoods by using weighted multi-hop motif adjacency matrices. In [64], a motif-level generation is developed for that motif is an essential feature in the community detection task. In [65], Sankar et al. propose a motif convolution method which captures semantic higher-order relationships to improve the semi-supervised node classification. In [41], Qiu et al. leverage motif ego network to learn universal properties of graphs for graph pre-training. In [66], Rong et al. conduct graph representation learning by predicting contextual motifs for transfer learning. Inspired by these works, we hypothesize that motif-based graph augmentation can boost GCL performance for it can capture higher-order graph characteristics.

TABLE 1
Summary of the main notations used throughout the paper.

Symbol	Description
$\mathbf{M}$, $\mathbf{m}_i$	matrix, i-th row of the matrix
$\mathcal{G}$	attributed graph
$\mathcal{V}$	the set of nodes in the graph
$\mathcal{E}$	the set of edges in the graph
$N = \|\mathcal{V}\|$	number of nodes in the graph
$\mathbf{A} \in \mathbb{R}^{N\times N}$	adjacency matirx of the graph
$\mathbf{X} \in \mathbb{R}^{N\times d}$	raw features of the graph
$\mathbf{Y} \in \mathbb{R}^{N\times 1}$	labels of nodes in the graph
$f(\mathbf{X}, \mathbf{A}; \mathbf{\Theta})$	GNN encoder with parameters $\mathbf{\Theta}$
$S(\cdot,\cdot)$	cosine similarity
$\mathcal{G}_f$, $\mathcal{G}_t$	feature graph and topology graph
$\mathbf{X}_f$, $\mathbf{X}_t \in \mathbb{R}^{N\times h}$	node features of feature and topology graph
$\mathbf{A}_f$, $\mathbf{A}_t \in \mathbb{R}^{N\times N}$	adjacency matrix of feature and topology graph
$\mathbf{E} \in \mathbb{R}^{N\times d'}$	node2vec embeddings
$\mathbf{F}, \mathbf{T} \in \mathbb{R}^{N\times h}$	feature embeddings and topology embeddings
$\mathbf{H} \in \mathbb{R}^{N\times h}$	node embeddings from contrastive learning
$\mathbf{Z} \in \mathbb{R}^{N\times h}$	final embeddings
$\tilde{\mathbf{X}} \in \mathbb{R}^{N\times d}$	reconstructed feature
$\mathbf{X}'$, $\mathbf{A}'$	attacked node features and adjacency matrix

3 Preliminaries

In this section, we first define graph contrastive learning and the main notations used in the paper, then we give a brief introduction to graph neural networks.

3.1 Problem Definition

Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ denote an attributed graph, where $\mathcal{V} = \{v_1, v_2, \cdots, v_N\}$ is the node set with N nodes and $\mathcal{E} \subseteq \mathcal{V}\times\mathcal{V}$ is the edge set. The adjacency matrix and the feature matrix are denoted as $\mathbf{A} \in \{0,1\}^{N\times N}$ and $\mathbf{X} \in \mathbb{R}^{N\times d}$ respectively, where $\mathbf{A}_{ij} = 1$ iff $(v_i, v_j) \in \mathcal{E}$. $\mathbf{x}_i \in \mathbb{R}^{1\times d}$ is the raw attribute of node v_i, and d is the raw feature dimension. Given an attributed graph $\mathcal{G}$ with no avaliable labels, our goal is to train a graph encoder $f(\mathbf{X}, \mathbf{A}; \mathbf{\Theta})$ with $\mathbf{X}$ and $\mathbf{A}$ as input. The output embeddings $\mathbf{H} = f(\mathbf{X}, \mathbf{A}; \mathbf{\Theta})$ can be directly applied to downstream tasks by training a Multi-Layer Perceptron (MLP) in a freezing mode, or the trained encoder $f(\mathbf{X}, \mathbf{A}; \mathbf{\Theta})$ can be trained end-to-end with an MLP classifier in a full fine-tuning mode. More implementation details are available in Section 5. The detailed notation table is listed in Table 1.

3.2 Graph Neural Networks

With the fast development of deep learning, graph neural networks have become the de-facto standard feature extractor for graph-structured data in recent years. Generally, it follows the "propagation-transformation" pattern to extract features from neighbor nodes iteratively to fuse embeddings:

$$\mathbf{h}_t^{(l+1)} \leftarrow \mathbf{Transform}\left(\underset{\forall s \in \mathcal{N}(t)}{\mathbf{Propagate}}\left(\mathbf{h}_s^{(l)}; \mathbf{h}_t^{(l)}\right)\right), \quad (1)$$

where $\mathbf{h}_t^{(l)}$ is the node embedding of node v_t at the (l)-th layer, $\mathcal{N}(t)$ is the node set of v_t's neighbor nodes. In our scenarios, any kind of GNN can be applied here as the feature extractor, and we choose a two-layer GCN [1] as the backbone model structure for its efficiency and generalizability, which can be formalized as:

$$f(\mathbf{X}, \mathbf{A}; \mathbf{\Theta}) = \mathbf{H}^{(2)} = \hat{\mathbf{A}}\sigma(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}, \quad (2)$$

where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}(\mathbf{A} + \mathbf{I}_N)\tilde{\mathbf{D}}^{-1/2}$ with $\tilde{\mathbf{D}}$ being the degree matrix of $\mathbf{A} + \mathbf{I}_N$ and $\mathbf{I}_N$ being the identity matrix, $\mathbf{W}^{(l)}$ are learnable weight matrices (i.e., $\mathbf{\Theta}$) and $\sigma(\cdot)$ is activation function.

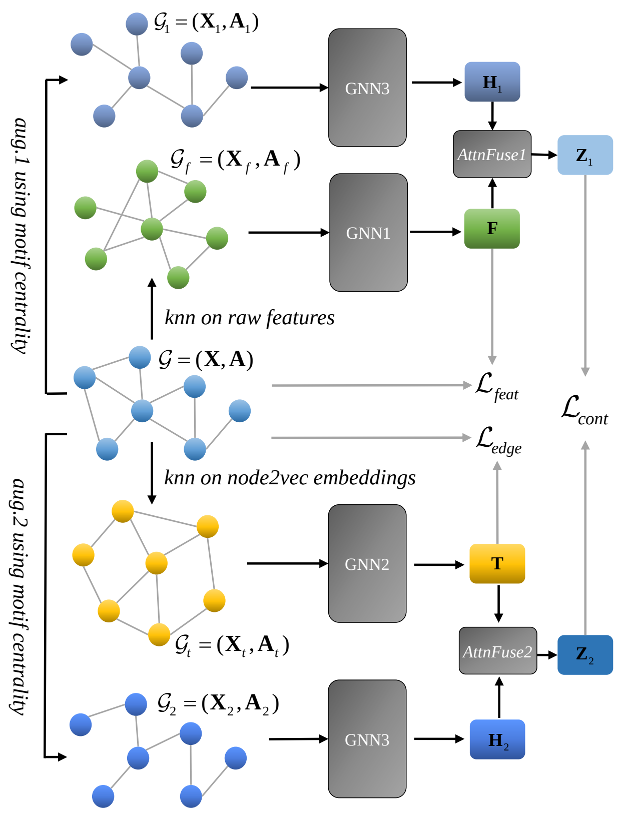
4 Methodologies

In this section, we present AEGCL in detail. The overall flowchart of AEGCL is sketched in Fig. 2. For the initial graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$, the key idea is to generate feature graph $\mathcal{G}_f$ and topology graph $\mathcal{G}_t$ via $k$nn on raw features and node2vec embeddings to extract embeddings $\mathbf{F}$ dedicated to node features and embeddings $\mathbf{T}$ dedicated to graph structures via GNN1 and GNN2, respectively. The two-fold information is fused with motif centrality-based GCL embeddings $\mathbf{H}_1$ and $\mathbf{H}_2$ generated from shared GNN3 via *AttnFuse*, after which we get $\mathbf{Z}_1$ and $\mathbf{Z}_2$. The loss function is defined as the weighted sum of contrastive loss $\mathcal{L}_{cont}$, raw feature recovery loss $\mathcal{L}_{feat}$, and edge reconstruction loss $\mathcal{L}_{edge}$, which are combined in a multi-task learning manner. The trained encoder GNN3 will be used for downstream tasks.

4.1 Feature and Topology Graph Construction

In order to decouple the two-fold node feature and graph topology information, following [19], we refer to k-nearest neighbor (kNN) to construct orthogonal feature graph $\mathcal{G}_f$ and topology graph $\mathcal{G}_t$. Specifically, for feature graph $\mathcal{G}_f$, we simply choose the top-k nearest nodes of each node in the raw attribute space $\mathbf{X}$ via cosine similarity:

$$\mathbf{S}(v_i, v_j) = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{|\mathbf{x}_i||\mathbf{x}_j|}. \quad (3)$$



$\mathcal{G}_1 = (\mathbf{X}_1, \mathbf{A}_1)$
$\mathcal{G}_f = (\mathbf{X}_f, \mathbf{A}_f)$
$\mathcal{G} = (\mathbf{X}, \mathbf{A})$
$\mathcal{G}_t = (\mathbf{X}_t, \mathbf{A}_t)$
$\mathcal{G}_2 = (\mathbf{X}_2, \mathbf{A}_2)$
aug.1 using motif centrality
aug.2 using motif centrality
knn on raw features
knn on node2vec embeddings
GNN3
GNN1
GNN2
GNN3
$\mathbf{H}_1$
AttnFuse1
$\mathbf{Z}_1$
$\mathbf{F}$
$\mathbf{T}$
AttnFuse2
$\mathbf{Z}_2$
$\mathbf{H}_2$
$\mathcal{L}_{feat}$
$\mathcal{L}_{edge}$
$\mathcal{L}_{cont}$

Fig. 2. The overall framework of AEGCL. Firstly, we generate feature graph $\mathcal{G}_f$ and topology graph $\mathcal{G}_t$ using $k$nn on raw attributes and node2vec embeddings, together with two graph augmentations $\mathcal{G}_1$ and $\mathcal{G}_2$ using motif centrality. Then, feature graph and topology graph are fed into dedicated GNN (GNN1 and GNN2) to capture their respective information, while two graph augmentations are fed into a shared GNN (GNN3). Finally, the learned feature embeddings $\mathbf{F}$ and topology embeddings $\mathbf{T}$ are aggregated with augmented graph embeddings $\mathbf{H}_1, \mathbf{H}_2$ via *AttnFuse*. The trained encoder GNN3 will be used for downstream tasks.

For topology graph $\mathcal{G}_t$, we first perform node2vec [36] on the initial graph topology structure to get node2vec embeddings $\mathbf{E} \in \mathbb{R}^{N \times d'}$ for each node, and then perform kNN on $\mathbf{E}$ in the same way as the feature graph. The reason for using node2vec is that it is a PageRank-like random walk solely based on topology structure so it can exploit topology information to form node embeddings.

It is worth noting that by setting k to a relatively small number, the feature graph and topology graph become very sparse compared to the initial graph, for that the number of edges is kN in both graphs. Specifically, for the Amazon-Photo dataset with $k = 5$, the number of edges in both feature and topology graphs is only 16% of that in the original graph. Besides, since we choose k nearest neighbors evenly for each node, the feature graph and the topology graph could avoid centrality biases caused by the long-tailed distribution. The detailed analysis of hyper-parameter k is performed in Section 5.

4.2 Adaptive Contrastive Learning Module with Motif Centrality

In this section, we apply the SimCLR [15] paradigm as an example to describe AEGCL since it is the state-of-the-art contrastive learning framework [17].

4.2.1 SimCLR overview

Given a graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$, two arbitrary augmentation views $\mathcal{G}_1 = (\mathbf{X}_1, \mathbf{A}_1)$ and $\mathcal{G}_2 = (\mathbf{X}_2, \mathbf{A}_2)$ are firstly generated via random edge dropping, attribute masking etc. Then they are fed into a shared GNN encoder, after which we get node embeddings $\mathbf{H}_1 = f(\mathbf{X}_1, \mathbf{A}_1; \mathbf{\Theta}) \in \mathbb{R}^{N \times h}$ and $\mathbf{H}_2 = f(\mathbf{X}_2, \mathbf{A}_2; \mathbf{\Theta}) \in \mathbb{R}^{N \times h}$. The key of SimCLR is to define the positive and negative node pairs. Take v_i in view $\mathcal{G}_1$ as an example, they are defined as follows:

Pos: Node v_i in $\mathcal{G}_1$ and $\mathcal{G}_2$.
Neg: (1) Intra-view: node v_i and another node v_j in view $\mathcal{G}_1$;
(2) Inter-view: node v_i in view $\mathcal{G}_1$ and another node v_k in view $\mathcal{G}_2$.

The contrastive loss can thus be defined with InfoNCE [67] following [17] as:

$$l_1(\mathbf{u}_i, \mathbf{v}_i) = \log \frac{\text{pos}}{\text{pos} + \text{neg}}, \tag{4}$$

$$\text{pos} = \exp\{\mathbf{S}(\mathbf{u}_i, \mathbf{v}_i)/\tau\}, \tag{5}$$

$$\text{neg} = \sum_{j \neq i} \exp\{\mathbf{S}(\mathbf{u}_i, \mathbf{v}_j)/\tau\} + \sum_{k \neq i} \exp\{\mathbf{S}(\mathbf{u}_i, \mathbf{v}_k)/\tau\}, \tag{6}$$

where $\mathbf{u}_*$ and $\mathbf{v}_*$ are node embeddings of v_* in $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively, and τ is a temperature parameter. The objective for node v_i in view $\mathcal{G}_2$, i.e., $l_2(\mathbf{v}_i, \mathbf{u}_i)$ can be defined in the same way.

4.2.2 Motif-based adaptive augmentation

Different from the random augmentation discussed above, adaptive augmentation devotes to generating graph views which are adaptive to the input graph so that it can reflect the graph's intrinsic properties [17]. Specifically, edge dropping and attribute masking are still adopted, but the dropping and masking probability is skewed for *less* important edges or attributes, and the importance is based on centrality, which could be degree, eigenvector, or PageRank. It will force the model to highlight important graph structures. However, due to (1) prevalent node centrality measures are all based on low-level node connection, which leaves semantic higher-order graph structure agnostic; (2) in real-world scenarios node degree always follows a long-tailed distribution, i.e., most nodes have a small degree, thus dropping less important nodes can easily degenerate to trivial random drop, which is not beneficial to gain robustness. Targeted to remit these two issues, we first propose to leverage the higher-order information at the level of subgraph, e.g., motif [61], then we argue that dropping *more* important edges [68] can force the model to learn essential information from the graph, which can help boost model performance.

Motif centrality. Motif is defined as a small dense subgraph pattern occurring in a complex network at numbers that are significantly higher than those of randomized subgraphs preserving the same degree of nodes [69], [70]. It is a frequently-used representation for small dense subgraph patterns. Therefore, we propose a brand new adaptive augmentation method using motif, called motif centrality.

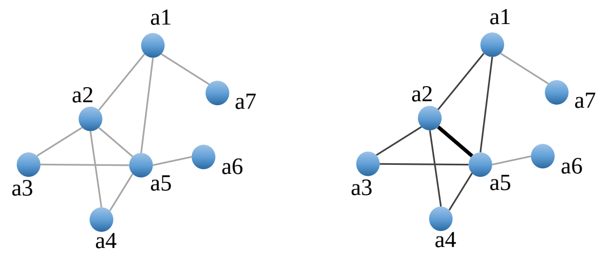


Fig. 3. Illustration of motif centrality. Left: original unweighted graph. Right: weighted graph according to triangle motifs. Thickness of edges represents edge weight.

We leverage the triangle motif to calculate the motif centrality as it is the most simple and representative higher-order graph structure and has been wildly applied in the literature [21], [60], [69], [70]. Specifically, for the unweighted original graph, we count the number of triangle motifs to which each edge belongs, and use this frequency as edge weight. Like in Fig. 3, (a_2, a_5) belongs to three triangle motifs (a_2, a_5, a_3), (a_2, a_5, a_4), (a_2, a_5, a_1), so the edge weight between (a_2, a_5) is 4 while edge (a_1, a_7) does not belong to any triangle motif, its edge weight remains 1. Then the edge dropping can be applied according to the weights as centrality measure.

TABLE 2
Demo experiment about the effect of the two improvements on Cora.

Pipeline	GCA	GCA$_{vr1}$	GCA$_{vr2}$	GCA$_{vr3}$
Accuracy(%)	78.10	80.10	79.40	80.70

A demo empirical study on real-world dataset Cora (Table 2) shows that our two improvements, i.e., (1) motif centrality, (2) drop more important edges with a higher probability, can both improve the performance of GCA. Here, GCA is the initial GCA model with degree centrality adaptive augmentation which performs the best over degree centrality, eigenvalue centrality, and PageRank centrality proposed in [17], GCA$_{vr1}$ is the variant with (1), GCA$_{vr2}$ is the variant with (2) and GCA$_{vr3}$ is the variant with both (1) and (2). Detailed experiments can be found in Section 5.

4.3 Fusion Strategy

Feature graph and topology graph generated from $k$nn are extracted by two GNN encoders to get embeddings $\mathbf{F}$ and $\mathbf{T}$ dedicated for node features and graph structures. Node embeddings $\mathbf{H}_1$ and $\mathbf{H}_2$ are also generated via a shared GNN encoder from a GCL module with motif centrality-based augmentations. Now we need to design a module to fuse them such that (1) InfoNCE loss is still adoptable to preserve the GCL structure; (2) graph reconstruction can be easily applied; (3) information from both sides can be fused automatically. We design a simple *AttnFuse* based on attention mechanism to fuse symmetric embedding pairs $(\mathbf{T}, \mathbf{H}_1)$ and $(\mathbf{F}, \mathbf{H}_2)$ to form the final embeddings via $\mathbf{Z}_1 = AttnFuse_1(\mathbf{T}, \mathbf{H}_1)$ and $\mathbf{Z}_2 = AttnFuse_2(\mathbf{F}, \mathbf{H}_2)$.

And we take the formation of $\mathbf{Z}_1$ as an example:

$$\omega_T = \mathbf{q}^{\mathrm{T}} \tanh(\mathbf{W}\mathbf{T}^{\mathrm{T}} + \mathbf{b}), \tag{7}$$

$$\omega_{H_1} = \mathbf{q}^{\mathrm{T}} \tanh(\mathbf{W}\mathbf{H}_1^{\mathrm{T}} + \mathbf{b}), \tag{8}$$

$$\alpha_T = \mathrm{softmax}(\omega_T) = \frac{\exp(\omega_T)}{\exp(\omega_T) + \exp(\omega_{H_1})}, \tag{9}$$

$$\alpha_{H_1} = \mathrm{softmax}(\omega_{H_1}) = \frac{\exp(\omega_{H_1})}{\exp(\omega_T) + \exp(\omega_{H_1})}, \tag{10}$$

$$\mathbf{Z}_1 = \alpha_T \mathbf{T} + \alpha_{H_1} \mathbf{H}_1. \tag{11}$$

Here, $\mathbf{q} \in \mathbb{R}^{h' \times 1}$ is the shared attention vector, $\mathbf{W} \in \mathbb{R}^{h' \times h}$ is a trainable weight matrix and $\mathbf{b} \in \mathbb{R}^{h' \times 1}$ is a bias vector. The fusion of $AttnFuse(\mathbf{F}, \mathbf{H}_2)$ is performed in the same way and we can get $\mathbf{Z}_2$. Now, $\mathbf{z}_{1_i}$ and $\mathbf{z}_{2_i}$ can be used to substitute $\mathbf{u}_i$ and $\mathbf{v}_i$ in Eq. (4).

4.4 Objective Function

In order to introduce graph autoencoder to GCL, we devote to considering the GCL loss InfoNCE and reconstruction loss in a multi-task learning way.

4.4.1 *Contrastive objective function*

Based on the single node objective loss Eq. (4), we can define the overall contrastive loss function following [17] as:

$$\mathcal{L}_{cont} = \frac{1}{2N} \sum_{i=1}^{N} [l_1(\mathbf{z}_{1_i}, \mathbf{z}_{2_i}) + l_2(\mathbf{z}_{2_i}, \mathbf{z}_{1_i})]. \tag{12}$$

4.4.2 *Reconstruction objective function*

The key to graph autoencoder is to leverage graph property reconstruction as the objective function. Here, hence we have already decoupled the original attributed graph into a topology graph and a feature graph, it is natural to reconstruct the original edge and attribute based on the two decoupled graphs. And the reconstruction losses can be conjoint with the contrastive learning loss in a multi-task learning way. From another perspective, the two reconstruction losses can be interpreted as two regulation terms to restrict the contrastive learning [24]. Specifically, to maintain the model's simplicity and make the analysis easier, we adopt vanilla but effective designs. For edge reconstruction, we use inner production to implement the edge generator on topology graph embeddings as:

$$\tilde{\mathbf{A}} = \mathrm{softmax}(\mathbf{T} \cdot \mathbf{T}^{\mathrm{T}}), \tag{13}$$

Therefore, the loss function for training this edge generator can be defined as:

$$\mathcal{L}_{edge} = \frac{1}{N} \|\mathbf{A} \odot (\tilde{\mathbf{A}} - \mathbf{A})\|^2, \tag{14}$$

where $\odot$ is the hadmard product such that $\mathcal{L}_{edge}$ only pays attention to edges existing in the original graph following [71], [72]. Similarly, we can define the loss function for raw attribute reconstruction as the mean squared error:

$$\mathcal{L}_{feat} = \frac{1}{N} \sum_{i \in \mathcal{V}} \|\tilde{\mathbf{x}}_i - \mathbf{x}_i\|^2, \tag{15}$$

where $\tilde{\mathbf{X}} = \mathrm{MLP}(\mathbf{F})$ is the reconstructed raw attributes using feature graph embeddings.

Algorithm 1 Training process for AEGCL

```
**Input:** $\mathcal{G} = (\mathbf{X}, \mathbf{A})$
**Output:** trained GCN encoder $f(\mathbf{X}, \mathbf{A}; \mathbf{\Theta})$
 1: Perform node2vec on $\mathbf{A}$ to get $\mathbf{E}$
 2: Perform $k$nn on $\mathbf{X}$, $\mathbf{E}$ to get $\mathcal{G}_f$, $\mathcal{G}_t$
 3: while not converge do
 4:     Obtain embedding $\mathbf{F}$ of $\mathcal{G}_f$ using encoder $f_f$
 5:     Obtain embedding $\mathbf{T}$ of $\mathcal{G}_t$ using encoder $f_t$
 6:     Generate two graph augmentations $\mathcal{G}_1$, $\mathcal{G}_2$ via adaptive motif centrality
 7:     Obtain embeddings $\mathbf{H}_1$, $\mathbf{H}_2$ using encoder $f$
 8:     Obtain aggregated embeddings $\mathbf{Z}_1$, $\mathbf{Z}_2$ using $AttnFuse_1(\mathbf{F}, \mathbf{H}_1)$ and $AttnFuse_2(\mathbf{T}, \mathbf{H}_2)$
 9:     Back propagate via Eq. (12), Eq. (14), Eq. (15), Eq. (16)
10: end while
```

Combining the two constraints as regulation terms with the contrastive loss function, we get the following overall objective function:

$$\mathcal{L} = \mathcal{L}_{cont} + \beta \mathcal{L}_{edge} + \gamma \mathcal{L}_{feat}, \tag{16}$$

where β and γ are weight parameters of the constraint terms. The back propagation is adopted to solve the optimization of the overall objective function in an end-to-end manner without any labeled data.

4.5 Complexity Analysis

The entire training process of AEGCL is shown in Algorithm 1, based on which we analyze the time complexity. It is worth mentioning that the feature graph $\mathcal{G}_f$ and the topology graph $\mathcal{G}_t$ in line 1-2 could be obtained before training. Assuming h is the hidden dimension, L is the number of GCN layer, $|\mathcal{E}|$ is the number of edges, and N is the number of nodes. For line 4-7, $\mathcal{G}_f$, $\mathcal{G}_t$, $\mathcal{G}_1$, $\mathcal{G}_2$ are fed into GNNs with time complexity $\mathcal{O}(hL|\mathcal{E}| + Nh^2L)$, as GNN costs $\mathcal{O}(h|\mathcal{E}|)$ to propagate features and $\mathcal{O}(Nh^2)$ to multiply by the weight matrix in each layer. For line 8, the time complexity for each $AttnFuse$ module is $\mathcal{O}(Nh)$. Therefore, a forward pass of AEGCL has a time complexity of $\mathcal{O}(hL|\mathcal{E}| + Nh^2L + Nh)$. As L is always small in GCN (e.g., $L = 2$ in our case), the time complexity is deduced to $\mathcal{O}(h|\mathcal{E}| + Nh^2 + Nh) = \mathcal{O}(h|\mathcal{E}| + Nh^2)$, which is lightweight.

5 Experiments

In this section, extensive experiments are conducted to evaluate the effectiveness and robustness of AEGCL.

5.1 Experimental Setup

5.1.1 *Datasets*

The main experiments are conducted on six publicly available 10,000-scale real-world graph benchmarks widely used in the literature from PyTorch Geometric library [73]. The statistical descriptions of these datasets are shown in Table 3. We give a brief description of the datasets as follows.

- **Amazon-Photo** and **Amazon-Computers** [74] are two co-purchase networks constructed from Amazon where nodes represent products and edges represent

TABLE 3
Statistics of datasets used in experiments.

Datasets	#Nodes	#Edges	#Features	#Classes
Amazon-Photo	7,650	119,081	745	8
Amazon-Computers	13,752	245,861	767	10
Coauthor-CS	18,333	81,894	6,805	15
PubMed	19,717	44,338	500	3
DBLP	17,716	105,734	1,639	4
CoraFull	19,793	65,311	8,710	70

co-purchase relations. Each product node has a raw bag-of-words feature encoding product reviews and is labeled with its category.

- **Coauthor-CS** [74] is an academic network where nodes represent authors and edges represent co-author relationships. Each author node has a raw bag-of-words feature encoding keywords of his/her publication and is labeled with the most related research field.
- **PubMed** [75], **DBLP**, and **CoraFull** [76] are three academic networks where nodes represent papers and edges represent citation relations. Each paper has a raw bag-of-words feature encoding its keywords and is labeled with its related research area.

5.1.2 Baselines

We compare AEGCL with two types of baselines, namely self-supervised approaches and semi-supervised approaches to show its effectiveness. Self-supervised methods include node2vec [36], Graph Rec [22], GraphSAGE [77], GAE [12], VGAE [12], DGI [44], GMI [45], MVGRL [46], GRACE [47], ARIEL [11], SimGCL [43], GCA [17], and BGRL [48]. Semi-supervised methods include GCN [1], GAT [2], GraphBert [22], and LINKX [20]. We also report the performance obtained using an MLP classifier on raw node features (Raw features).

5.1.3 Configurations

All the baselines are tested based on official/popular implementations with parameters suggested by their papers, and we also further tune them to get optimal performance. For our AEGCL, we set the hidden layer dimension of GCN as $\{128, 256, 512\}$ and output layer dimension as $\{64, 128, 256\}$. We use Adam as the optimizer with learning rate as $\{0.0005, 0.001, 0.01\}$ and weight decay as $\{1e-4, 1e-5\}$. The candidate activation functions in GCN are $\{ReLU, LeakyReLU, PReLU, RReLU\}$. The weights β and γ for $\mathcal{L}_{edge}$ and $\mathcal{L}_{feat}$ are searched in $\{0, 1e-3, 1e-2, \dots, 1e+3\}$, and k in k-nearest neighbor graph is searched in $\{1, 2, \dots, 10\}$. The detailed hyper-parameter specification for each dataset is elaborated in Table 4 for repeatability. Here, hid_dim, out_dim, attn_dim and n2v_dim correspond to the dimensions of the hidden layer of GCN encoder, the output layer of GCN encoder, the attention vector, and node2vec embeddings respectively; τ is the temperature parameter in Eq. (4); lr and weight-decay are the learning rate and weight decay of Adam optimizer; k, β and γ are three hyper-parameters which will be discussed in Section 5.5. The evaluation metric in term of node classification accuracy is reported for all the experiments.

5.2 Node Classification

As self-supervised approaches and semi-supervised approaches are both leveraged as baseline methods, for a fair comparison, we also adopt two modes of AEGCL, i.e., freezing mode AEGCL and full fine-tuning mode AEGCL-ft, as discussed in Section 3. The main difference between the two modes is that, after the self-supervised model training with the whole graph in a transductive setting, we apply the learned *embeddings* directly to downstream tasks in the freezing mode, while in the full fine-tuning mode, we treat the self-supervised learning as the *pre-training phase* imitating Bert [25], and then fine-tune its parameters based on the downstream node classification task. In other words, data labels are agnostic to the model in the freezing mode, while a fraction of them (training set) are visible in the full fine-tuning mode. In the evaluation phase, a $l2$-regularized MLP is leveraged in both modes as the classification head with Adam optimizer of learning rate 0.01 for the downstream task as in [17], [41]. For the PubMed dataset, we apply public splits [1] to split it into training/validation/testing sets for the downstream node classification tasks. For the other five datasets, we randomly split them into training/validation/testing sets with proportions 10%, 10%, 80% respectively following [17], [48] since there are no publicly accessible splits. We follow [78] to report the testing set performance when the performance on the validation set achieves the best result. The results in terms of accuracy are reported in Table 5 for the freezing mode and Table 6 for the full fine-tuning mode, respectively. Note that GCA-DE($vr.$) is the GCA-DE variant which drops more important edges with a higher probability as proposed in Section 4.2.2, and GCA-MO is the GCA variant with motif centrality-based adaptive augmentation.

Based on both tables, we can see that our proposed method in freezing mode (AEGCL) and in full fine-tuning mode (AEGCL-ft) both outperform all the baseline methods on almost all datasets except on CoraFull in the freezing mode.

Specifically, for the freezing mode, adaptive augmentations GCA-DE, GCA-EV, and GCA-PR can boost the performance of random edge dropping and attribute masking. And our proposed motif centrality-based augmentation GCA-MO can further improve model performance, which we attribute to motif centrality can capture higher-order graph semantic structure. Besides, we find that GCA-DE ($vr.$) generally outperforms GCA-DE, which demonstrates the effectiveness of dropping more important edges as discussed in Section 4.2.2. Overall, AEGCL can achieve consistent improvements over all datasets, especially on PubMed with almost 2% performance gain, which demonstrates the stability and universality of combining graph autoencoder with GCL.

For the full fine-tuning mode, it is intuitive that the semi-supervised methods/full fine-tuned methods generally perform better than self-supervised methods or their self-supervised counterparts for that a fraction of labeled data is accessible during training. And we can see that the

TABLE 4
Specified model Hyper-parameters.

Datasets	hid_dim	out_dim	attn_dim	n2v_dim	σ	τ	lr	weight-decay	epochs	k	β	γ
Amazon-Photo	512	256	16	128	$ReLU$	0.3	0.001	1e-5	2,000	5	1	1
Amazon-Computers	256	128	16	128	$PReLU$	0.2	0.01	1e-5	2,000	5	0.1	0.001
Coauthor-CS	512	256	16	128	$PReLU$	0.4	0.0005	1e-5	2,000	6	0.001	0.01
PubMed	512	256	16	128	$ReLU$	0.2	0.001	1e-5	2,000	7	1	1
DBLP	512	256	16	128	$PReLU$	0.4	0.0005	1e-5	1,500	6	1	0.1
CoraFull	256	128	16	128	$ReLU$	0.2	0.001	1e-5	1,500	5	0.1	0.1

TABLE 5
Node classification results (%) and their standard deviations for 5 runs on six real-world datasets obtained by the self-supervised learning methods. The best results are highlighted in **boldface**. $\mathbf{X}$ and $\mathbf{A}$ correspond to node features and graph adjacency matrix respectively. OOM indicates Out-Of-Memory on a 24GB GeForce GTX 3090 GPU.

Method	Training Data	Amazon-Photo	Amazon-Computers	Coauthor-CS	PubMed	DBLP	CoraFull
Raw features	$\mathbf{X}$	72.27±0.00	73.81±0.00	90.37±0.00	73.42±0.00	71.62±0.00	31.37±0.00
node2vec	$\mathbf{A}$	87.76±0.10	84.39±0.08	85.08±0.03	62.13±0.05	77.14±0.12	49.49±0.12
Graph Rec	$\mathbf{X}, \mathbf{A}$	77.81±0.21	82.48±0.38	90.39±0.17	72.78±0.26	75.60±0.33	22.13±0.19
GAE	$\mathbf{X}, \mathbf{A}$	91.62±0.13	85.27±0.19	90.01±0.71	73.74±0.28	79.73±0.47	54.25±0.28
VGAE	$\mathbf{X}, \mathbf{A}$	92.20 ±0.11	86.37±0.21	92.11±0.09	75.17±0.18	80.81±0.20	55.82±0.31
DGI	$\mathbf{X}, \mathbf{A}$	91.61±0.22	83.95±0.47	92.15±0.63	76.78±0.33	79.51±0.86	55.38±0.39
GMI	$\mathbf{X}, \mathbf{A}$	90.68±0.17	82.21±0.31	OOM	79.90±0.49	78.43±0.77	55.97±0.28
MVGRL	$\mathbf{X}, \mathbf{A}$	91.74±0.07	87.52±0.11	92.11±0.12	79.35±0.18	81.26±0.26	57.08±0.19
BGRL	$\mathbf{X}, \mathbf{A}$	91.09±0.33	88.21±0.39	91.97±0.20	80.03±0.42	82.43±0.55	57.42±0.25
GRACE	$\mathbf{X}, \mathbf{A}$	91.13±0.27	87.24±0.37	92.68±0.12	79.46±0.26	82.59±0.93	**58.21**±0.26
ARIEL	$\mathbf{X}, \mathbf{A}$	91.80±0.24	87.07±0.33	92.39±0.28	74.23±0.82	81.01±0.22	57.41±0.73
SimGCL	$\mathbf{X}, \mathbf{A}$	91.09±0.34	82.58±0.27	92.34±0.20	75.20±0.93	78.18±0.35	54.59±0.88
GCA-DE	$\mathbf{X}, \mathbf{A}$	91.36±0.20	87.59±0.33	92.70±0.17	79.43±0.23	82.62±0.77	57.62±0.16
GCA-EV	$\mathbf{X}, \mathbf{A}$	91.34±0.25	87.64±0.47	92.73±0.15	79.38±0.32	82.72±0.83	57.57±0.20
GCA-PR	$\mathbf{X}, \mathbf{A}$	91.25±0.28	87.49±0.43	92.69±0.10	79.35±0.24	82.61±0.86	57.73±0.23
GCA-MO	$\mathbf{X}, \mathbf{A}$	91.53±0.31	87.68±0.37	92.71±0.08	79.58±0.27	82.95±0.86	57.79±0.15
GCA-DE ($vr.$)	$\mathbf{X}, \mathbf{A}$	91.47±0.25	87.62±0.42	92.68±0.10	79.78±0.31	82.75±0.66	57.68±0.18
AEGCL	$\mathbf{X}, \mathbf{A}$	**92.65**±0.32	**88.38**±0.25	**92.76**±0.36	**80.42**±0.40	**84.03**±0.51	57.86±0.29

TABLE 6
Node classification results (%) and their standard deviations for 5 runs on six real-world datasets obtained by semi-supervised/fine-tuned methods. The best results are highlighted in **boldface**. $\mathbf{X}$, $\mathbf{A}$ and $\mathbf{Y}$ correspond to node features, graph adjacency matrix and node labels respectively.

Method	Training Data	Amazon-Photo	Amazon-Computers	Coauthor-CS	PubMed	DBLP	CoraFull
GCN	$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	92.42±0.22	86.51±0.54	93.03±0.31	78.89±0.30	79.17±0.60	56.91±0.28
GAT	$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	92.56±0.35	86.93±0.29	92.31±0.24	78.93±0.30	78.56±0.46	45.23±0.30
GraphBert	$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	90.69±0.06	85.27±0.17	91.27±0.10	79.30±0.07	76.28±0.15	25.37±0.18
LINKX	$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	84.46±0.83	81.89±0.64	78.16±0.69	58.13±0.30	72.36±0.52	37.37±0.88
GRACE-ft	$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	92.82±0.17	89.38±0.23	93.24±0.15	78.35±0.18	82.91±0.42	63.37±0.27
GCA-DE-ft	$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	92.78±0.14	89.35±0.18	93.32±0.10	78.55±0.24	83.02±0.36	63.35±0.15
GCA-EV-ft	$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	92.74±0.21	89.41±0.21	93.28±0.09	78.50±0.22	82.88±0.30	63.42±0.21
GCA-PR-ft	$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	92.63±0.15	89.25±0.26	93.24±0.13	78.51±0.29	82.95±0.31	63.44±0.18
AEGCL-ft	$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	**93.35**±0.30	**89.89**±0.18	**93.50**±0.31	**80.53**±0.55	**84.10**±0.72	**64.08**±0.39

AEGCL variant in the full fine-tuning mode (AEGCL-ft) again performs the best. Besides, it is interesting to see that LINKX which is also dedicated to decomposing the graph into orthogonal feature and topology components does not perform as we expect. We analyze the reason lies in that it adopts simple MLP instead of GNN to encode the two-fold information, which damages the representation ability.

5.3 Ablation Study

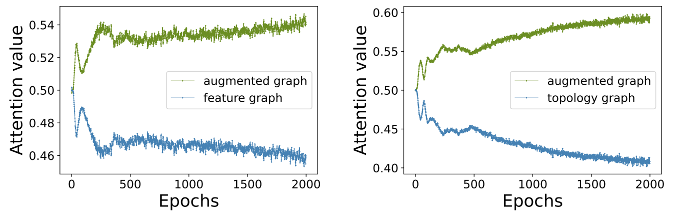
We conduct a series of ablation studies to show how each component affects the model. Note that the comparison between GRACE, GCA-DE, GCA-EV, GCA-PR, GCA-MO, and GCA-DE($vr.$) in Table 5 has demonstrated the effectiveness of dropping more important edges and motif centrality-based adaptive augmentation. Here, we further conduct the ablation study by zooming in on three aspects, namely the reconstruction method, graph augmentation pattern, and aggregation pattern. The results are shown in Table 7, where $vr1$ to $vr6$ means six variants of the AEGCL method.

5.3.1 Graph autoencoder

Two folds of graph autoencoder are leveraged in AEGCL, i.e., edge reconstruction and raw feature reconstruction.

TABLE 7
Ablation study results (%) on five real-world datasets.

Variants	Rec. Method		Aug. Pattern		Aggrga. Pattern		Amazon-Photo	Amazon-Computers	Coauthor-CS	PubMed	DBLP
	edge	feat.	motif	other	*AttnFuse*	mean					
$vr1$	✓	✓	-	-	-	-	71.42	80.42	88.63	72.60	75.59
$vr2$	-	-	✓	-	-	-	91.40	87.64	92.73	79.50	82.72
$vr3$	✓	-	✓	-	✓	-	92.35	87.30	92.57	79.30	81.09
$vr4$	-	✓	✓	-	✓	-	92.14	86.98	92.66	77.50	83.31
$vr5$	✓	✓	-	✓	✓	-	92.37	86.89	92.67	80.30	81.85
$vr6$	✓	✓	✓	-	-	✓	92.43	85.89	92.51	79.70	81.85
AEGCL	✓	✓	✓	-	✓	-	92.65	88.38	92.76	80.40	84.03



(a) Changing trend of attention value in *AttnFuse1* (b) Changing trend of attention value in *AttnFuse2*

Fig. 4. The attention changing trends w.r.t. epochs on Amazon-Photo.

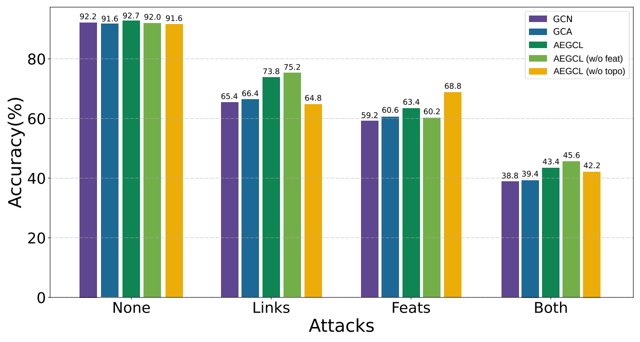


Fig. 5. Network attack results on Amazon-Photo.

Here, we test their effectiveness by disabling either or both of them, by setting β or γ to 0.

From the comparison between AEGCL and $vr2/vr3/vr4$, we can see that simultaneously considering two-fold graph autoencoder tasks — edge reconstruction and feature reconstruction — boosts the model performance. And different datasets show different sensitivity of the two reconstruction terms, possibly due to intrinsic data distribution. Quantitative analysis of β and γ is elaborated in Section 5.5.

5.3.2 Graph augmentation pattern

We also test how the motif centrality-based adaptive augmentation method contributes to the final performance of AEGCL. Specifically, we test its performance against degree centrality, eigenvector centrality, and PageRank centrality-based adaptive graph augmentation methods proposed in GCA. And the item "other" in the table means the best variant of the three.

From the comparison of $vr5$ and AEGCL, we can see that our motif centrality outperforms other adaptive graph augmentation methods.

5.3.3 Aggregation pattern

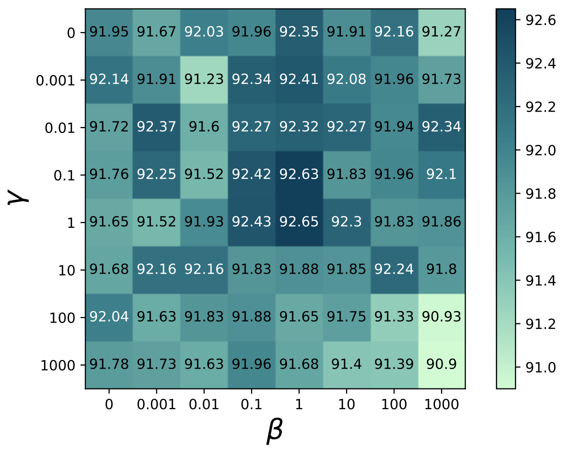
We also conduct experiments to validate the effectiveness of the attention-based *AttnFuse* module when aggregating node feature embeddings $\mathbf{F}$ and graph topology embeddings $\mathbf{T}$ with augmented node embeddings $\mathbf{H}_1$ and $\mathbf{H}_2$ from graph contrastive learning methods. And we compare *AttnFuse* against mean operator. We also show the changing trends of the attention value in *AttnFuse* on the Amazon-Photo dataset, as shown in Fig. 4.

From the comparison between $vr6$ and AEGCL in Table 7, we can see that the *AttnFuse* module generally performs better than the mean operator, for that it can leverage the important information from topology/feature graph and augmented graph counterparts automatically. From Fig. 4, we can see that the attention of topology graph and feature graph together with the two augmented graphs converge to distinct values, which illustrates that the two augmented graphs play different weight roles, and the *AttnFuse* module can distinguish them automatically.

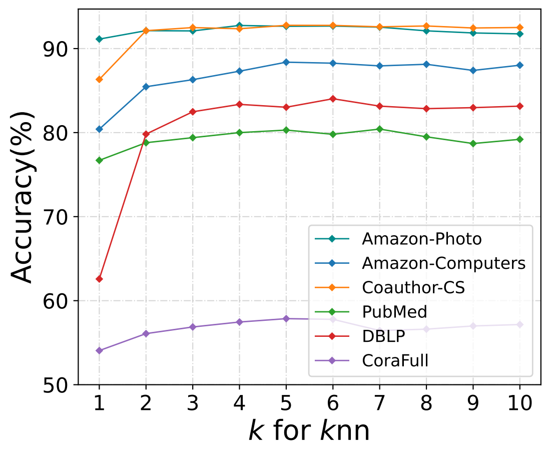
Besides, we also design a variant ($vr1$) which does not apply contrastive learning but only leverages two reconstruction tasks. And we can see that $vr1$ performs the worse. It demonstrates the effectiveness of the whole contrastive learning schema.

5.4 Network Attack

Since the proposed AEGCL method leverages the node feature reconstruction and graph topology reconstruction explicitly, it is intuitive to wonder to what extent can the two-fold information be preserved, i.e., the robustness of AEGCL. Therefore, experiments are conducted under the graph attack dedicated to node features or graph topology, or both. Here, we apply single-node direct evasion attack *nettack* proposed in [49], where the trained GNN encoder $f(\mathbf{X}, \mathbf{A}; \mathbf{\Theta})$ remains unchanged throughout the attack. The attacker *nettack* takes the original node features $\mathbf{X}$, the original graph adjacency matrix $\mathbf{A}$, the node labels $\mathbf{Y}$, the target node v_t and the trained GNN encoder $f(\mathbf{X}, \mathbf{A}; \mathbf{\Theta})$ as



(a) Analysis of parameter β, γ



(b) Analysis of parameter k for $k$nn

Fig. 6. Hyper-parameter analysis of AEGCL w.r.t. weight coefficients β, γ and k for $k$nn on Amazon-Photo.

inputs and generates perturbed (attacked) node features $\mathbf{X}'$ and graph adjacency matrix $\mathbf{A}'$, i.e.,

$$\mathbf{X}', \mathbf{A}' = nettack(\mathbf{X}, \mathbf{A}, v_n, f(\mathbf{X}, \mathbf{A}; \mathbf{\Theta})), \tag{17}$$

then the embeddings of the attacked graph is

$$\mathbf{Z}' = f(\mathbf{X}', \mathbf{A}'; \mathbf{\Theta}). \tag{18}$$

During the experiment, we set the attack perturbation intensity $n_{perturb}$ to 1 and randomly select 500 nodes as the attack target. The results on the Amazon-Photo dataset are shown in Fig. 5. Here, AEGCL (w/o feat) stands for the variant which only aggregates topology graph with graph augmentations, and AEGCL (w/o topo) stands for the variant which only aggregates feature graph with graph augmentations.

From the figure, we can see that when suffering the attack, the performance of the semi-supervised method namely GCN drops the fastest, which conforms to the conclusion in [51]. For the other five self-supervised methods, the robustness of GCA is the worst, as it does not train the two-fold node feature and graph topology information explicitly. By contrast, the performances of AEGCL (w/o feat), AEGCL (w/o topo), and AEGCL are better. When leveraging only node feature or graph topology information during training, we can see that the performance of AEGCL (w/o feat) shows superior result under sole link attack while AEGCL (w/o topo) shows superior result under sole feature attack. Although it would be premature to conclude that the topology reconstruction-based and raw feature reconstruction-based graph autoencoders are bound to improve model robustness since only *nettack* is tested, the results do demonstrate that the model stability is preserved when facing single-node direct evasion attack. And we attribute it to leveraging the feature and topology information explicitly.

5.5 Hyper-parameter Analysis

We conduct this experiment to investigate the sensitivity of AEGCL on different hyper-parameters, i.e., multi-task learning parameters β, γ, and k in $k$nn leveraged to construct feature graph and topology graph.

5.5.1 Analysis of weight coefficients β and γ

We test the coefficients for edge and feature reconstruction weights β and γ in Eq. (16) by varying both of them in range $\{0, 0.001, 0.01, 0.1, 1, 10, 100, 1000\}$. The results are shown in Fig. 6(a) for Amazon-Photo, from which we can see that with the increase of single β or γ, the performances show a firstly-raising-then-dropping trend. The best performance is obtained when β and γ are in range $\{0.1, 1, 10\}$. Generally speaking, the model is stable for that its relative performance perturbation range is only 1.76% w.r.t. β and γ.

5.5.2 Analysis of k-nearest neighbor graph k

We need to sample k neighbor nodes for each node in raw feature space and node2vec embedding space for the feature graph and topology graph. Here, we test k in the range from 1 to 10 for all the datasets in Fig. 6(b). From the figure, we can see that the accuracy is relatively low when k=1 for all the datasets, especially on DBLP. By increasing k, the accuracy rises, and it becomes stable when k is in the range from 4 to 7, after which it starts to drop a little. This may be because when k is low, the generated feature graph and topology graph are not informative enough as few neighbors are considered. And when k is too large, more noisy neighbors are introduced to the two graphs, which might confuse the encoder when distilling feature or topology information. We conduct a hyper-parameter search of k in the range from 4 to 7 in practice for the six datasets, which is given in Table 4.

5.6 Indepth Analysis about AEGCL

5.6.1 Analysis of feature and topology graph construction

Apart from applying the $k$nn algorithm, which is wildly adopted in the literature [19] to construct the feature and topology graphs, we also explore other methods for the graph construction, including fixed threshold and learnable threshold. Given pairwise similarity matrices of node2vec embeddings and raw node attributes, the fixed threshold is implemented via selecting the top-k most similar node pairs as generated edges, where k is the number of edges in the raw graph; and the learnable threshold is implemented

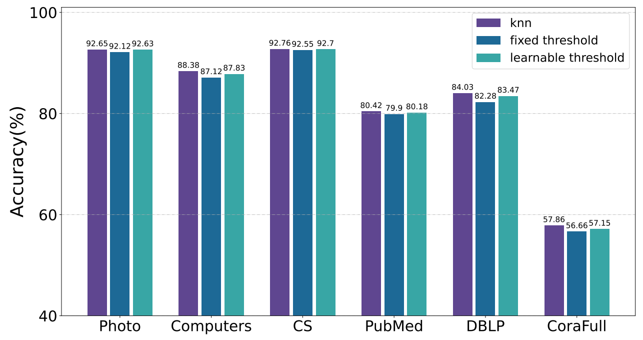


Fig. 7. Comparision about different methods to construct feature and topology graphs.

TABLE 8
Analysis of different motifs on six real-world datasets. Node classification accuracy (%) is reported.

Datasets	3-clique	4-clique	5-clique
Amazon-Photo	92.65	92.58	92.21
Amazon-Computers	88.38	87.76	87.58
Coauthor-CS	92.76	92.70	92.61
PubMed	80.42	79.97	79.83
DBLP	84.03	82.71	82.12
CoraFull	57.86	57.68	57.48

via training a dynamic threshold where node pairs whose similarities are larger than the threshold will be generated as edges. The results are illustrated in Fig. 7. From the figure we can see that the $k$nn method still achieves the best performance, and the fixed threshold and learnable threshold methods perform slightly worse than the $k$nn method. We attribute this to the fact that by setting a threshold, the constructed graphs easily degenerate to the original graph, which may not provide sufficient information for training the model.

5.6.2 *Analysis of other motifs*

Apart from the triangle motif (3-clique motif), we also conduct an experiment on other motifs, including 4-clique motif and 5-clique motif, which is shown in Table 8. From the table, we can see that the accuracy drops from 3-clique motif to 5-clique motif on the six datasets. We attribute this phenomenon to the fact that the complex 5-clique motif is scarcer than the simple 3-clique motif, which results in the motif centrality being more uniform across the graph. Nevertheless, we can see that using the 5-clique motif can still achieve competitive results compared with the baselines.

5.7 Discussion about Model Generalizability

We discuss the generalizability of AEGCL in this section focusing on other node-level downstream tasks and other kinds of networks, like protein network, which is highly dependent on the graph geometry.

5.7.1 *Node clustering*

Follwing [79], we conduct node clustering by training a k-means algorithm on the node embeddings learned by different models, where k is set to the number of classes. The results in terms of normalized mutual information (NMI) and adjusted rand index (ARI) on Amazon-Computers are reported in Table 9. It is clear to see that AEGCL outperforms other models on both metrics, which we attribute to the fact that the two reconstruction functions facilitate tighter intra-cluster bonds.

TABLE 9
Node clustering results in terms of NMI and ARI on the Amazon-Computers dataset.

Method	NMI	ARI
GAE	0.441±0.000	0.258±0.000
VGAE	0.423±0.000	0.238±0.000
DGI	0.318±0.002	0.165±0.002
MVGRL	0.244±0.000	0.141±0.000
BGRL	0.484±0.000	0.295±0.000
GCA	0.426±0.000	0.246±0.000
AEGCL	**0.486**±0.001	**0.308**±0.002

TABLE 10
Link prediction results in terms of Area Under the Receiver Operating Characteristic Curve (ROC AUC) (%) and Average Precision (AP) (%) on the Amazon-Photo dataset.

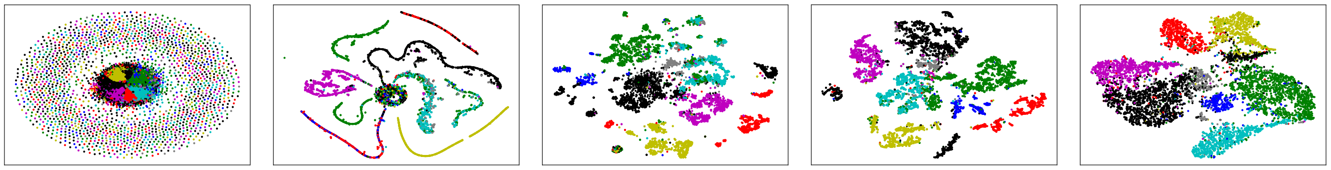
Method	ROC AUC	AP
Raw features	88.44±0.09	85.86±0.06
node2vec	96.31±0.10	92.37±0.13
Graph Rec	96.90±0.09	96.42±0.11
GAE	89.59±0.14	89.29±0.08
VGAE	85.73±0.13	85.80±0.09
GCA	98.21±0.05	97.88±0.07
AEGCL	**98.26**±0.05	**97.95**±0.06

TABLE 11
Node classification results in terms of Micro-F1 (%) on inductive task with protein network PPI.

Method	Training Data	PPI
Raw features	$\mathbf{X}$	42.2
GraphSAGE-GCN	$\mathbf{X}, \mathbf{A}$	46.5
GraphSAGE-mean	$\mathbf{X}, \mathbf{A}$	48.6
GraphSAGE-LSTM	$\mathbf{X}, \mathbf{A}$	48.2
GraphSAGE-pool	$\mathbf{X}, \mathbf{A}$	50.2
DGI	$\mathbf{X}, \mathbf{A}$	63.80±0.20
GMI	$\mathbf{X}, \mathbf{A}$	65.00±0.02
BGRL	$\mathbf{X}, \mathbf{A}$	68.98±0.14
GRACE	$\mathbf{X}, \mathbf{A}$	66.20±0.10
AEGCL	$\mathbf{X}, \mathbf{A}$	**69.37**±0.18
GAT	$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	97.30±0.20

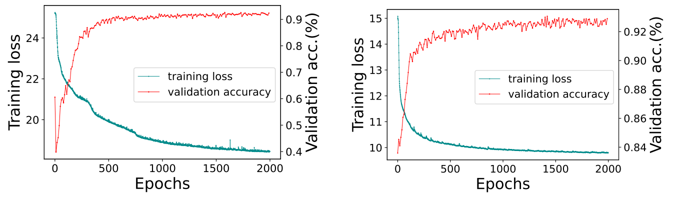
5.7.2 *Link prediction*

Following [12], we split the edges into training/validation/testing sets with proportions 70%, 20%, 10%, respectively, and train a linear classifier with multi-class binary cross entropy loss based on the embeddings learned by different models. The results in terms of Area Under the Receiver Operating Characteristic Curve (ROC AUC) and Average Precision (AP) on Amazon-Photo are reported in Table 10, where AEGCL achieves the best performance. We attribute it to the topology reconstruction function which directly targets at link structure, as the



(a) Raw features (b) Graph reconstruction (c) node2vec (d) GCA (e) AEGCL

Fig. 8. Visualization of the pre-trained embeddings on Amazon-Photo. Each point indicates a product and its color indicates the category.



(a) Amazon-Photo (b) Coauthor-CS

Fig. 9. The training procedure on Amazon-Photo and Coauthor-CS.

simple baseline Graph Rec also achieves outstanding performance.

5.7.3 Experiment on protein network

Apart from the six datasets from co-purchase networks, co-author networks, and citation in Section 5.2, we also conduct an experiment on protein network PPI [80] as it is highly dependent on the graph geometry. Furthermore, it is an inductive task where 20 graphs are for training, 2 graphs are for validation, and 2 graphs are for testing following [77]. For this dataset, each node has multiple labels that are a subset of gene ontology sets (121 in total), and node features include positional gene sets, motif gene sets, and immunological signatures (50 in total). The node classification results in terms of Micro-F1 is elaborated in Table 11, from which we can see that AEGCL still performs the best compared to other self-supervised learning methods, which shows the generalizability of AEGCL across different type of graphs. Interestingly, there is a huge gap between self-supervised methods and supervised methods like GAT [2], as supervision is necessary for multi-label classification tasks. It is a potential future direction to reduce the performance gap between supervised and self-supervised learning paradigms for multi-label classification tasks.

5.8 Visualization

Besides statistical analysis, we also visualize the embeddings learned by some self-supervised methods to achieve an intuitive understanding. Specifically, we utilize t-SNE [81] to project the node embeddings learned by graph reconstruction, node2vec, GCA, and AEGCL together with raw features on Amazon-Photo into a 2-dimensional space. The results are shown in Fig.8, where each node is colored by its label.

According to the figure, we can see that graph reconstruction and node2vec struggle to present clear clusters as they have blurry boundaries. For GCA, nodes of different classes are still scattered to some degree. AEGCL performs the best apparently, for that Fig. 8(e) shows a compact structure and relatively clear boundaries.

Besides, to better understand the training process, we also visualize the training records on two datasets, i.e., Amazon-Photo and Coauthor-CS to show their converge trends, as shown in Fig. 9. Both figures show training loss and classification accuracy (%) on validation set w.r.t. training epochs. We can see that the training process is rather stable for that training loss drops and validation accuracy increases steadily on both datasets. It takes around 500 epochs for both datasets to converge, and the validation accuracy perturbs on a small scale afterward. We reduce this influence by reporting the test accuracy when the best validation accuracy is obtained.

6 Conclusion and Future Work

In this paper, motivated to improve the effectiveness and robustness of graph contrastive learning, we propose AEGCL by introducing graph AutoEncoder to Graph Contrastive Learning. Benefiting from explicit edge reconstruction and raw attribute reconstruction, the robustness of GCL can be improved. Furthermore, we also propose a motif centrality-based adaptive graph augmentation method which leverages rich-semantic higher-order graph characteristics to boost the performance of GCL empirically. Extensive experiments compared with both self-supervised learning methods and semi-supervised learning methods demonstrate the superior effectiveness and robustness of AEGCL in downstream node classification tasks with satisfactory interpretability. In the future, we expect three promising development avenues: (1) adapting AEGCL to graph-level representation learning like graph classification and graph generation; (2) further exploring the robustness of graph contrastive learning when faced with adversarial graph attacks; and (3) leveraging the easy-to-conduct graph autoencoder task with other downstream tasks to boost their performance.

Acknowledgments

This work was supported by National Key Research and Development Program of China (2021YFF1201200), NSFC (62276277), Guangdong Basic and Applied Basic Research Foundation (2022B1515120059), and Guangdong Province Key Laboratory of Computational Science at the Sun Yat-sen University (2020B1212060032).

References

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

[2] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[3] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *ICML*, 2020, pp. 874–883.

[4] D. Jin, Z. Yu, P. Jiao, S. Pan, D. He, J. Wu, P. S. Yu, and W. Zhang, "A survey of community detection approaches: From statistical modeling to deep learning," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 2, pp. 1149–1170, 2023.

[5] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *NeurIPS*, 2018, pp. 5171–5181.

[6] T. Chen and R. C. Wong, "Handling information loss of graph neural networks for session-based recommendation," in *KDD*, 2020, pp. 1172–1180.

[7] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," in *NeurIPS*, 2020, pp. 5812–5823.

[8] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. S. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," in *ICLR*, 2020.

[9] N. Jovanovic, Z. Meng, L. Faber, and R. Wattenhofer, "Towards robust graph contrastive learning," *arXiv preprint arXiv:2102.13085*, 2021.

[10] J. Xia, L. Wu, J. Chen, B. Hu, and S. Z. Li, "Simgrace: A simple framework for graph contrastive learning without data augmentation," in *WWW*, 2022, pp. 1070–1079.

[11] S. Feng, B. Jing, Y. Zhu, and H. Tong, "Adversarial graph contrastive learning with information regularization," in *WWW*, 2022, pp. 1362–1371.

[12] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *NIPS Workshop on Bayesian Deep Learning*, 2016.

[13] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, "A simple framework for contrastive learning of visual representations," in *ICML*, 2020, pp. 1597–1607.

[14] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, "Momentum contrast for unsupervised visual representation learning," in *CVPR*, 2020, pp. 9726–9735.

[15] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, "A simple framework for contrastive learning of visual representations," in *ICML*, 2020, pp. 1597–1607.

[16] M. Sun, J. Xing, H. Wang, B. Chen, and J. Zhou, "Mocl: Contrastive learning on molecular graphs with multi-level domain knowledge," *arXiv preprint arXiv:2106.04509*, 2021.

[17] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Graph contrastive learning with adaptive augmentation," in *WWW*, 2021, pp. 2069–2080.

[18] Z. Yu, D. Jin, Z. Liu, D. He, X. Wang, H. Tong, and J. Han, "AS-GCN: adaptive semantic architecture of graph convolutional networks for text-rich networks," in *ICDM*, 2021, pp. 837–846.

[19] X. Wang, M. Zhu, D. Bo, P. Cui, C. Shi, and J. Pei, "AM-GCN: adaptive multi-channel graph convolutional networks," in *KDD*, 2020, pp. 1243–1253.

[20] D. Lim, F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao, and S. N. Lim, "Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods," *NeurIPS*, 2021.

[21] P. Li, L. Huang, C. Wang, and J. Lai, "Edmot: An edge enhancement approach for motif-aware community detection," in *KDD*, 2019, pp. 479–487.

[22] J. Zhang, H. Zhang, C. Xia, and L. Sun, "Graph-bert: Only attention is needed for learning graph representations," *arXiv preprint arXiv:2001.05140*, 2020.

[23] Z. Hu, Y. Dong, K. Wang, K. Chang, and Y. Sun, "GPT-GNN: generative pre-training of graph neural networks," in *KDD*, 2020, pp. 1857–1867.

[24] Y. You, T. Chen, Z. Wang, and Y. Shen, "When does self-supervision help graph convolutional networks?" in *ICML*, 2020, pp. 10 871–10 880.

[25] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019, pp. 4171–4186.

[26] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[27] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *NeurIPS*, 2019, pp. 5754–5764.

[28] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021.

[29] H. Bao, L. Dong, and F. Wei, "Beit: Bert pre-training of image transformers," *arXiv preprint arXiv:2106.08254*, 2021.

[30] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," *arXiv preprint arXiv:2111.06377*, 2021.

[31] A. Kumagai, T. Iwata, and Y. Fujiwara, "Transfer metric learning for unseen domains," *Data Science and Engineering*, vol. 5, no. 2, pp. 140–151, 2020.

[32] J. You, B. Liu, Z. Ying, V. S. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *NeurIPS*, 2018, pp. 6412–6422.

[33] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *ICML*, 2018, pp. 5694–5703.

[34] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.

[35] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[36] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[37] Z. Hou, X. Liu, Y. Cen, Y. Dong, H. Yang, C. Wang, and J. Tang, "Graphmae: Self-supervised masked graph autoencoders," *arXiv preprint arXiv:2205.10803*, 2022.

[38] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *ICLR*, 2014.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.

[40] L. Wu, H. Lin, Z. Gao, C. Tan, S. Li *et al.*, "Self-supervised on graphs: Contrastive, generative, or predictive," *arXiv preprint arXiv:2105.07342*, 2021.

[41] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, "GCC: graph contrastive coding for graph neural network pre-training," in *KDD*, 2020, pp. 1150–1160.

[42] P. Trivedi, E. S. Lubana, Y. Yan, Y. Yang, and D. Koutra, "Augmentations in graph contrastive learning: Current methodological flaws & towards better practices," in *WWW*, 2022, pp. 1538–1549.

[43] J. Yu, H. Yin, X. Xia, T. Chen, L. Cui, and Q. V. H. Nguyen, "Are graph augmentations necessary?: Simple graph contrastive learning for recommendation," in *SIGIR*, 2022, pp. 1294–1303.

[44] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *ICLR*, 2019.

[45] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang, "Graph representation learning via graphical mutual information maximization," in *WWW*, 2020, pp. 259–270.

[46] K. Hassani and A. H. K. Ahmadi, "Contrastive multi-view representation learning on graphs," in *ICML*, 2020, pp. 4116–4126.

[47] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning," *arXiv preprint arXiv:2006.04131*, 2020.

[48] S. Thakoor, C. Tallec, M. G. Azar, R. Munos, P. Veličković, and M. Valko, "Bootstrapped representation learning on graphs," *arXiv preprint arXiv:2102.06514*, 2021.

[49] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *KDD*, 2018, pp. 2847–2856.

[50] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *ICLR*, 2019.

[51] W. Jin, Y. Li, H. Xu, Y. Wang, S. Ji, C. Aggarwal, and J. Tang, "Adversarial attacks and defenses on graphs: A review, a tool and empirical studies," *arXiv preprint arXiv:2003.00653*, 2020.

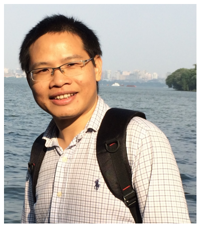
[52] Y. Li, W. Jin, H. Xu, and J. Tang, "Deeprobust: A pytorch library for adversarial attacks and defenses," *arXiv preprint arXiv:2005.06149*, 2020.

[53] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples for graph data: Deep insights into attack and defense," in *AAAI*, 2019, pp. 4816–4823.

[54] K. Xu, H. Chen, S. Liu, P. Chen, T. Weng, M. Hong, and X. Lin, "Topology attack and defense for graph neural networks: An optimization perspective," in *IJCAI*, 2019, pp. 3961–3967.

[55] J. Chen, Y. Wu, X. Xu, Y. Chen, H. Zheng, and Q. Xuan, "Fast gradient attack on network embedding," *arXiv preprint arXiv:1809.02797*, 2018.
[56] Y. Sun, S. Wang, X. Tang, T.-Y. Hsieh, and V. Honavar, "Node injection attacks on graphs via reinforcement learning," *arXiv preprint arXiv:1909.06543*, 2019.
[57] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *ICML*, 2018, pp. 1123–1132.
[58] Y. Ma, S. Wang, T. Derr, L. Wu, and J. Tang, "Graph adversarial attack via rewiring," in *KDD*, 2021, pp. 1161–1169.
[59] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR*, 2018.
[60] C. E. Tsourakakis, J. Pachocki, and M. Mitzenmacher, "Scalable motif-aware graph clustering," in *WWW*, 2017, pp. 1451–1460.
[61] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.
[62] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, vol. 353, no. 6295, pp. 163–166, 2016.
[63] J. B. Lee, R. A. Rossi, X. Kong, S. Kim, E. Koh, and A. Rao, "Graph convolutional networks with motif-based attention," in *CIKM*, 2019, pp. 499–508.
[64] Y. Jia, Q. Zhang, W. Zhang, and X. Wang, "CommunityGAN: Community detection with generative adversarial nets," in *WWW*, 2019, pp. 784–794.
[65] A. Sankar, X. Zhang, and K. C. Chang, "Meta-GNN: metagraph neural network for semi-supervised learning in attributed heterogeneous information networks," in *ASONAM*, 2019, pp. 137–144.
[66] Y. Rong, Y. Bian, T. Xu, W. Xie, Y. Wei, W. Huang, and J. Huang, "Self-supervised graph transformer on large-scale molecular data," in *NeurIPS*, 2020.
[67] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.
[68] S. Suresh, P. Li, C. Hao, and J. Neville, "Adversarial graph augmentation to improve graph contrastive learning," in *NeurIPS*, 2021, pp. 15 920–15 933.
[69] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.
[70] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, vol. 353, no. 6295, pp. 163–166, 2016.
[71] T. Zhao, X. Zhang, and S. Wang, "Graphsmote: Imbalanced node classification on graphs with graph neural networks," in *WSDM*, 2021, pp. 833–841.
[72] D. Jin, C. Huo, C. Liang, and L. Yang, "Heterogeneous graph neural network via attribute completion," in *WWW*, 2021, pp. 391–400.
[73] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
[74] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," in *Relational Representation Learning Workshop@NeurIPS*, 2018.
[75] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Mag.*, vol. 29, no. 3, pp. 93–106, 2008.
[76] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," in *ICLR*, 2018.
[77] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.
[78] C. Park, D. Kim, J. Han, and H. Yu, "Unsupervised attributed multiplex network embedding," in *AAAI*, 2020, pp. 5371–5378.
[79] W. Li, L. Huang, C. Wang, and Y. Ye, "Stargat: Star-shaped hierarchical graph attentional network for heterogeneous network representation learning," in *ICDM*, 2021, pp. 1198–1203.
[80] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinform.*, vol. 33, no. 14, pp. i190–i198, 2017.
[81] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, 2008.

Wen-Zhi Li received his B.Sc. honors degree in computer science from Jilin University in 2021 and is currently a master's student in computer science in the School of Computer Science and Engineering, Sun Yat-sen University. He has published several papers in international conferences such as KDD and ICDM. His research interest includes machine learning and data mining.



Chang-Dong Wang received the Ph.D. degree in computer science in 2013 from Sun Yat-sen University, Guangzhou, China. He joined Sun Yat-sen University in 2013, where he is currently an associate professor with School of Computer Science and Engineering. His current research interests include machine learning and data mining. He has published over 80 scientific papers in international journals and conferences such as IEEE TPAMI, IEEE TKDE, IEEE TCYB, IEEE TNNLS, KDD, AAAI, IJCAI, CVPR, ICDM, CIKM and SDM. His ICDM 2010 paper won the Honorable Mention for Best Research Paper Awards. He won 2012 Microsoft Research Fellowship Nomination Award. He was awarded 2015 Chinese Association for Artificial Intelligence (CAAI) Outstanding Dissertation. He is an Associate Editor in Journal of Artificial Intelligence Research (JAIR).

Jian-Huang Lai received the M.Sc. degree in applied mathematics in 1989 and the Ph.D. degree in mathematics in 1999 from Sun Yat-sen University, China. He joined Sun Yat-sen University in 1989 as an Assistant Professor, where he is currently a Professor with the School of Computer Science and Engineering. His current research interests include the areas of digital image processing, pattern recognition, multimedia communication, wavelet and its applications. He has published more than 200 scientific papers in the international journals and conferences on image processing and pattern recognition, such as IEEE TPAMI, IEEE TKDE, IEEE TNNLS, IEEE TIP, IEEE TSMC-B, Pattern Recognition, ICCV, CVPR, IJCAI, ICDM and SDM. Prof. Lai serves as a Standing Member of the Image and Graphics Association of China, and also serves as a Standing Director of the Image and Graphics Association of Guangdong.

Philip S. Yu is a Distinguished Professor in Computer Science at the University of Illinois at Chicago and also holds the Wexler Chair in Information Technology. Before joining UIC, Dr. Yu was with IBM, where he was manager of the Software Tools and Techniques group at the Watson Research Center. His research interest is on big data, including data mining, data stream, database and privacy. He has published more than 1,000 papers in refereed journals and conferences. He holds or has applied for more than 300 US patents.

Dr. Yu is a Fellow of the ACM and the IEEE. He is on the steering committee of the ACM Conference on Information and Knowledge Management and was a member of the steering committee of IEEE Data Engineering and IEEE Conference on Data Mining. He was the Editor-in-Chief of ACM Transactions on Knowledge Discovery from Data (2011-2017) and the Editor-in-Chief of IEEE Transactions on Knowledge and Data Engineering (2001-2004). Dr. Yu is the recipient of ACM SIGKDD 2016 Innovation Award for his influential research and scientific contributions on mining, fusion and anonymization of big data, the IEEE Computer Society's 2013 Technical Achievement Award for "*pioneering and fundamentally innovative contributions to the scalable indexing, querying, searching, mining and anonymization of big data*", and the Research Contributions Award from IEEE Intl. Conference on Data Mining (ICDM) in 2003 for his pioneering contributions to the field of data mining. He also received the ICDM 2013 10-year Highest-Impact Paper Award, and the EDBT Test of Time Award (2014). He had received several IBM honors including 2 IBM Outstanding Innovation Awards, an Outstanding Technical Achievement Award, 2 Research Division Awards and the 94th plateau of Invention Achievement Awards. He was an IBM Master Inventor. Dr. Yu received the B.S. Degree in E.E. from National Taiwan University, the M.S. and Ph.D. degrees in E.E. from Stanford University, and the M.B.A. degree from New York University.