Data-centric Graph Learning: A Survey

Yuxin Guo*, Deyu Bo*, Cheng Yang†, Zhiyuan Lu, Zhongjian Zhang, Jixi Liu, Yufei Peng, Chuan Shi†

***Abstract*—The history of artificial intelligence (AI) has witnessed the significant impact of high-quality data on various deep learning models, such as ImageNet for AlexNet and ResNet. Recently, instead of designing more complex neural architectures as model-centric approaches, the attention of AI community has shifted to data-centric ones, which focuses on better processing data to strengthen the ability of neural models. Graph learning, which operates on ubiquitous topological data, also plays an important role in the era of deep learning. In this survey, we comprehensively review graph learning approaches from the data-centric perspective, and aim to answer three crucial questions: *(1) when to modify graph data*, *(2) what part of the graph data needs modification* to unlock the potential of various graph models, and *(3) how to safeguard graph models* from problematic data influence. Accordingly, we propose a novel taxonomy based on the stages in the graph learning pipeline, and highlight the processing methods for different data structures in the graph data, *i.e.,* topology, feature and label. Furthermore, we analyze some potential problems embedded in graph data and discuss how to solve them in a data-centric manner. Finally, we provide some promising future directions for data-centric graph learning.**

***Index Terms*—Data-centric Learning, Graph Neural Network.**

I. Introduction

RECENT advancements in the non-Euclidean domain draw substantial attention from the artificial intelligence (AI) community. Graphs, as typical non-Euclidean data, are ubiquitous in the real world and have been widely used in many fields, such as recommendation, security, bioinformatics, etc. Over the past decade, the progress of graph-related research has been propelled by innovations in graph models, ranging from graph kernels [1], [2] to graph embeddings [3], [4], and culminating in the latest advancements represented by graph neural networks (GNNs) [5], [6]. Conversely, little research has been directed toward the intrinsic aspects of graph data, including quality, diversity, security, and so on.

Generally, the revolutions in AI have consistently been initiated by the availability of vast amounts of high-quality data, subsequently followed by powerful models. A notable example is the success of ImageNet [7], which significantly contribute to the development of deep convolutional neural networks, *e.g.*, AlexNet [8] and ResNet [9]. Similarly, the importance of graph data in GNNs learning cannot be overstated. Graphs naturally represent complex relationships and interactions, making them crucial for enhancing the performance of GNNs. For example, the construction of graphs has a substantial impact on the performance of graph learning. Even when employing the same GNN model and training methods, the choice between weighted and unweighted graphs, directed and undirected graphs, as well as heterogeneous and homogeneous graphs can result in vastly different final model performance. As the significance of data becomes increasingly acknowledged, recently, the attention of the AI community has shifted from model-centric approaches to data-centric ones [10], [11].

The emerging data-centric AI emphasizes producing suitable data to improve the performance of a given model. Specifically, data-centric graph learning focuses on optimizing the quality and applicability of graph data, utilizing methods such as graph augmentation [12]–[31], graph sampling [32]–[39], and feature engineering [40]–[47], etc. By properly modifying graph data, these approaches address critical challenges such as sparsity and overfitting, ultimately leading to more effective and efficient graph models. "*How to process the graph data to unlock the full potential of graph models?*" A well-informed answer can help us understand the relationship between graph data and graph models. However, unlike Euclidean data such as images and tabular data, the irregular nature of graphs poses several questions for data-centric graph learning: Firstly, *when should we modify graph data to benefit graph models?* Data modification may occur in different stages of graph learning. For example, we can heuristically perturb the edges before training, while we can also estimate new graph structures from the node representations during training. Secondly, *which part of the graph data should we modify?* Graph data involves various structures, including edges, nodes, features, and labels, each of which plays an important role in graph representation learning. Thirdly, *how to prevent the graph models from being affected by the problematic graph data?* Graph data may inevitably introduce noise and bias, due to the manually defined relations and features, which makes the models untrustworthy.

This survey systematically reviews and categorizes existing graph learning methods from the data-centric perspective. In particular, to answer the first question, we divide the graph learning process into four stages: preparation, pre-processing, training, and inference, as illustrated in Figure 1. We discuss the significance of each stage for graph data. Next, we further categorize existing methods from a structural perspective to address the second question. Specifically, we consider how to handle the topology, features, and labels of graph data, respectively. Finally, we analyze the potential problems in existing graph data, including vulnerability, unfairness, selection bias, and heterophily. We further discuss how to solve these issues in a data-centric way.

Related Surveys. Currently, there is some literature on

* The first two authors contributed equally to this work.
† Corresponding authors: {yangcheng,shichuan}@bupt.edu.cn
School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China, 100876.

data-centric AI [10], [11]. However, they mainly focus on the Euclidean domain, and there is little discussion about non-Euclidean data. Additionally, there are many surveys on the topic of model-centric graph learning. For example, Cui *et al.* [48] summarizes network embedding methods, Wu *et al.* [49] divides GNNs into four representative frameworks, and Bronstein *et al.* [50] studies the equivariant and invariant models on the geometric data. These surveys introduce various powerful graph models but are orthogonal to our data-centric survey. On the other hand, there are also works covering some specific data-centric approaches, *e.g.*, graph augmentation [51], graph sampling [52], and graph structure learning [53], which can be seen as a part of our survey. The recent work [54] also provides a review for data-centric graph learning. However, it does not distinguish the different stages of data-centric graph learning.

Contributions. The contribution of this paper is summarized as follows:

- **Novel Taxonomy.** We categorize existing data-centric graph learning methods by the stages in the graph learning pipeline, including pre-processing, training, and inference. For each stage, we introduce its goal and importance for data-centric graph learning.
- **Multiple Perspectives.** We highlight how to process different data structures in the graph data, including topology, feature, and label, to unlock the potential of given graph models.
- **Comprehensive Discussion.** We analyze the potential influence of problematic graph data on the graph models and discuss how to alleviate these problems in a data-centric manner. Moreover, we suggest three possible future directions for data-centric graph learning, which may benefit the development of this field.

Organization. The rest of this survey is organized as follows: Section II outlines the background of data-centric graph learning, and describes how graph data is manually processed. Sections III-V introduce the data-centric graph learning methods in the pre-processing, training, and inference stages, respectively. Section VI introduces the potential problems of graph data and discusses how to deal with these issues. Finally, Section VII provides a summarization of this paper and poses several promising future directions.

II. Definition and Background

The first occurrence of the term "graph" dates back to 1878, when J. J. Sylvester [55] used it to establish a connection between mathematics and chemical structures. Before that, graph theory can be traced back to 1735, when Euler [56] solved the Seven Bridge of Königsberg problem and proved that it is impossible to walk through all seven bridges exactly once. There are also some other significant graph theory problems [57], such as Diagram-Tracing Puzzles, Four-Color Problem, etc.

A graph consists of different nodes and their connections, which can be defined as:

Definition 1: A graph is represented as $G = (V, E)$, where V is the vertex set and E is the edge set. Let $|V|$=N, the adjacency matrix of graph G is represented as $\mathbf{A} \in \{0,1\}^{N\times N}$, where $A_{ij} = 1$ if there is an edge between nodes i and j, and $A_{ij} = 0$ otherwise. A graph including node attributes is defined as an attributed graph, where $\mathbf{X} \in \mathbb{R}^{N\times d}$ is the node feature matrix and the i-th row $\mathbf{X}_i$ indicates the attributes of node v_i. The Laplacian matrix of graph G is denoted as $\mathbf{L}$, and $\hat{\mathbf{L}} = \mathbf{I}_N - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is the normalized graph Laplacian matrix with $\mathbf{D}$ the diagonal degree matrix.

Definition 2: Data-centric graph learning aims at improving the quality and utility of graph data (such as topology, features, and labels) across various stages of graph learning (including preparation, pre-processing, training, and inference). Model-centric graph learning, which prioritizes model architecture improvements while assuming the dataset is fixed, often overlooks the complexity, scope, and relevance of the data. In contrast, data-centric approaches emphasize refining the data itself, with the model remaining relatively unchanged. Consequently, data-centric graph learning tends to be more adaptable and transferable across different problems.

Building upon the graph definition, we can abstract the real-world objectives and their relations into a graphical representation. For example, in the Königsberg problem, Euler simplifies the islands into nodes and bridges into edges. In contrast to naturally occurring images and languages, creating graphs demands significant artificial priors. Consequently, graph construction stands as the **first step of data-centric graph learning**.

Initially, the definition of edges contains meaningful semantics. On the one hand, different definitions of edges convey disparate information. Consider the example of a molecular graph where atoms serve as nodes. Edges can be defined as Chemical bonds between atoms, reflecting their chemical properties. Alternatively, edges also can be defined as connections between atoms within a specific radius, *i.e.*, K-Nearest Neighbor (KNN) graphs, representing the positional information in Euclidean space. On the other hand, various auxiliary information introduces diverse edge types as shown in Figure 1. For instance, directed graphs consider the directions of edges, heterogeneous graphs [58] cover different relations between nodes, hypergraphs [59] capture high-order correlations, and temporal graphs [60] encode the time stamps of edges.

In addition to the definition of edges, there are also various definitions of node features. For example, within text-attributed graphs, textual information serves as node features. In the Planetoid datasets [61], the text is represented as the bag-of-words, while in the OGB datasets [62], the text representations are encoded by the pre-trained language models. These two different definitions of node features result in a significant performance gap. Therefore, how to process the node features to improve the performance of graph models is also crucial for data-centric graph learning.

III. Pre-processing Stage

In this section, we will discuss data-centric methods at the pre-processing stage, which aim to heuristically modify graph

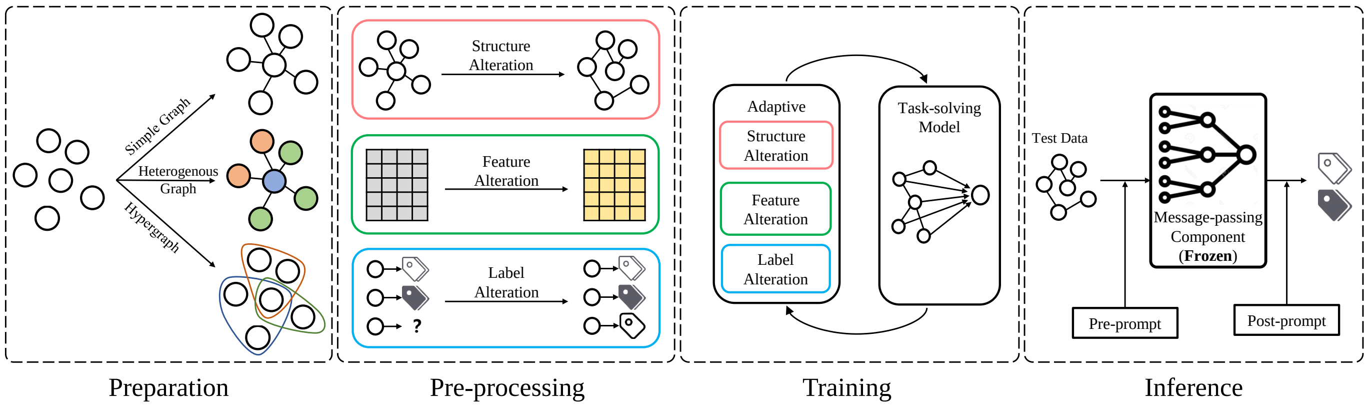

Fig. 1. Pipeline of data-centric graph learning. The first step is to construct different graphs from the data as needed. The graph structure, node features, or labels are then pre-processed to facilitate the learning of graph models. During the training phase, graph data is collaboratively processed with the task-solving model to improve its performance. Ultimately, prompts are designed to imbue the graph models with enhanced predictive capabilities during the inference stage.

TABLE I
TAXONOMY AND REPRESENTATIVE WORKS OF DATA-CENTRIC GRAPH LEARNING.

Perspective	Method	Stage	Category & Reference
Topology	Graph Reduction	(§ III-A3) Pre-processing	Sparsification [63]–[76], Coarsening [77]–[84], Condensation [85]–[92]
	Graph Generation	(§ III-A2) Pre-processing	Node [93]–[97], Adjacency matrix [98], [99], Embedding [100], [101], Eigenvalue and eigenvector [102], [103]
	Graph Augmentation	(§ III-A1) Pre-processing	Dropping [43], [104]–[108], Subgraph [109]–[111], Diffusion [112]–[114]
		(§ IV-A1) Training	Structure [12]–[28], Spectrum [29]–[31], Automated [115]–[119]
	Graph Sampling	(§ III-A5) Pre-processing	Random [32]–[34], Importance [35]–[39]
		(§ IV-A2) Training	Node-wise [120]–[124], Layer-wise [125], [126]
	Graph Curriculum Learning	(§ III-A4) Pre-processing	Node-level [127]–[134], Graph-level [135]–[139]
		(§ IV-A4) Training	Edge [140]–[143], Node [144]–[146], Graph [147]–[149]
	Graph Structure Learning	(§ IV-A3) Training	Discrete [150]–[157], weighted [6], [12], [14], [158]–[171]
Feature	Feature Augmentation	(§ III-B1) Pre-processing	Pre-Attribute [40]–[47], Attribute-Free [3], [4], [172], [173]
	Position Encoding	(§ III-B2) Pre-processing	Absolute [174]–[178], Relative [179]–[182]
	Feature Selection	(§ IV-B2) Training	Task-independent [183]–[188] , Task-specific [189], [190]
	Feature Completion	(§ IV-B1) Training	Homogeneous [191]–[194], Heterogeneous [195]–[199]
	Graph Prompt	(§ V) Inference	Pre-prompt [200]–[202], Post-prompt [203], [204]
Label	Label Mixup	(§ III-C1) Pre-processing	Mixup [205]–[209], Knowledge Distillation [210]–[221]
	Pseudo Labeling	(§ IV-C2) Training	Multi-stage [222]–[225]
	Active Learning	(§ IV-C1) Training	Node-independent [226]–[230], Node-correlated [231]–[243]

data without leveraging the information from task-solving graph models. Specifically, we consider different data structures in each subsection, including topology, features, and labels. We also introduce how to add, delete, or change each type of data in detail.

A. Topology

Topology is the most important part of graph data, representing its structural information. In this section, we first introduce how to enrich the topology information of graphs, including augmentation and generation. Next, we refer to the topology reduction, which removes some redundant edges and nodes in the topology. Finally, we present curriculum learning and sampling, aiming to speed up the training of graph models by changing the original graph data distribution.

1) Graph Augmentation: Due to the scarcity and sparsity of graph data, it is difficult for graph models to fit the underlying distribution of graph data and easily fall into local optimum. Therefore, it is important to enrich the topology information in a low-overhead manner. Under this circumstance, graph augmentation is proposed to alleviate the over-fitting of graph models by perturbing the graph topology without changing its crucial information. Here we introduce some off-the-training heuristic methods, including masking, substitution, and diffusion.

Edge and Node Masking. Stochastically masking the edges or nodes in the graphs is a basic but extensively employed technique for graph augmentation. DropEdge [108] first proposes randomly masking edges when training GNNs to alleviate the over-smoothing problem. Subsequently, You *et al.* [43] follow a similar scheme that randomly deletes nodes and their associated edges from the graph. In addition to dealing with the over-fitting and over-smoothing issues, these masking methods have been extensively adopted in graph self-supervised learning (SSL). On the one hand, graph contrastive learning [43], [104], [105] leverages these incomplete topologies as different views of a graph and learns to capture their consistency. On the other hand, some generative graph SSL methods [106], [107] treat the masked edges and nodes as self-supervised signals. By reconstructing the masked information, these methods can learn meaningful graph representations.

Subgraph Substitution. The masking-based methods can only affect the local topology of a graph, overlooking its global structures. In response, subgraph substitution methods have been introduced to replace the specific substructures within graphs as a means of augmentation. MoCL [109] employs biomedical knowledge to augment molecular graphs by substituting important substructures, such as functional groups, which injects domain-specific insights into graph augmentation. SubMix [110] utilizes importance sampling to extract and exchange the connected and clustered subgraphs from a pair of graphs, which effectively fuse their topologies. M-Evolve [111] uses motifs to augment graph data by selecting target motifs and adjusting edges within them.

Graph Diffusion. Graph diffusion [112] provides a way to enrich the topology information by aggregating the neighbors at different distances, which can be formulated as:

$$\tilde{\mathbf{A}} = \sum_{k=0}^{\infty} \theta_k \mathbf{T}^k, \tag{1}$$

where $\tilde{\mathbf{A}}$ is the augmented adjacency matrix, $\mathbf{T} = \mathbf{A}\mathbf{D}^{-1}$ is the random walk transition matrix, k is the distance, and θ_k denotes the weight coefficients. There are two special examples of graph diffusion: Personalized PageRank (PPR) and heat kernel. Specifically, PPR uses the random walk transition matrix with coefficients $\theta_k = \alpha(1-\alpha)^k$ and heat kernel uses $\theta_k = e^{-s}\frac{s^k}{k!}$ to exponentially reduce the influence of remote nodes, where α and s are hyperparameters.

The aforementioned graph augmentation methods are intuitive, concise, and provide adequate performance in most cases. Additionally, they can be aggregated to form a complex enhancement strategy, which will be further discussed in Section IV-A1.

2) Graph Generation: Although graph augmentation can initially enrich the topology information, it inevitably introduces noise and hurts the model performance. As an advanced approach, graph generation [244], [245] aims to generate graph samples with high quality and diversity. In this section, we introduce some representative graph generation methods based on the varied generated data, including sequence, embedding, adjacency matrix, and eigenvector.

Node Sequence. Simplifying a graph into a sequence is the first idea of graph generation, resulting in the autoregressive graph generation methods. Generally, autoregressive methods aim to generate graphs node-by-node based on the pre-sampled ordering. However, due to the non-uniqueness and high-dimensional nature of graphs, utilizing node ordering as input needs to consider the permutation-invariance property. To address this challenge, Li *et al.* [93] randomly sample node orderings from the full permutation of the nodes to guarantee the invariance. This strategy is effective but inefficient. Subsequently, GraphRNN [94], MolecularRNN [95], GraphGen [97], and GraphDF [96] employ Breadth-First Search (BFS) or depth-first search (DFS) to guarantee the uniqueness of node orderings and maintain high scalability.

Adjacency Matrix. In addition to sequence generation, another natural idea is to directly generate the adjacency matrix of the graph. Methods belonging to this category, *e.g.*, DiGress [98] and EDGE [99], mainly use the emerging diffusion model as the generation framework. Typically, they first gradually add Gaussian noise into the nodes or edges in the adjacency matrix. After that, a standard de-noising process is emplyed to recover the adjacency matrix from the noisy graph structures. A major advantage of this process is that the generation is permutation-equivariant and effective in generating high-quality small graphs.

Node Embedding. Generating adjacency matrices of graphs is usually time-consuming and cannot scale to large graphs. One possible solution is to generate the graph in an indirect way. For example, the adjacency matrix can be represented by the node embeddings: $\mathbf{A} = \mathbf{H} \cdot \mathbf{H}^\top$. In this way, we only need to generate a small tensor $\mathbf{H} \in \mathbb{R}^{N \times d}$, rather than the large adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, where $d \ll N$. For example, VGAE [100] and GNF [101] first utilize GNNs to learn the node representations and then map them into a multivariate Gaussian distribution, where the mean and variance are defined by the node representations. In the training process, these methods will reconstruct the graph structure via a link prediction loss function. In the generation process, these methods directly sample node representations from the learned multivariate Gaussian distribution to generate the graph topology. Although generating node embeddings can scale to large graphs, it still suffers from the high computation cost as it needs to post-process the graphs for alignment.

Eigenvalue and Eigenvector. Another indirect graph generation method is to generate the eigenvalues and eigenvectors of graphs. SPECTRE [102] and GSDM [103] are the two representative methods, which leverage generative adversarial network and diffusion model to generate the eigenvalues and eigenvectors of graphs, respectively. Since the spectrum of a graph encodes its global structure, these generation methods can capture the shape information of graphs and therefore result in better generative performance.

3) Graph Reduction: The inherent irregular and non-Euclidean structure of graphs prevents the graph models from scaling to large scales [246], [247]. To reduce the computational complexity and improve scalability, one possible way is to reduce the redundant nodes and edges in the graphs without changing the crucial structural information. Therefore, the graph models trained on the original and reduced graphs will have similar performance. Graph reduction can be divided into three categories: edge reduction, node reduction, and co-reduction.

Edge Reduction. Removing the redundant edges in a graph is known as the graph sparsification method, which aims to get a sparsified graph $G_s = (V, E_s)$ by removing some edges of the original graph G, where $E_s \subset E$. In general, G_s should preserve some crucial properties of G, including cut weights, spectral similarity, and shortest path, resulting in three types of graph sparsification methods: cut sparsification [63]–[65], spectral sparsification [66], and spanner [67].

Cut sparsification aims to reduce edges while preserving the value of the graph cut $C = (V', V - V')$, which divides a graph into two subgraphs with node sets V' and $V - V'$, respectively. The value of a graph cut $w(C)$ is defined as the total weight of the edges with the endpoint on each side of a cut. Generally, G_s is a ϵ-cut sparsifier of G if it satisfies the following equation for arbitrary graph cut:

$$(1-\epsilon)w_G(C) \leq w_S(C) \leq (1+\epsilon)w_G(C), \tag{2}$$

where $\epsilon \in (0, 1)$. This property has been widely used in many graph theories, such as graph connectivity, the maximum flow problem, and the minimum bisection problem.

Spectral sparsification ensures the original and sparse graphs have similar quadratic forms, *i.e.*, smoothness. G_s is a ϵ-spectral sparsifier of G if it satisfies the following equation for any vector $\mathbf{q} \in \mathbb{R}^{N\times 1}$:

$$(1-\epsilon)\mathbf{q}^T\mathbf{L}_G\mathbf{q} \leq \mathbf{q}^T\mathbf{L}_S\mathbf{q} \leq (1+\epsilon)\mathbf{q}^T\mathbf{L}_G\mathbf{q}, \tag{3}$$

where $\mathbf{q}^\top\mathbf{L}\mathbf{q}$ is the quadratic form. Spectral sparsification preserves the most important spectral property that represents the global structural information.

Spanner is designed for tasks that focus on the distance of node pairs. G_s is a t-spanner of G if their shortest paths d_S and d_G satisfy the following equation for all node pairs $(v_i, v_j), v_i, v_j \in V$:

$$d_G(v_i, v_j) \leq d_S(v_i, v_j) \leq t \cdot d_G(v_i, v_j), \tag{4}$$

where $t \geq 1$. The t-spanner can capture the underlying graph structure of a network and perform well in some special graphs, such as rings, meshes, trees, butterflies, and cube-connected cycles. [68]

There are many methods to implement the aforementioned sparsifiers, including sampling [65], [66], [76] and dynamic streaming [69]–[71]. Sampling methods aim to assign each edge a possibility. A larger possibility means a higher probability of being removed. Dynamic streaming methods continuously add or remove some edges in a stream until reach the requirement of the sparsification. Recent studies have also shown great potential to use graph sparsification to alleviate over-fitting and over-smoothing of GNNs [72], [73], which may be a future direction of research in graph sparsification.

Node Reduction. Merging a set of closely connected nodes into a super node can also reduce the size of graphs, which is also known as graph coarsening. The coarsened graph is defined as $G_s = (V_s, E_s)$, where $|V_s| = n < N$. The basic approach of graph coarsening is pairwise aggregation, which merges a pair of nodes into a supernode based on some similarity measures [78]–[80]. In recent years, spectrum-preserving coarsening methods have received more attention because of their superiority in capturing important structural information.

Loukas *et al.* [81] propose restricted spectral similarity (RSS) to evaluate whether the coarsened graph can learn the spectrum of the original graph, which is defined as:

$$(1-\zeta_k)\lambda_k \leq \mathbf{u}_k^T\widetilde{\mathbf{L}}\mathbf{u}_k \leq (1+\zeta_k)\lambda_k, \tag{5}$$

where λ_k and $\mathbf{u}_k$ indictae the k-th eigenvalue and eigenvector of $\mathbf{L}$, respectively. $\widetilde{\mathbf{L}} \in \mathbb{R}^{N\times N}$ is the approximation of $\mathbf{L}$, which bridges the gap between the original and coarsened graphs. Intuitively, $\mathbf{L}_s$ preserves the information of $\mathbf{L}$ as it approximates the total variation of the eigenvectors, *i.e.*, smoothness.

Many others follow and continue to work on the graph Laplacian, proposing various methods to measure the distance between the coarse graph and the original graph. For example, Jin *et al.* [82] proposes spectral distances to capture structural differences between graphs.

Co-reduction. Simultaneously removing the edges and nodes of a graph is defined as co-reduction. The representative work is graph condensation, which synthesizes a condensed graph G_s with fewer nodes and edges. The graph models trained on the original and the condensed graphs can have similar performance.

Jin *et al.* [85] introduce the concept of graph condensation, and provide the first framework of graph condensation. This framework leverages the gradient matching [86] method to align the distributions of original and condensed graphs. Specifically, the optimization process can be formulated as follows:

$$\boldsymbol{\Upsilon}_{\mathcal{S}} = \nabla_{\boldsymbol{\theta}}\mathcal{L}\left(f_{\boldsymbol{\theta}}\left(\mathbf{A}_s, \mathbf{X}_s\right), \mathbf{Y}_s\right), \tag{6}$$

$$\boldsymbol{\Upsilon} = \nabla_{\boldsymbol{\theta}}\mathcal{L}\left(f_{\boldsymbol{\theta}}(\mathbf{A}, \mathbf{X}), \mathbf{Y}\right), \tag{7}$$

$$Dis\left(\boldsymbol{\Upsilon}_{\mathcal{S}}, \boldsymbol{\Upsilon}\right) = \frac{\boldsymbol{\Upsilon}_{\mathcal{S}} \cdot \boldsymbol{\Upsilon}}{\|\boldsymbol{\Upsilon}_{\mathcal{S}}\| \, \|\boldsymbol{\Upsilon}\|}, \tag{8}$$

where $f_{\boldsymbol{\theta}}$ denotes the graph models parameterized with $\boldsymbol{\theta}$, $\mathcal{L}$ is the loss function, and $\boldsymbol{\Upsilon}_{\mathcal{S}}$, $\boldsymbol{\Upsilon}$ are the gradients on the condensed and origina graphs, respectively. By minimizing the distance Dis between gradients, the condensed graph can preserve the structure and feature information of the original graph.

To overcome the computational inefficiency caused by bilevel optimization, DosCond [87] proposes an aggressive approximation of the gradient matching loss, which only matches the gradients of the initial model, avoiding the need for the inner loop. GCDM [89] optimizes the synthetic graph by minimizing the distance between the distribution of receptive fields in the original and synthetic graphs. SFGC [88] solely optimizes the node features of the synthetic graph, treating the Laplacian matrix constructed by structure information as an identity matrix. To improve the suboptimal performance caused by the inexact approximation methods aforementioned, KIDD [90] adopts kernel ridge regression for an exact solution and proposes a series of variants to further boost efficiency. SGDD [91] constructs the structure of the synthetic graph via a structure learning model integrating both feature and auxiliary information. GCEM [92] also tackles the cross-architecture problem from the spectral perspective. Unlike SGDD, GCEM directly optimizes the eigenbasis by aligning the subspaces represented as the outer product of the eigenbasis of the original and synthetic graph respectively, and then constructs the synthetic graph using the learned eigenbasis and the spectrum of the original graph.

4) Graph Curriculum Learning: Curriculum learning [248] aims to imitate the human learning process, advocating that the model starts learning from easy samples and gradually advances to complex samples. Graph Curriculum Learning (CuL) [249] can benefit the convergence of graph models and improve the generalization ability. Most graph CuL methods have two important functions: difficulty measurer and training scheduler. The former evaluates the difficulty of the training data to give the learning prior, and the latter decides how to learn from easy to hard samples. According to the data structures used in these two functions, we divide existing methods into two categories: node-level and graph-level.

Node-level CuL. The basic idea is to determine the difficulty of nodes utilizing some statistical information. For example, CLNode [127] uses node features, node structure, and label noise to get the difficulty of training samples. MTGNN [128] evaluates the difficulty by the length of the predicted steps, while Tuneup [129] uses node degree to measure the difficulty. Data relationship is also a major perspective in designing the difficult measurer. For example, DiGCL [130] and HSAN [131] leverage contrastive loss to judge difficulty, DualGCN [132] employs the cross-review strategy, GNN-CL [133] focuses on homophily and smoothness of node and their neighbors, GCN-WSRS [134] emphasizes the correlation between samples.

Graph-level CuL. Different from the above Node-level Cul, Graph-level Cul incorporates graph information into the difficulty measurer and training scheduler. CurGraph [135] calculates the difficulty based on the within-class and between-class distributions of the embeddings of the samples., CuCo [136] determines difficulty based on the similarity of embeddings between negative and positive samples, and SMMCL [137] evaluates difficulty with multimodal consistency and multimodal uncertainty. Some methods, such as HACL [138] and CHEST [139], utilize both properties and data relationships to design the difficulty measurer. HACL employs the complexity of the vocabulary in the sample and the consistency between the sentiment expressed by the vocabulary in the sample and the expected sentiment to evaluate the difficulty. CHEST defines the difficulty measurer on multiple pre-training tasks, namely three subgraph context information tasks related to nodes, edges, meta-paths, and a graph contrastive learning task.

5) Graph Sampling: Graph sampling methods focus on selecting the nodes that are important for the training of graph models, which can restrict the memory overhead within a fixed budget and speed up the convergence of graph models. In this part, we discuss the heuristic sampling methods, which can be further divided into two categories: random sampling and importance sampling.

Random Sampling. Since nodes in graphs have different numbers of neighbors, directly aggregating the information of neighbors will lead to uncontrollable expenses. The basic way to control the overhead of graph models is to randomly sample the same number of neighbors for each node, making the time and memory overhead controllable during training. Specifically, GraphSAGE [32] is the first proposed graph sampling algorithm for GNNs. For each node, GraphSAGE randomly samples a fixed number of neighbor nodes with equal probability in each convolutional layer of the model for node feature aggregation. Different from GraphSAGE, Cluster-GCN [33] performs random sampling in units of subgraphs. It first divides the original graph into multiple clusters by using some graph clustering algorithms and randomly selects a fixed number of clusters, combined into a subgraph for training. Parallelize Graph Sampling [34] is proposed to accurately and efficiently train large-scale graph data, which also randomly samples nodes when generating subgraphs for training.

In summary, random sampling treats the sampled nodes as obeying a uniform distribution. This method overcomes the limitation of neighborhood explosion when aggregating node features and avoids the out-of-memory issue.

Importance Sampling. Different from random sampling, in order to sample more informative nodes, importance sampling assigns each node a different sampling probability. Importance samplings can reduce the variance caused by sampling, leading to a more stable training process and enhanced model performance, which can be classified into node-wise sampling, layer-wise sampling, and subgraph-based sampling based on previous work [250], [251].

Specifically, FastGCN [35] and LADIES [36] are typical layer-wise sampling algorithms. FastGCN performs node sampling independently at each layer based on a pre-set probability, with the sampling probability calculated by the degree of the node. It believes that nodes with higher degrees contain more information. Differently, LADIES calculates the layer-dependent laplacian matrix for nodes to obtain the sampling probability of each node. GraphSAINT [37] is a subgraph-based sampling algorithm, which additionally introduces the calculation method of the sampling probability on the edge. These probabilities are also related to the adjacency matrix of the graph and are independent of the training of the model. Additionally, some methods are considered as node-wise sampling by us and they are based on random walk such as PinSage [38] and HetGNN [39], which preferentially sample nodes with a high number of visits after the random walk. We also consider the number of visits as the importance of the node.

In Chapter IV-A2, we will discuss adaptive and learnable sampling methods, which perform better sampling of the original graph as the model is trained.

B. Feature

In this section, we first delve into feature augmentation, dividing our discussion into two distinct categories: pre-Attribute augmentation and attribute-free augmentation. Following this, we explore position encoding methods, which are crucial for capturing structural information in graph networks. This exploration is further categorized into two types: absolute position encoding and relative position encoding.

1) Feature Augmentation: By creating or modifying node features, feature augmentation introduces additional, relevant information into the dataset, which helps the model generalize better and avoid overfitting. Depending on the initial availability of node features in a dataset, graph feature augmentation methods can be divided into two main categories: vanilla methods and feature-free methods.

Pre-Attribute Augmentation. In graphs equipped with pre-existing features, augmentation methods aim to enhance the utility and diversity of these features through various intuitive adjustments. These methods can be categorized as follows: feature corruption [40]–[42], which introduces controlled noise to the features; feature shuffling, which rearranges the features; feature masking [43], [44], which selectively hides certain features; feature addition, which introduces new features; feature rewriting [45], [46], which alters existing features; feature propagation, which spreads features across the graph; and feature mixing [47], which combines features from different nodes. Further details and examples of these methods can be found in a comprehensive survey on the topic [51].

Attribute-Free Augmentation. For nodes without initial features, several methods have been proposed to generate meaningful features. A prominent approach is the utilization of random walks to capture structural information. Perozzi introduces DeepWalk *et al.* [3], which initiates multiple random walks from each node and uses the sequence of nodes in these walks to generate node embeddings utilizing word2vec [172]. Building on this, node2vec [4] extends DeepWalk by incorporating a flexible probability mechanism for guiding the random walks, thereby offering a more nuanced exploration of the graph structure. In contrast, an alternative approach known as SDNE [173] employs an encoder-decoder architecture for node feature learning. In this method, each column of the adjacency matrix is treated as the initial node embeddings and serves as input to the encoder. The model computes its loss by assessing the disparity between these initial embeddings and the embeddings generated by the decoder, effectively capturing the underlying structural essence of the graph.

In general, feature augmentation exhibits diversity and flexibility, allowing for customized enhancements tailored to the specific requirements of a given problem.

2) Position Encoding: It is well-recognized that the expressive power of message-passing neural networks (MPNNs) is bounded by the 1-Weisfeiler-Lehman (WL) test and cannot distinguish the isomorphism graphs [252]. To break this limitation, a popular way is to augment the node features with some positional information, known as the positional encodings. In this section, we introduce two types of position encodings: absolute methods and relative methods.

Absolute Position Encoding (APE). The objective of APE is to assign each node a position representation to indicate its unique position in the whole graph. A popular approach for APE involves utilizing the eigenvectors of the graph Laplacian, which is first introduced by Dwivedi *et al.* [174] in the graph Transformer architecture and then adopted by [181].

However, the eigenvectors suffer from the sign- and basis-ambiguity, implying that randomly reflecting the signs or rotating the coordinates still satisfies the definition of eigenvalue decomposition [175]. To solve these issues, SAN [176] proposes to use the Transformer to learn a rotations-equivariant APE. Due to the permutation-invariant property of self-attention, SAN is invariant to the rotations of eigenvectors' coordinates. Additionally, SignNet [175] proposes to simultaneously take the positive and negative eigenvectors as input, so as to address the sign-flipping problem. And then use invariant and equivariant GNNs to learn positional representations from the eigenvectors.

There are also other APE methods, such as random features [177] and random walk features [178]. However, they suffer from either poor generalization or limited receptive field, which limits their widespread use in graph models.

Relative Position Encoding (RPE).RPE captures the relational information between two nodes by employing the distances between them as position encodings. The existing methods are categorized into 1D-RPE and 2D-RPE based on the involved dimensionality.

Given a target node, 1D-RPE methods aim to leverage the distance between the anchor nodes and the target node as the positional representations. The pioneering 1D-RPE method, PGNN [179], adopts a strategy of random sampling anchor nodes and utilizing the distances to these anchor nodes as position encodings. Li *et al.* [180] further proposes distance encoding, which takes the geodesic distances between nodes as the relative positions and avoids the choice of anchor nodes.

2D-RPE methods frequently function as the inductive bias for graph structures, a key component widely employed in the graph Transformer architecture. For example, Graphormer [181] encodes the information of the shortest path between two nodes as the 2D-RPE and adds it to the self-attention matrix to preserve the structural information, which can be formulated as:

$$A_{ij} = \frac{(\mathbf{h}_i \mathbf{W}_Q)(\mathbf{h}_j \mathbf{W}_K)^T}{\sqrt{r}} + b_{\phi(v_i, v_j)}, \tag{9}$$

where $\mathbf{h}_i$ and $\mathbf{h}_j$ are the representations of nodes v_i and v_j, respectively, r is the dimension of representations, and $\phi(v_i, v_j)$ indicates the shortest path encoding function.

Generally, 1D-RPEs are more user-friendly, whereas 2D-RPEs provide a more comprehensive set of positional information. The 1D-RPEs can be transformed into 2D-RPEs. For example, PEG [182] uses the Euclidean distance between eigenvectors to reweight the adjacency matrix, which combines the advantages of both APE and RPE.

C. Label

In this section, we explore various techniques designed to augment the existing graph label data and mitigate issues related to overfitting and noisy labels. These include label mixup methods, which blend labels from different instances to create new training examples, and knowledge distillation, where a teacher model is employed to generate labels for unlabeled data. Additionally, we discuss alternative approaches that focus on refining or selectively using labels to improve the training process. These methodologies collectively contribute to the robustness and accuracy of GNNs in handling graph-structured data.

1) Label Mixup : Label mixup combines two different instances including their associated labels as a new instance, and employs the mixed instances to train GNN models. This approach effectively increases the diversity of the training data, helping the model to better handle a wider range of inputs, thereby making the learned models more generalized and less overfitted. Label mixup methods in graph learning can be broadly divided into graph-level mixup and node-level mixup based on the mixing object.

Graph-level Mixup. The graph-level mixup involves mixing labels of two distinct graphs. For instance, G-Mixup [205] starts by estimating a graphon [206] using graphs of the same class and then progresses to interpolating graphons of different classes in Euclidean space. This interpolation creates mixed graphons, from which synthetic graphs are generated through sampling. Concurrently, it blends the labels of these classes using a predefined parameter. Besides, GraphMAD [207] integrates topology by performing nonlinear graph mixup within a continuous domain characterized by graphons. Moreover, it employs convex clustering to learn data-driven mixup functions, allowing generated samples to exploit relationships among all graphs rather than just pairs of data. Similar to G-Mixup, it combines labels using a predefined weight. Meanwhile, Graph Transplant [208] combines irregular graphs by transplanting subgraphs between instances. This approach leverages node saliency to select meaningful subgraphs effectively and uses adaptive label assignment, demonstrating improved performance across diverse graph domains.

Node-level Mixup. Conversely, the node-level mixup focuses on mixing labels of different nodes, particularly for the node classification task. A notable approach proposed by Wang et al. [209] utilizes a two-stage Mixup framework. In the first stage, a standard feed-forward process within GNNs is employed to obtain node representations. Following this, in the second stage, Mixup is applied using representations of each node's neighbors derived from the first stage, and the label is mixed by a hyperparameter weight same as the aforementioned works.

2) Knowledge Distillation: Knowledge distillation [253], [254] enables a light-weight model (*i.e.*, student) to acquire knowledge by regarding the soft predictions made by a high-capacity model (*i.e.*, teacher) as pseudo labels for unlabeled nodes/graphs. In this survey, we focus on offline distillation techniques [210]–[221], which can be seen as data-centric enrichment of labels. Specifically, a static teacher model is utilized to produce labels for unlabeled data, and these freshly generated labels are then harnessed to train the student model. For instance, Yang et al. [212] introduce a knowledge distillation approach tailored for GCNs, utilizing a Local Structure Preserving (LSP) module to ensure topology-aware transfer of knowledge from a teacher model to a compact student model. BGNN [210] sequentially transfers knowledge from multiple GNNs into a student GNN, augmented by an adaptive temperature module and a weight boosting module to enhance learning effectiveness. Furthermore, CPF [214] extracts GNN knowledge from a pre-trained GNN model and infuses it into a simpler student model, which combines label propagation and feature transformation, to enhance prediction accuracy while preserving interpretability.

3) Others: Beyond the techniques of label mixup and knowledge distillation, which predominantly generate extra labels for unlabeled sets, there exists a body of work that specifically focuses on modifying [255]–[257] or omitting noisy labels [258] in the labeled set. For instance, Zhong et al. [257] employs a GCN to refine noisy predictions by establishing relationships between high-confidence snippets and low-confidence ones, thereby propagating anomaly information to correct erroneous labels. Similarly, GNN Cleaner [255] corrects noisy labels by generating pseudo labels through label propagation, and then adaptively and dynamically adjusting these labels during training. In a different vein, Better With Less [258] introduces a novel graph selector that identifies the most instructive data points based on predictive uncertainty and inherent properties of graphs.

IV. Training Stage

In this section, we introduce the graph data modification method in the training phase, where the data modification module and the task-solving model cooperate with each other to improve performance. Following Section III, we also consider how to operate (*i.e.,* add, delete, or change) different data structures (*i.e.,* topology, feature and label) in each subsection. The related methods can be seen in Table I.

A. Topology

In this section, we delve into diverse techniques crafted for modifying the graph structure throughout model training. Initially, we present graph adaptive augmentation, striving to seamlessly integrate augmentation procedures during the training phase. Subsequently, we discuss graph adaptive sampling methods capable of adjusting the sampling strategy according to the current model's performance. Besides, we also present graph structure learning which endeavors to uncover valuable graph structures from data. Finally, we explore self-paced learning which allows the model to measure the difficulty of instances and determine the training progress according to the current model state.

1) Graph Adaptive Augmentation: The conventional rule-based augmentation methods may not be sufficient when demanding increased robustness and improved performance, primarily due to their independence from the task-solving model's training process. Conversely, graph adaptive augmentation techniques integrate augmentation procedures seamlessly during the training phase, which we classify into four distinct categories. Edge-based methods revamp the adjacency matrix guided by loss functions, subgraph-based methods concentrate on extracting a more informative subgraph, spectrum-based methods propose to generate an augmentation from the graph spectrum view, and automated augmentation frameworks introduce to learn optimal augmentation strategy for varied scenarios.

Edge-based. To ensure that the edge augmentation process is guided by a differentiable loss function, some studies (e.g., GLCN [12], TO-GCN [13], Pro-GNN [14]) treat the graph as continuous rather than having exact edges. These works formulate loss functions by incorporating specific constraints, such as smoothness and sparsity, to generate gradients for refining the graph structure. As an example, Pro-GNN [14] proposes a loss function:

$$\mathcal{L} = ||\tilde{\mathbf{A}} - \mathbf{A}||_F^2 + \eta||\tilde{\mathbf{A}}||_1 + \beta||\tilde{\mathbf{A}}||_* + \rho(\mathbf{X}^T\hat{\mathbf{L}}\mathbf{X}) + \gamma\mathcal{L}_{GNN}, \tag{10}$$

where $\tilde{\mathbf{A}}$ is the augmented graph adjacency matrix, and $\hat{\mathbf{L}}$ is the normalized laplacian matrix. Specifically, the term $||\tilde{\mathbf{A}} - \mathbf{A}||_F^2$($|| \cdot ||_F$ means Frobenius norm) aims to make $\tilde{\mathbf{A}}$ close to A. The term $\eta||\tilde{\mathbf{A}}||_1$ ($|| \cdot ||_1$ means L_1 norm) and $\beta||\tilde{\mathbf{A}}||_*$ ($|| \cdot ||_*$ means nuclear norm) ensure sparsity and low-rank properties, respectively. The term $\rho(\mathbf{X}^T\hat{\mathbf{L}}\mathbf{X})$ controls feature smoothness, and $\gamma\mathcal{L}_{GNN}$ is the empirical loss.

Other studies do not directly implement alterations to the graph through gradients; instead, they iteratively generate or remove discrete edges during training. A representative work is AdaEdge [15], where edges are added or removed based on the classification results of their respective nodes. Similarly, GNNGuard [16] selectively removes edges between dissimilar nodes that may be malicious. Besides, TADropEdge [17] drops edges according to their weights, calculated from the graph spectrum. PTDNet [18] and NeuralSparse [19] leverage sparsity and low-rank properties to remove task-irrelevant edges. Furthermore, there are also some related works proposed to fulfill different demands, such as fairness ([20]), and anomaly detection ([21]).

Subgraph-based. Subgraph-based adaptive augmentation aims to find the most representative and informative subgraph (also known as the graph rationale), like the functional groups in a molecule, to enhance the model in terms of performance, interpretability, robustness, and so on ([22], [23], [24], [25]). For instance, GIB [22] extracts information from graph structure and node features, adopting an Information Bottleneck (IB) perspective, and encourages the learned representation to contain the minimum amount of information appropriate for downstream prediction tasks. Consequently, models trained using this approach demonstrate reduced susceptibility to overfitting and increased robustness against adversarial attacks. Moreover, GSAT [24] proposes stochastic attention to reduce information from the input graph, thus achieving better generalization and interpretation. GREA [25] introduces environment replacement augmentation, which identifies the rationale and separates it from the graph, then substitutes the remaining part, which they call the environment, to generate augmented data. Besides, several related methods garner attention within the out-of-distribution (OOD) research field ([26], [27], [28]). These approaches are devised to separate and leverage causal and non-causal, or invariant and variant subgraphs, with the aim of enhancing OOD generalization.

Spectrum-based. It is preferred to consider global information to better preserve graph properties when generating an augmentation [112]. Graph spectrum is such a tool that naturally incorporates global graph properties including clusterability, connectivity, d-regularity, etc., offering an effective way to gain rich information with respect to the overall structure of the graph. Specifically, the graph spectrum $\mathbf{\Lambda}$ is calculated by an eigendecomposition procedure:

$$\hat{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathbf{T}}, \tag{11}$$

With such a powerful tool, several spectral augmentation methods are proposed. SpCo [29] seeks to identify optimal contrastive pairs by amplifying high-frequency amplitudes, thereby introducing distinctions between the original and augmented graphs, while retaining low-frequency amplitudes to preserve invariant information within the graph. Simultaneously, it addresses the optimization problem by treating it as a matrix perturbation through Sinkhorn's Iteration. SPAN [30] also aims to maximize the spectral difference between two graphs, but the optimization is guided by gradients. Furthermore, SFA [31] presents two augmentation methods: spectral graph cropping, which involves the removal of the two smallest non-zero eigenvalues, and graph frequency component reordering, which permutes the eigenvectors associated with the top k eigenvalues.

Automated Augmentation. The methods mentioned above utilize fixed augmentation strategies. Nonetheless, the same augmentation strategy may not be suitable for diverse datasets. Consequently, a significant portion of research is transitioning towards automatic augmentation, which entails the learning of the optimal augmentation strategy during the model training process. For example, JOAO [115] sets up a bi-level optimization procedure to train the encoder and augmentation strategy simultaneously. Besides, many methods take advantage of reinforcement learning (RL) in data augmentation. LG2AR [117] learns a policy to assign probabilities to a set of strategies (edge dropping, feature masking, etc.) and samples from them in each training epoch. AutoGDA [118] introduces a community-customized graph data augmentation method that employs an RL-agent to determine the optimal strategy for each community within a given graph. GraphAug [119] focuses on providing label-invariance augmentations for different graphs, implemented by an RL-agent which takes graph embedding as input and samples a strategy.

2) Graph Adaptive Sampling: In this subsection, we discuss graph adaptive sampling algorithms, which can adjust the sampling strategy based on the current model's performance, facilitating more effective utilization of graph information during the training process and thus improving the model's performance. According to the strategy and scope during node sampling, we currently classify these methods into node-wise sampling and layer-wise sampling as mentioned in III-A5.

Node-wise Sampling. Node-wise sampling is a fundamental sampling technique that samples a fixed number of neighboring nodes for each target node layer by layer.

In particular, VR-GCN [120] performs a fixed number of random sampling of neighbor nodes like GraphSAGE. However, we classify it as a graph-adaptive sampling method because it retains and utilizes intermediate node information that evolves as the model learns to approximate unsampled nodes. This approach enables the model to gather a relatively greater amount of node information, ultimately enhancing model performance with a small number of samples. PASS [121] directly utilizes gradient information and task performance loss to train a sampling policy. Since the sampling operation is non-differentiable, PASS proposes to learn from the gradients to determine which neighbors provide valuable information and are therefore assigned high sampling probabilities. Besides, GCN-BS [122] and Thanos [123] consider the sampling problem from another perspective. GCN-BS reformulates the optimization of sampling variance by treating it as a bandit problem, establishing a connection between sampling variance and both the reward and the regret. To achieve adaptive sampling probabilities, GCN-BS calculates the reward and subsequently updates the sampler following back propagation. Based on GCN-BS, Thanos introduces a novel biased reward function and corresponding regret, while also loosening the model's data-related assumptions. Experimental results demonstrate that Thanos surpasses GCN-BS in terms of approximation error and converges at a nearly optimal rate. Furthermore, ANS-GT [124] is applied to the graph transformer, which develops a framework that combines multiple node-wise heuristic sampling strategies and designs learnable weights for each heuristic strategy to adaptively select the best-performing sampling algorithm during training, making the model achieve the best performance.

Layer-wise Sampling. Different from the above method, layer-wise sampling abandons sampling from nodes and, instead, within the realm of graph convolutional layers, selects a consistent number of nodes at each layer. For example, AS-GCN [125] is a typical adaptive layer-wise sampling method, which samples nodes for the lower layers based on the nodes in the top layer and enables the sharing of sampled neighbor nodes among different parent nodes. This approach facilitates control over the number of nodes at each layer, preventing excessive expansion. More importantly, the newly proposed layer-wise sampling method in AS-GCN is adaptive and explicitly reduces sampling variance, thereby enhancing the training effectiveness of this approach. Additionally, MVS-GNN [126] theoretically analyzes the components of sampling variance and introduces gradient-based minimization of variance sampling. It samples from the optimal importance sampling distribution by computing the gradient norm, ensuring that the sampled nodes consistently maintain the smallest sampling variance.

3) Graph Structure Learning: The effectiveness of GNNs in capturing expressive representations is significantly influenced by the quality of the underlying graph-structured data. However, real-world graph structures are frequently noisy or incomplete. Therefore, Graph Structure Learning (GSL) endeavors to uncover valuable graph structures from data to enhance the learning of graph representations. Depending on whether weight information on the edges is considered, existing GSL methodologies can be broadly classified into two categories: learning discrete graph structures and learning weighted graph structures.

Learning Discrete Graph Structures. The methods in this category consider graph structures as random variables, where discrete graph structures can be sampled from a certain probabilistic adjacency matrix. Various techniques, such as variational inference [150]–[154], bi-level optimization [155], [156], and reinforcement learning [157], are leveraged to jointly optimize both the probabilistic adjacency matrix and GNN parameters. For example, G^3NN [150] treats node features, graph structure, and node labels as random variables, then utilizes a flexible generative model to capture their joint distribution for the graph generation. VGCNs [151] aims to maximize the posterior over binary adjacency matrix given the observed data, namely node features and observed node labels. In contrast to variational inference approaches, LDS [155] and GSEBO [156] view the task, i.e., optimizing the graph structure and GNN parameters, as a bi-level optimization problem. Furthermore, DGM [157] employs reinforcement learning to tackle the non-differentiability challenge arising from the edge sampling operation during optimization.

Learning Weighted Graph Structures. Since learning discrete graph structures tends to optimize the adjacency matrix directly based on certain prior constraints on the graph properties, many of these approaches are unsuitable for the inductive learning setting where unseen nodes emerge during the inference phase [259]. Consequently, inspired by attention techniques [6], a class of methods focuses on learning weighted graph structures, i.e., edge weights between nodes. Based on various similarity measures, such as cosine-based similarity [158], [159], attention mechanisms [6], [160]–[164], kernel-based similarity [165], [166], many methods take node embedding to learn pairwise node similarity matrices. Some methods further integrate intrinsic edge embeddings [167], [168] or edge connection information [169] into the process of similarity learning. Additionally, graph regularization techniques [12], [14], [158], [170], [171] directly optimize the graph structure by considering various graph properties, such as smoothness, connectivity, low rank and sparsity. For instance, Pro-GNN [14] learns a refined graph structure from a perturbed graph by leveraging the sparsity, low rank, and smoothness properties of the graph. When a new structure is learned through the above methods, completely discarding the original graph structure may result in the loss of valuable information. In light of this, recent work [158], [165] suggests utilizing both the learned graph structure and the original graph structure through a linear combination.

4) Graph Self-paced Learning: As a special curriculum learning algorithm, self-paced learning allows the task-solving model to measure the difficulty of instances and determine the training progress according to the current model state [260]. Similar to Section III-A4, we categorize existing methods into node-level, link-level, and graph-level methods.

Node-level Self-paced Learning. The basic idea is to determine the training nodes according to the current training state. DSP-GCN [140] and SPC-GNN [141] gradually incorporate unlabeled nodes with higher confidence predictions into the training set, while SS-GSELM [142] and SPGCN [143] according to the loss value of the labeled nodes in each training, nodes with smaller loss values are prioritized for training.

Edge-level Self-paced Learning. Different from node-level, edge-level self-paced learning gradually introduces the relationships between nodes in the training process. For instance, SCCABG [144] and SCCBG [145] determine the reliability of each edge by an adaptive clustering similarity measure, and then the edges gradually are included in order of reliability. SANE [146] gradually introduces the semantic relationship between nodes into the network representation learning by considering node similarity.

Graph-level Self-paced Learning. The core insight of such approaches is to gradually determine the context of the center node. For example, SeedNE [147], relying on the sampling probability of nodes, progressively selects difficult negative context nodes to learn better node representations. In contrast, SPGCL [148] prioritizes the nodes with the largest mutual information as neighbors. Meanwhile, SPARC [149] selects graph contexts for model training based on the number of labels in different classes, gradually focusing on contexts associated with rare classes.

B. Feature

In this section, we explore the manipulation of graph features during model training, organizing our discussion into two distinct categories: feature completion and feature selection. Feature completion is designed to generate missing node features, addressing incomplete features in graph data. Conversely, feature selection aims to pinpoint highly valuable features during model training.

1) Feature Completion: The majority of GNNs assume that the node features on the graph are complete, but this assumption is often broken in practical applications, the reasons mainly come from the following aspects [191]: (1) machine or human errors during the data collection process; (2) collecting the dataset completely is very costly in practice; (3) many users are not willing to provide complete personal information due to privacy protection. Therefore, to address the problem of incomplete features in graph data, feature completion as a key solution aims to fill in the missing node features in the graph. According to different types of graph data, the existing methods can be roughly categorized into homogeneous and heterogeneous graph-based feature completion.

Homogeneous Graph-based Feature Completion. For homogeneous graphs, GCN$_{mf}$ [191] employs the Gaussian mixture model to represent the missing node features. Based on a shared-latent space assumption on graphs, SAT [192] designs a distribution matching GNN architecture for feature completion. Amer [193] develops a novel generative adversarial network to generate missing attributes. GINN [194] reconstructs a complete graph by using a GNN-based autoencoder network.

Heterogeneous Graph-based Feature Completion. For more complex heterogeneous graphs, a series of node feature completion methods capable of handling different types of nodes and edges have been proposed [195]–[199]. For example, AC-HEN [195] utilizes feature aggregation and structural aggregation to obtain multi-view embeddings for completing missing attributes. HGNN-AC [196] pre-learns topological embeddings and then utilizes them to guide the feature completion. HGCA [197] designs an augmentation network that captures the semantic relationships between nodes and attributes to achieve fine-grained attribute completion. AutoAC [198] models the attribute completion problem as an automatic search problem for each missing attribute node.

2) Feature Selection: The cost of model training significantly rises when data used in machine learning algorithms displays sparse features in a high-dimensional space, which we call the curse of dimensionality. As a result, Feature Selection (FS) emerges as one approach to mitigate this challenge. The goal of FS is to identify highly correlated features with the labels and prioritize them during model training. FS not only helps reduce the computational costs associated with high-dimensional data but also improves model performance by fitting meaningful features. In graph learning, the frequently utilized FS methods, incorporated into the model training phase, can be classified into two types based on their relationship with downstream tasks: task-independent FS and task-specific FS.

Task-independent FS. Such methods concentrate on generating superior features and seamlessly integrate with any GNN model or downstream tasks. To start, various works [183]–[186] center around introducing a regularization objective for feature selection. For example, Graph Lasso [183] incorporates a graph regularizer based on Lasso [261] for the feature graph to account for structural information. Further, AsGNNS [185] combines L2, 1/L1-norm regularized attribute selection and GNNs together to extract meaningful features and eliminate noisy ones. Except for regularization-based FS, [187] proposes a task-independent method, which first extends the feature selection algorithm presented via Gumbel Softmax to GNNs to extract features, and then implements a mechanism to rank the extracted features. ADAPT [188] introduces a framework for feature selection with the goal of identifying informative features that accurately describe the adaptive neighborhood structure of a network.

Task-specific FS. In contrast, the task-specific FS methods take the GNN model's task into consideration. Noticing that selective aggregation outperforms default aggregation in node classification tasks, Dual-Net GNN [189] suggests a classifier model trained on a subset of input node features to predict node labels and a selector model that learns to provide the optimal input subset to the classifier for achieving best performance. Lin *et al.* [190] introduce FS-GCN, an FS method that integrates an indicator matrix into the propagation process of GCN. The optimization of the indicator matrix involves minimizing the cross-entropy loss derived from the semi-supervised node classification task, coupled with a sparsity-based regularization.

C. Label

In this section, we explore the augmentation of labels during model training, organizing our discussion into two distinct categories: active learning and pseudo labeling. Active learning is devised to strategically select the most impactful data and label it for enhancing the task-solving model. Additionally, pseudo labeling aims to expand the label set by employing a trained model and assigning pseudo-labels.

1) Active Learning: In real-world scenarios, demanding a large quantity of labeled data to achieve an excellent model is both expensive and unrealistic. Therefore, the concept of Active Learning (AL) has been introduced. It aims to select the most effective data for the task-solving model from the data set and label it to get the best model performance when the labeling cost is limited. While numerous AL methods have been employed in graph data, the majority remains centered on node classification problems. Categorically, the existing AL methods applied to graph data can be segregated into two groups: node-independent and node-correlated AL methods.

Node-independent AL. These methods usually select nodes in unlabeled data for labeling based on metrics and rules. They regard each node selection as an independent process and believe that the current node selection process will not have an impact on other nodes. For example, AGE [226] uses three indicators: information entropy, information density, and graph centrality to select the most informative nodes for labeling. ANRMAB [227] focuses on improving AGE by employing a multi-armed bandit mechanism to dynamically learn weights for balancing the aforementioned three metrics. Similar to ANRMAB, ActiveHNE [228] is generalized to heterogeneous graphs. Besides, SmartQuery [229] introduces degree and PageRank informativeness measurements to select nodes. Furthermore, ALG [230] introduces a novel metric called Effective Reception Field (ERF), which combines receptive field with node effectiveness measurements, leading to the involvement of more nodes in training when ERF is maximized.

Overall, these methods use greedy ideas to independently select nodes to label based on different measurements, and cannot guarantee that the model achieves long-term optimal performance.

Node-correlated AL. Differing from node-independent AL, such AL methods assume nodes are correlated and consider interactions between nodes. This idea avoids that the selected nodes are similar and clustered together, which leads to no duplication of information. Consider a straightforward example where, in node-independent methods, two similar and informative nodes might be chosen and labeled sequentially, resulting in a failure to maximize information acquisition for enhancing model performance within limited labeling costs. In contrast, node-correlated methods address this limitation. Currently, these methods can be broadly categorized into three groups: reinforcement learning (RL)-based, influence function-based, and clustering-based.

Many RL-based methods model active learning on graphs as a Markov decision process. The selection of nodes for marking is regarded as an action, and the performance of the model based on the selected nodes is viewed as a reward, so as to consider the interaction between nodes and obtain the long-term performance of the model. Specifically, GPA [231] parameterizes the policy network as a GNN and utilizes reinforcement learning to train the policy network. Subsequently, the network generates a probability for each unlabeled node and then samples a node for labeling based on the probability. Inspired by GPA, DAG [232] learns a graph-specific policy based on a universal policy for each graph when solving the transferable problem. The knowledge learned by graph-specific policies is dynamically distilled into the universal policy by minimizing the KL divergence between graph-specific policies. Similarly, ALLIE [233] applies an imbalance-aware reinforcement learning based graph policy network to find unlabeled nodes that maximize model performance, and uses a graph coarsening strategy to make it applicable to the large-scale graph. BIGENE [234] introduces multi-agent reinforcement learning into batch active learning to improve sampling efficiency and considers both node informativeness and diversity.

There are still some methods based on the influence function that consider the information of each node. After message passing, the most effective nodes are selected through the aggregation of node information from a full-graph perspective to obtain long-term effects. For example, Grain [235] connects active learning on graphs with social influence maximization and proposes a diversified influence maximization strategy to select nodes. SAG [236] introduces the L1-norm of the expected Jacobian matrix as the influence of nodes after k-layer GCN propagation. IGP [237] proposes relaxed query and soft label conditions, and selects nodes by maximizing a new criterion called information gain propagation. Moreover, the influence function is employed in JuryGCN [238] to quantify jackknife uncertainty for each node, and nodes exhibiting high jackknife uncertainty are then selected for active learning.

Clustering-based active learning methods for graph data ensure the diversity of nodes to the greatest extent and avoid redundant information. For example, FeatProp [239] uses propagated node features for clustering and labels the nodes at the center of each cluster. Unlike FeatProp, LSCALE [240] clusters node embeddings in a latent space that contains two key attributes: low label requirements and informative distances. In ScatterSample [241], the uncertainty of all nodes is first calculated, and then the top uncertain nodes are clustered to ensure the diversity of sampling.

Some other different advanced technologies have also been proposed. SEAL [242] introduces adversarial learning into graph active learning for the first time. And MetAL [243] evaluates the importance of nodes based on meta-gradients rather than heuristic rules.

In general, no matter what strategy is used (RL-based, influence function-based and clustering-based, etc.), the node-correlated AL methods always do not treat the selection of each node as an independent process. They consider both the informativeness and diversity of nodes to different extents to avoid redundant information and achieve long-term effects.

2) Pseudo Labeling: With a multi-stage training paradigm, Pseudo Labeling [222]–[225] utilizes a trained model to expand the label set by assigning a pseudo-label, then fine-tunes the trained model or re-trains a new model. For example, Li et al. [222] first present a pseudo-labeled GCN, where the top K high-confidence unlabeled nodes are selected to expand the training set for model retraining. MT-GCN [223] takes the pseudo label to accomplish the mutual teaching process across the two GCNs. M3S [224] employs a deep clustering model to assign pseudo-labels. InfoGNN [225] considers both informativeness and prediction confidence of pseudo-labeled nodes, to further solve the problem of information redundancy and noisy pseudo-labels in existing methods.

V. Inference Stage

The inference stage refers to the phase where a pretrained graph model is used for the prediction of downstream tasks. Moreover, inference data refers to the graph data utilized during the inference phase of pretrained models. In this stage, we can adjust the inference data to fulfill different trustworthy requirements or reformulate the downstream tasks to align with the pretext tasks ensuring high-quality knowledge transfer. From a data-centric perspective, using prompts to modify the inference data can help obtain the desired objectives without changing model parameters. In this section, we discuss prompt learning methods which are gradually gaining popularity in the realm of graphs. To elaborate, we classify existing graph prompting methods into two categories: pre-prompt and post-prompt, based on whether the task-specific prompts operate before or after the message passing module as shown in Figure1.

A. Pre-prompt

In the case of pre-prompt methods, existing works such as [200]–[202], [262]–[264] modify the input graph data either in terms of topology or node features before message passing to facilitate downstream task adaptation, or they construct a prompt graph to promote the model's adaptation to downstream tasks.

A classical example that applies prompt learning to graph neural networks is AAGOD [200], which utilizes prompt learning to modify the graph topology thus achieving adaptation for out-of-distribution (OOD) detection tasks without changing the parameters of GNN backbones. Using a parameterized matrix as a learnable instance-specific prompt, AAGOD superimposes this prompt on the adjacency matrix of the original input graph and reuses the well-trained GNN to encode the modified graph into vector representations. The instance-specific prompt helps highlight potential patterns in the in-distribution (ID) graphs, thereby increasing the difference between OOD and ID graphs.

In a similar vein, a notable contribution in this area is a multi-task prompt method [201] which uses prompts to modify the input feature. It bridges the gap between graph pre-training and downstream tasks under the multi-task background. This work initially constructs induced graphs for nodes and edges through τ-hop neighbors, thereby reformulating node-level and edge-level tasks into graph-level tasks. Additionally, it designs prompt tokens for the input graph and modifies the features of each node by weighting all the prompt tokens before message passing. Finally, using the meta-learning paradigm, the prompt parameters are updated for multi-task scenarios.

Another approach to designing prompts for graphs involves utilizing a prompt graph to assist model training. Prodigy [202] proposes a graph prompt learning design that not only reformulates downstream tasks into a unified template but also introduces a prompt graph with regard to few-shot tuning. In the process of few-shot prompting for downstream tasks, Prodigy first retrieves the corresponding context from the source graph data. Subsequently, it abstracts the corresponding task into a task graph and combines it with the data graph to form a prompt graph. Once the prompt graph is obtained, leveraging a proficiently trained GNN via specifically crafted pre-training tasks enables the acquisition of predicted labels for each data point.

B. Post-prompt

As for post-prompt methods [203], [204], task-specific prompts often operate on representations that have already undergone message passing to enable downstream task adaptation.

GraphPrompt [203] is also among the early attempts at prompt learning in the field of graphs. It similarly aims to bridge the gap between pretext tasks and downstream tasks but in a post-prompt manner. This framework begins with pre-training on unlabeled graphs employing a self-supervised link prediction task. It unifies node classification tasks and graph classification tasks into a link prediction form by adding pseudo nodes, thereby eliminating the gap between pretext tasks and downstream tasks. Subsequently, it utilizes learnable prompts to guide each downstream task. This learnable prompt can be understood as a weighted mask to the readout representation. As a result, it can be applied to various tasks, each with a distinct emphasis on different feature channels.

Another pioneering work is GPPT [204]. Unlike GraphPrompt which achieves a form of multi-task unification for tasks at graph-level, node-level, and edge-level, GPPT primarily focuses on node classification tasks. Similarly, GPPT utilizes link prediction as a pretraining task, although it differs in prompt design by concatenating task-specific prompts with node representations to guide adaptation.

VI. Problematic Graph Data

Manually defined and processed graph data inevitably introduces noise and problems. The aforementioned methods are usually used in general graph data, without considering the specific issues hidden in the graph structure. In this section, we list a series of commonly introduced problematic graph data and discuss how to deal with these issues in a data-centric approach.

Vulnerability. Recent advances in graph adversarial learning show that graph structures are vulnerable [265], where a small perturbation in the structures, features, or labels can significantly affect the predictions of the graph models [266]. However, existing methods focus on designing graph defense models to handle adversarial edges and lack generalization. The emerging certificate method [267] provides a data-centric way to consistently improve the robustness of data against perturbations, which has been widely used in graph data. For example, Bojchevski *et al.* [268] first propose the verifying certifiable robustness of graph data, improving the robustness by constraining the local and global certificates. Tao *et al.* [269] further present the immunization method for graph data, which injects protective edges to improve the robustness of graphs against perturbations.

Unfairness. Existing literature shows that graph models may have inherent prejudice if the training data contains sensitive attributes or specific structures. Feature fairness requires graph models to make predictions without using sensitive attributes, such as gender and race. Agarwal *et al.* [270] propose fairness-aware graph augmentation, which utilizes the counterfactual perturbation to make graph models learn invariant representations against sensitive attributes. Structural fairness refers to the phenomenon that graph models have different accuracy on nodes with different structures, *e.g.*, high-degree and low-degree nodes [271]. GRADE [272] first finds that graph contrastive learning has better structural fairness than semi-supervised GNNs. Based on this discovery, they propose interpolation-based and purification-based graph augmentations for low- and high-degree nodes.

Selection Bias. Due to the influence of human factors, the collection of graph data will inevitably introduce selection bias, making the distribution of training data and test data inconsistent. For example, existing molecular graph datasets can only cover part of the overall molecular distribution, resulting in distribution shifts. One data-centric approach to mitigating this problem is stable learning [273], which can be viewed as a special case of data sampling. For example, SGL [274] introduces a stable graph learning framework, which can learn invariant patterns against selection bias in an unsupervised way. DGNN [275] re-weights the node weights to remove the spurious correlations in node representations.

Heterophily. Most graph models rely on the homophily assumption, *i.e.*, nodes belonging to the same class tend to connect with each other, to learn representations. However, real-world graphs have mixed patterns, where heterophilic graphs exist, such as protein-protein interaction networks [276]. The homophilic ratio of graphs significantly affects the performance of graph models [277]. Some methods use graph structure learning to find the non-local neighbors to alleviate the heterophily of graphs. For example, Geom-GCN [278] constructs a latent geometric graph to enrich the potential homophilic neighbors. AM-GCN [279] proposes to fuse the original graph and kNN graph of node features, which leverages the feature similarity to enhance the graph homophily.

VII. Future Directions

Data-centric AI is an emerging topic in deep learning. Here we present several promising future directions for data-centric graph learning.

A. Standardized Graph Data Processing

Existing graph construction and data processing methods rely on expert priors to define the structures, features, and labels, which seriously hinders the transferability of graph data across different domains. For example, the graph models trained on citation networks cannot be used for social networks, due to their discrepancy in node features. *Is there a way that we can standardize the processing principles of graph data?* If so, the ubiquitous graph data can be unified and the knowledge can be transferred across domains. One possible approach is to use large language models (LLMs) to process graph data, unifying the node features in the language space [280].

B. Continuous Learning of Graph Data

Continuous learning aims to endow deep learning models with the ability to continuously learn new knowledge from a stream of data. An interesting question is that *can graph data also learn knowledge from the predictions of graph models?* Graph models can learn semantic information from the raw graph data and remove some noise. Based on the predictions of graph models, graph data can also be continuously optimized. In this way, graph data can learn in conjunction with the graph model and adaptively adjust based on various graph models and downstream tasks, rather than heavily depending on prior knowledge or assumptions. For example, graph condensation methods [85], [87] aim to use the gradients of graph models to generate new graph data, which can be seen as a special case of data continuous learning.

C. Graph Data & Models Co-development

While we have frequently highlighted the importance of high-quality graph data for the success of model-centric graph learning, it is crucial to acknowledge the reciprocal relationship. It is foreseeable that optimal data manipulation strategies and model design mutually influence each other, and there is no single set of graph strategies/models that consistently performs best when paired with different graph models/strategies. Therefore, *how to further encourage the collaborative development of graph data and models* is an important task for data-centric graph learning. The key to this lies in blurring the boundary between graph data and models, followed by the collaborative design of data-centric operations and model-centric methods. GraphStorm [281] adeptly focuses on both the development of graph data and the deployment of graph models, resulting in heightened effectiveness and efficiency.

D. Graph Data for LLM-based Graph Models

Motivated by the success of LLMs in language area, many works propose to explore graph models based on LLMs [280]. Similar to general LLMs, which require high-quality data for pre-training and whose capabilities largely depend on the pre-trained corpus and its preprocessing [282], improvements in the quality and quantity of graph data are also key factors contributing to the effectiveness of graph models based on LLMs [280]. Despite the numerous data modification methods discussed above, most of these techniques are typically tailored for traditional GNNs. Therefore, there is a pressing need for further exploration into how to effectively modify graph data for LLM-based graph models.

VIII. Conclusion

In this survey, we give a comprehensive review of data-centric graph learning. We categorize existing methods from two perspectives: One is the learning stage, including pre-processing, training, and inference. Another is the data structure, including topology, feature, and label. Through these two views, we carefully explain when to modify graph data and how to modify graph data to unlock the potential of the graph models. Besides, we also introduce some potential issues of graph data and discuss how to solve them in a data-centric approach. Finally, we propose several promising future directions in this field. To sum up, we believe that data-centric AI is a viable path to general artificial intelligence, and data-centric graph learning will play an important role in graph data mining.

IX. ACKNOWLEDGEMENTS

Writing a survey always involves more people than just the authors. We would like to express our sincere thanks to all those who worked with us on this paper. They are Boyu Chen, Haoran Dai, Ao Sun, Yue Yu, Yixin Xiao, Qi Zhang and Chunchen Wang. Additionally, this work was supported by the National Key Research and Development Program of China (No.2023YFC3303800).

References

[1] K. M. Borgwardt and H. Kriegel, "Shortest-path kernels on graphs," in *ICDM*, 2005.

[2] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *AISTATS*, 2009.

[3] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *KDD*, 2014.

[4] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016.

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

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

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2015.

[10] D. Zha, Z. P. Bhat, K.-H. Lai, F. Yang, Z. Jiang, S. Zhong, and X. Hu, "Data-centric artificial intelligence: A survey," *ArXiv*, vol. abs/2303.10158, 2023.

[11] J. Jakubik, M. Vossing, N. Kuhl, J. Walk, and G. Satzger, "Data-centric artificial intelligence," *ArXiv*, vol. abs/2212.11854, 2022.

[12] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo, "Semi-supervised learning with graph learning-convolutional networks," in *CVPR*, 2019.

[13] L. Yang, Z. Kang, X. Cao, D. Jin, B. Yang, and Y. Guo, "Topology optimization based graph convolutional network," in *IJCAI*, 2019.

[14] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *KDD*, 2020.

[15] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *AAAI*, 2020.

[16] X. Zhang and M. Zitnik, "Gnnguard: Defending graph neural networks against adversarial attacks," in *NeurIPS*, 2020.

[17] Z. Gao, S. Bhattacharya, L. Zhang, R. S. Blum, A. Ribeiro, and B. M. Sadler, "Training robust graph neural networks with topology adaptive edge dropping," *CoRR*, 2021.

[18] D. Luo, W. Cheng, W. Yu, B. Zong, J. Ni, H. Chen, and X. Zhang, "Learning to drop: Robust graph neural network via topological denoising," in *WSDM*, 2021.

[19] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Robust graph representation learning via neural sparsification," in *ICML*, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 11 458–11 468.

[20] I. Spinelli, S. Scardapane, A. Hussain, and A. Uncini, "Fairdrop: Biased edge dropout for enhancing fairness in graph representation learning," *IEEE Trans. Artif. Intell.*, 2022.

[21] T. Zhao, B. Ni, W. Yu, Z. Guo, N. Shah, and M. Jiang, "Action sequence augmentation for early graph-based anomaly detection," in *CIKM*, 2021.

[22] J. Yu, T. Xu, Y. Rong, Y. Bian, J. Huang, and R. He, "Graph information bottleneck for subgraph recognition," in *ICLR*, 2021.

[23] S. Chang, Y. Zhang, M. Yu, and T. S. Jaakkola, "Invariant rationalization," in *ICML*, 2020.

[24] S. Miao, M. Liu, and P. Li, "Interpretable and generalizable graph learning via stochastic attention mechanism," in *ICML*, 2022.

[25] G. Liu, T. Zhao, J. Xu, T. Luo, and M. Jiang, "Graph rationalization with environment-based augmentations," in *KDD*, 2022.

[26] Y. Wu, X. Wang, A. Zhang, X. He, and T. Chua, "Discovering invariant rationales for graph neural networks," in *ICLR*, 2022.

[27] Y. Chen, Y. Zhang, Y. Bian, H. Yang, K. Ma, B. Xie, T. Liu, B. Han, and J. Cheng, "Learning causally invariant representations for out-of-distribution generalization on graphs," in *NeurIPS*, 2022.

[28] H. Li, Z. Zhang, X. Wang, and W. Zhu, "Learning invariant graph representations for out-of-distribution generalization," in *NeurIPS*, 2022.

[29] N. Liu, X. Wang, D. Bo, C. Shi, and J. Pei, "Revisiting graph contrastive learning from the perspective of graph spectrum," in *NeurIPS*, 2022.

[30] L. Lin, J. Chen, and H. Wang, "Spectral augmentation for self-supervised learning on graphs," in *ICLR*. OpenReview.net, 2023.

[31] A. Ghose, Y. Zhang, J. Hao, and M. Coates, "Spectral augmentations for graph contrastive learning," in *AISTATS*, ser. Proceedings of Machine Learning Research, vol. 206. PMLR, 2023, pp. 11 213–11 266.

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

[33] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *KDD*, 2019, pp. 257–266.

[34] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Accurate, efficient and scalable graph embedding," in *IPDPS*. IEEE, 2019, pp. 462–471.

[35] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

[36] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," *NeurIPS*, 2019.

[37] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," *arXiv preprint arXiv:1907.04931*, 2019.

[38] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 974–983.

[39] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *KDD*, 2019, pp. 793–803.

[40] L. Yang, L. Zhang, and W. Yang, "Graph adversarial self-supervised learning," *NeurIPS*, vol. 34, pp. 14 887–14 899, 2021.

[41] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *arXiv preprint arXiv:1809.10341*, 2018.

[42] F. Feng, X. He, J. Tang, and T.-S. Chua, "Graph adversarial training: Dynamically regularizing based on graph structure," *IEEE TKDE*, vol. 33, no. 6, pp. 2493–2504, 2019.

[43] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *NeurIPS*, vol. 33, pp. 5812–5823, 2020.

[44] S. Thakoor, C. Tallec, M. G. Azar, M. Azabou, E. L. Dyer, R. Munos, P. Veličković, and M. Valko, "Large-scale representation learning on graphs via bootstrapping," *arXiv preprint arXiv:2102.06514*, 2021.

[45] Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi, "Nodeaug: Semi-supervised node classification with data augmentation," in *KDD*, 2020, pp. 207–217.

[46] Z. Xu, B. Du, and H. Tong, "Graph sanitation with application to node classification," in *WWW*, 2022, pp. 1136–1147.

[47] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio, "Manifold mixup: Better representations by interpolating hidden states," in *ICML*. PMLR, 2019, pp. 6438–6447.

[48] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE TKDE*, 2017.

[49] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[50] M. M. Bronstein, J. Bruna, Y. LeCun, A. D. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE SPM*, 2016.

[51] K. Ding, Z. Xu, H. Tong, and H. Liu, "Data augmentation for deep graph learning: A survey," *ArXiv*, vol. abs/2202.08235, 2022.

[52] P. Hu and W. Lau, "A survey and taxonomy of graph sampling," *ArXiv*, vol. abs/1308.5865, 2013.

[53] Y. Zhu, W. Xu, J. Zhang, Y. Du, J. Zhang, Q. Liu, C. Yang, and S. Wu, "A survey on graph structure learning: Progress and opportunities," *ArXiv*, 2021.

[54] X. Zheng, Y. Liu, Z. Bao, M. Fang, X. Hu, A. W.-C. Liew, and S. Pan, "Towards data-centric graph machine learning: Review and outlook," *ArXiv*, vol. abs/2309.10979, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:262067202

[55] J. J. Sylvester, "On an application of the new atomic theory to the graphical representation of the invariants and covariants of binary quantics, with three appendices," *AM J MATH*, vol. 1, no. 1, pp. 64–104, 1878.

[56] L. Euler, "Solutio problematis ad geometriam situs pertinentis," *Commentarii academiae scientiarum Petropolitanae*, pp. 128–140, 1741.

[57] J. L. Gross and J. Yellen, *Handbook of graph theory*. CRC press, 2003.

[58] X. Wang, D. Bo, C. Shi, S. Fan, Y. Ye, and P. S. Yu, "A survey on heterogeneous graph embedding: Methods, techniques, applications and sources," *IEEE TBD*, vol. 9, no. 2, pp. 415–436, 2023.

[59] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *AAAI*, 2019.

[60] E. Rossi, B. P. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *ArXiv*, vol. abs/2006.10637, 2020.

[61] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[62] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *NeurIPS*, vol. 33, pp. 22 118–22 133, 2020.

[63] A. A. Benczúr and D. R. Karger, "Approximating s-t minimum cuts in Õ(n2) time," in *STOC*. New York, NY, USA: Association for Computing Machinery, 1996, p. 47–55. [Online]. Available: https://doi.org/10.1145/237814.237827

[64] A. Benczur and D. R. Karger, "Randomized approximation schemes for cuts and flows in capacitated graphs," *arXiv preprint cs/0207078*, 2002.

[65] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi, "A general framework for graph sparsification," in *STOC*, 2011, pp. 71–80.

[66] R. S. Srinivasa, C. Xiao, L. Glass, J. Romberg, and J. Sun, "Fast graph attention networks using effective resistance based graph sparsification," *arXiv preprint arXiv:2006.08796*, 2020.

[67] R. Ahmed, G. Bodwin, F. D. Sahneh, K. Hamm, M. J. L. Jebelli, S. Kobourov, and R. Spence, "Graph spanners: A tutorial review," *Computer Science Review*, vol. 37, p. 100253, 2020.

[68] D. Peleg and A. A. Schäffer, "Graph spanners," *Journal of graph theory*, vol. 13, no. 1, pp. 99–116, 1989.

[69] Y. Chen, S. Khanna, and H. Li, "On weighted graph sparsification by linear sketching," in *FOCS*. IEEE, 2022, pp. 474–485.

[70] A. Filtser, M. Kapralov, and N. Nouri, "Graph spanners by sketching in dynamic streams and the simultaneous communication model," in *SODA*. SIAM, 2021, pp. 1894–1913.

[71] M. Kapralov, A. Mousavifar, C. Musco, C. Musco, N. Nouri, A. Sidford, and J. Tardos, "Fast and space efficient spectral sparsification in dynamic streams," in *SIAM*. SIAM, 2020, pp. 1814–1833.

[72] Y. Ye and S. Ji, "Sparse graph attention networks," *IEEE TKDE*, vol. 35, no. 1, pp. 905–916, 2021.

[73] V. N. Ioannidis, S. Chen, and G. B. Giannakis, "Pruned graph scattering transforms," in *ICLR*, 2020.

[74] J. Li, T. Zhang, H. Tian, S. Jin, M. Fardad, and R. Zafarani, "Graph sparsification with graph convolutional networks," *INT J DATA SCI*, pp. 1–14, 2022.

[75] ——, "Sgcn: A graph sparsifier based on graph convolutional networks," in *PAKDD*. Springer, 2020, pp. 275–287.

[76] V. Sadhanala, Y.-X. Wang, and R. Tibshirani, "Graph sparsification approaches for laplacian smoothing," in *AISTATS*. PMLR, 2016, pp. 1250–1259.

[77] J. Chen, Y. Saad, and Z. Zhang, "Graph coarsening: from scientific computing to machine learning," *SeMA Journal*, pp. 1–37, 2022.

[78] K. Chen, *Matrix preconditioning techniques and applications*. Cambridge University Press, 2005, vol. 19.

[79] A. C. Muresan and Y. Notay, "Analysis of aggregation-based multigrid," *SIAM Journal on Scientific Computing*, vol. 30, no. 2, pp. 1082–1103, 2008.

[80] G. Karypis and V. Kumar, "Multilevel graph partitioning schemes," in *ICPP (3)*, 1995, pp. 113–122.

[81] A. Loukas and P. Vandergheynst, "Spectrally approximating large graphs with smaller graphs," in *ICML*. PMLR, 2018, pp. 3237–3246.

[82] Y. Jin, A. Loukas, and J. JaJa, "Graph coarsening with preserved spectral properties," in *AISTATS*. PMLR, 2020, pp. 4452–4462.

[83] C. Cai, D. Wang, and Y. Wang, "Graph coarsening with neural networks," *arXiv preprint arXiv:2102.01350*, 2021.

[84] Z. Huang, S. Zhang, C. Xi, T. Liu, and M. Zhou, "Scaling up graph neural networks via graph coarsening," in *KDD*, 2021, pp. 675–684.

[85] W. Jin, L. Zhao, S. Zhang, Y. Liu, J. Tang, and N. Shah, "Graph condensation for graph neural networks," *arXiv preprint arXiv:2110.07580*, 2021.

[86] B. Zhao, K. R. Mopuri, and H. Bilen, "Dataset condensation with gradient matching," *arXiv preprint arXiv:2006.05929*, 2020.

[87] W. Jin, X. Tang, H. Jiang, Z. Li, D. Zhang, J. Tang, and B. Yin, "Condensing graphs via one-step gradient matching," in *KDD*, 2022, pp. 720–730.

[88] X. Zheng, M. Zhang, C. Chen, Q. V. H. Nguyen, X. Zhu, and S. Pan, "Structure-free graph condensation: From large-scale graphs to condensed graph-free data," *arXiv preprint arXiv:2306.02664*, 2023.

[89] M. Liu, S. Li, X. Chen, and L. Song, "Graph condensation via receptive field distribution matching," *arXiv preprint arXiv:2206.13697*, 2022.

[90] Z. Xu, Y. Chen, M. Pan, H. Chen, M. Das, H. Yang, and H. Tong, "Kernel ridge regression-based graph dataset distillation," in *KDD*, 2023, pp. 2850–2861.

[91] B. Yang, K. Wang, Q. Sun, C. Ji, X. Fu, H. Tang, Y. You, and J. Li, "Does graph distillation see like vision dataset counterpart?" *arXiv preprint arXiv:2310.09192*, 2023.

[92] Y. Liu, D. Bo, and C. Shi, "Graph condensation via eigenbasis matching," *arXiv preprint arXiv:2310.09202*, 2023.

[93] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

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

[95] M. Popova, M. Shvets, J. Oliva, and O. Isayev, "Molecularrnn: Generating realistic molecular graphs with optimized properties," *arXiv preprint arXiv:1905.13372*, 2019.

[96] Y. Luo, K. Yan, and S. Ji, "Graphdf: A discrete flow model for molecular graph generation," in *ICML*, 2021.

[97] N. Goyal, H. V. Jain, and S. Ranu, "Graphgen: A scalable approach to domain-agnostic labeled graph generation," in *WWW*, 2020.

[98] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard, "Digress: Discrete denoising diffusion for graph generation," *arXiv preprint arXiv:2209.14734*, 2022.

[99] X. Chen, J. He, X. Han, and L. Liu, "Efficient and degree-guided graph generation via discrete diffusion modeling," in *ICML*, vol. 202, 2023, pp. 4585–4610.

[100] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[101] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky, "Graph normalizing flows," *NeurIPS*, vol. 32.

[102] K. Martinkus, A. Loukas, N. Perraudin, and R. Wattenhofer, "Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators," in *ICML*. PMLR, 2022, pp. 15 159–15 179.

[103] T. Luo, Z. Mo, and S. J. Pan, "Fast graph generation via spectral diffusion." *CoRR*, 2022.

[104] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning," *CoRR*, 2020.

[105] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," in *NeurIPS*, 2020.

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

[107] Q. Tan, N. Liu, X. Huang, R. Chen, S. Choi, and X. Hu, "MGAE: masked autoencoders for self-supervised learning on graphs," *CoRR*, 2022.

[108] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2020.

[109] M. Sun, J. Xing, H. Wang, B. Chen, and J. Zhou, "Mocl: Data-driven molecular fingerprint via knowledge-aware contrastive learning from molecular graph," in *KDD*, 2021.

[110] J. Yoo, S. Shim, and U. Kang, "Model-agnostic augmentation for accurate graph classification," in *WWW*, 2022.

[111] J. Zhou, J. Shen, and Q. Xuan, "Data augmentation for graph classification," in *CIKM*, 2020.

[112] J. Klicpera, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," in *NeurIPS*, 2019.

[113] K. Hassani and A. H. K. Ahmadi, "Contrastive multi-view representation learning on graphs," in *ICML*, 2020.

[114] J. Yuan, H. Yu, M. Cao, M. Xu, J. Xie, and C. Wang, "Semi-supervised and self-supervised classification with multi-view graph neural networks," in *CIKM*, 2021.

[115] Y. You, T. Chen, Y. Shen, and Z. Wang, "Graph contrastive learning automated," in *ICML*. PMLR, 2021, pp. 12 121–12 132.

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

[117] K. Hassani and A. H. K. Ahmadi, "Learning graph augmentations to learn graph representations," *CoRR*, 2022.

[118] T. Zhao, X. Tang, D. Zhang, H. Jiang, N. Rao, Y. Song, P. Agrawal, K. Subbian, B. Yin, and M. Jiang, "Autogda: Automated graph data augmentation for node classification," in *LoG*, 2022.

[119] Y. Luo, M. McThrow, W. Y. Au, T. Komikado, K. Uchino, K. Maruhashi, and S. Ji, "Automated data augmentations for graph classification," in *ICLR*, 2023.
[120] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," *arXiv preprint arXiv:1710.10568*, 2017.
[121] M. Yoon, T. Gervet, B. Shi, S. Niu, Q. He, and J. Yang, "Performance-adaptive sampling strategy towards fast and accurate graph neural networks," in *KDD*, 2021, pp. 2046–2056.
[122] Z. Liu, Z. Wu, Z. Zhang, J. Zhou, S. Yang, L. Song, and Y. Qi, "Bandit samplers for training graph neural networks," *NeurIPS*, pp. 6878–6888, 2020.
[123] Q. Zhang, D. Wipf, Q. Gan, and L. Song, "A biased graph neural network sampler with near-optimal regret," *NeurIPS*, pp. 8833–8844, 2021.
[124] Z. Zhang, Q. Liu, Q. Hu, and C.-K. Lee, "Hierarchical graph transformer with adaptive node sampling," *NeurIPS*, vol. 35, pp. 21 171–21 183, 2022.
[125] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," *NeurIPS*, vol. 31, 2018.
[126] W. Cong, R. Forsati, M. Kandemir, and M. Mahdavi, "Minimal variance sampling with provable guarantees for fast training of graph neural networks," in *KDD*, 2020, pp. 1393–1403.
[127] X. Wei, X. Gong, Y. Zhan, B. Du, Y. Luo, and W. Hu, "Clnode: Curriculum learning for node classification," in *WSDM*, 2023.
[128] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *KDD*, 2020.
[129] W. Hu, K. Cao, K. Huang, E. W. Huang, K. Subbian, and J. Leskovec, "Tuneup: A training strategy for improving generalization of graph neural networks," *arXiv preprint arXiv:2210.14843*, 2022.
[130] Z. Tong, Y. Liang, H. Ding, Y. Dai, X. Li, and C. Wang, "Directed graph contrastive learning," *NeurIPS*, 2021.
[131] Y. Liu, X. Yang, S. Zhou, X. Liu, Z. Wang, K. Liang, W. Tu, L. Li, J. Duan, and C. Chen, "Hard sample aware network for contrastive deep graph clustering," in *AAAI*, 2023.
[132] X. Dong, C. Long, W. Xu, and C. Xiao, "Dual graph convolutional networks with transformer and curriculum learning for image captioning," in *MM*, 2021.
[133] X. Li, L. Wen, Y. Deng, F. Feng, X. Hu, L. Wang, and Z. Fan, "Graph neural network with curriculum learning for imbalanced node classification," *arXiv preprint arXiv:2202.02529*, 2022.
[134] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *KDD*, 2018.
[135] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Curgraph: Curriculum learning for graph classification," in *WWW*, 2021.
[136] G. Chu, X. Wang, C. Shi, and X. Jiang, "Cuco: Graph representation with curriculum contrastive learning." in *IJCAI*, 2021.
[137] C. Gong, J. Yang, and D. Tao, "Multi-modal curriculum learning over graphs," *TIST*, 2019.
[138] U. Ahmed, J. C.-W. Lin, and G. Srivastava, "Hyper-graph-based attention curriculum learning using a lexical algorithm for mental health," *PRL*, 2022.
[139] H. Wang, K. Zhou, X. Zhao, J. Wang, and J.-R. Wen, "Curriculum pre-training heterogeneous subgraph transformer for top-n recommendation," *TOIS*, 2023.
[140] L. Yang, Z. Chen, J. Gu, and Y. Guo, "Dual self-paced graph convolutional network: Towards reducing attribute distortions induced by topology." in *IJCAI*, 2019.
[141] M. Gong, H. Zhou, A. Qin, W. Liu, and Z. Zhao, "Self-paced co-training of graph neural networks for semi-supervised node classification," *IEEE TNNLS*, 2022.
[142] L. Li, K. Zhao, J. Gan, S. Cai, T. Liu, H. Mu, and R. Sun, "Robust adaptive semi-supervised classification method based on dynamic graph and self-paced learning," *IPMt*, vol. 58, no. 1, p. 102433, 2021.
[143] L. Chen, H. Xu, Z. Wang, C. Wang, and Y. Jiang, "Self-paced learning based graph convolutional neural network for mixed integer programming (student abstract)," in *AAAI*, vol. 37, no. 13, 2023, pp. 16 188–16 189.
[144] P. Zhou, X. Liu, L. Du, and X. Li, "Self-paced adaptive bipartite graph learning for consensus clustering," *TKDD*, 2023.
[145] P. Zhou, L. Du, and X. Li, "Self-paced consensus clustering with bipartite graph," in *CIKM*, 2021, pp. 2133–2139.
[146] C. Huang, B. Shi, X. Zhang, X. Wu, and N. V. Chawla, "Similarity-aware network embedding with self-paced learning," in *CIKM*, 2019, pp. 2113–2116.
[147] H. Gao and H. Huang, "Self-paced network embedding," in *KDD*, 2018, pp. 1406–1415.
[148] R. Li, T. Zhong, X. Jiang, G. Trajcevski, J. Wu, and F. Zhou, "Mining spatio-temporal relations via self-paced graph contrastive learning," in *KDD*, 2022, pp. 936–944.
[149] D. Zhou, J. He, H. Yang, and W. Fan, "Sparc: Self-paced network representation for few-shot rare category characterization," in *KDD*, 2018, pp. 2807–2816.
[150] J. Ma, W. Tang, J. Zhu, and Q. Mei, "A flexible generative framework for graph-based semi-supervised learning," *NeurIPS*, vol. 32, 2019.
[151] Y. Zhang, S. Pal, M. Coates, and D. Ustebay, "Bayesian graph convolutional neural networks for semi-supervised classification," in *AAAI*, vol. 33, no. 01, 2019, pp. 5829–5836.
[152] P. Elinas, E. V. Bonilla, and L. Tiao, "Variational inference for graph convolutional networks in the absence of graph data and adversarial settings," *NeurIPS*, vol. 33, pp. 18 648–18 660, 2020.
[153] S. Pal, S. Malekmohammadi, F. Regol, Y. Zhang, Y. Xu, and M. Coates, "Non parametric graph learning for bayesian graph neural networks," in *UAI*. PMLR, 2020, pp. 1318–1327.
[154] Q. Sun, J. Li, H. Peng, J. Wu, X. Fu, C. Ji, and S. Y. Philip, "Graph structure learning with variational information bottleneck," 2022.
[155] L. Franceschi, M. Niepert, M. Pontil, and X. He, "Learning discrete structures for graph neural networks," in *ICML*. PMLR, 2019, pp. 1972–1982.
[156] N. Yin and Z. Luo, "Generic structure extraction with bi-level optimization for graph structure learning," *Entropy*, 2022.
[157] A. Kazi, L. Cosmo, S.-A. Ahmadi, N. Navab, and M. Bronstein, "Differentiable graph module (dgm) for graph convolutional networks," *arXiv preprint arXiv:2002.04999*, 2020.
[158] Y. Chen, L. Wu, and M. Zaki, "Iterative deep graph learning for graph neural networks: Better and robust node embeddings," *NeurIPS*, 2020.
[159] J. Zhao, X. Wang, C. Shi, B. Hu, G. Song, and Y. Ye, "Heterogeneous graph structure learning for graph neural networks," in *AAAI*, 2021.
[160] Y. Chen, L. Wu, and M. J. Zaki, "Graphflow: Exploiting conversation flow with graph neural networks for conversational machine comprehension," *arXiv preprint arXiv:1908.00059*, 2019.
[161] ——, "Reinforcement learning based graph-to-sequence model for natural question generation," *arXiv preprint arXiv:1908.04942*, 2019.
[162] K.-W. On, E.-S. Kim, Y.-J. Heo, and B.-T. Zhang, "Cut-based graph learning networks to discover compositional structure of sequential video data," in *AAAI*, vol. 34, no. 04, 2020, pp. 5315–5322.
[163] Z. Yang, J. Zhao, B. Dhingra, K. He, W. W. Cohen, R. R. Salakhutdinov, and Y. LeCun, "Glomo: Unsupervised learning of transferable relational graphs," *NeurIPS*, vol. 31, 2018.
[164] D. Huang, P. Chen, R. Zeng, Q. Du, M. Tan, and C. Gan, "Location-aware graph convolutional networks for video question answering," in *AAAI*, vol. 34, no. 07, 2020, pp. 11 021–11 028.
[165] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *AAAI*, vol. 32, no. 1, 2018.
[166] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.
[167] P. Liu, S. Chang, X. Huang, J. Tang, and J. C. K. Cheung, "Contextualized non-local neural networks for sequence learning," in *AAAI*, vol. 33, no. 01, 2019, pp. 6762–6769.
[168] S. Liu, Y. Chen, X. Xie, J. Siow, and Y. Liu, "Retrieval-augmented generation for code summarization via hybrid gnn," *arXiv preprint arXiv:2006.05405*, 2020.
[169] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo, "Semi-supervised learning with graph learning-convolutional networks," in *IEEE/CVF*, 2019, pp. 11 313–11 320.
[170] L. Yang, Z. Kang, X. Cao, D. Jin, B. Yang, and Y. Guo, "Topology optimization based graph convolutional network." in *IJCAI*, 2019, pp. 4054–4061.
[171] X. Gao, W. Hu, and Z. Guo, "Exploring structure-adaptive graph learning for robust semi-supervised classification," in *ICME*, 2020.
[172] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
[173] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, 2016, pp. 1225–1234.
[174] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *ArXiv*, 2020.
[175] D. Lim, J. D. Robinson, L. Zhao, T. E. Smidt, S. Sra, H. Maron, and S. Jegelka, "Sign and basis invariant networks for spectral graph representation learning," in *ICLR*, 2023.

[176] D. Kreuzer, D. Beaini, W. L. Hamilton, V. Létourneau, and P. Tossou, "Rethinking graph transformers with spectral attention," in *NeurIPS*, 2021, pp. 21 618–21 629.
[177] R. Abboud, İ. İ. Ceylan, M. Grohe, and T. Lukasiewicz, "The surprising power of graph neural networks with random node initialization," in *IJCAI*, 2021, pp. 2112–2118.
[178] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Graph neural networks with learnable structural and positional representations," in *ICLR*. OpenReview.net, 2022.
[179] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *ICML*, vol. 97, 2019, pp. 7134–7143.
[180] P. Li, Y. Wang, H. Wang, and J. Leskovec, "Distance encoding: Design provably more powerful neural networks for graph representation learning," in *NeurIPS*, 2020.
[181] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T. Liu, "Do transformers really perform badly for graph representation?" in *NeurIPS*, 2021, pp. 28 877–28 888.
[182] H. Wang, H. Yin, M. Zhang, and P. Li, "Equivariant and stable positional encoding for more powerful graph neural networks," in *ICLR*, 2022.
[183] J. Ye and J. Liu, "Sparse methods for biomedical data," *ACM Sigkdd Explorations Newsletter*, vol. 14, no. 1, pp. 4–15, 2012.
[184] D. Ming, C. Ding, and F. Nie, "A probabilistic derivation of lasso and l12-norm feature selections," in *AAAI*, vol. 33, no. 01, 2019, pp. 4586–4593.
[185] B. Jiang, B. Wang, and B. Luo, "Sparse norm regularized attribute selection for graph neural networks," *Pattern Recognition*, vol. 137, p. 109265, 2023.
[186] Q. Huang, T. Xia, H. Sun, M. Yamada, and Y. Chang, "Unsupervised nonlinear feature selection from high-dimensional signed networks," in *AAAI*, vol. 34, no. 04, 2020, pp. 4182–4189.
[187] D. B. Acharya and H. Zhang, "Feature selection and extraction for graph neural networks," in *ACMSE*, 2020, pp. 252–255.
[188] J. Li, R. Guo, C. Liu, and H. Liu, "Adaptive Unsupervised Feature Selection on Attributed Networks," in *KDD*. Association for Computing Machinery, 2019, pp. 92–100.
[189] S. K. Maurya, X. Liu, and T. Murata, "Not All Neighbors are Friendly: Learning to Choose Hop Features to Improve Node Classification," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. ACM, 2022, pp. 4334–4338.
[190] Z. Lin, M. Luo, Z. Peng, J. Li, and Q. Zheng, "Nonlinear feature selection on attributed networks," *Neurocomputing*, vol. 410, pp. 161–173, 2020.
[191] H. Taguchi, X. Liu, and T. Murata, "Graph convolutional networks for graphs containing missing features," *Future Generation Computer Systems*, vol. 117, pp. 155–168, 2021.
[192] X. Chen, S. Chen, J. Yao, H. Zheng, Y. Zhang, and I. W. Tsang, "Learning on attribute-missing graphs," *TPAMI*, vol. 44, no. 2, pp. 740–757, 2020.
[193] D. Jin, R. Wang, T. Wang, D. He, W. Ding, Y. Huang, L. Wang, and W. Pedrycz, "Amer: A new attribute-missing network embedding approach," *IEEE Transactions on Cybernetics*, 2022.
[194] I. Spinelli, S. Scardapane, and A. Uncini, "Missing data imputation with adversarially-trained graph convolutional networks," *Neural Networks*, vol. 129, pp. 249–260, 2020.
[195] D. Jin, C. Huo, C. Liang, and L. Yang, "Heterogeneous graph neural network via attribute completion," in *WWW*, 2021, pp. 391–400.
[196] K. Wang, Y. Yu, C. Huang, Z. Zhao, and J. Dong, "Heterogeneous graph neural network for attribute completion," *Knowledge-Based Systems*, vol. 251, p. 109171, 2022.
[197] D. He, C. Liang, C. Huo, Z. Feng, D. Jin, L. Yang, and W. Zhang, "Analyzing heterogeneous networks with missing attributes by unsupervised contrastive learning," *TNNLS*, 2022.
[198] G. Zhu, Z. Zhu, W. Wang, Z. Xu, C. Yuan, and Y. Huang, "Autoac: Towards automated attribute completion for heterogeneous graph neural network," *arXiv preprint arXiv:2301.03049*, 2023.
[199] C. Li, Y. Yan, J. Fu, Z. Zhao, and Q. Zeng, "Hetregat-fc: Heterogeneous residual graph attention network via feature completion," *Information Sciences*, vol. 632, pp. 424–438, 2023.
[200] Y. Guo, C. Yang, Y. Chen, J. Liu, C. Shi, and J. Du, "A data-centric framework to endow graph neural networks with out-of-distribution detection ability," *KDD*, 2023.
[201] X. Sun, H. Cheng, J. Li, B. Liu, and J. Guan, "All in one: Multi-task prompting for graph neural networks," 2023.
[202] Q. Huang, H. Ren, P. Chen, G. Kržmanc, D. Zeng, P. Liang, and J. Leskovec, "Prodigy: Enabling in-context learning over graphs," *arXiv preprint arXiv:2305.12600*, 2023.
[203] Z. Liu, X. Yu, Y. Fang, and X. Zhang, "Graphprompt: Unifying pre-training and downstream tasks for graph neural networks," in *WWW*, 2023.
[204] M. Sun, K. Zhou, X. He, Y. Wang, and X. Wang, "Gppt: Graph pre-training and prompt tuning to generalize graph neural networks," in *KDD*, 2022, pp. 1717–1727.
[205] X. Han, Z. Jiang, N. Liu, and X. Hu, "G-mixup: Graph data augmentation for graph classification," in *ICML*. PMLR, 2022, pp. 8230–8248.
[206] E. M. Airoldi, T. B. Costa, and S. H. Chan, "Stochastic blockmodel approximation of a graphon: Theory and consistent estimation," *NeurIPS*, vol. 26, 2013.
[207] M. Navarro and S. Segarra, "Graphmad: Graph mixup for data augmentation using data-driven convex clustering," in *ICASSP*. IEEE, 2023, pp. 1–5.
[208] J. Park, H. Shim, and E. Yang, "Graph transplant: Node saliency-guided graph mixup with local structure preservation," in *AAAI*, vol. 36, no. 7, 2022, pp. 7966–7974.
[209] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Mixup for node and graph classification," in *WWW*, 2021, pp. 3663–3674.
[210] Z. Guo, C. Zhang, Y. Fan, Y. Tian, C. Zhang, and N. V. Chawla, "Boosting graph neural networks via adaptive knowledge distillation," in *AAAI*, vol. 37, no. 6, 2023, pp. 7793–7801.
[211] H. He, J. Wang, Z. Zhang, and F. Wu, "Compressing deep graph neural networks via adversarial knowledge distillation," in *KDD*, 2022, pp. 534–544.
[212] Y. Yang, J. Qiu, M. Song, D. Tao, and X. Wang, "Distilling knowledge from graph convolutional networks," in *CVPR*, 2020, pp. 7074–7083.
[213] Q. Tan, D. Zha, S.-H. Choi, L. Li, R. Chen, and X. Hu, "Double wins: Boosting accuracy and efficiency of graph neural networks by reliable knowledge distillation," 2022.
[214] C. Yang, J. Liu, and C. Shi, "Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework," in *WWW*, 2021, pp. 1227–1237.
[215] J. Guo, D. Chen, and C. Wang, "Alignahead: Online cross-layer knowledge extraction on graph neural networks," in *IJCNN*. IEEE, 2022, pp. 1–8.
[216] X. Deng and Z. Zhang, "Graph-free knowledge distillation for graph neural networks," *arXiv preprint arXiv:2105.07519*, 2021.
[217] C. Yang, Y. Guo, Y. Xu, C. Shi, J. Liu, C. Wang, X. Li, N. Guo, and H. Yin, "Learning to distill graph neural networks," in *WSDM*, 2023, pp. 123–131.
[218] C. Zhang, J. Liu, K. Dang, and W. Zhang, "Multi-scale distillation from multiple graph neural networks," in *AAAI*, vol. 36, no. 4, 2022, pp. 4337–4344.
[219] L. Wu, H. Lin, Y. Huang, and S. Z. Li, "Knowledge distillation improves graph structure augmentation for graph neural networks," *NurIPS*, vol. 35, pp. 11 815–11 827, 2022.
[220] Y. Dong, B. Zhang, Y. Yuan, N. Zou, Q. Wang, and J. Li, "Reliant: Fair knowledge distillation for graph neural networks," in *SDM*. SIAM, 2023, pp. 154–162.
[221] Y. He and Y. Ma, "Sgkd: A scalable and effective knowledge distillation framework for graph representation learning," in *ICDMW*. IEEE, 2022, pp. 666–673.
[222] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, vol. 32, no. 1, 2018.
[223] K. Zhan and C. Niu, "Mutual teaching for graph convolutional networks," *Future Generation Computer Systems*, vol. 115, pp. 837–843, 2021.
[224] K. Sun, Z. Lin, and Z. Zhu, "Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes," in *AAAI*, vol. 34, no. 04, 2020, pp. 5892–5899.
[225] Y. Li, J. Yin, and L. Chen, "Informative pseudo-labeling for graph neural networks with few labels," *Data Mining and Knowledge Discovery*, vol. 37, no. 1, pp. 228–254, 2023.
[226] H. Cai, V. W. Zheng, and K. C.-C. Chang, "Active learning for graph embedding," *arXiv preprint arXiv:1705.05085*, 2017.
[227] L. Gao, H. Yang, C. Zhou, J. Wu, S. Pan, and Y. Hu, "Active discriminative network representation learning," in *IJCAI*, 2018.
[228] X. Chen, G. Yu, J. Wang, C. Domeniconi, Z. Li, and X. Zhang, "Activehne: Active heterogeneous network embedding," *arXiv preprint arXiv:1905.05659*, 2019.
[229] X. Li, Y. Wu, V. Rakesh, Y. Lin, H. Yang, and F. Wang, "Smartquery: An active learning framework for graph neural networks through hybrid uncertainty reduction," in *CIKM*, 2022.
[230] W. Zhang, Y. Shen, Y. Li, L. Chen, Z. Yang, and B. Cui, "Alg: Fast and accurate active learning framework for graph convolutional networks," in *ICMD*, 2021.

[231] S. Hu, Z. Xiong, M. Qu, X. Yuan, M.-A. Côté, Z. Liu, and J. Tang, "Graph policy network for transferable active learning on graphs," *NeurIPS*, 2020.

[232] S. Hu, M. Qu, Z. Liu, and J. Tang, "Transfer active learning for graph neural networks," 2019.

[233] L. Cui, X. Tang, S. Katariya, N. Rao, P. Agrawal, K. Subbian, and D. Lee, "Allie: Active learning on large-scale imbalanced graphs," in *WWW*, 2022.

[234] Y. Zhang, H. Tong, Y. Xia, Y. Zhu, Y. Chi, and L. Ying, "Batch active learning with graph neural networks via multi-agent deep reinforcement learning," in *AAAI*, 2022.

[235] W. Zhang, Z. Yang, Y. Wang, Y. Shen, Y. Li, L. Wang, and B. Cui, "Grain: Improving data efficiency of graph neural networks via diversified influence maximization," *arXiv preprint arXiv:2108.00219*, 2021.

[236] T. Yang, M. Zhou, Y. Wang, Z. Lin, L. Pan, B. Cui, and Y. Tong, "Mitigating semantic confusion from hostile neighborhood for graph active learning," in *CIKM*, 2023.

[237] W. Zhang, Y. Wang, Z. You, M. Cao, P. Huang, J. Shan, Z. Yang, and B. Cui, "Information gain propagation: a new way to graph active learning with soft labels," *arXiv preprint arXiv:2203.01093*, 2022.

[238] J. Kang, Q. Zhou, and H. Tong, "Jurygcn: quantifying jackknife uncertainty on graph convolutional networks," in *KDD*, 2022.

[239] Y. Wu, Y. Xu, A. Singh, Y. Yang, and A. Dubrawski, "Active learning for graph neural networks via node feature propagation," *arXiv preprint arXiv:1910.07567*, 2019.

[240] J. Liu, Y. Wang, B. Hooi, R. Yang, and X. Xiao, "Lscale: Latent space clustering-based active learning for node classification," in *ECML-PKDD*, 2022.

[241] Z. Dai, V. Ioannidis, S. Adeshina, Z. Jost, C. Faloutsos, and G. Karypis, "Scattersample: Diversified label sampling for data efficient graph neural network learning," *arXiv preprint arXiv:2206.04255*, 2022.

[242] Y. Li, J. Yin, and L. Chen, "Seal: Semisupervised adversarial active learning on attributed graphs," *TNNLS*, 2020.

[243] K. Madhawa and T. Murata, "Active learning on graphs via meta learning," in *ICML Workshop*, 2020.

[244] X. Guo and L. Zhao, "A systematic survey on deep generative models for graph generation," *TPAMI*, 2022.

[245] Y. Zhu, Y. Du, Y. Wang, Y. Xu, J. Zhang, Q. Liu, and S. Wu, "A survey on deep graph generation: Methods and applications," in *LoG*, 2022.

[246] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE TNNLS*, vol. 32, no. 1, pp. 4–24, 2020.

[247] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, pp. 1–23, 2019.

[248] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *TPAMI*, 2021.

[249] H. Li, X. Wang, and W. Zhu, "Curriculum graph machine learning: A survey," *ArXiv*, vol. abs/2302.02926, 2023.

[250] X. Liu, M. Yan, L. Deng, G. Li, X. Ye, and D. Fan, "Sampling methods for efficient training of graph convolutional networks: A survey," *IEEE/CAA JAS*, pp. 205–234, 2021.

[251] X. Liu, M. Yan, L. Deng, G. Li, X. Ye, D. Fan, S. Pan, and Y. Xie, "Survey on graph neural network acceleration: An algorithmic perspective," *arXiv preprint arXiv:2202.04822*, 2022.

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

[253] Y. Tian, S. Pei, X. Zhang, C. Zhang, and N. Chawla, "Knowledge distillation on graphs: A survey," *ArXiv*, vol. abs/2302.00219, 2023.

[254] J. Liu, T. Zheng, G. Zhang, and Q. Hao, "Graph-based knowledge distillation: A survey and experimental evaluation," *ArXiv*, vol. abs/2302.14643, 2023.

[255] J. Xia, H. Lin, Y. Xu, C. Tan, L. Wu, S. Li, and S. Z. Li, "Gnn cleaner: Label cleaner for graph structured data," *IEEE TKDE*, 2023.

[256] Y. Li, J. Yin, and L. Chen, "Unified robust training for graph neural networks against label noise," in *PKDD*. Springer, 2021, pp. 528–540.

[257] J.-X. Zhong, N. Li, W. Kong, S. Liu, T. H. Li, and G. Li, "Graph convolutional label noise cleaner: Train a plug-and-play action classifier for anomaly detection," in *IEEE/CVF*, 2019, pp. 1237–1246.

[258] J. Xu, R. Huang, X. JIANG, Y. Cao, C. Yang, C. Wang, and Y. Yang, "Better with less: Data-active pre-training of graph neural networks," 2022.

[259] L. Wu, P. Cui, J. Pei, L. Zhao, and X. Guo, "Graph neural networks: foundation, frontiers and applications," in *KDD*, 2022, pp. 4840–4841.

[260] M. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," *NeurIPS*, vol. 23, 2010.

[261] R. Tibshirani, "Regression shrinkage and selection via the lasso," *JRSSB*, vol. 58, no. 1, pp. 267–288, 1996.

[262] W. Jin, T. Zhao, J. Ding, Y. Liu, J. Tang, and N. Shah, "Empowering graph representation learning with test-time graph transformation," *ArXiv*, vol. abs/2210.03561, 2022.

[263] T. Fang, Y. Zhang, Y. Yang, C. Wang, and L. Chen, "Universal prompt tuning for graph neural networks," *NeurIPS*, 2023.

[264] Y. Zhu, J. Guo, and S. Tang, "Sgl-pt: A strong graph learner with graph prompt tuning," *arXiv preprint arXiv:2302.12449*, 2023.

[265] L. Sun, Y. Dou, C. Yang, K. Zhang, J. Wang, S. Y. Philip, L. He, and B. Li, "Adversarial attack and defense on graph data: A survey," *IEEE TKDE*, 2022.

[266] D. Zügner, O. Borchert, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on graph neural networks: Perturbations and their patterns," *TKDD*, vol. 14, no. 5, pp. 1–31, 2020.

[267] J. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *ICML*, vol. 97, 2019, pp. 1310–1320.

[268] A. Bojchevski and S. Günnemann, "Certifiable robustness to graph perturbations," in *NeurIPS*, 2019, pp. 8317–8328.

[269] S. Tao, H. Shen, Q. Cao, L. Hou, and X. Cheng, "Adversarial immunization for certifiable robustness on graphs," in *WSDM*, 2021, pp. 698–706.

[270] C. Agarwal, H. Lakkaraju, and M. Zitnik, "Towards a unified framework for fair and stable graph representation learning," in *UAI*, ser. PMLR, vol. 161. AUAI Press, 2021, pp. 2114–2124.

[271] Z. Liu, Y. N. Li, N.-F. Chen, Q. Wang, B. Hooi, and B. He, "A survey of imbalanced learning on graphs: Problems, techniques, and future directions," *ArXiv*, vol. abs/2308.13821, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:261245374

[272] R. Wang, X. Wang, C. Shi, and L. Song, "Uncovering the structural fairness in graph contrastive learning," in *NeurIPS*, 2022.

[273] K. Kuang, P. Cui, S. Athey, R. Xiong, and B. Li, "Stable prediction across unknown environments," in *KDD*. ACM, 2018, pp. 1617–1626.

[274] Y. He, P. Cui, J. Ma, H. Zou, X. Wang, H. Yang, and P. S. Yu, "Learning stable graphs from multiple environments with selection bias," in *KDD*. ACM, 2020, pp. 2194–2202.

[275] S. Fan, X. Wang, C. Shi, K. Kuang, N. Liu, and B. Wang, "Debiased graph neural networks with agnostic label selection bias," *IEEE TNNLS*, 2022.

[276] M. Newman, "Mixing patterns in networks." *Physical review. E, Statistical, nonlinear, and soft matter physics*, 2002.

[277] D. Bo, X. Wang, C. Shi, and H. Shen, "Beyond low-frequency information in graph convolutional networks," in *AAAI*, 2021, pp. 3950–3957.

[278] H. Pei, B. Wei, K. C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," in *ICLR*, 2020.

[279] X. Wang, M. Zhu, D. Bo, P. Cui, C. Shi, and J. Pei, "Am-gcn: Adaptive multi-channel graph convolutional networks," in *KDD*, 2020.

[280] J. Liu, C. Yang, Z. Lu, J. Chen, Y. Li, M. Zhang, T. Bai, Y. Fang, L. Sun, P. S. Yu *et al.*, "Towards graph foundation models: A survey and beyond," *arXiv preprint arXiv:2310.11829*, 2023.

[281] X. Zheng, Y. Liu, Z. Bao, M. Fang, X. Hu, A. W.-C. Liew, and S. Pan, "Towards data-centric graph machine learning: Review and outlook," *arXiv preprint arXiv:2309.10979*, 2023.

[282] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.