Acceleration Algorithms in GNNs: A Survey

Lu Ma, Zeang Sheng, Xunkai Li, Xinyi Gao, Zhezheng Hao, Ling Yang, Xiaonan Nie, Jiawei Jiang, Wentao Zhang and Bin Cui, *Fellow, IEEE*

Abstract—Graph Neural Networks have demonstrated remarkable effectiveness in various graph-based tasks, but their inefficiency in training and inference poses significant challenges for scaling to real-world, large-scale applications. To address these challenges, a plethora of algorithms have been developed to accelerate GNN training and inference, garnering substantial interest from the research community. This paper presents a systematic review of these acceleration algorithms, categorizing them into three main topics: training acceleration, inference acceleration, and execution acceleration. For training acceleration, we discuss techniques like graph sampling and GNN simplification. In inference acceleration, we focus on knowledge distillation, GNN quantization, and GNN pruning. For execution acceleration, we explore GNN binarization and graph condensation. Additionally, we review several libraries related to GNN acceleration, including our Scalable Graph Learning library, and propose future research directions. A comprehensive summary is available in our GitHub repository: https://github.com/PKU-DAIR/SGL/blob/main/Awsome-GNN-Acceleration.md.

Index Terms—Graph Neural Networks, Scalability, Acceleration Algorithms, Graph Learning Library

1 Introduction

Graph-structured data are ubiquitous in the real world, representing objects and their relationships in varied domains, including, but are not limited to, social networks [1], [2], e-commerce networks [3], molecular graphs [4], [5], traffic networks [6], [7], and knowledge graphs [8], [9]. These graphs are characterized by complex structures containing rich underlying information and values [10]. Consequently, leveraging deep learning methods to analyze graph-structured data has garnered significant research attention in recent years.

Graph Neural Networks (GNNs) [11], [12], [13], [14], which excel in extracting graph structure information and revealing interactions and potential connections between nodes, currently represent the state-of-the-art in learning node, edge, and whole graph representations. GNNs have demonstrated remarkable success and effectiveness in various graph-based applications and tasks, including chemistry and biology [15], [16], [17], computer vision [18], [19], [20], natural language processing [21], [22], and recommendation systems [23], [24], [25].

Despite the widespread adoption of GNNs, their training and inference processes are known to be inefficient, posing significant challenges in scaling GNNs for real-world and large-scale graph applications [26], [27]. Generally, GNNs rely on the adjacency matrix of the graph and the node features. The adjacency matrix, which represents the connections between nodes, grows quadratically with the number of nodes, leading to substantial memory and computational demands. Similarly, the node features, which encapsulate the attributes of each node, add to the complexity and storage requirements. Meanwhile, the exponential growth in the size of graph data exacerbates these challenges. Previously, many graph-based tasks were often conducted on toy datasets, which is harmful to practical usages. Currently, large-scale graph datasets are thereby proposed in realistic applications [28], [29]. For instance, Facebook's social network graph contains over 2 billion nodes and 1 trillion edges. Training on such a massive-scale graph generates approximately 100 terabytes of data. This enormous volume of data necessitates extensive computational resources and sophisticated data management strategies. Furthermore, GNNs predominantly utilize the message passing (MP) paradigm [17], which involves recursive neighborhood aggregation. While this approach enhances performance by capturing local dependencies, it also results in repetitive neighborhood expansion, significantly increasing computational and I/O overhead [27]. Each iteration of MP involves aggregating information from an expanding set of

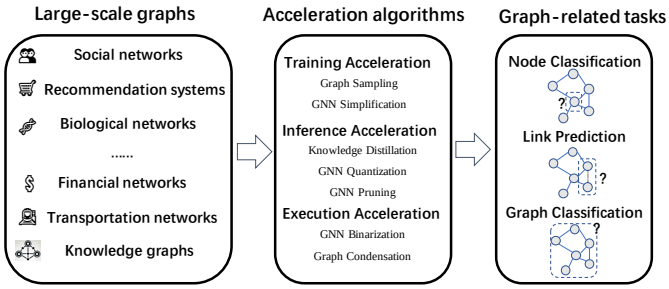


Fig. 1: An overview for acceleration algorithms in GNNs. Large-scale graphs, including social networks, recommendation systems, biological networks, financial networks, transportation networks, and knowledge graphs, necessitate the use of various acceleration algorithms. These algorithms enhance the efficiency of GNNs across three main dimensions: training acceleration, inference acceleration, and execution acceleration. Such acceleration enable the effective handling of diverse graph-related tasks.

Lu Ma, Zeang Sheng, Ling Yang, Xiaonan Nie and Bin Cui are with the School of CS, Peking University (e-mail: maluqaq@stu.pku.edu.cn; shengzeang18@pku.edu.cn; yangling0818@163.com; xiaonan.nie@pku.edu.cn; bin.cui@pku.edu.cn).
Xunkai Li is with the School of CS, Beijing Institute of Technology (e-mail: cs.xunkai.li@gmail.com).
Xinyi Gao is with the University of Queensland (e-mail: xinyi.gao@uq.edu.au).
Zhezheng Hao is with the School of AI, Northwestern Polytechnical University (e-mail: haozhezheng@outlook.com).
Jiawei Jiang is a professor with the School of Computer Science, Wuhan University, China. (e-mail: jiawei.jiang@whu.edu.cn).
Wentao Zhang is with the Center for machine learning research, Peking University (e-mail: wentao.zhang@pku.edu.cn).
Corresponding author: Wentao Zhang (e-mail: wentao.zhang@pku.edu.cn).

neighbors, leading to exponential growth in the number of nodes involved in computation. This not only amplifies time complexity but also exacerbates memory challenges, as storing intermediate states and activation outputs strains GPU memory. In addition, the irregular nature of graph data storage complicates efficient data access.

Acceleration Algorithms in GNNs, encompassing training acceleration and inference acceleration, emerge as a promising research direction to reduce the storage and computational demands of GNNs. These algorithms are pivotal for achieving scalable GNN implementations and have attracted considerable interest from the research and industry communities. Zhang et al. [30] conduct a comprehensive survey on GNN acceleration methods from the perspective of algorithms, systems and customized hardware. Chen et al. [31] dissucs about GNN acceleration from a hardware perspective. Liu et al. [32] review the acceleration algorithms for GNNs, focusing on graph-level and model-level optimizations. However, both of them have limitations in terms of comprehensive coverage and the absence of a taxonomy specifically focused on the purpose of algorithms. In contrast, we provide a thorough review of the literature on the advances in acceleration algorithms in GNNs and offer a systematic and comprehensive survey. The key contributions of this survey are summarized as follows:

• **New Taxonomy**. We propose a systematic classification schema to organize existing acceleration algorithms in GNNs, categorizing them into three primary topics: training acceleration, inference acceleration, and execution acceleration (i.e., both training and inference). Our taxonomy not only categorizes these techniques but also highlights their primary focus and contributions to the field of GNN acceleration. Notably, the execution acceleration category stands out due to its dual focus on optimizing both training and inference, offering a unique and comprehensive approach compared to the other categories. By organizing the acceleration algorithms into these distinct categories, we aim to provide a clear and comprehensive understanding of the current landscape, guiding researchers and practitioners in selecting and developing appropriate methods for their specific needs.

• **Comprehensive review**. For each acceleration topic, we focus on various techniques to accelerate GNNs and introduce the fundamental definition and representative algorithms for each technique. Specifically, for training acceleration topic, we focus on graph sampling [13], [33], [34] and GNN simplification [35], [36]. We focus on knowledge distillation on graphs [20], [37], GNN quantization [38], [39] and GNN pruning [40], [39] in inference acceleration topic. For execution acceleration topic, we focus on GNN Binarization [41] and graph condensation [42], [43].

• **Future research**. We discuss the limitations of current methods and propose six potential future directions for acceleration algorithms in GNNs, such as extending to other scenarios and integrating with cutting-edge applications. This discussion aims to guide researchers seeking scalable GNNs.

The remainder of this article is organized as follows: Section 2 introduces the preliminaries of GNNs and the challenges of scalable GNNs. Sections 3 to 5 summarize and discuss training acceleration, inference acceleration, and execution acceleration algorithms in GNNs. Section 6 reviews libraries related to acceleration in GNNs and discusses our **Scalable Graph Learning** (SGL) library. Finally, we highlight six future directions and conclude the survey in Section 7. By disseminating this survey, we hope to foster further research and practical applications of acceleration in GNNs.

2 PRELIMINARY

In this section, we first provide a brief introduction to the fundamental formulations and paradigms of GNNs, specifically the MP paradigm. Subsequently, we discuss two key challenges associated with acceleration algorithms in GNNs: time complexity and memory complexity, before moving on to the next section.

2.1 Graph Neural Networks

Consider a graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$. Node features are represented as $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times f}$, where n is the number of nodes and f is the dimension of the node features. The neighborhood set and the degree of node v are denoted by $\mathcal{N}(v)$ and $d(v)$. The adjacency matrix is defined as $\mathbf{A} \in \{0,1\}^{n \times n}$. $\mathbf{A}[u,v] = 1$ if $(u,v) \in \mathcal{E}$, otherwise, $\mathbf{A}[u,v] = 0$. We use bold uppercase letters (e.g., $\mathbf{X}$) and bold lowercase letters (e.g., $\mathbf{x}$) to represent matrices and vectors, respectively.

Most GNNs can be formulated using the MP paradigm (e.g., GCN [11], GraphSAGE [13], GAT [12], etc.). In this paradigm, each layer of GNNs utilizes an aggregation function and an update function. The l^{th} MP layer of GNNs is formulated as follows:

$$\begin{aligned} \mathbf{m}_v^{(l)} &\leftarrow \text{aggre gate}\left(\mathbf{h}_v^{(l-1)}, \left\{\mathbf{h}_u^{(l-1)} \mid u \in \mathcal{N}(v)\right\}\right), \\ \mathbf{h}_v^{(l)} &\leftarrow \text{update}\left(\mathbf{m}_v^{(l)}, \mathbf{h}_v^{(l-1)}\right), \end{aligned} \tag{1}$$

where we define $\mathbf{h}_v^{(l)}$ as the representation of node v in the l^{th} MP layer. Additionally, $\mathbf{m}_v^{(l)}$ denotes the message associated with node v. The aggregation function is represented by $\mathrm{aggregate}$, while the update function is denoted by update. Specifically, a message vector $\mathbf{m}_v^{(l)}$ is computed using the representations of neighboring nodes $\mathcal{N}(v)$ through the aggregation function $\mathrm{aggregate}$, and subsequently updated by the update function update. Node representation typically begins with node features, denoted as $\mathbf{H}^{(0)} = \mathbf{X}$, and the final representation is obtained after L MP layers as $\mathbf{H} = \mathbf{H}^{(L)}$. The matrix version of the l^{th} MP layer is as follows:

$$\mathbf{H}^{(l)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}\right), \tag{2}$$

where $\hat{\mathbf{A}}$ is the graph propagation matrix, which is usually a function of $\mathbf{A}$ and different for various GNNs.

The Graph Convolutional Network (GCN) [11], one of the earliest GNNs, operates by performing a linear approximation to spectral graph convolutions. The l^{th} MP layer is formulated using the MP paradigm as follows:

$$\begin{aligned} \mathbf{m}_v^{(l)} &= \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(l-1)} / \sqrt{\tilde{d}_v \tilde{d}_u}, \\ \mathbf{h}_v^{(l)} &= \sigma\left(\mathbf{W}^{(l)} \mathbf{m}_v^{(l)}\right), \end{aligned} \tag{3}$$

where we define $\tilde{d}(v)$ as the degree of node v obtained from the adjacency matrix with self-connections, denoted by $\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{A}$. $\mathbf{W}^{(l)}$ represents the learnable weights. In the matrix version, the graph propagation matrix of GCN is a degree-normalized version of the adjacency matrix with self-loops, denoted as $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$, where $\tilde{\mathbf{D}}$ is the diagonal degree matrix of $\tilde{\mathbf{A}}$.

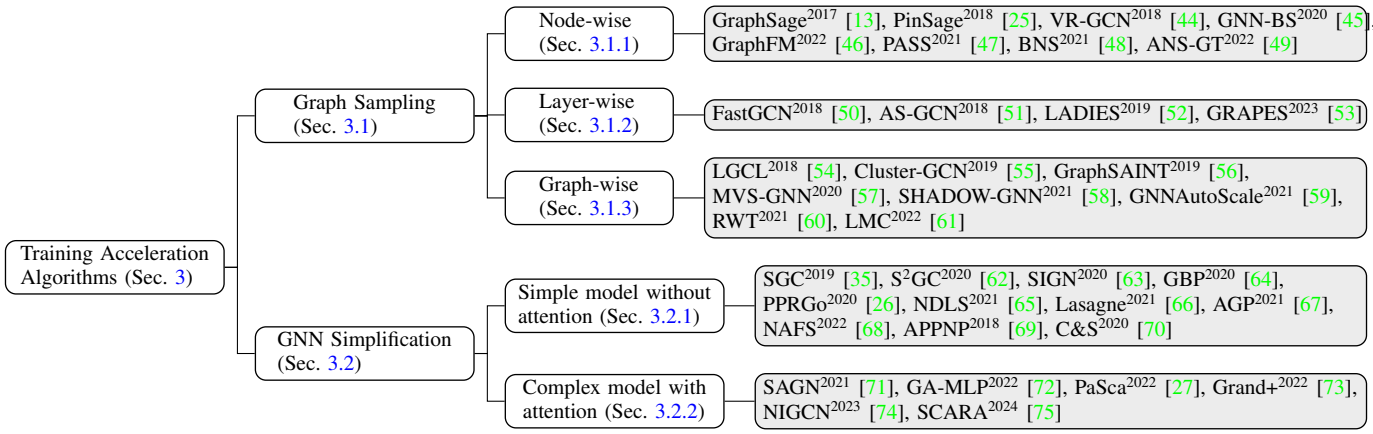


Fig. 2: The taxonomy of training acceleration algorithms in GNNs. The number in the top right corner represents the publication year.

2.2 Challenges of scalable GNNs

GNNs are notorious for inefficient training and inference, thus scalable GNNs (i.e. GNNs designed to handle large-scale graphs) face twofold challenges: time complexity and memory complexity.

2.2.1 Time Complexity Challenges

Scalable GNNs frequently encounter challenges related to time complexity during both training and inference phases. Unlike independent samples, nodes in a graph cannot be treated as such. Consequently, GNNs necessitate the resource-intensive and time-consuming MP for both training and inference. Computing the hidden representation for a specific node involves aggregating information from its neighbors. Consequently, The neighbors, in turn, must consider their own neighbors, leading to exponential expansion with each MP layer. This growth results in substantial computational costs. Additionally, due to graph irregularity, data is often stored non-contiguously in memory, causing high I/O overhead during aggregation operations. The processes of information aggregation, transformation, and node representation updates pose significant time complexity challenges.

2.2.2 Memory Complexity Challenges

Another significant challenge for scalable GNNs pertains to memory complexity. Training full-batch GNNs, such as GCN, requires storing the entire graph and node features in GPU memory, leading to substantial memory overflow issues, especially with graphs containing millions of nodes. Even batch-training methods demand considerable memory resources to process large-scale graphs. Efficiently storing and accessing graph data presents further challenges. Additionally, storing the activation output of each layer is essential for computing the gradient during the backward pass. As the number of GNN layers increases, the activation footprints progressively occupy more GPU memory during training.

Approaches with acceleration in training, inference, and execution, reviewed in later sections, aim to address the two aspects of challenges.

3 Training Acceleration

In this section, we discuss two types of training acceleration methods: graph sampling and GNN simplification. For each category, we elucidate the fundamental definitions and properties, followed by examples of typical work within these categories. Table1 presents memory usage and hardware throughput for GNN training acceleration methods [36], and Table 2 presents time complexity and memory complexity for GNN training acceleration methods.

3.1 Graph Sampling

Graph sampling [33], [34], [13], a common yet sophisticated technique, is employed to accelerate the training process of GNNs. This technique utilizes mini-batch training, leveraging sampled sub-graphs to approximate node representations. It addresses the neighbor explosion issue, significantly reducing memory and computation consumption during training. For completeness, we reiterate the unified formulation of graph sampling methods as follows:

$$\mathbf{m}_v^{(l)} \leftarrow \text{aggregate}\left(\mathbf{h}_v^{(l-1)}, \left\{\mathbf{h}_u^{(l-1)} \mid (u,v) \in Block^{(l)}\right\}\right),$$
$$\mathbf{h}_v^{(l)} \leftarrow \text{update}\left(\mathbf{m}_v^{(l)}, \mathbf{h}_v^{(l-1)}\right), \tag{4}$$

where we define $Block^{(l)}$ as the bipartite graph between the $(l-1)^{th}$ MP layer and the l^{th} MP layer: $Block^{(l)} = \{(u,v) \mid u \in n^{(l-1)}, v \in n^{(l)}, (u,v) \in \mathcal{E}\}$; and $n^{(l)}$ represents the set of sampled nodes at the l^{th} MP layer. Notably, during mini-batch training, the nodes in the final MP layer remain fixed as the batched nodes for each training iteration.

When considering a target node v at the $(l+1)^{th}$ MP layer, graph sampling methods deviate from the traditional approach of aggregating embeddings from the full neighborhood $\mathcal{N}(v)$ at the l^{th} MP layer. Instead, they selectively choose a subset of neighboring nodes, denoted as $\mathcal{N}(v) \cap n^{(l)}$, which contains the most crucial information for prediction. The fundamental distinction among graph sampling methods lies in how they sample the node sets $\{n^{(0)}, \ldots, n^{(L-1)}, n^{(L)}\}$. These methods can be categorized based on the scale at which they sample $n^{(l)}$, resulting in node-wise sampling, layer-wise sampling, and graph-wise sampling methods, as illustrated in Figure 3.

3.1.1 Node-wise Sampling

Node-wise sampling methods focus on sampling a fixed-size set of neighbors for each node in each MP layer, mitigating the issue of neighbor explosion and reduces the memory complexity from $\mathcal{O}(d^L)$ to $\mathcal{O}(r^L)$, where d represents the average node degree.

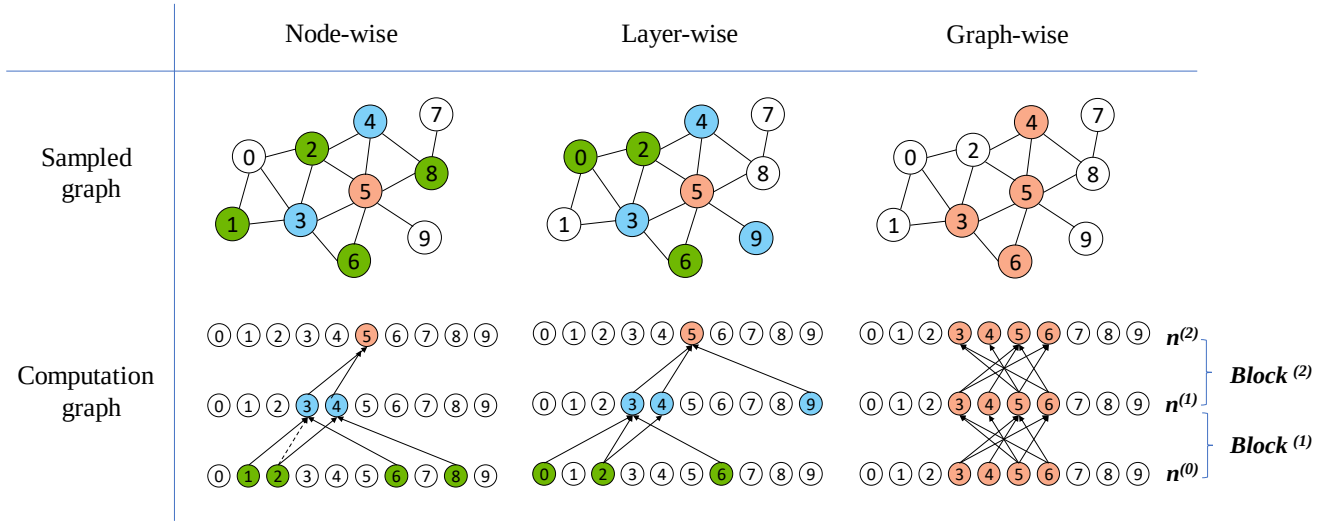


Fig. 3: Illustration of different graph sampling methods: red nodes represent selected nodes in the current batch as $n^{(L)}$, blue nodes are sampled in the first layer as $n^{(1)}$, and green nodes are sampled in the second layer as $n^{(0)}$. $n^{(0)}$ and $n^{(1)}$ form $Block^{(1)}$, while $n^{(1)}$ and $n^{(2)}$ form $Block^{(2)}$. The node-wise sampling method samples two nodes for each node (e.g., sampling v_1 and v_6 for v_3 in layer 1). The layer-wise sampling method samples three nodes for each GNN layer (e.g., sampling v_0, v_2 and v_6 inn layer 0). The graph-wise sampling method samples the same sub-graph (i.e. sampling v_3, v_4, v_5 and v_6) for all layers.

Diving into node-sampling process, sampled nodes at the l^{th} MP layer (excluding L^{th} MP layer) can be formulated as:

$$n^{(l)} = \bigcup_{v \in n^{(l+1)}} \{u \mid u \sim r \cdot \mathbb{P}_{\mathcal{N}(v)}\}, \tag{5}$$

where $n^{(L)}$ are the selected nodes in current batch; $r \cdot \mathbb{P}$ represents sampling r nodes from a sampling distribution $\mathbb{P}$; $\mathcal{N}(v)$ denotes the neighborhood of node v as the sampling space; and parameter r corresponds to the number of samples. Specifically, considering the set of sampled nodes $n^{(l)}$ at the l^{th} MP layer, node-wise sampling methods select r neighbors for each node v at the $(l+1)^{th}$ MP layer, expressed as $\{u \mid u \sim r \cdot \mathbb{P}_{\mathcal{N}(v)}, v \in n^{(l+1)}\}$.

GraphSAGE [13] pioneered graph sampling, which focuses on sampled sub-graphs instead of entire graphs. In GraphSAGE, a fixed-size set of target nodes forms a mini-batch, and a tree rooted at each target node is randomly sampled by recursively expanding the neighborhood around the root node. For each l^{th} layer, GraphSAGE samples a fixed number of neighbor nodes $r^{(l)}$ for each node. Notably, each neighbor node u in $\mathcal{N}(v)$ has an equal probability of $\frac{1}{\mathcal{N}(v)}$ to be sampled. During inference, for each sampled tree (which also serves as the sampled graph), GraphSAGE computes the hidden representation of the target node by aggregating node representations from bottom to top. GraphSAGE shows good performance not only on tiny datasets but also on their constructed industrial Reddit dataset. In addition, GraphSAGE naturally generalizes to unseen nodes and graphs, enabling inductive learning on graphs. Furthermore, **PinSAGE** [25] extends GraphSAGE by incorporating importance scores for neighbors through random walks as sampling probabilities. This approach mitigates information loss caused by unweighted aggregation and scales GNNs for industrial web-scale recommender systems.

Based on the GraphSAGE workflow, **VR-GCN** [44] not only employs the same sampling operation and distribution as GraphSAGE but also proposes utilizing historical representations from the previous epoch to mitigate variance from neighbor sampling and accelerate training convergence. Specifically, VR-GCN maintains the node representations from the previous epoch for each node v at each l^{th} MP layer as an affordable approximation, denoted as $\overline{\mathbf{h}}_v^{(l)}$. Whenever $\mathbf{h}_v^{(l)}$ is computed, VR-GCN updates $\overline{\mathbf{h}}_v^{(l)}$ with $\mathbf{h}_v^{(l)}$. Assuming that GNN weights change gradually during training and that $\mathbf{h}_v^{(l)}$ is similar to $\overline{\mathbf{h}}_v^{(l)}$, we can estimate $\mathbf{h}_v^{(l)}$ using the update function: $\text{update}\left(\mathbf{m}_v^{(l)}, \mathbf{h}_v^{(l-1)}, \overline{\mathbf{h}}_v^{(l)}\right)$. However, it's important to note that VR-GCN requires storing the previous representations of all nodes, leading to increased time and space complexity. Following the concept of VR-GCN, many advanced algorithms utilize historical embeddings as approximate embeddings to enhance performance and reduce the number of sampled neighbors [57], [59], [46].

BNS [48] adopts a policy to stochastically block the ongoing expansion of neighboring nodes, effectively reducing the exponential increase in computation and memory complexity of GNNs. Meanwhile, **GNN-BS** [45] formulates neighbor sampling as a bandit problem and proposes a learnable sampler aimed at minimizing variance. Additionally, **PASS** [47] introduces a performance-adaptive sampler that selectively samples neighbors informative for a target task, outperforming many existing samplers.

Overall, node-wise sampling methods sample a fixed-size set of neighbors for each node in each MP layer, addressing the neighbor explosion issue, reducing the memory complexity from $\mathcal{O}(d^L)$ to $\mathcal{O}(r^L)$, and enabling efficient training of GNNs on large-scale graphs. However, the neighbor explosion issue has not been solved completely as the sampled nodes exponentially with MP layers, and the estimation variance of embedding approximation in the sampling process inevitably leads to some loss of graph information, leading to a decline in performance.

3.1.2 *Layer-wise sampling*

To address the neighbor explosion issue in node-wise sampling methods, layer-wise sampling methods focus on sampling a fixed-size set of nodes for each MP layer. This reduction in sampling size helps manage memory complexity, reducing it from $\mathcal{O}(r^L)$ to $\mathcal{O}(rL)$, thereby enabling efficient training of GNNs on large-scale

graphs. Diving into the layer-sampling process, sampled nodes at the l^{th} MP layer (excluding L^{th} MP layer) can be formulated as:

$$n^{(l)} = \{u \mid u \sim r \cdot \mathbb{P}_{\mathcal{V}}\}. \tag{6}$$

Specifically, when considering a target node v at the $(l+1)^{th}$ MP layer, layer-wise graph sampling methods aggregate information from the embeddings of $\mathcal{N}(v) \cap n^{(l)}$ at the l^{th} MP layer to perform message passing.

FastGCN [50], as an inaugural layer-wise sampling method, interprets graph convolutions as integral transforms of embedding functions. It samples a fixed number of nodes in each MP layer using a node-degree-based probability distribution: $p(u) \propto d(u)$. FastGCN conducts independent sampling operations at each MP layer while using the same sampling distribution. Despite its efficiency, FastGCN does not guarantee connectivity between sampled nodes across different MP layers due to independent sampling. Consequently, this approach may result in sparse computational graphs, causing significant variance in the approximate embeddings.

To address this issue, **LADIES** [52] adopts a similar importance sampling strategy as FastGCN but confines the sampling domain to the neighborhood of the sampled nodes. Given the set of sampled nodes $n^{(l+1)}$ at the $(l+1)^{th}$ MP layer, LADIES selects $r^{(l)}$ nodes for $n^{(l)}$, which can be denoted as $\{u \mid u \sim r^{(l)} \cdot \mathbb{P}_{\bigcup_{v \in n^{(l+1)}} \mathcal{N}(v)}\}$. This sampler guarantees that each sampled node will have at least one connected node in the previous MP layer. As a result, LADIES achieves substantially reduced variance compared to FastGCN and demonstrates high accuracy even with a small sample size, making it suitable for training very deep and large GNNs.

Moreover, **AS-GCN** [51] enhances LADIES by introducing an adaptive and applicable sampler for explicit variance reduction during the training phase, ensuring higher accuracy. Additionally, **GRAPES** [53] proposes an adaptive layer-wise sampler that learns the importance of nodes by adapting to the classification loss. This approach constructs a sampled sub-graph of influential nodes for improved performance.

While layer-wise sampling methods effectively control neighbor expansion, ensuring that the total number of sampled nodes grows linearly with the network depth, reducing the memory complexity from $\mathcal{O}(r^L)$ to $\mathcal{O}(rL)$. However, due to layer-wise sampling, some nodes may not have a sufficient number of neighboring nodes, leading to sparsity in the computational graph. When the parameter r is insufficient, this sparsity can result in a decline in model accuracy and instability in outcomes.

3.1.3 Graph-wise sampling

Graph-wise sampling methods concentrate on sampling the same sub-graph for each MP layer based on a specific sampling strategy $\mathbb{P}_{\mathcal{G}}$. The resulting sampled sub-graph is significantly smaller, allowing full-batch MP without memory overflow concerns. Diving into the graph-sampling process, sampled nodes within a batch can be formulated as:

$$n^{(0)} = \cdots = n^{(L-1)} = n^{(L)} = \{u \mid u \sim r \cdot \mathbb{P}_{\mathcal{V}}\}. \tag{7}$$

In summary, graph-wise sampling allows us to substitute MP on sub-graphs for inefficient full MP on large-scale graphs.

ClusterGCN [55] introduces mini-batch training by leveraging the graph clustering algorithm. Cluster-GCN operates as follows: the entire graph is initially partitioned into smaller non-overlapping clusters using clustering algorithms such as METIS [76]. During each training iteration, certain clusters are selected as a sub-graph for further full-batch training, and inter-cluster edges are reintroduced within the sub-graph. This straightforward yet effective approach significantly improves memory and computational efficiency, making it suitable for handling large-scale graphs.

GraphSAINT [56] is a widely used graph-wise sampling method, that first samples the training graph and then constructs a full-batch GNN on the sampled sub-graph. To enhance training quality, GraphSAINT analyzes the bias and variance of mini-batches defined on sampled sub-graphs. It proposes normalization techniques to eliminate bias and sampling algorithms for variance reduction. GraphSAINT provides a flexible and extensible framework, including sub-graph sampling strategies and GNN architectures, achieving high accuracy and rapid convergence.

Additionally, **SHADOW-GNN** [58] proposes extracting sub-graphs for each target node and applying GNNs to these sub-graphs to address scalability challenges. **GNNAutoScale** [59] leverages historical embeddings to generate messages outside the sampled sub-graph, thereby maintaining the expressiveness of the original GNNs. Furthermore, **LMC** [61] introduces the first graph-wise sampling method with provable convergence. This method employs efficient compensations to correct biases in mini-batch gradients, thereby accelerating convergence. Empirical evidence demonstrates that LMC achieves high accuracy while maintaining efficiency.

Overall, graph-wise sampling methods are applicable to a broad range of GNNs by directly running them on sampled sub-graphs. Meanwhile, these graph-wise samplers can operate entirely in parallel or even prior to the training process. However, sampling the entire graph can cause the GNN to focus only on the nodes within sub-graphs, thereby neglecting global information. Additionally, as the sampling process solely focuses on the original graph, the partitioning of the graph could have a significant impact on the stability and efficiency of training.

3.2 GNN Simplification

GNN simplification [35], [27] is a recent direction to accelerate GNN training by decoupling the standard MP paradigm. The primary concept behind GNN simplification methods is to either precompute the propagated features during preprocessing stages or propagate predictions during postprocessing stages, thereby effectively separating the aggregation operation from the update operation [36]. These methods offer two primary advantages. First, they can directly utilize sparse matrix multiplication, performed only once on CPUs, to obtain propagated features or predictions during preprocessing without training on GPUs. This avoids the inefficiencies of the MP process, significantly reducing time complexity. Second, because node dependencies are fully addressed in the pre- or postprocessing stage, it becomes feasible to divide the training nodes into smaller mini-batches. This enables batch training, significantly reducing memory consumption during training. However, GNN simplification models lack sufficient expressiveness due to the absence of a trainable aggregation process, limiting their applicability. To reflect the development of these methods, we categorize them into two types: simple methods without attention techniques and complex methods with attention techniques.

TABLE 1: The memory usage of activations and the hardware throughput (higher is better) for some GNN training acceleration methods. The hardware here is an RTX 3090 GPU.

	Flickr		Reddit		ogbn-products	
	Act Mem. (MB)	Throughput (iteration/s)	Act Mem. (MB)	Throughput (iteration/s)	Act Mem. (MB)	Throughput (iteration/s)
GraphSAGE	230.63	65.96	687.21	27.62	415.94	37.69
FastGCN	19.77	226.93	22.53	87.94	11.54	93.05
ClusterGCN	18.45	171.46	20.84	79.91	10.62	156.01
GraphSAINT	16.51	151.77	21.25	70.68	10.95	143.51
SGC	0.01	115.02	0.02	89.91	0.01	267.31
SIGN	16.99	96.20	16.38	75.33	16.21	208.52
SAGN	72.94	55.28	72.37	43.45	71.81	80.04

TABLE 2: Summary of time complexity and memory complexity for GNN training acceleration methods: n, m, c, and f are the number of nodes, edges, classes, and feature dimensions, respectively. b is the batch size, and r refers to the number of sampled nodes. L and K corresponds to the number of times we aggregate features and labels, respectively. Besides, P and Q are the number of layers in MLP classifiers trained with features and labels, respectively. For GraphSAINT, $d = \frac{m}{n}$ is the average degree of $\mathcal{G}$ and the number of batches is $\frac{n}{b}$. For GBP, ϵ denote the error threshold. For simplicity we omit the memory for storing the graph or sub-graphs since they are fixed and usually not the main bottleneck.

Type	Model	Pre-processing Time	Training Time	Training Memory
	GCN	-	$\mathcal{O}\left(Lmf + Lnf^2\right)$	$\mathcal{O}\left(Lnf + Lf^2\right)$
Graph sampling	GraphSAGE	-	$\mathcal{O}\left(k^L nf^2\right)$	$\mathcal{O}\left(br^L f + Lf^2\right)$
	VR-GCN	-	$\mathcal{O}\left(Lmf + Lnf^2 + r^L nf^2\right)$	$\mathcal{O}\left(Lnf + Lf^2\right)$
	FastGCN	-	$\mathcal{O}\left(kLnf^2\right)$	$\mathcal{O}\left(bkLf + Lf^2\right)$
	Cluster-GCN	$\mathcal{O}(m)$	$\mathcal{O}\left(Lmf + Lnf^2\right)$	$\mathcal{O}\left(bLf + Lf^2\right)$
	GraphSAINT	-	$\mathcal{O}\left(Lbdf + Lnf^2\right)$	$\mathcal{O}\left(bLf + Lf^2\right)$
GNN simplification without attention	SGC	$\mathcal{O}(Lmf)$	$\mathcal{O}\left(nf^2\right)$	$\mathcal{O}\left(bf + f^2\right)$
	S^2GC	$\mathcal{O}(Lmf)$	$\mathcal{O}\left(nf^2\right)$	$\mathcal{O}\left(bf + f^2\right)$
	SIGN	$\mathcal{O}(Lmf)$	$\mathcal{O}\left(LPnf^2\right)$	$\mathcal{O}\left(bLf + Pf^2\right)$
	GBP	$\mathcal{O}\left(Lnf + L\frac{\sqrt{m \lg n}}{\varepsilon}\right)$	$\mathcal{O}\left(Pnf^2\right)$	$\mathcal{O}\left(bf + Pf^2\right)$
	APPNP	-	$\mathcal{O}\left(Lmf + Pnf^2\right)$	$\mathcal{O}\left(nf + Pf^2\right)$
	C&S	$\mathcal{O}\left(Kmc\right)$	$\mathcal{O}\left(Pnf^2\right)$	$\mathcal{O}\left(bf + Pf^2\right)$
GNN simplification with attention	SAGN	$\mathcal{O}(Lmf)$	$\mathcal{O}\left(LPnf^2\right)$	$\mathcal{O}\left(bLf + Pf^2\right)$
	GAMLP	$\mathcal{O}(Lmf + Kmc)$	$\mathcal{O}\left(Pnf^2 + Qnc^2\right)$	$\mathcal{O}\left(bf + Pf^2 + Qc^2\right)$

3.2.1 Simple Model without Attention

SGC [35] is a pioneering work in GNN simplification, demonstrating that empirically removing intermediate nonlinearities from GCNs does not adversely affect accuracy in many downstream applications (e.g., node classification tasks). Consequently, by simplifying the GCN, the formulation of SGC can be denoted as follows:

$$\begin{aligned} Y &= \operatorname{softmax}\left(\hat{\mathbf{A}} \cdots \hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(1)} \cdots \mathbf{W}^{(L)}\right) \\ &= \operatorname{softmax}\left(\hat{\mathbf{A}}^L \mathbf{X}\mathbf{W}\right), \end{aligned} \tag{8}$$

where the propagated features $\hat{\mathbf{A}}^L$ can be precomputed on CPUs, resulting in a significant reduction in training time, while maintaining comparable accuracy to standard GNNs. For instance, on the Reddit dataset, SGC can be trained up to two orders of magnitude faster than graph sampling methods. In addition to its effectiveness, SGC serves as a starting point for developing more complex GNN simplification methods.

Following the design principle of SGC, Frasca et al. [63] propose **SIGN** inspired by the inception module [77], which can be formulated as:

$$\begin{aligned} \mathbf{X}^{'} &= [\mathbf{X}\mathbf{W}^{(0)}, \mathbf{A}^{(1)}\mathbf{X}\mathbf{W}^{(1)}, \ldots, \mathbf{A}^{(L)}\mathbf{X}\mathbf{W}^{(L)}], \\ \mathbf{Y} &= \operatorname{softmax}\left(\mathbf{X}^{'}\mathbf{W}\right), \end{aligned} \tag{9}$$

where $\mathbf{X}^{'}$ represents the intermediate node embeddings that incorporate information about neighbors from multiple hops, $\mathbf{W}, \mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}$ are learnable weights, and $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(L)}$ are various graph propagation matrices corresponding to different iteration counts (i.e. $\mathbf{A}^{(i)} = \hat{\mathbf{A}}^i$). By employing diverse graph propagation matrices, SIGN can obtain multi-scale propagated node features capturing various connectivity patterns. As a result, SIGN outperforms SGC. Zhu et al. [62] introduce **$\mathbf{S^2}$GC**, which balances low- and high-pass filter bands to capture both global and local contexts of each node. Meanwhile, **GBP** [64] further refines the combination process through weighted averaging. Additionally, GBP proposes an approximation algorithm to compute $\mathbf{X}^{'}$, achieving sub-linear time complexity for both precomputation and training phases.

To address the variation in optimal propagation iterations across different nodes, **NDLS** [65] provides a bound to guide the control of smoothness for individual nodes. By calculating a node-specific smoothing iteration based on this bound, each node in NDLS undergoes personalized aggregation iterations, thereby enhancing model performance. Meanwhile, **NAFS** [68] focuses on improving feature smoothing operations and generating node embeddings in a training-free manner. NAFS proposes node-adaptive feature smoothing, which generates smoothed features tailored to each node's individual properties. It also employs different ensemble strategies to combine multiple smoothed features from diverse knowledge extractors. Additionally, the concept of approximate

graph propagation (**AGP**) has been proposed by Wang et al. [67], unifying various proximity measures. Wang et al. also introduce a unified randomized graph propagation algorithm that efficiently computes proximity queries and graph feature propagation with nearly optimal computation time and a theoretical error guarantee.

In addition to the aforementioned precomputation methods, several label propagation methods can also accelerate GNN training. **PPNP** [69] trains on raw features using a multi-layer perceptron (MLP), and then propagates predictions using a graph propagation matrix during training process. Directly calculating the personalized PageRank matrix is inefficient and a fast approximation of PPNP, called **APPNP**, is introduced. Although the propagation operations of APPNP do not require any additional parameters for training, APPNP focuses on integrating label propagation into the training process and executes propagation operations in each training epoch, making it only marginally faster than GCN and challenging to perform on large-scale graphs. To address these issues, **C&S** [70] utilizes label propagation as two simple post-processing steps: correcting base predictions with error correlation and smoothing final predictions with prediction correlation. These post-processing steps leverage the fact that errors and labels on connected nodes tend to be positively correlated. Compared to APPNP, C&S is faster to train and easily scales to large-scale graphs.

3.2.2 Complex Model with Attention

To enhance the performance of the aforementioned simple methods, SAGN [71] replaces the redundant concatenation operation in SIGN with attention techniques, as denoted by:

$$\begin{aligned} \mathbf{H}^{(l)} &= \mathrm{MLP}(\hat{\mathbf{A}}^l\mathbf{X}), \\ \mathbf{X}^{'} &= \sum_{l=0}^{L} \Theta^{(l)}\mathbf{H}^{(l)}, \end{aligned} \tag{10}$$

where $\Theta^{(l)}$ is the diagonal attention matrix. The i-th entry $\Theta_i^{(l)}$ of $\Theta^{(l)}$ is calculated as follows:

$$\Theta_i^{(l)} = \mathrm{softmax}(\mathrm{LeakyReLU}([\mathbf{H}_i^{(0)}, \mathbf{H}_i^{(l)}]) \cdot a), \tag{11}$$

where a is the learnable attention vector and $\cdot$ represents the dot product. This attention technique effectively collect neighbor information from various iterations in an adaptive manner, leading to improved prediction performance. Additionally, **GAMLP** [72] introduces two node-adaptive attention mechanisms: recursive attention and JK attention [78]. These mechanisms enable each node to combine propagated features across different iterations. As a result, GAMLP pushes the performance boundaries of scalable GNNs, particularly on large-scale and sparse industrial graphs.

Furthermore, **PaSca** [27] provides a unified framework for GNN simplification methods and introduces the Scalable Graph Neural Architecture Paradigm (**SGAP**) for GNN simplification methods. As depicted in Figure 4, SGAP typically involves three independent stages: (1) Preprocessing: propagating raw features to generate propagated features for each iteration and aggregating these propagated features to generate the final combined message for each node, (2) Training: feeding the propagated and aggregated information into an MLP for training, and (3) Postprocessing: propagating predictions to generate propagated predictions and

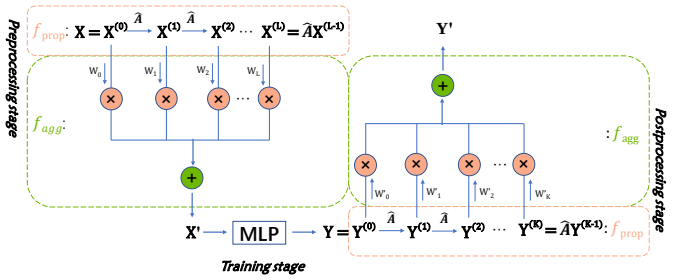


Fig. 4: Illustration of SGAP for GNN simplification methods. Preprocessing: $\mathbf{X}'$ is obtained by propagation f_{prop} and aggregation f_{agg} on raw features $\mathbf{X}$. Training: $\mathbf{Y}$ is generated through an MLP on $\mathbf{X}'$. Post-processing: $\mathbf{Y}'$ is obtained by propagation f_{prop} and aggregation f_{agg} on raw predictions $\mathbf{Y}$.

then aggregating them to obtain the final predictions. The formulation of SGAP is denoted as follows:

$$\begin{aligned} &\text{Preprocessing:} \quad \mathbf{M} \leftarrow f_{\mathrm{prop}}\left(\tilde{\mathbf{A}}, \mathbf{X}\right), \mathbf{X}^{'} \leftarrow f_{\mathrm{agg}}\left(\mathbf{M}\right), \\ &\text{Training:} \quad \mathbf{Y} = \mathrm{MLP}\left(\mathbf{X}^{'}\right) \\ &\text{Postprocessing:} \quad \mathbf{M}^{'} \leftarrow f_{\mathrm{prop}}\left(\tilde{\mathbf{A}}, \mathbf{Y}\right), \mathbf{Y}^{'} \leftarrow f_{\mathrm{agg}}\left(\mathbf{M}^{'}\right), \end{aligned} \tag{12}$$

where $\mathbf{M} = \{\mathbf{X}^{(0)}, \mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(L)}\}$ represents the set of propagated features, with $\mathbf{X}^{(i+1)} = \tilde{\mathbf{A}}\mathbf{X}^{(i)}$, and $\mathbf{X}^{(0)} = \mathbf{X}$; $\mathbf{M}^{'} = \{\mathbf{Y}^{(0)}, \mathbf{Y}^{(1)}, \ldots, \mathbf{Y}^{(K)}\}$ denotes the set of propagated predictions, with $\mathbf{Y}^{(i+1)} = \tilde{\mathbf{A}}\mathbf{Y}^{(i)}$, and $\mathbf{Y}^{(0)} = \mathbf{Y}$; $f_{\mathrm{prop}}(\cdot)$ signifies the propagation function, and $f_{\mathrm{agg}}(\cdot)$ denotes the aggregation function. Generally, SGAP first obtains propagated features or predictions across different iterations using various propagation matrices. Then, SGAP aggregates these propagated features either with or without attention techniques. Subsequently, SGAP trains MLP on the propagated and aggregated features to obtain predictions. Finally, SGAP propagates and aggregates these predictions. It is worth noting that the first aggregation function in the Preprocessing stage will be transferred to the Training stage if it contains learnable parameters; and the second aggregation function in the Postprocessing stage is prohibited from containing learnable parameters due to scalability.

Building upon on the SGAP paradigm, PaSca introduces a comprehensive design space, encompassing 150k possible designs of GNN simplification methods(e.g., SGC, SIGN, GAMLP, etc.). Instead of concentrating on individual designs, PaSca emphasizes the overall design space, resulting in superior performance and scalability on both academic datasets and an industry short-form video recommendation graph from one partner enterprise. Additionally, rather than optimizing predictive performance alone, PaSca can tackle the accuracy-efficiency trade-off through multi-objective optimization, and provides architectures to meet different needs for performance and inference time.

Overall, GNN simplification is a technique aimed at accelerating GNN training by decoupling the MP paradigm. This method offers significant advantages, such as utilizing sparse matrix multiplication during the preprocessing stage to reduce time consumption and enabling mini-batch training to lower memory consumption. However, the absence of a trainable aggregation process limits the expressiveness of GNN simplification models, restricting their applicability.

4 GNN Inference Acceleration

GNNs have demonstrated significant success in handling graph-structured data due to their ability to model complex relationships and interactions. However, their inefficient inference makes the deployment of large-scale and complex graphs challenging in resource-constrained environments, such as mobile devices or edge computing platforms [79], [88]. In this section, we discuss three types of inference acceleration methods: knowledge distillation, GNN quantization, and GNN pruning. For each category, we first introduce the fundamental definitions and properties, followed by examples of typical works.

4.1 Knowledge Distillation

Knowledge distillation (KD) is a prominent method for model compression and acceleration. This approach facilitates the extraction and transfer of knowledge from a "teacher" model to a "student" model, garnering significant attention in recent years [107], [108], [109]. The process begins with the independent training of the teacher model, followed by the training of the student model using the pretrained teacher as a fixed reference. The primary goal of KD is to enable the student model to mimic the teacher by leveraging diverse knowledge forms such as logits, activations, neurons, and features. This knowledge underpins the learning trajectory of the student model. Typically, KD involves guiding the student model to minimize the knowledge distillation loss, $\mathcal{L}_{KD}$, which measures the discrepancy between the student and teacher models. This is formally defined as:

$$\mathcal{L}_{KD} = DIV(k^t, k^s). \tag{13}$$

Here, $DIV(\cdot,\cdot)$ represents the divergence function, such as the Kullback-Leibler divergence $D_{KL}(\cdot,\cdot)$, and k^t and k^s represent the knowledge from the teacher and student models, respectively. In addition to computing $\mathcal{L}_{KD}$, the student model is usually trained with a dual objective: mastering the original downstream task through cross-entropy (CE) loss and assimilating the teacher's knowledge through $\mathcal{L}_{KD}$. The former objective supports task-specific classification learning, while the latter facilitates the knowledge transfer from the teacher to the student.

Applying KD to GNNs primarily aims to enhance performance or compress models, thereby accelerating GNN inference [37], [20]. In this context, the student model often achieves greater efficiency than the teacher model. Such efficiency improvements may include reductions in the number of layers, neuron counts, or hidden dimensions per layer, as well as fewer parameters or adoption of a more scalable model architecture like an MLP. After distillation, the student model operates more efficiently while still delivering performance comparable to the original teacher GNN. In this survey, KD techniques for GNN inference acceleration are categorized into two types: GNN-to-GNN KD methods and GNN-to-MLP KD methods, based on the type of student models involved.

4.1.1 GNN-to-GNN KD

As the pioneering KD method tailored for GNNs, **LSP** [20] distills local node structures from a teacher GNN to a student GNN. LSP defines the local structure of a node v with vectors $LS_v \in \mathbb{R}^{d(v)}$, capturing the similarity between node v and all its immediate neighbors. Specifically, the vector element $LS_v(u)$ encodes the structural relationship from node v to node u. The computation of $LS_v(u)$ is expressed as:

$$LS_v(u) = \frac{e^{\mathrm{SIM}(h_v,h_u)}}{\sum_{i\in\mathcal{N}(v)} e^{\mathrm{SIM}(h_v,h_i)}}, u \in \mathcal{N}(v), \tag{14}$$

where $\mathrm{SIM}(\cdot,\cdot)$, representing a similarity function, typically computes the squared Euclidean distance between node feature vectors h_i and h_j as $\|h_i - h_j\|_2^2$. Consequently, based on these vectors, LSP introduces a local structure preserving loss $\mathcal{L}_{LSP}$ as:

$$\begin{aligned}\mathcal{L}_{LSP} &= \frac{1}{n}\sum_{v\in\mathcal{V}} D_{KL}(LS_v^s || LS_v^t) \\ &= \frac{1}{n}\sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{N}(v)} LS_v^s(u)\log\frac{LS_v^s(u)}{LS_v^t(u)}.\end{aligned} \tag{15}$$

LSP has demonstrated excellent performance in knowledge distillation across various domains, such as node classification and 3D object recognition [110].

Drawing inspiration from KD on graphs, **TinyGNN**, as proposed in [79], integrates a peer-aware module that explicitly captures local structural details through peer node data and a neighbor distillation technique. This technique assimilates knowledge from a more complex teacher GNN, enabling TinyGNN to achieve performance comparable to or better than more intricate models, while providing a 7.73×–126.59× speed enhancement during inference across four real-world large-scale graphs, incluing Facebook web graph, Chameleon network, Squirrel network and AliGraph constructed from realworld traffic logs in Taobao.

To further improve the performance of student GNNs, **G-CRD** [81] not only preserves local structural information but also endeavors to capture global structural information. G-CRD enhances student GNN performance by aligning node embeddings with those of a teacher GNN within a shared representation space using contrastive learning, which helps preserve global topology implicitly. This approach has been shown to boost the performance and robustness of student GNNs, surpassing LSP. Concurrently, **GKD** [82] investigates the role of the heat kernel in GNNs and introduces the Neural Heat Kernel (NHK) to capture global structures. By leveraging NHK, GKD delineates the geometric properties of graphs' underlying manifolds, facilitating the transfer of global geometric knowledge from a teacher to a student GNN. **GraphAKD** [84] employs adversarial learning to distill both local and global node representations, enhancing the student GNN's overall performance.

In the realm of graph classification tasks, **GFKD** [80] proposes a novel KD approach that operates effectively even without graph data. GFKD simulates graph topology structures using a multivariate Bernoulli distribution and facilitates knowledge transfer from a teacher GNN to a student GNN through the generation of synthetic graphs.

4.1.2 GNN-to-MLP KD

GNN-to-GNN KD methods aim to train student GNNs with fewer parameters to accelerate inference. Despite these improvements, the time-intensive MP remains necessary during inference. To address this issue, recent research has explored distilling knowledge from trained GNNs into MLPs, which are typically faster and simpler than GNNs. This approach removes the reliance on the inference graph and facilitates efficient deployment in latency-sensitive applications, eliminating the need for time-intensive MP.

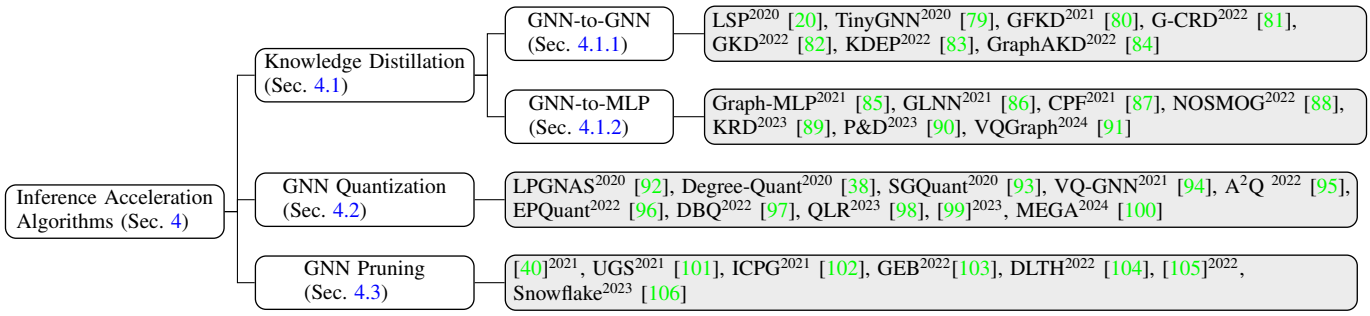


Fig. 5: The taxonomy of inference acceleration algorithms in GNNs. The number in the top right corner represents the publication year.

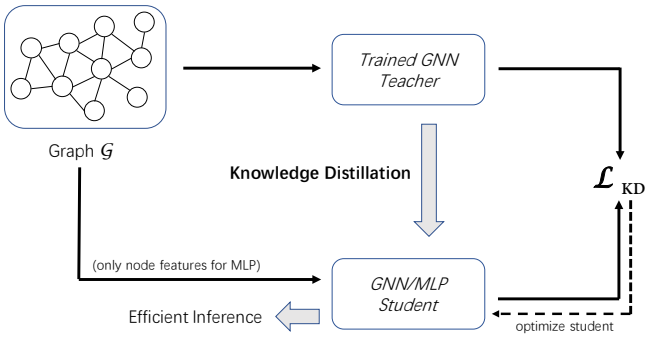


Fig. 6: An overview for knowledge distillation on graphs. In this process, a graph $\mathcal{G}$ is processed by both the trained teacher GNN and the trainable student model. Knowledge distillation guides the student model to emulate the teacher GNN through the loss function $\mathcal{L}_{KD}$, thus optimizing the performance of the student model.

The combination of MLP simplicity and rich knowledge distilled from GNNs significantly enhances inference speed without markedly compromising performance.

GLNN [86] exemplifies this approach. Following the standard KD workflow, GLNN trains an MLP student to mimic the output logits of the GNN teacher by leveraging the teacher's predictions to guide learning. This method distills topological knowledge implicitly, without explicit dependence on the graph structure. In a production environment, for both transductive and inductive predictions across seven datasets, the GLNN student MLP surpasses standalone MLPs by an average of 12.36% and matches the performance of GNNs on six of the seven datasets. However, it is important to recognize that GLNN may underperform when node labels are predominantly influenced by the graph structure rather than node features.

Structural information can also be transferred in GNN-to-MLP KD. This process involves encoding the graph's structural patterns into the MLP student, enabling it to interpret and predict graph-based relationships without directly engaging in message-passing operations. **Graph-MLP** [85] seeks to train an MLP student model by integrating CE loss for node classification with Neighbor Contrastive (NContrast) loss to capture graph structure information. For node v, the NContrast loss is defined as:

$$\mathcal{L}_{NContrast_v} = -\log \frac{\sum_{u \in B} \gamma_{vu} e^{\text{SIM}(h_v, h_u)/\tau}}{\sum_{k \in B} e^{\text{SIM}(h_v, h_k)/\tau}}, \qquad (16)$$

where SIM is the cosine similarity, τ is the temperature parameter, B is the set of sampled nodes, and γ_{vu} represents the connection strength between node v and node u, which is non-zero only for nodes within certain hops of node v. Remarkably, when combined with the distilled structural knowledge from a teacher GNN, the student MLP often achieves comparable or superior performance in node classification tasks with a simpler and more efficient structure. To further enhance the performance of student MLPs, Tian et al. [88] introduce **NOSMOG**, a method for developing Noise-robust, Structure-aware MLPs on Graphs. NOSMOG mitigates the impact of lacking graph structural information and sensitivity to noisy node features by incorporating positional features to align MLPs with GNNs, using representational similarity distillation to utilize structural insights from a teacher GNN, and implementing adversarial feature augmentation to bolster robustness against feature noise.

Moreover, Wu et al. [89] quantified the reliability of nodes in GNNs by evaluating the invariance of their information entropy to noise perturbations. To facilitate reliable knowledge distillation, they developed an effective method named **KRD**. This approach assesses each node's probability of being an informative and dependable knowledge source. Based on these probabilities, KRD selects additional reliable nodes to guide the training of student MLPs. However, the representation space of GNNs may not fully capture diverse local graph structures, which can restrict knowledge transfer to MLPs. To overcome this limitation, Yang et al. [91] introduced **VQGRAPH**, a framework that integrates GNNs and MLPs by developing a robust graph representation space. VQGRAPH employs an encoder derived from vector-quantized variational autoencoders (VQ-VAEs) to function as a structure-aware graph tokenizer, establishing a significant codebook. Utilizing this codebook, VQGRAPH implements a token-based distillation strategy with soft token assignments, effectively facilitating the transfer of structural knowledge from GNNs to MLPs.

Overall, KD on graphs is designed to train scalable student models by transferring knowledge from complex teacher GNNs to simpler models, thereby accelerating GNN inference. KD is applicable to a wide range of tasks, including natural language inference, image segmentation, and various graph learning tasks such as node and graph classification. Both GNN-to-GNN and GNN-to-MLP strategies provide unique advantages depending on the application requirements and resource constraints, merging the capabilities of GNNs with simpler models to achieve efficient, scalable, and high-performing models. Nevertheless, the success of KD hinges on the quality of distilled knowledge, choices in model architecture, distillation techniques, and the design of distillation algorithms, underscoring the significance and challenge of advancing KD on graphs.

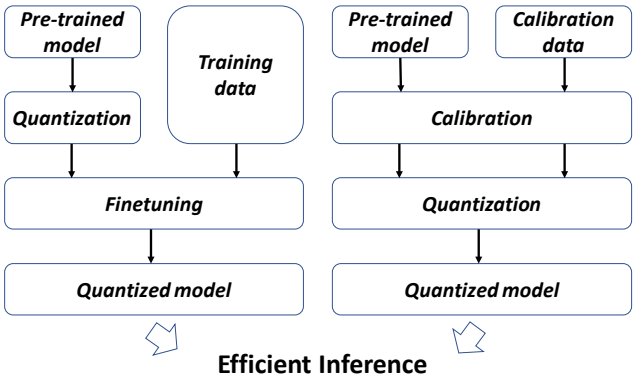


Fig. 7: An overview of Quantization Aware Training (QAT) and Post-Training Quantization (PTQ) techniques. In QAT, a pretrained model is quantized and subsequently fine-tuned with training data to adjust parameters and mitigate accuracy loss. Conversely, in PTQ, a pretrained model is calibrated using a small subset of training data to determine clipping ranges and scaling factors, after which the model undergoes quantization based on these calibration results.

4.2 GNN Quantization

Quantization is the process of mapping continuous data, such as neural network weights and activations, to more compact representations [111], [112], [113]. The primary objective of model quantization is to reduce both computational demands and memory requirements. Typically, this process involves converting high-precision numerical values to lower precision formats (e.g., from 32-bit floating point to 8-bit integers). Such quantized models are not only smaller in size but also facilitate faster inference, owing to decreased memory access and enhanced speed of low-precision arithmetic operations.

As illustrated in Figure 7, quantization can be categorized into two categories: Quantization Aware Training (QAT) and Post-Training Quantization (PTQ). QAT generally requires more computational resources and time but often results in higher model accuracy compared to PTQ, which is widely recognized as the standard method for creating robust quantized models with minimal error [114], [115]. In its basic form, QAT introduces the numerical errors caused by quantization during the forward pass and employs the Straight Through Estimator [116] to compute gradients as though quantization had not occurred. Specifically, quantization of a tensor x in the forward pass is typically implemented as follows:

$$\mathrm{Q}(x) = \min(q_{max}, \max(q_{min}, [\frac{x}{s} + z])), \tag{17}$$

where $\mathrm{Q}(\cdot)$ denotes the quantization operator, x represents a real-valued input (activation or weight), s is a scaling factor that adjusts x within the $[q_{min}, q_{max}]$ range, and z is an integer zero point.

To investigate QAT specifically designed for GNNs, Tailor et al. [38] analyze why standard QAT often fails for GNNs and identify that high in-degree nodes are primarily responsible for poor gradient estimates, especially during the aggregation phase. Based on these findings, they propose an architecture-agnostic QAT method for GNNs, termed Degree-Quant, in which high in-degree nodes are maintained at full precision while other nodes operate at reduced precision. Although quantization typically results in performance degradation, INT8 GNNs employing Degree-Quant not only achieve results comparable to those of FP32 models in most cases but also demonstrate up to 4.7× speedups and 4-8× reductions in memory usage relative to FP32 models.

Inspired by the differential impact of various layers on the loss function, mixed-precision quantization has been developed to assign different bitwidths to different layers, enhancing model compression [117]. Zhu et al. [95] introduced a mixed-precision quantization method for GNNs, termed Aggregation-Aware mixed-precision Quantization ($\mathbf{A^2Q}$). This approach autonomously learns and allocates optimal bitwidths to individual nodes within the graph, achieving an impressive 18.6x compression ratio with minimal accuracy loss, while significantly reducing time and memory demands during GNN inference.

In addition to the standard QAT methods described above, **LPGNAS** [92] pioneers the integration of network architecture search (NAS) with quantization and establishes a quantization search space for GNNs. This innovative NAS mechanism ensures that both architecture and quantization decisions remain differentiable. LPGNAS autonomously optimizes the architecture and quantization strategy for various GNN components through back-propagation in a single search cycle. Furthermore, Feng et al. [93] introduced a PTQ method for GNNs, named SGQuant. SGQuant utilizes an automated bit-selection technique to identify the most appropriate bits for quantization. However, SGQuant focuses on quantizing node features while maintaining weights at full precision, which may limit its efficiency and practical application.

Moreover, recent studies have investigated vector quantization for GNNs, utilizing vectors from a codebook developed during training rather than the original features [94], [96]. For example, VQ-GNN [94] enhances mini-batch MP within each GNN layer by updating the codebook and approximating MP across mini-batches of nodes and codewords. This method achieves significant efficiency gains in terms of memory usage, training and inference times, and convergence speed. However, it is crucial to acknowledge that searching for quantized vectors within the codebook presents substantial computational challenges for vector quantization techniques.

In the context of GNNs, most quantization approaches aim to enable the use of low-precision integer arithmetic during inference to expedite model processing and minimize memory consumption while preserving accuracy. Although this review primarily discusses quantization for inference acceleration, it is important to recognize that several studies [118], [119], [120] have also explored quantization for training acceleration. In particular, advancements in half-precision and mixed-precision training have been the key factors driving a tenfold increase in throughput for AI accelerators [121]. Nonetheless, achieving precision less than half-precision often requires extensive tuning, and recent research in quantization has predominantly focused on inference. This survey specifically highlights GNN quantization strategies for inference.

4.3 GNN Pruning

The process of eliminating non-critical or redundant neurons from neural networks is known as *pruning* [122], [123], [39]. This method is widely used to reduce the size and computational demands of trained networks, thereby enhancing inference speed without significantly impacting accuracy. During pruning, neurons that minimally affect the model's output or loss function are selectively removed, creating a sparse computational graph. The trade-off between inference speed and accuracy is crucial;

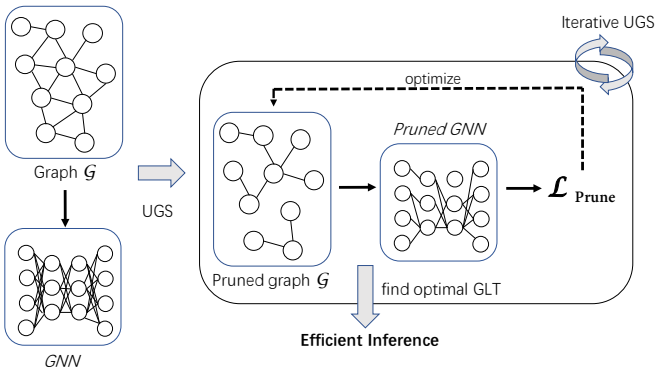


Fig. 8: An overview of obtaining a GLT through iterative UGS is as follows: In a single iteration of UGS, redundant edges and weights are eliminated from the graph and the GNN to identify a sub-optimal solution, as dictated by the loss function $\mathcal{L}_{prune}$. Subsequently, these sub-optimal solutions serve as the basis for further iterations of UGS, where the graph and GNN are co-optimized to achieve the final, optimal GLT.

as more parameters are pruned, the model operates faster but potentially at the cost of accuracy, and the reverse is also true. Specifically, pruning GNNs introduces additional complexities, as GNN inference depends on both node features and the local graph structure, making it more complex than typical inference processes on independent data instances.

To enhance large-scale and real-time GNN inference, **Zhou et al.** [40] present a method to prune the dimensions in each GNN layer. This approach formulates the GNN pruning problem as a LASSO optimization task, selecting input channels to effectively approximate the output. For full inference, pruned GNN models achieve an average of 3.27× speedup with only 0.002 drop in F1-Micro on five popular datasets. Moreover, this method is applicable to a wide range of GNN architectures.

As a significant contribution to the field of model pruning, Frankle et al. [124] introduced the Lottery Ticket Hypothesis (**LTH**), revealing that highly sparse and independently trainable sub-networks can be identified from dense models through iterative pruning. To extend LTH's applicability to GNNs, Chen et al. [101] define a graph lottery ticket (**GLT**) as a pair consisting of a core sub-dataset and a sparse sub-network. These GLTs can be trained in isolation to match the performance of the full model and graph.

To efficiently extract GLTs from the full graph and the original GNN, a unified GNN sparsification (**UGS**) framework is proposed, which simultaneously reduces edges in the graph and parameters in GNNs. Under UGS, pruned graphs and GNNs are co-optimized to find sub-optimal solutions for GLT during training through the loss function $\mathcal{L}_{prune}$ as:

$$\begin{aligned}\mathcal{L}_{prune} =& \mathcal{L}\left(\{\boldsymbol{m}_g \odot \boldsymbol{A}, \boldsymbol{X}\}, \boldsymbol{m}_\theta \odot \Theta\right) \\ &+ \gamma_1 \left\|\boldsymbol{m}_g\right\|_1 + \gamma_2 \left\|\boldsymbol{m}_\theta\right\|_1,\end{aligned} \tag{18}$$

where $\mathcal{L}$ is CE loss for classification tasks; $\odot$ is the element-wise product; $\boldsymbol{m}_g$ and $\boldsymbol{m}_\theta$ are two differentiable masks for indicating the insignificant connections and weights in the graph and GNNs; γ_1 and γ_1 are the hyparameters to control the ℓ_1 sparsity regularizers of $\boldsymbol{m}_g$ and $\boldsymbol{m}_\theta$, respectively.

As shown in Figure 8, through iterative UGS, optimal GLTs can be found for further inference. The identified GLTs can achieve 20% to 98% MAC savings, with 5% to 58.19% sparsity in graphs and 20% to 97.75% sparsity in GNN models, with minimal to no performance degradation for node classification tasks.

Following the concepts of GLT and UGS, Hui et al. [105] discovered that the performance of pruned GNNs significantly degrades as graph sparsity increases beyond a certain threshold. To address this issue, they introduce two techniques aimed at enhancing UGS performance in high-sparsity scenarios. Additionally, in a novel exploration, Wang et al. [104] tackled the complex problem of transferring a random ticket to a graph lottery ticket within GNNs. Unlike GLT, which pre-trains dense GNNs to recognize graph lottery tickets, their approach of transforming a random ticket into a high-performance sparse network and core subgraph proves more practical and valuable. Building upon this idea, they further propose the Dual Graph Lottery Ticket (**DGLT**) framework. DGLT consistently outperforms GLT across various graph/network sparsity levels in GNN benchmarks, offering a triple-win solution: high sparsity, high performance, and good explainability.

Overall, pruning in GNNs is a crucial technique for achieving faster and more scalable GNN inference by removing non-essential neurons while maintaining accuracy. However, it presents challenges, including balancing the trade-off between speed and accuracy, managing increased complexity from graph dependencies, and mitigating performance degradation at high sparsity levels. Despite these challenges, pruning remains a valuable and worthwhile method for large-scale and real-time applications.

5 GNN Execution Acceleration

GNN execution acceleration methods can accelerate both training and inference of GNNs. In this section, we discuss two types of GNN execution acceleration methods, called GNN binarization and graph condensation. We first discuss the fundamental definition and property of these methods, and exemplify typical work belonging to these categories.

5.1 GNN Binarization

Binarization is a technique that pushes model quantization to the extreme by employing only a single bit for weights and activations, minimizing memory usage [146], [147], [148]. By binarizing neural networks, heavy matrix multiplications can be replaced with lightweight bitwise XNOR and Bitcount operations. Consequently, binary neural networks outperform other compression methods (such as knowledge distillation, pruning, and quantization) by significantly reducing storage requirements and computational overhead in both inference and training processes. However, binarization inherently leads to substantial information loss, and its discontinuity poses challenges for deep network optimization. To mitigate these issues, researchers have proposed various algorithms, achieving promising progress. For instance, BinaryConnect [149] binarizes network parameters, replacing most floating-point multiplications with additions, resulting in near state-of-the-art performance.

In the realm of GNNs, Wang et al. [41] pioneered the concept of Binary GCN (**Bi-GCN**). This approach significantly reduces memory consumption by 30x for both network parameters and input node attributes. Additionally, Bi-GCN achieves an average 47x acceleration in inference on citation networks. To enhance training speed as well, the authors introduce a novel backpropagation method that considers relationships among binary weights during

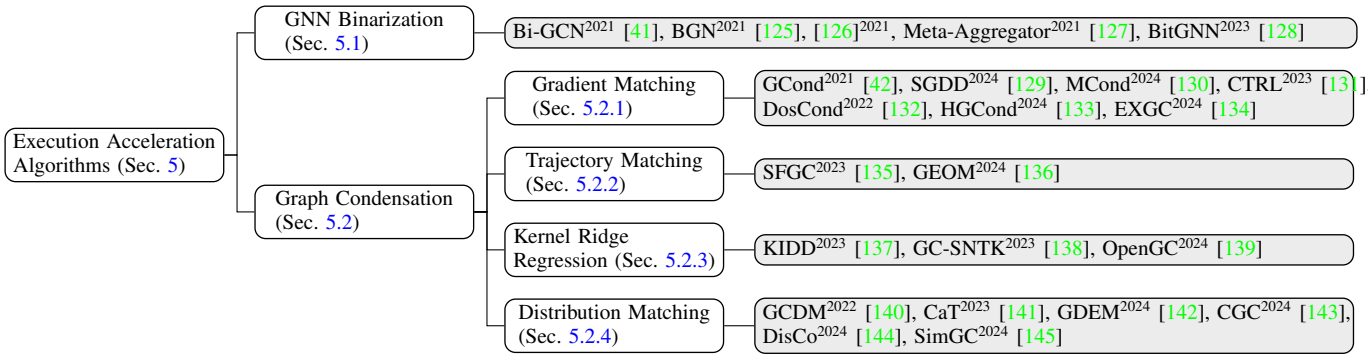


Fig. 9: The taxonomy of execution acceleration algorithms in GNNs. The number in the top right corner represents the publication year.

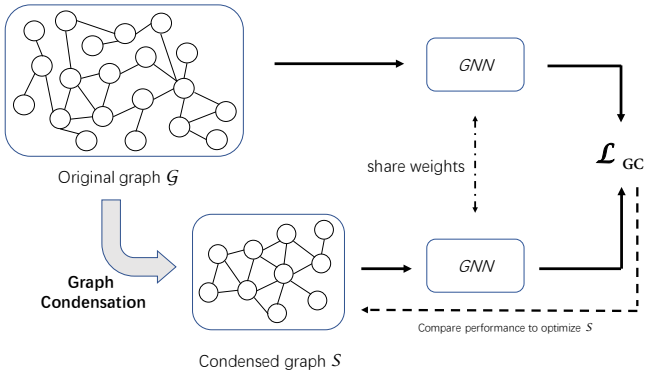


Fig. 10: An overview of graph condensation: both the original graph $\mathcal{G}$ and the condensed graph $\mathcal{S}$ are processed by the same GNN, resulting in respective classification outcomes. By comparing these classification results, $\mathcal{S}$ is optimized according to the loss function $\mathcal{L}_{GC}$.

the back propagation process. Remarkably, despite the substantial memory savings and acceleration, Bi-GCN delivers performance comparable to the standard GCN on various benchmark datasets.

Furthermore, Wang et al. [125] propose **BGN**, which leverages binarized vectors and bitwise operations to generate binary graph representations. BGN can be seamlessly integrated into various GNNs, immediately reducing their memory consumption. The efficient bit-wise operations between binary vectors can significantly speed up model inference. In addition, BGN's gradient estimator enables efficient backpropagation, further accelerating the training process.

Jing et al. [127] investigated a new meta aggregation scheme for binarizing GNNs, aptly named **Meta-Aggregator**. They observe that a single bit GNN framework lacks sufficient discriminative power to distinguish graph topologies, resulting in a significant performance drop. To address this, they introduce two aggregators: the Greedy Gumbel Aggregator (GNA) and the Adaptable Hybrid Aggregator (ANA). GNA learns to select an optimal aggregator from a candidate pool, while ANA learns to combine the benefits of multiple individual aggregators through hybrid aggregation behavior. These two aggregators enhance binary training accuracy.

5.2 Graph Condensation

Graph condensation (GC), also known as graph distillation, is a data-centric technique designed to synthesize a small yet highly informative graph dataset that encompasses both node attributes and topology structure, effectively representing large original graphs. Through task-driven optimization, models trained on these small condensed datasets can achieve performance comparable to those trained on large original graphs [43], [42]. Once these highly informative synthetic graphs are obtained, they can be used to train and infer subsequent GNNs. Consequently, graph condensation accelerates both training and inference, benefiting node classification and graph classification tasks.

Similar to GC is graph reduction, which includes methods like graph sparsification [150] and graph coarsening [151]. These methods simplify graphs to expedite graph algorithms while maintaining the essential characteristics of large graphs. However, they rely on heuristic methods, resulting in poor generalization across different GNN architectures or downstream tasks, making them far less effective than GC.

To explore the mathematical representation of GC, the objective of GC is to compress a large, original graph $\mathcal{G} = (\mathbf{A}, \mathbf{X}, \mathbf{Y})$ into a small, synthetic, and highly informative graph, denoted as $\mathcal{S} = (\mathbf{A}^{GC}, \mathbf{X}^{GC}, \mathbf{Y}^{GC})$, where $\mathbf{A}^{GC} \in \mathbb{R}^{n' \times n'}, \mathbf{X}^{GC} \in \mathbb{R}^{n' \times f}, \mathbf{Y}^{GC} \in \mathbb{R}^{n'}$ with $n' \ll n$. To this end, the loss for the original graph $\mathcal{G}$ and synthetic graph $\mathcal{S}$ is defined as follows:

$$\begin{aligned} \mathcal{L}^{\mathcal{G}}(\theta) &= \ell\left(f_\theta(\mathcal{G}), \mathbf{Y}\right), \\ \mathcal{L}^{\mathcal{S}}(\theta) &= \ell\left(f_\theta(\mathcal{S}), \mathbf{Y}^{GC}\right), \end{aligned} \tag{19}$$

where ℓ is the task-specific objective, such as CE loss, and $f_\theta(\cdot)$ is a relay GNN, parameterized by θ. Therefore, the graph condensation is formulated as follows:

$$\begin{aligned} &\min_{\mathcal{S}} \mathcal{L}^{\mathcal{G}}\left(\theta^{\mathcal{S}}\right) \\ &\text{s.t. } \theta^{\mathcal{S}} = \arg\min_{\theta} \mathcal{L}^{\mathcal{S}}(\theta). \end{aligned} \tag{20}$$

Addressing the objective requires the incorporation of the task-specific objective ℓ and resolving a nested loop optimization, which involves unrolling the entire training trajectory of the relay model optimization. This process can be prohibitively expensive. To alleviate the complexity of the optimization process, various approximation methods have been proposed, including gradient matching, trajectory matching, kernel ridge regression, and distribution matching. As shown in Figure 10, these methods focus on comparing the performance of models trained on $\mathcal{G}$ and $\mathcal{S}$ to optimize $\mathcal{S}$ according to $\mathcal{L}_{GC}$.

5.2.1 Gradient Matching

Gradient matching aims to align the gradients of the GNN parameters trained on the original graph with those obtained from

the condensed graph. This alignment ensures that the learning dynamics on the smaller graph mimic those on the original graph, resulting in similar model performance.

A notable method in this context is **GCond**, proposed by Jin et al. [42]. This approach optimizes the condensed graph by minimizing the difference between the gradients of the loss function with respect to the GNN parameters on both the original and condensed graphs, using $\mathcal{L}_{cond}$ as follows:

$$\mathcal{L}_{GC} = \mathbb{E}_{\theta_0 \sim \Theta}\left[\sum_{t=1}^{T} \mathcal{D}\left(\nabla_\theta \mathcal{L}^{\mathcal{G}}\left(\theta_t\right), \nabla_\theta \mathcal{L}^{\mathcal{S}}\left(\theta_t\right)\right)\right] \quad \text{s.t. } \theta_{t+1} = \text{opt}\left(\mathcal{L}^{\mathcal{S}}\left(\theta_t\right)\right), \tag{21}$$

where θ_0 is the initialization of $\theta^{\mathcal{T}}$, $\theta^{\mathcal{S}}$, Θ is a specific distribution for relay model initialization, and $\text{opt}(\cdot)$ is the model parameter optimizer.

As shown in Equation 21, GCond computes gradients for both graphs and employs a gradient descent algorithm to iteratively update the condensed graph, ensuring that the gradients align closely. This direct targeting of the training process helps retain critical learning signals from the original graph. GCond has been shown to maintain over 95% of the original test accuracy while reducing the graph size by more than 99.9%. Additionally, the same condensed graph can be transferred to train different GNNs to achieve good performance.

Moreover, Jin et al. [132] introduced **DosCond**, which extends GC to graph classification tasks. DosCond learns discrete, synthetic graphs for condensing graph datasets, capturing the discrete structure via a graph probabilistic model that can be learned in a differentiable manner. As a gradient matching method, DosCond proposes a one-step gradient matching scheme, accelerating the traditional gradient matching process by performing gradient matching in a single step without training network weights. Theoretical analysis supports the rationality of this one-step gradient matching.

MCond [130] focuses on accelerating GNN inference. In practical graph systems, newly presented nodes are connected to existing nodes in the original graph. Consequently, the extensive size of large original graphs poses a considerable challenge for real-time inference. To perform efficient inference on inductive nodes, MCond adopts GC and proposes learning a mapping from original nodes to condensed nodes, which can seamlessly integrate new nodes into the synthetic graph. This approach allows direct information propagation on the small condensed graph, significantly accelerating the inference process. Notably, on the Reddit dataset, MCond achieves up to a 121.5× inference speedup and a 55.9× reduction in storage requirements compared to methods relying solely on the original graph.

Additionally, **EXGC** presents a new GC technique that enhances efficiency through Mean-Field variational approximation and introduces explainability via the gradient information bottleneck. This method prunes redundant parameters and prioritizes informative nodes for training. Gao et al. [133] present **HGCond**, a method for efficient GNN training on large heterogeneous graphs using cluster-based feature initialization and an OPS exploration strategy, outperforming existing approaches.

5.2.2 Trajectory Matching

Gradient matching primarily aligns single-step gradients; however, it may accumulate errors when the relay GNN is iteratively updated using condensed data across multiple steps. To address this issue, trajectory matching is proposed. Trajectory matching involves aligning the final points of the optimization trajectory of the GNN parameters trained on the condensed data with those observed on the original data.

Following this framework, Zheng et al. [135] introduce the Structure-Free Graph Condensation paradigm (**SFGC**) to effectively distill large-scale real-world graphs into small-scale synthetic data without graph structures. Under SFGC, the training trajectory meta-matching scheme and the graph neural tangent kernel work collaboratively to synthesize small-scale, graph-free data with superior effectiveness and generalization ability. Anticipated as a bridge between GNNs and industrial MLPs, SFGC addresses graph data scalability while preserving the expressive performance of graph learning. Moreover, **GEOM** makes the first attempt toward lossless graph condensation, condensing Citeseer to 0.9%, Cora to 1.3%, Ogbn-arxiv to 5%, Flickr to 1%, and Reddit to 5% without any performance loss when training a GCN. This method innovates by employing curriculum learning for diverse expert trajectory training and an expanding window matching technique, leading to efficient knowledge transfer into a compact graph that preserves the performance of GNNs.

5.2.3 Kernel Ridge Regression

Kernel ridge regression (KRR) addresses the limitations of gradient and trajectory matching, which involve complex bi-level optimization leading to slow convergence and reduced performance [152]. This method reformulates the original task as a regression problem by substituting the neural network with KRR. This approach is not only computationally efficient but also effective as it can find the exact solution of the distillation objective.

Regarding graph condensation, **KIDD** [137] proposes to distill a given graph dataset by precisely solving a bilevel distillation objective using kernel ridge regression. To further accelerate computation and expand functionality, KIDD also investigates a series of model enhancements specifically designed for graph dataset distillation scenario. Moreover, **Wang et al.** [138] propose a principled way to improve graph condensation efficiency by replacing the inner GNN training with a structure-based neural tangent kernel in the KRR paradigm. This approach demonstrates powerful generalization capability across various GNN architectures.

5.2.4 Distribution Matching

Distribution matching focuses on aligning the statistical distributions of node features and edges between the original and condensed graphs. Accordingly, the optimization objective of distribution matching is formulated as follows:

$$\mathcal{L}_{GC} = \mathbb{E}_{\theta_0 \sim \Theta}\left[\mathcal{D}\left(f_\theta\left(\mathcal{T}\right), f_\theta\left(\mathcal{S}\right)\right)\right], \tag{22}$$

where the distance measurement $\mathcal{D}(\cdot, \cdot)$ is specifically designed for class-wise comparison, requiring the use of class labels in distribution matching. This approach ensures that the condensed graph retains similar statistical properties, which is essential for preserving the overall characteristics of the original graph.

Liu et al. [140] view the original graph as a distribution of receptive fields and propose Graph Condensation via Receptive Field Distribution Matching (**GCDM**). GCDM aims to synthesize a small graph whose receptive fields share a similar distribution to

the original graph. GCDM optimizes the synthetic graph by minimizing a distribution matching loss quantified using maximum mean discrepancy. The resulting synthetic graph can be effectively transferred to various GNNs, significantly improving condensing speed. Additionally, Liu et al. [142] propose **GDEM**, a novel distribution matching graph distillation framework that mitigates the dependence on GNNs by matching the eigenbasis instead of the entire graph structure. It is theoretically demonstrated that GDEM preserves essential spectral similarity during distillation.

Overall, graph condensation provides powerful techniques to accelerate GNN training and inference by creating smaller, highly representative graphs. By employing optimization strategies such as gradient matching, trajectory matching, kernel ridge regression, and distribution matching, significant reductions in computational load can be achieved while maintaining model performance. GC contains vast application prospects and is increasingly being utilized across various fields, including hyper-parameter/neural architecture search [153], graph continual learning [154] and federated learning [155]. Despite the promising results of these graph condensation methods, their reliance on accurately imitating GNN training behavior limits the transferability of condensed graphs across different GNN architectures.

6 Libraries for GNNs Acceleration

With the rapid development of GNNs, numerous graph libraries have been developed in recent years. PyTorch Geometric (PyG)[1] [156] and Deep Graph Library (DGL)[2] [157] are the most widely used libraries for graph machine learning. PyG, built upon PyTorch, is designed for geometric deep learning and employs sparse tensor abstraction and acceleration to achieve high data throughput. DGL is an easy-to-use, high-performance, and portable Python library that makes the graph the central programming abstraction and provides flexible and efficient message-passing APIs. However, both libraries are primarily designed for small-scale graphs and are not optimal for large-scale graphs. Consequently, several libraries tailored for large-scale graph like PSGraph [159], AliGraph[3] [160], PGL[4],and Euler[5] have emerged. These libraries either focus on distributed scenarios or lack comprehensive support for various acceleration methods (e.g., GNN simplification methods) and diverse types of graphs (e.g., heterogeneous graphs).

To address these issues, we introduce our geometric library called Scalable Graph Learning (SGL)[6], a toolkit for scalable graph learning on a single machine, which enables deep graph learning on extremely large datasets. Figure 11 illustrates SGL's framework. The main characteristics of SGL are fourfold:

- **High scalability:** SGL can handle graph data with billions of nodes and edges. For training acceleration methods, SGL can follow the message passing paradigm to implement graph sampling methods and the SGAP paradigm to implement GNN simplification methods. For inference acceleration methods, SGL can implement knowledge distillation methods.
- **Ease of use:** SGL offers user-friendly interfaces for easily implementing and evaluating existing scalable GNNs on various downstream tasks, including node classification, node clustering, and link prediction.
- **Data-centric:** SGL integrates data-centric graph machine learning (DC-GML) methods (e.g., graph data augmentation methods) to enhance graph data quality and representation. Additionally, SGL also has customized acceleration algorithms for heterogeneous graphs.

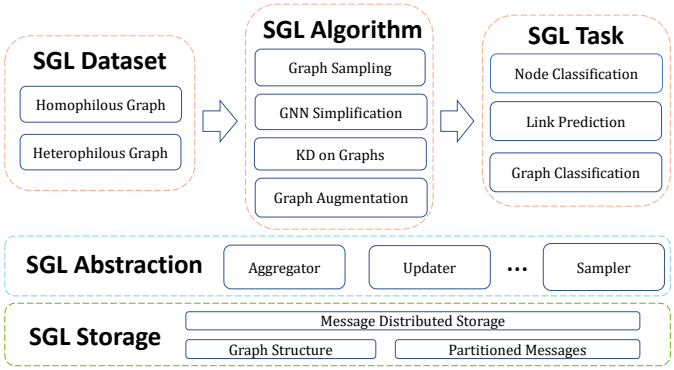


Fig. 11: The architecture of SGL framework. SGL implements SGL Dataset and SGL Algorithm based on SGL Storage and SGL Abstraction to solve the tasks in SGL Task. SGL can implement graph sampling methods and GNN simplification methods to accelerate GNN training. SGL can also implement knowledge distillation methods to accelerate GNN inference.

The main design goal of our SGL is to support scalable graph learning. SGL allows users to easily implement scalable graph neural networks including graph sampling methods, GNN simplification methods and knowledge distillation methods. In addition, users can evaluate its performance on various downstream tasks like node classification, node clustering, and link prediction. SGL also supports auto neural architecture search functionality based on OpenBox [161].

7 Future Directions and Conclusion

7.1 Future Directions

7.1.1 Extend to complex graph types

Extensive research in graph machine learning has focused on complex graph types, including temporal graphs, heterogeneous graphs, and dynamic graphs. Temporal graphs, which incorporate time-evolving nodes and edges, present unique challenges in maintaining and processing temporal information efficiently [162], [163]. Heterogeneous graphs, consisting of multiple types of nodes and edges, require sophisticated methods to manage the diversity in relationships and attributes [164]. Dynamic graphs, characterized by their rapidly changing structures, demand real-time updates and adaptations to ensure accurate and efficient learning [165]. For each of these graph types, the development of specialized acceleration algorithms is essential. Researchers could prioritize creating scalable solutions that can handle the unique challenges posed by these graph type.

7.1.2 Combine with DC-GML

The growing emphasis on DC-GML, encompassing the full spectrum of graph data activities such as collection, exploration, enhancement, exploitation, and maintenance, has garnered significant scholarly interest in recent years [166], [167]. This heightened

1. https://www.pyg.org/
2. https://www.dgl.ai/
3. https://github.com/alibaba/graph-learn
4. https://github.com/PaddlePaddle/PGL
5. https://github.com/alibaba/euler
6. https://github.com/PKU-DAIR/SGL

TABLE 3: Libraries of Deep Learning on Graphs

Name	Parallel Processing Architecture	Language /Framework	Key Characteristics
PyTorch Geometric [156]	Single Machine Multiple GPUs	PyTorch	MP paradigm, unified operations, comprehensive existing methods
Deep Graph Library [157]	Single Machine Multiple GPUs	PyTorch/TensorFlow	MP paradigm, unified operations, improved efficiency
NeuGraph [158]	Single Machine Multiple GPUs	TensorFlow	vertex-centric, pipeline scheduling
PSGraph [159]	Multi-Machine Distributed	PyTorch	scalability, vertex-centric
AliGraph [160]	Multi-Machine Distributed	Unknown	scalability, vertex-centric
Euler	Multi-Machine Distributed	C++/TensorFlow	scalability, edge-centric
PGL	Multi-Machine Distributed	PaddlePaddle	scalability, MP paradigm
SGL	Single Machine Multiple GPUs	Pytorch	scalability, data-centric

focus is driven by the recognition that effective handling of graph data can substantially improve the performance and applicability of GNNs. Key methodologies within DC-GML, such as graph structure learning [168], [169], graph feature enhancement [170], and graph self-supervised learning [171], [172], [173], are pivotal for handling and leveraging the inherent information of large-scale graphs. To harness the full potential of these methodologies, researchers could develop novel acceleration algorithms specifically tailored for DC-GML techniques. In addition, some DC-GML methodologies can be used for accelerating GNNs (e.g. graph reduction, graph condensation), enabling scalable GNNs [43], [150]. Consequently, future work could explore the combination of acceleration algorithms and DC-GML techniques, which not only promises the computational challenges posed by DC-GML when dealing with massive graph datasets, but also improves the performance of GNNs from the data point of view, thus facilitating more robust and scalable GNN applications.

7.1.3 Customized acceleration for applications

Customized acceleration techniques are gaining critical importance across diverse applications such as AI for Science, encompassing fields like biology [16] and chemistry [15], and recommendation systems [23]. In AI-driven scientific research, disciplines such as genomics and materials science present complex and large-scale data challenges. Developing bespoke algorithms that enhance computational efficiency and accuracy is crucial [174]. Recommendation systems require optimized acceleration methods to handle extensive user interaction data and generate personalized content suggestions in real-time [24]. Addressing scalability while maintaining responsiveness is a key challenge. Future work could explore adaptive methods that dynamically adjust to varying data loads and user demands. The design of these customized acceleration techniques is pivotal in addressing the specific challenges and harnessing the full potential of large-scale data in these applications.

7.1.4 Extend to distributed scenarios

Conducting GNN training in a distributed manner has emerged as a prevalent strategy to expedite the processing of large-scale graphs [155], [175]. However, this approach introduces several complexities that must be addressed to ensure efficient performance. One significant challenge is the presence of super nodes, or nodes with an exceptionally high degree, which can lead to severe workload imbalances across computing devices. Furthermore, variations in network bandwidth between devices within the same machine and those spanning different machines add another layer of complexity to distributed GNN training [176]. Consequently, it remains an open question whether existing GNN acceleration algorithms can maintain their performance under these intricate and heterogeneous distributed environments. Future work could explore addressing these challenges, which is critical for the development of robust and scalable GNN training methodologies that can fully leverage the advantages of distributed computing.

7.1.5 Extend to out-of-core scenarios

The rapid growth of graph datasets, often reaching hundreds of gigabytes or even terabytes [28], [29], exceeds the memory capacity of standard machines, requiring the use of external storage like SSDs. While recent studies [177], [178] have explored external storage for GNN training, the high latency of data retrieval remains a major bottleneck [177]. However, the process of retrieving graph data from external storage introduces substantial latency, significantly impeding the efficiency of GNN training [177]. This bottleneck highlights the urgent need for the future research of advanced acceleration algorithms specifically tailored for out-of-core GNN execution.

7.1.6 Extend to new graph learning paradigm

Deploying GNNs in real-world applications requires scalable graph learning paradigms. Emerging approaches like graph unlearning [179] and continual learning [180] show promise but face scalability challenges [181]. Addressing these limitations requires the development of scalable architectures that can implement these advanced graph learning paradigms on industrial-scale graphs. Such advancements are crucial for translating theoretical research into practical applications, ensuring that GNNs can be robust, adaptable, and efficient in diverse and dynamic real-world environments, which could be a promising research direction.

7.2 Conclusion

In conclusion, we presented a systematic view of acceleration algorithms for GNNs on large-scale graphs. Our survey proposes a new taxonomy for acceleration algorithms, which covers three main topics based on their purpose: training acceleration, inference acceleration, and execution acceleration. In particular, we thoroughly discuss, categorize, and summarize the existing algorithms according to the proposed taxonomy. Additionally,

we discuss our Scalable Graph Learning library and share our thoughts on future directions. Acceleration algorithms can pave the way for more efficient GNN pipelines, enabling the use of GNN models on more large-scale graphs. We hope this survey serves as a useful resource for researchers and advances the future work of acceleration algorithms in GNNs

REFERENCES

[1] X. Li, L. Sun, M. Ling, and Y. Peng, "A survey of graph neural network based recommendation in social networks," *Neurocomputing*, 2023.

[2] K. Sharma, Y.-C. Lee, S. Nambi, A. Salian, S. Shah, S.-W. Kim, and S. Kumar, "A survey of graph neural networks for social recommender systems," *ACM Computing Surveys*, 2022.

[3] K. Wei, J. Huang, and S. Fu, "A survey of e-commerce recommender systems," in *2007 international conference on service systems and service management*. IEEE, 2007, pp. 1–5.

[4] Y. Wang, Z. Li, and A. Barati Farimani, "Graph neural networks for molecules," in *Machine Learning in Molecular Sciences*. Springer, 2023, pp. 21–66.

[5] O. Wieder, S. Kohlbacher, M. Kuenemann, A. Garon, P. Ducrot, T. Seidel, and T. Langer, "A compact review of molecular property prediction with graph neural networks," *Drug Discovery Today: Technologies*, vol. 37, pp. 1–12, 2020.

[6] S. Rahmani, A. Baghbani, N. Bouguila, and Z. Patterson, "Graph neural networks for intelligent transportation systems: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 8, pp. 8846–8885, 2023.

[7] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Expert systems with applications*, vol. 207, p. 117921, 2022.

[8] Z. Ye, Y. J. Kumar, G. O. Sing, F. Song, and J. Wang, "A comprehensive survey of graph neural networks for knowledge graphs," *IEEE Access*, vol. 10, pp. 75 729–75 741, 2022.

[9] S. Arora, "A survey on graph neural networks for knowledge graph completion," *arXiv preprint arXiv:2007.12374*, 2020.

[10] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 249–270, 2020.

[11] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[12] Z. Liu and J. Zhou, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[13] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[14] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[15] P. Reiser, M. Neubert, A. Eberhard, L. Torresi, C. Zhou, C. Shao, H. Metni, C. van Hoesel, H. Schopmans, T. Sommer *et al.*, "Graph neural networks for materials science and chemistry," *Communications Materials*, 2022.

[16] R. Li, X. Yuan, M. Radfar, P. Marendy, W. Ni, T. J. O'Brien, and P. M. Casillas-Espinosa, "Graph signal processing, graph neural network and graph learning on biological data: a systematic review," *R-BME*, 2021.

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[18] C. Chen, Y. Wu, Q. Dai, H.-Y. Zhou, M. Xu, S. Yang, X. Han, and Y. Yu, "A survey on graph neural networks and graph transformers in computer vision: a task-oriented perspective," *arXiv preprint arXiv:2209.13232*, 2022.

[19] L. Jiao, J. Chen, F. Liu, S. Yang, C. You, X. Liu, L. Li, and B. Hou, "Graph representation learning meets computer vision: A survey," *IEEE Transactions on Artificial Intelligence*, vol. 4, no. 1, pp. 2–22, 2022.

[20] Y. Yang, J. Qiu, M. Song, D. Tao, and X. Wang, "Distilling knowledge from graph convolutional networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7074–7083.

[21] L. Wu, Y. Chen, K. Shen, X. Guo, H. Gao, S. Li, J. Pei, B. Long *et al.*, "Graph neural networks for natural language processing: A survey," *Foundations and Trends® in Machine Learning*, 2023.

[22] P. Pham, L. T. Nguyen, W. Pedrycz, and B. Vo, "Deep learning, graph-based text representation and classification: a survey, perspectives and challenges," *Artificial Intelligence Review*, vol. 56, no. 6, pp. 4893–4927, 2023.

[23] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," *ACM Computing Surveys*, 2022.

[24] C. Gao, X. Wang, X. He, and Y. Li, "Graph neural networks for recommender system," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 2022, pp. 1623–1625.

[25] 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.

[26] A. Bojchevski, J. Gasteiger, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, M. Lukasik, and S. Günnemann, "Scaling graph neural networks with approximate pagerank," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2464–2473.

[27] W. Zhang, Y. Shen, Z. Lin, Y. Li, X. Li, W. Ouyang, Y. Tao, Z. Yang, and B. Cui, "Pasca: A graph neural architecture search system under the scalable paradigm," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1817–1828.

[28] 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," *arXiv preprint arXiv:2005.00687*, 2020.

[29] A. Khatua, V. S. Mailthody, B. Taleka, T. Ma, X. Song, and W.-m. Hwu, "Igb: Addressing the gaps in labeling, features, heterogeneity, and size of public graph datasets for deep learning research," *arXiv preprint arXiv:2302.13522*, 2023.

[30] S. Zhang, A. Sohrabizadeh, C. Wan, Z. Huang, Z. Hu, Y. Wang, J. Cong, Y. Sun *et al.*, "A survey on graph neural network acceleration: Algorithms, systems, and customized hardware," *arXiv preprint arXiv:2306.14052*, 2023.

[31] S. Chen, J. Liu, and L. Shen, "A survey on graph neural network acceleration: A hardware perspective," *Chinese Journal of Electronics*, vol. 33, no. 3, pp. 601–622, 2024.

[32] 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.

[33] 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 Journal of Automatica Sinica*, vol. 9, no. 2, pp. 205–234, 2021.

[34] M. Serafini and H. Guan, "Scalable graph neural network training: The case for sampling," *ACM SIGOPS Operating Systems Review*, vol. 55, no. 1, pp. 68–76, 2021.

[35] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.

[36] K. Duan, Z. Liu, P. Wang, W. Zheng, K. Zhou, T. Chen, X. Hu, and Z. Wang, "A comprehensive study on large-scale graph training: Benchmarking and rethinking," *Advances in Neural Information Processing Systems*, vol. 35, pp. 5376–5389, 2022.

[37] Y. Tian, S. Pei, X. Zhang, C. Zhang, and N. V. Chawla, "Knowledge distillation on graphs: A survey," *arXiv preprint arXiv:2302.00219*, 2023.

[38] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," in *International Conference on Learning Representations*, 2020.

[39] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[40] H. Zhou, A. Srivastava, H. Zeng, R. Kannan, and V. Prasanna, "Accelerating large scale real-time gnn inference using channel pruning," *Proceedings of the VLDB Endowment*, vol. 14, no. 9, pp. 1597–1605, 2021.

[41] J. Wang, Y. Wang, Z. Yang, L. Yang, and Y. Guo, "Bi-gcn: Binary graph convolutional network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1561–1570.

[42] W. Jin, L. Zhao, S. Zhang, Y. Liu, J. Tang, and N. Shah, "Graph condensation for graph neural networks," in *International Conference on Learning Representations*, 2021.

[43] X. Gao, J. Yu, W. Jiang, T. Chen, W. Zhang, and H. Yin, "Graph condensation: A survey," *arXiv preprint arXiv:2401.11720*, 2024.

[44] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *International Conference on Machine Learning*. PMLR, 2018, pp. 942–950.

[45] Z. Liu, Z. Wu, Z. Zhang, J. Zhou, S. Yang, L. Song, and Y. Qi, "Bandit samplers for training graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6878–6888, 2020.

[46] H. Yu, L. Wang, B. Wang, M. Liu, T. Yang, and S. Ji, "Graphfm: Improving large-scale gnn training via feature momentum," in *International Conference on Machine Learning*. PMLR, 2022, pp. 25 684–25 701.

[47] 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 *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2046–2056.

[48] K.-L. Yao and W.-J. Li, "Blocking-based neighbor sampling for large-scale graph neural networks." in *IJCAI*, 2021, pp. 3307–3313.

[49] Z. Zhang, Q. Liu, Q. Hu, and C.-K. Lee, "Hierarchical graph transformer with adaptive node sampling," *Advances in Neural Information Processing Systems*, vol. 35, pp. 21 171–21 183, 2022.

[50] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," in *International Conference on Learning Representations*, 2018.

[51] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," *Advances in neural information processing systems*, vol. 31, 2018.

[52] 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," *Advances in neural information processing systems*, vol. 32, 2019.

[53] T. Younesian, T. Thanapalasingam, E. van Krieken, D. Daza, and P. Bloem, "Grapes: Learning to sample graphs for scalable graph neural networks," in *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023.

[54] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 1416–1424.

[55] 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 *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 257–266.

[56] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *International Conference on Learning Representations*, 2019.

[57] W. Cong, R. Forsati, M. Kandemir, and M. Mahdavi, "Minimal variance sampling with provable guarantees for fast training of graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1393–1403.

[58] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan, V. Prasanna, L. Jin, and R. Chen, "Decoupling the depth and scope of graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 19 665–19 679, 2021.

[59] M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec, "Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings," in *International conference on machine learning*. PMLR, 2021, pp. 3294–3304.

[60] J. Bai, Y. Ren, and J. Zhang, "Ripple walk training: A subgraph-based training framework for large and deep graph neural network," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[61] Z. Shi, X. Liang, and J. Wang, "Lmc: Fast training of gnns via subgraph sampling with provable convergence," in *The Eleventh International Conference on Learning Representations*, 2022.

[62] H. Zhu and P. Koniusz, "Simple spectral graph convolution," in *International conference on learning representations*, 2020.

[63] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, "Sign: Scalable inception graph neural networks," *arXiv preprint arXiv:2004.11198*, 2020.

[64] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. Du, and J.-R. Wen, "Scalable graph neural networks via bidirectional propagation," *Advances in neural information processing systems*, vol. 33, pp. 14 556–14 566, 2020.

[65] W. Zhang, M. Yang, Z. Sheng, Y. Li, W. Ouyang, Y. Tao, Z. Yang, and B. Cui, "Node dependent local smoothing for scalable graph learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20 321–20 332, 2021.

[66] X. Miao, W. Zhang, Y. Shao, B. Cui, L. Chen, C. Zhang, and J. Jiang, "Lasagne: A multi-layer graph convolutional network framework via node-aware deep architecture," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 1721–1733, 2021.

[67] H. Wang, M. He, Z. Wei, S. Wang, Y. Yuan, X. Du, and J.-R. Wen, "Approximate graph propagation," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1686–1696.

[68] W. Zhang, Z. Sheng, M. Yang, Y. Li, Y. Shen, Z. Yang, and B. Cui, "Nafs: a simple yet tough-to-beat baseline for graph representation learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 26 467–26 483.

[69] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *International Conference on Learning Representations*, 2018.

[70] Q. Huang, H. He, A. Singh, S.-N. Lim, and A. Benson, "Combining label propagation and simple models out-performs graph neural networks," in *International Conference on Learning Representations*, 2020.

[71] C. Sun, H. Gu, and J. Hu, "Scalable and adaptive graph neural networks with self-label-enhanced training," *arXiv preprint arXiv:2104.09376*, 2021.

[72] W. Zhang, Z. Yin, Z. Sheng, Y. Li, W. Ouyang, X. Li, Y. Tao, Z. Yang, and B. Cui, "Graph attention multi-layer perceptron," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4560–4570.

[73] W. Feng, Y. Dong, T. Huang, Z. Yin, X. Cheng, E. Kharlamov, and J. Tang, "Grand+: Scalable graph random neural networks," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 3248–3258.

[74] K. Huang, J. Tang, J. Liu, R. Yang, and X. Xiao, "Node-wise diffusion for scalable graph learning," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 1723–1733.

[75] N. Liao, D. Mo, S. Luo, X. Li, and P. Yin, "Scalable decoupling graph neural network with feature-oriented optimization," *The VLDB Journal*, vol. 33, no. 3, pp. 667–683, 2024.

[76] G. Karypis and V. Kumar, "Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997.

[77] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[78] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International conference on machine learning*, 2018.

[79] B. Yan, C. Wang, G. Guo, and Y. Lou, "Tinygnn: Learning efficient graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1848–1856.

[80] X. Deng and Z. Zhang, "Graph-free knowledge distillation for graph neural networks," in *IJCAI*, 2021.

[81] C. K. Joshi, F. Liu, X. Xun, J. Lin, and C. S. Foo, "On representation knowledge distillation for graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[82] C. Yang, Q. Wu, and J. Yan, "Geometric knowledge distillation: Topology compression for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 35, pp. 29 761–29 775, 2022.

[83] R. He, S. Sun, J. Yang, S. Bai, and X. Qi, "Knowledge distillation as efficient pre-training: Faster convergence, higher data-efficiency, and better transferability," in *CVPR*, 2022.

[84] H. He, J. Wang, Z. Zhang, and F. Wu, "Compressing deep graph neural networks via adversarial knowledge distillation," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 534–544.

[85] Y. Hu, H. You, Z. Wang, Z. Wang, E. Zhou, and Y. Gao, "Graph-mlp: Node classification without message passing in graph," *arXiv preprint arXiv:2106.04051*, 2021.

[86] S. Zhang, Y. Liu, Y. Sun, and N. Shah, "Graph-less neural networks: Teaching old mlps new tricks via distillation," in *International Conference on Learning Representations*, 2021.

[87] C. Yang, J. Liu, and C. Shi, "Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework," in *Proceedings of the web conference 2021*, 2021, pp. 1227–1237.

[88] Y. Tian, C. Zhang, Z. Guo, X. Zhang, and N. Chawla, "Learning mlps on graphs: A unified view of effectiveness, robustness, and efficiency," in *The Eleventh International Conference on Learning Representations*, 2022.

[89] L. Wu, H. Lin, Y. Huang, and S. Z. Li, "Quantifying the knowledge in gnns for reliable distillation into mlps," in *International Conference on Machine Learning*. PMLR, 2023, pp. 37 571–37 581.

[90] Y.-M. Shin and W.-Y. Shin, "Propagate & distill: Towards effective graph learners using propagation-embracing mlps," in *The Second Learning on Graphs Conference*, 2023.

[91] L. Yang, Y. Tian, M. Xu, Z. Liu, S. Hong, W. Qu, W. Zhang, B. Cui, M. Zhang, and J. Leskovec, "Vqgraph: Graph vector-quantization for bridging gnns and mlps," in *International Conference on Learning Representations*, 2024.

[92] Y. Zhao, D. Wang, D. Bates, R. Mullins, M. Jamnik, and P. Lio, "Learned low precision graph neural networks," *arXiv preprint arXiv:2009.09232*, 2020.

[93] B. Feng, Y. Wang, X. Li, S. Yang, X. Peng, and Y. Ding, "Sgquant: Squeezing the last bit on graph neural networks with specialized quantization," in *2020 IEEE 32nd international conference on tools with artificial intelligence (ICTAI)*. IEEE, 2020, pp. 1044–1052.

[94] M. Ding, K. Kong, J. Li, C. Zhu, J. Dickerson, F. Huang, and T. Goldstein, "Vq-gnn: A universal framework to scale up graph neural networks using vector quantization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 6733–6746, 2021.

[95] Z. Zhu, F. Li, Z. Mo, Q. Hu, G. Li, Z. Liu, X. Liang, and J. Cheng, "A^2Q: Aggregation-aware quantization for graph neural networks," in *International Conference on Learning Representations*, 2022.

[96] L. Huang, Z. Zhang, Z. Du, S. Li, H. Zheng, Y. Xie, and N. Tan, "Epquant: A graph neural network compression approach based on product quantization," *Neurocomputing*, vol. 503, pp. 49–61, 2022.

[97] Y. Guo, Y. Chen, X. Zou, X. Yang, and Y. Gu, "Algorithms and architecture support of degree-based quantization for graph neural networks," *Journal of Systems Architecture*, vol. 129, p. 102578, 2022.

[98] S. Wang, B. Eravci, R. Guliyev, and H. Ferhatosmanoglu, "Low-bit quantization for deep graph neural networks with smoothness-aware message propagation," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023, pp. 2626–2636.

[99] M. Eliasof, B. J. Bodner, and E. Treister, "Haar wavelet feature compression for quantized graph convolutional networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[100] Z. Zhu, F. Li, G. Li, Z. Liu, Z. Mo, Q. Hu, X. Liang, and J. Cheng, "Mega: A memory-efficient gnn accelerator exploiting degree-aware mixed-precision quantization," *HPCA*, 2024.

[101] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, "A unified lottery ticket hypothesis for graph neural networks," in *International conference on machine learning*. PMLR, 2021, pp. 1695–1706.

[102] Y. Sui, X. Wang, T. Chen, X. He, and T.-S. Chua, "Inductive lottery ticket learning for graph neural networks," 2021.

[103] H. You, Z. Lu, Z. Zhou, Y. Fu, and Y. Lin, "Early-bird gcns: Graph-network co-optimization towards more efficient gcn training and inference via drawing early-bird lottery tickets," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8910–8918.

[104] K. Wang, Y. Liang, P. Wang, X. Wang, P. Gu, J. Fang, and Y. Wang, "Searching lottery tickets in graph neural networks: A dual perspective," in *The Eleventh International Conference on Learning Representations*, 2022.

[105] B. Hui, D. Yan, X. Ma, and W.-S. Ku, "Rethinking graph lottery tickets: Graph sparsity matters," in *International Conference on Learning Representations*, 2022.

[106] K. Wang, G. Li, S. Wang, G. Zhang, K. Wang, Y. You, X. Peng, Y. Liang, and Y. Wang, "The snowflake hypothesis: Training deep gnn with one node one receptive field," *arXiv preprint arXiv:2308.10051*, 2023.

[107] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, 2021.

[108] P. Chen, S. Liu, H. Zhao, and J. Jia, "Distilling knowledge via knowledge review," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5008–5017.

[109] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[110] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[111] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*, 2022.

[112] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks," *arXiv preprint arXiv:2205.07877*, 2022.

[113] O. Weng, "Neural network quantization for efficient inference: A survey," *arXiv preprint arXiv:2112.06126*, 2021.

[114] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 8612–8620.

[115] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8bert: Quantized 8bit bert," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 2019, pp. 36–39.

[116] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[117] W. Chen, P. Wang, and J. Cheng, "Towards mixed-precision quantization of neural networks via constrained optimization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5350–5359.

[118] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," *Neural information processing systems*, 2018.

[119] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," *Neural Information Processing Systems*, 2018.

[120] Z. Liu, K. Zhou, F. Yang, L. Li, R. Chen, and X. Hu, "Exact: Scalable graph neural networks training via extreme activation compression," in *International Conference on Learning Representations*, 2021.

[121] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.

[122] S. Vadera and S. Ameen, "Methods for pruning deep neural networks," *IEEE Access*, vol. 10, pp. 63 280–63 300, 2022.

[123] S. Xu, A. Huang, L. Chen, and B. Zhang, "Convolutional neural network pruning: A survey," in *2020 39th Chinese Control Conference (CCC)*. IEEE, 2020, pp. 7458–7463.

[124] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks." in *International Conference on Learning Representations*, 2019.

[125] H. Wang, D. Lian, Y. Zhang, L. Qin, X. He, Y. Lin, and X. Lin, "Binarized graph neural network," *World Wide Web*, vol. 24, pp. 825–848, 2021.

[126] M. Bahri, G. Bahl, and S. Zafeiriou, "Binary graph neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 9492–9501.

[127] Y. Jing, Y. Yang, X. Wang, M. Song, and D. Tao, "Meta-aggregator: Learning to aggregate for 1-bit graph neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5301–5310.

[128] J.-A. Chen, H.-H. Sung, X. Shen, S. Choudhury, and A. Li, "Bitgnn: Unleashing the performance potential of binary graph neural networks on gpus," in *Proceedings of the 37th International Conference on Supercomputing*, 2023, pp. 264–276.

[129] 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?" *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[130] X. Gao, T. Chen, Y. Zang, W. Zhang, Q. V. H. Nguyen, K. Zheng, and H. Yin, "Graph condensation for inductive node representation learning," in *IEEE International Conference on Data Engineering*, 2024.

[131] T. Zhang, Y. Zhang, B. Yang, K. Wang, T. Li, K. Zhang, W. Shao, P. Liu, J. T. Zhou, and Y. You, "Ctrl: Graph condensation via crafting rational trajectory matching," 2023.

[132] W. Jin, X. Tang, H. Jiang, Z. Li, D. Zhang, J. Tang, and B. Yin, "Condensing graphs via one-step gradient matching," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 720–730.

[133] J. Gao, J. Wu, and J. Ding, "Heterogeneous graph condensation," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[134] J. Fang, X. Li, Y. Sui, Y. Gao, G. Zhang, K. Wang, X. Wang, and X. He, "Exgc: Bridging efficiency and explainability in graph condensation," in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 721–732.

[135] 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.

[136] Y. Zhang, T. Zhang, K. Wang, Z. Guo, Y. Liang, X. Bresson, W. Jin, and Y. You, "Navigating complexity: Toward lossless graph condensation via expanding window matching," *arXiv preprint arXiv:2402.05011*, 2024.

[137] Z. Xu, Y. Chen, M. Pan, H. Chen, M. Das, H. Yang, and H. Tong, "Kernel ridge regression-based graph dataset distillation," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 2850–2861.

[138] L. Wang, W. Fan, J. Li, Y. Ma, and Q. Li, "Fast graph condensation with structure-based neural tangent kernel," *arXiv preprint arXiv:2310.11046*, 2023.

[139] X. Gao, T. Chen, W. Zhang, Y. Li, X. Sun, and H. Yin, "Graph condensation for open-world graph learning," *arXiv preprint arXiv:2405.17003*, 2024.

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

[141] Y. Liu, R. Qiu, and Z. Huang, "Cat: Balanced continual graph learning with graph condensation," in *2023 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2023, pp. 1157–1162.

[142] Y. Liu, D. Bo, and C. Shi, "Graph distillation with eigenbasis matching," in *Forty-first International Conference on Machine Learning*.

[143] X. Gao, T. Chen, W. Zhang, J. Yu, G. Ye, Q. V. H. Nguyen, and H. Yin, "Rethinking and accelerating graph condensation: A training-free approach with class partition," *arXiv preprint arXiv:2405.13707*, 2024.

[144] Z. Xiao, S. Liu, Y. Wang, T. Zheng, and M. Song, "Disentangled condensation for large-scale graphs," *arXiv preprint arXiv:2401.12231*, 2024.

[145] Z. Xiao, Y. Wang, S. Liu, H. Wang, M. Song, and T. Zheng, "Simple graph condensation," *arXiv preprint arXiv:2403.14951*, 2024.

[146] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, 2020.

[147] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artificial Intelligence Review*, vol. 53, pp. 5113–5155, 2020.

[148] C. Yuan and S. S. Agaian, "A comprehensive review of binary neural network," *Artificial Intelligence Review*, vol. 56, no. 11, pp. 12 949–13 013, 2023.

[149] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[150] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 2008, pp. 563–568.

[151] A. Loukas and P. Vandergheynst, "Spectrally approximating large graphs with smaller graphs," in *International conference on machine learning*. PMLR, 2018, pp. 3237–3246.

[152] T. Nguyen, Z. Chen, and J. Lee, "Dataset meta-learning from kernel ridge-regression," *arXiv preprint arXiv:2011.00050*, 2020.

[153] B. M. Oloulade, J. Gao, J. Chen, T. Lyu, and R. Al-Sabri, "Graph neural architecture search: A survey," *Tsinghua Science and Technology*, vol. 27, no. 4, pp. 692–708, 2021.

[154] F. G. Febrinanto, F. Xia, K. Moore, C. Thapa, and C. Aggarwal, "Graph lifelong learning: A survey," *IEEE Computational Intelligence Magazine*, vol. 18, no. 1, pp. 32–51, 2023.

[155] X. Fu, B. Zhang, Y. Dong, C. Chen, and J. Li, "Federated graph machine learning: A survey of concepts, techniques, and applications," *ACM SIGKDD Explorations Newsletter*, vol. 24, no. 2, pp. 32–47, 2022.

[156] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[157] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai *et al.*, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[158] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai, "{NeuGraph}: Parallel deep neural network computation on large graphs," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 443–458.

[159] J. Jiang, P. Xiao, L. Yu, X. Li, J. Cheng, X. Miao, Z. Zhang, and B. Cui, in *IEEE International Conference on Data Engineering*, 2020.

[160] H. Yang, "Aligraph: A comprehensive graph neural network platform," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 3165–3166.

[161] Y. Li, Y. Shen, W. Zhang, Y. Chen, H. Jiang, M. Liu, J. Jiang, J. Gao, W. Wu, Z. Yang *et al.*, "Openbox: A generalized black-box optimization service," in *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2021, pp. 3209–3219.

[162] S. Gupta and S. Bedathur, "A survey on temporal graph representation learning and generative modeling," *arXiv preprint arXiv:2208.12126*, 2022.

[163] A. Longa, V. Lachi, G. Santin, M. Bianchini, B. Lepri, P. Lio, F. Scarselli, and A. Passerini, "Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities," *arXiv preprint arXiv:2302.01018*, 2023.

[164] R. Bing, G. Yuan, M. Zhu, F. Meng, H. Ma, and S. Qiao, "Heterogeneous graph neural networks analysis: a survey of techniques, evaluations and applications," *Artificial Intelligence Review*, 2023.

[165] Z. Feng, R. Wang, T. Wang, M. Song, S. Wu, and S. He, "A comprehensive survey of dynamic graph neural networks: Models, frameworks, benchmarks, experiments and challenges," *arXiv preprint arXiv:2405.00476*, 2024.

[166] 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.

[167] C. Yang, D. Bo, J. Liu, Y. Peng, B. Chen, H. Dai, A. Sun, Y. Yu, Y. Xiao, Q. Zhang *et al.*, "Data-centric graph learning: A survey," *arXiv preprint arXiv:2310.04987*, 2023.

[168] 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 preprint arXiv:2103.03036*, 2021.

[169] Y. Liu, Y. Zheng, D. Zhang, H. Chen, H. Peng, and S. Pan, "Towards unsupervised deep graph structure learning," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1392–1403.

[170] W. Li, J. Wang, Y. Gao, M. Zhang, R. Tao, and B. Zhang, "Graph-feature-enhanced selective assignment network for hyperspectral and multispectral data classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–14, 2022.

[171] Y. Liu, M. Jin, S. Pan, C. Zhou, Y. Zheng, F. Xia, and S. Y. Philip, "Graph self-supervised learning: A survey," *IEEE transactions on knowledge and data engineering*, vol. 35, no. 6, pp. 5879–5900, 2022.

[172] Y. Xie, Z. Xu, J. Zhang, Z. Wang, and S. Ji, "Self-supervised learning of graph neural networks: A unified review," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 2, pp. 2412–2429, 2022.

[173] L. Wu, H. Lin, C. Tan, Z. Gao, and S. Z. Li, "Self-supervised learning on graphs: Contrastive, generative, or predictive," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 4216–4235, 2021.

[174] H. Wang, T. Fu, Y. Du, W. Gao, K. Huang, Z. Liu, P. Chandak, S. Liu, P. Van Katwyk, A. Deac *et al.*, "Scientific discovery in the age of artificial intelligence," *Nature*, vol. 620, no. 7972, pp. 47–60, 2023.

[175] Y. Shao, H. Li, X. Gu, H. Yin, Y. Li, X. Miao, W. Zhang, B. Cui, and L. Chen, "Distributed graph neural network training: A survey," *ACM Computing Surveys*, vol. 56, no. 8, pp. 1–39, 2024.

[176] Q. W. Khan, A. N. Khan, A. Rizwan, R. Ahmad, S. Khan, and D.-H. Kim, "Decentralized machine learning training: a survey on synchronization, consolidation, and topologies," *IEEE Access*, vol. 11, pp. 68 031–68 050, 2023.

[177] Y. Park, S. Min, and J. W. Lee, "Ginex: Ssd-enabled billion-scale graph neural network training on a single machine via provably optimal in-memory caching," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2626–2639, 2022.

[178] R. Waleffe, J. Mohoney, T. Rekatsinas, and S. Venkataraman, "Marius-gnn: Resource-efficient out-of-core training of graph neural networks," in *Proceedings of the Eighteenth European Conference on Computer Systems*, 2023, pp. 144–161.

[179] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang, "Graph unlearning," in *Proceedings of the 2022 ACM SIGSAC conference on computer and communications security*, 2022, pp. 499–513.

[180] Z. Tian, D. Zhang, and H.-N. Dai, "Continual learning on graphs: A survey," *arXiv preprint arXiv:2402.06330*, 2024.

[181] E. Chien, C. Pan, and O. Milenkovic, "Certified graph unlearning," *arXiv preprint arXiv:2206.09140*, 2022.

Lu Ma is expected to receive his B.S from the School of Information at Renmin University of China in May 2024. He is expected to enroll in the School of Electronic and Computer Engineering at Peking University in September 2024, advised by Prof. Bin Cui. His research interest is scalable graph learning and data-centric ML.

Zeang Sheng is currently working toward the PhD degree with the school of Computer Science, Peking University, advised by Prof. Bin Cui. He received the BS degree in computer science from Peking University in 2022. His research interest lies in efficient and scalable graph learning from both algorithm and system designs. He has published 5+ papers in top conferences such as SIGKDD, NeurIPS and ICML.

Xunkai Li is currently working toward the PhD degree with the school of Computer Science, Beijing Institute of Technology, advised by Prof. Rong-hua Li. He received the BS degree in computer science from Shandong University in 2022. His research interest lies in general Data-centric ML and Graph-ML in complex scenarios (directed, signed, hypergraph, and distributed). He has publishe 5+ papers in top DB/DM/AI conferences such as VLDB, WWW, AAAI.

Xinyi Gao is presently pursuing a Ph.D. in Electrical Engineering and Computer Science at The University of Queensland, under the guidance of Prof. Hongzhi Yin. He earned his Bachelor's and Master's degrees in Information and Communications Engineering from Xi'an Jiaotong University, China. Xinyi's research is focused on advancing efficient graph representation learning, encompassing areas such as graph condensation, scalable graph neural networks, and inference acceleration.

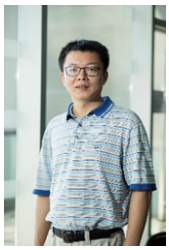
Zhezheng Hao is currently working toward the M.S. degree with the School of Artificial Intelligence, Optics and ElectroNics (iOPEN) at Northwestern Polytechnical University, advised by Prof. Feiping Nie. His current research interests include graph learning and data mining. He has published 5+ papers about graph learning in top journals and conferences such as TKDE, WWW, AAAI.

Ling Yang is currently working toward the PhD degree in Peking University (PKU), Beijing, China. His research interests include graph machine learning and diffusion models. He has published 10+ papers in top journals and conferences such as ICML, ICLR, NeurIPS, AAAI, TKDE, CVPR. He serves as a reviewer for TPAMI, NeurIPS, ICML, ICLR, CVPR, KDD, AAAI.

Xiaonan Nie received the PhD degree in computer science from Peking University, China, in 2024. His research interests include distributed deep learning systems, heterogeneous computing, and sparse neural networks.

Jiawei Jiang received his PhD degree in computer science from Peking University, China, in 2018. He is currently a professor with the School of Computer Science, Wuhan University, China. His research interests include database, big data management and analytics, and machine learning systems.

Wentao Zhang is an assistant professor in the Center for Machine Learning Research at Peking University (PKU). Wentao's research focuses on Graph ML, Data-centric ML, ML systems and AI4Science. Wentao has published 40+ papers, including 10+ first author papers in the top DB (SIGMOD, VLDB, ICDE), DM (KDD, WWW), and ML (ICML, NeurIPS, ICLR) venues. Wentao is the contributor or designer of several system projects, including Angel, SGL, and OpenBox. His research works have been powering several billion-scale applications in Tencent, and some of them have been recognized by multiple best paper awards, including the Best Paper Runner Up Award at APWeb-WAIM 2023, and the Best Student Paper Award at WWW 2022.



Bin Cui is a Professor in the School of CS at Peking University. He is a fellow of IEEE (2024) and Cheung Kong distinguished Professor (2016). His research interests include database system, big data management and analytics, ML/DL system. Prof. Cui has published more than 100 research papers and has served in the Technical Program Committee of various international conferences, including SIGMOD, VLDB, ICDE, and KDD, and as General Co-Chair of VLDB 2024, PC Co-Chair of APWeb 2015, WAIM 2016, DASFAA 2020 and DSS 2022, Area Chair of ICDE 2011&2018, Demo Co-Chair of ICDE 2014, Area Chair of VLDB 2014. He was a recipient of VLDB 2022 SDS Best paper award, WWW 2022 Best student paper award, and NSR 2016 best paper award. He was awarded Second Prize of Natural Science Award of MOE China (2014), CCF Young Scientist award (2009), and Microsoft Young Professorship award (MSRA 2008).