

# A Teacher-Free Graph Knowledge Distillation Framework With Dual Self-Distillation

Lirong Wu<sup>✉</sup>, Haitao Lin<sup>✉</sup>, Zhangyang Gao<sup>✉</sup>, Guojian Zhao<sup>✉</sup>, and Stan Z. Li<sup>✉</sup>, *Fellow, IEEE*

**Abstract**—Recent years have witnessed great success in handling graph-related tasks with Graph Neural Networks (GNNs). Despite their great *academic* success, Multi-Layer Perceptrons (MLPs) remain the primary workhorse for practical *industrial* applications. One reason for such an academic-industry gap is the neighborhood-fetching latency incurred by data dependency in GNNs. To reduce their gaps, Graph Knowledge Distillation (GKD) is proposed, usually based on a standard teacher-student architecture, to distill knowledge from a large teacher GNN into a lightweight student GNN or MLP. However, we found in this paper that neither teachers nor GNNs are necessary for graph knowledge distillation. We propose a *Teacher-Free Graph Self-Distillation* (TGS) framework that does not require any teacher model or GNNs during both training and inference. More importantly, the proposed TGS framework is purely based on MLPs, where structural information is only implicitly used to guide *dual knowledge self-distillation* between the target node and its neighborhood. As a result, TGS enjoys the benefits of graph topology awareness in training but is free from data dependency in inference. Extensive experiments have shown that the performance of vanilla MLPs can be greatly improved with dual self-distillation, e.g., TGS improves over vanilla MLPs by 15.54% on average and outperforms state-of-the-art GKD algorithms on six real-world datasets. In terms of inference speed, TGS infers  $75\times$ – $89\times$  faster than existing GNNs and  $16\times$ – $25\times$  faster than classical inference acceleration methods.

**Index Terms**—Graph neural networks, graph knowledge distillation, inference acceleration.

## I. INTRODUCTION

RECENTLY, Graph Neural Networks (GNNs) [1], [2], [3], [4], [5] have demonstrated their powerful capability to handle graph-structured data in a variety of applications, including meteorology [6], [7], life sciences [8], [9], and transportation [10], [11]. Despite their great academic success, practical deployments of GNNs in the industry are still less popular. One reason for their gaps is the neighborhood-fetching inference latency incurred by data dependency in GNNs [12]. Most existing GNNs [13], [14], [15] rely on message passing to

aggregate neighborhood features for capturing data dependency between nodes within a local subgraph. As a result, neighborhood fetching caused by data dependency is one of the major sources of GNN latency during inference. For example, to infer a single node with a  $L$ -layer GNN on a graph with average node degree  $R$ , it requires fetching and aggregating  $\mathcal{O}(R^L)$  fetched nodes. However,  $R$  can be large for real-world graphs, e.g., 19 for the Amazon-com dataset, and  $L$  is getting deeper for the latest GNN architectures, e.g.,  $L=1001$  layers for RevGNN-Deep [16]. Compared with GNNs, MLPs do not suffer from the data dependency problem and infer much faster than GNNs. However, due to the lack of modeling data dependencies, MLPs may fail to take full advantage of the graph topological information, which limits their performance on various downstream tasks.

To connect the two worlds of topology-aware GNNs and inference-efficient MLPs, graph knowledge distillation [17], [18], [19], [20], [21] is proposed to distill knowledge from large teacher GNNs to small student GNNs or MLPs, yielding two branches, namely GNN-to-GNN [22], [23], [24] and GNN-to-MLP [25], [26], [27], [28], [29], as shown in Fig. 1(a). Different from the standard teacher-student KD, there is a similar technique called GNN Self-Distillation [30], [31], which regularizes a GNN model by *distilling its own knowledge within GNNs* (e.g., across different layers or nodes) without other teacher models. Similarly, MLP Self-Distillation [32], [33] is proposed to implicitly self-distill knowledge from structural information, but it does not involve any teachers or GNNs and is purely based on MLPs.

Considering both inference accuracy and inference time, the above four types of graph knowledge distillation methods are two completely different worlds. As shown in Fig. 1(b), GNN-to-GNN KD and GNN Self-Distillation directly deploy well-distilled GNNs for inference, which helps to improve the inference accuracy but does not help much in inference acceleration with the data dependency in GNNs unresolved. On the other hand, GNN-to-MLP KD and MLP Self-Distillation require only MLPs for inference, which greatly improves their inference speed, but brings limited performance improvement. In particular, MLP Self-Distillation completely removes GNNs, which disables it from being topology-aware, making its inference fastest, but at the cost of undesired performance drops.

In this paper, we found that neither teachers nor GNNs are necessary to achieve *high-accuracy* and *high-efficiency* inference on graphs. Therefore, we propose a simple *Teacher-Free Graph Self-Distillation* (TGS) framework that does not require

Manuscript received 31 August 2023; revised 3 March 2024; accepted 4 March 2024. Date of publication 20 March 2024; date of current version 7 August 2024. This work was supported in part by the National Science and Technology Major Project under Grant 2022ZD0115101, and in part by the National Natural Science Foundation of China under Grant U21A20427 and Grant WU2022A009 from the Center of Synthetic Biology and Integrated Bioengineering of Westlake University. Recommended for acceptance by M.A. Cheema. (Corresponding author: Stan Z. Li.)

The authors are with the AI Lab, Research Center for Industries of the Future, Westlake University, Hangzhou 310000, China (e-mail: wulirong@westlake.edu.cn; linhaitao@westlake.edu.cn; gaozhangyang@westlake.edu.cn; guojianz@alummi.cmu.edu; stan.zq.li@westlake.edu.cn).

Digital Object Identifier 10.1109/TKDE.2024.3374773

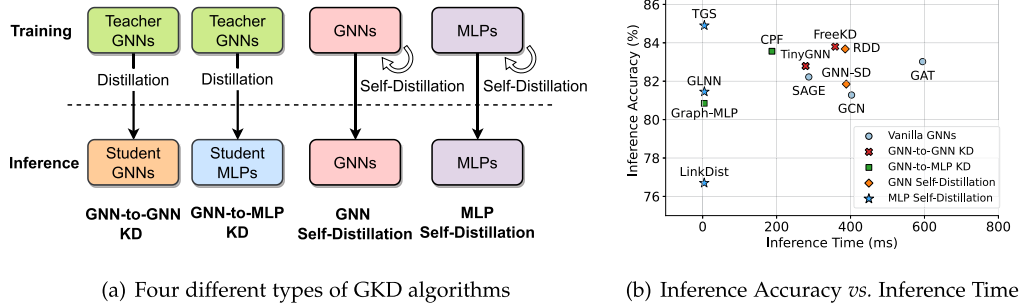


Fig. 1. (a): Illustration of four different types of graph knowledge distillation algorithms, depending on whether teacher models and GNNs/MLPs are included in training and inference. (b): Inference accuracy (%) versus inference time (ms) on the Cora dataset. If not specifically mentioned, all GKD algorithms adopt GCN as the backbone by default.

any teacher model or GNNs during both training and inference. The proposed TGS framework is purely based on MLPs, where structural information is only implicitly used to guide *dual knowledge self-distillation* between the target node and its neighborhood, substituting the explicit information propagation as in GNNs. As a result, the resulting model enjoys the benefits of graph topology-awareness in training but reduces time overhead in inference. Moreover, we propose an edge sampling strategy for batch-style self-distillation instead of feeding the entire graph into memory to reduce memory usage when scaling to large-scale graphs. Extensive experiments show that TGS infers as fast as MLPs, but its inference accuracy is comparable to or even better than state-of-the-art GKD algorithms. We believe that this work will inspire researchers to rethink the necessity of teachers and GNNs for graph knowledge distillation. Codes will be public at <https://github.com/LirongWu/TGS>.

## II. RELATED WORK

### A. Graph Neural Networks

Graph Neural Networks (GNNs) can be mainly divided into two categories, i.e., spectral-based GNNs and spatial-based GNNs. The spectral-based GNNs, such as ChebyNet [34] and GCN [35], define graph convolution kernels in the spectral domain based on the graph signal processing theory. Instead, the spatial-based GNNs, such as GraphSAGE [15] and GAT [35], directly define updating rules in the spatial space and focus on the design of neighborhood aggregation functions. The closest work to ours is APPNP [36], which shares some similar design insights with TGS in that both first use graph-independent models to extract node embeddings. However, their main difference is that APPNP applies Personalized PageRank to *explicitly* encode the adjacency matrix via message passing, while TGS uses Mixup and two inference layers to *implicitly* exploit the adjacency matrix as supervision signals. We refer interested readers to the recent survey [1], [37] for more GNN architectures. Despite their great progress, the above GNNs all share the de facto design that structural information is explicitly used for message passing, which leaves neighborhood fetching still one major source of GNN inference latency. To reduce multiplication and

accumulation operations, many *inference acceleration* technologies have been applied, including Pruning [38], Quantizing [39], and Neighborhood Sampling [40].

### B. Graph Knowledge Distillation

Several previous works on graph distillation try to distill knowledge from large teacher GNNs to smaller student GNNs, termed as *GNN-to-GNN KD* [41]. The student model in FreeKD [42] and TinyGNN [17] is a GNN with fewer parameters, but not necessarily fewer layers, which makes them still suffer from the neighborhood-fetching latency. The other branch of graph knowledge distillation is to distill from large teacher GNNs to lightweight student MLPs, termed as *GNN-to-MLP KD*. For example, CPF [26] proposes to distill knowledge from teacher GNNs to student MLPs, but it takes advantage of label propagation [43] in MLPs to improve performance and thus remains heavily data-dependent. Besides, GLNN [25] directly distills knowledge from GNNs to student MLPs, which has a great advantage in inference speed, but its performance gains are limited.

Different from the standard teacher-student KD architecture, there is a similar technique known as *GNN Self-Distillation* [44], which regularizes a GNN model by distilling its own knowledge without other teacher models. For example, GNN-SD [30] directly self-distills knowledge across different GNN layers. Besides, RDD [31] builds an ensemble teacher using multiple versions of the model itself without introducing an external teacher model. Compared to GNNs, MLPs have no data dependency and infer much faster. There are some attempts on *MLP Self-Distillation*, such as Graph-MLP [32] and LinkDist [33], which are purely based on MLP and self-distill knowledge from structural information by contrastive or consistency constraints. Despite the great advantage of MLP Self-Distillation in inference speed, their inference accuracy still cannot match the state-of-the-art competitors. More related work on graph knowledge distillation can be found in a recent survey [45]. The closest work to ours is Graph-MLP, but the essential difference between Graph-MLP and TGS is that the former is a contrastive learning method, while the latter is based on knowledge distillation. Their main differences are in the following four aspects:

(1) TGS utilizes mixup as data augmentation, while Graph-MLP [32], as a contrastive learning method, instead discards this; (2) Graph-MLP treats all nodes within an  $r$ -hop neighborhood as positive samples with  $\mathcal{O}(R^r)$  complexity, whereas TGS performs distillation only within a 1-hop neighborhood with a complexity of  $\mathcal{O}(R)$ , where  $R$  is the average node degree; (3) Graph-MLP treats all nodes within a batch as negative samples, while TGS only takes one randomly selected node as negative sample; and (4) as a contrastive learning method, Graph-MLP relies on InfoNCE to perform contrasting, whereas TGS directly minimizes the MSE losses between samples.

### C. Graph Contrastive Learning (GCL)

Recent years have witnessed the great success of graph contrastive learning in learning graph representation [46]. Graph contrastive learning generates multiple views for each instance through various data augmentation methods and maximizes the agreement between two positive samples (as measured by mutual information) against a large number of negative samples. For example, Deep Graph Infomax (DGI) [47] is proposed to contrast the patch representations and corresponding high-level summary of graphs. GraphCL [48] is one of the pioneering works that extends contrastive learning to graphs, and it defines four graph augmentation methods that achieve good results on the molecular property prediction task. Its follow-up, GCA [49], combines adaptive augmentation, which further improves the performance. Furthermore, InfoGCL [50] investigates how information is transformed and transferred during contrastive learning based on the information bottleneck, aimed at minimizing the loss of task-relevant information. There has been some recent work exploring the necessity of negative samples for graph contrastive learning. Inspired by BYOL [51], BGRL [52] proposes to perform the self-supervised learning that *does not require negative samples*, thus getting rid of the potentially quadratic bottleneck. Moreover, in addition to the commonly used InfoNCE [53] as a loss function for measuring agreement, MSE is also widely used in GCL, such as SelfGNN [54], DMGI [55], etc.

### III. PRELIMINARIES

**Notations and Problem Statement:** Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  denote a graph, where  $\mathcal{V}$  is the set of  $|\mathcal{V}| = N$  nodes with features  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N] \in \mathbb{R}^{N \times d}$  and  $\mathcal{E}$  is the set of edges between nodes. Each node  $v_i \in \mathcal{V}$  is associated with a  $d$ -dimensional features vector  $\mathbf{x}_i$ . Following the common semi-supervised node classification setting, only a subset of node  $\mathcal{V}_L = \{v_1, v_2, \dots, v_L\}$  with corresponding labels  $\mathcal{Y}_L = \{y_1, y_2, \dots, y_L\}$  are known, and we denote the labeled set as  $\mathcal{D}_L = (\mathcal{V}_L, \mathcal{Y}_L)$  and unlabeled set as  $\mathcal{D}_U = (\mathcal{V}_U, \mathcal{Y}_U)$ , where  $\mathcal{V}_U = \mathcal{V} \setminus \mathcal{V}_L$ . The task of node classification aims to learn a mapping  $\Phi: \mathcal{V} \rightarrow \mathcal{Y}$  on labeled data  $\mathcal{D}_L$ , so that it can be used to infer the labels  $\mathcal{Y}_U$  [56].

**Graph Neural Networks:** A general GNN framework consists of two key computations for each node  $v_i$  at every layer: (1)

AGGREGATE operation: aggregating messages from neighborhood  $\mathcal{N}_i$ ; (2) UPDATE operation: updating node representation from its representation in the previous layer and aggregated messages. Considering a  $L$ -layer GNN, the formulation of the  $l$ -th layer is as follows,

$$\begin{aligned} \mathbf{m}_i^{(l)} &= \text{AGGREGATE}^{(l)} \left( \left\{ \mathbf{h}_j^{(l-1)} : v_j \in \mathcal{N}_i \right\} \right) \\ \mathbf{h}_i^{(l)} &= \text{UPDATE}^{(l)} \left( \mathbf{h}_i^{(l-1)}, \mathbf{m}_i^{(l)} \right) \end{aligned} \quad (1)$$

where  $0 \leq l \leq L-1$ ,  $\mathbf{h}_i^{(l)}$  is the embedding of node  $v_i$  in the  $l$ -th layer, and  $\mathbf{h}_i^{(0)} = \mathbf{x}_i$  is the input feature. After  $L$  message-passing layers, the final node embeddings  $\mathbf{h}_i^{(L)}$  can be passed through an additional linear inference layer  $\mathbf{y}_i = f_\theta(\mathbf{h}_i^{(L)})$  for node classification on the target node  $v_i$ .

### IV. METHODOLOGY

Motivated by the complementary strengths and weaknesses of GNNs and MLPs, we propose in this paper a simple but effective *Teacher-Free Graph Self-Distillation* (TGS) framework, as illustrated in Fig. 2. The proposed TGS framework enjoys the benefits of GNN-like topology-awareness in training but keeps the inference-efficiency of MLPs in inference. The following subsections focus on three key aspects: (1) backbone architecture, how to construct a “boosted” MLP as backbone; (2) dual self-distillation, how to self-distill knowledge between the target node and its neighborhood; (3) training and inference, how to solve training difficulties and make inference with the trained model.

#### A. Backbone Architecture

The TGS framework is based on pure MLPs, with each layer composed of a linear transformation, an activation function, a batch normalization, and a dropout function, defined as:

$$\mathbf{H}^{(l+1)} = \text{Dropout} \left( \text{BN} \left( \sigma \left( \mathbf{H}^{(l)} \mathbf{W}^{(l)} \right) \right) \right), \quad \mathbf{H}^{(0)} = \mathbf{X} \quad (2)$$

where  $0 \leq l \leq L-1$ ,  $\sigma = \text{ReLU}(\cdot)$  denotes an activation function,  $\text{BN}(\cdot)$  denotes the batch normalization, and  $\text{Dropout}(\cdot)$  is the dropout function. In addition,  $\mathbf{W}^{(0)} \in \mathbb{R}^{d \times F}$  and  $\mathbf{W}^{(l)} \in \mathbb{R}^{F \times F}$  ( $1 \leq l \leq L-1$ ) are layer-specific parameter matrices with the hidden dimension  $F$ .

Given a target node  $v_i$  and its neighboring nodes  $\mathcal{N}_i$ , we first feed their node features  $\mathbf{x}_i$  and  $\{\mathbf{x}_j\}_{j \in \mathcal{N}_i}$  into MLPs and encode them as hidden representations  $\mathbf{h}_i^{(L)}$  and  $\{\mathbf{h}_j^{(L)}\}_{j \in \mathcal{N}_i}$ . Then, we define two parameter-independent inference layers for label prediction, respectively, i.e.,  $\mathbf{y}_i = f_\theta(\mathbf{h}_i^{(L)}) \in \mathbb{R}^C$  and  $\mathbf{z}_j = g_\gamma(\mathbf{h}_j^{(L)}) \in \mathbb{R}^C$ , where  $C$  is the number of category. Both two inference layers are implemented with one layer of linear transformation by default in this paper. In the next subsections, we will discuss in detail how to implicitly self-distill graph knowledge between the target node  $v_i$  and its neighborhood nodes  $\mathcal{N}_i$ .



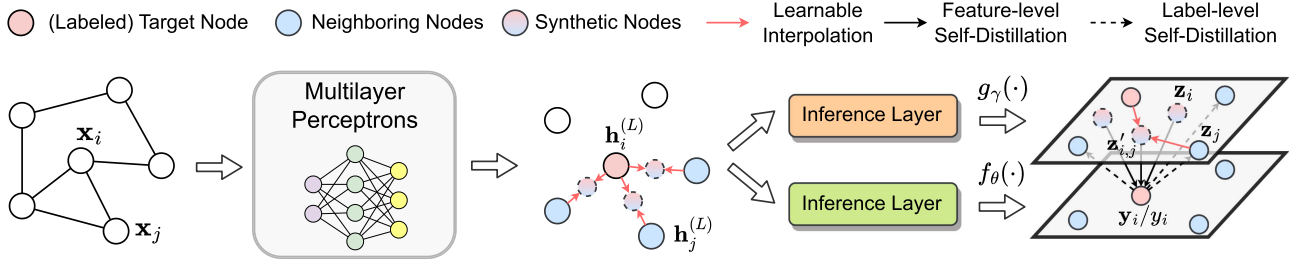


Fig. 2. Illustration of the proposed *Teacher-Free Graph Self-Distillation* (TGS) framework. In the training stage, the MLP and two inference layers  $f_\theta(\cdot)$ ,  $g_\gamma(\cdot)$  are jointly trained by the proposed dual feature-level and label-level self-distillation.

### B. Dual Knowledge Self-Distillation

In this subsection, we propose dual knowledge self-distillation to learn node features  $\mathbf{X}$  and labels  $\mathcal{Y}_L$  guided by structural information  $\mathcal{E}$ . It consists of two *bidirectional* modules, one *Feature-level Self-Distillation* that distills feature information from the neighborhood into the target node, and one *Label-level Self-Distillation* that distills label information from the target node into the neighborhood.

1) *Feature-Level Self-Distillation*: The graphs can be categorized into homophily and heterophily graphs, where the former fulfills the smoothness assumption while the latter does not. Considering the importance and prevalence of homophily graphs, for which a variety of classical GNNs including GCN, GAT, SAGE, and APPNP have been previously developed, we focus mainly on learning homophily graphs in this paper. The smoothness assumption indicates that neighboring nodes in a graph tend to share similar features and labels, while non-neighboring nodes should be far away. With such motivation, we perform feature-level self-distillation on the neighborhood by regularizing the consistency of the label distributions between the target node and its neighboring nodes. Along with connectivity, disconnectivity between nodes also carries important information that reveals the node's dissimilarity. However, the number of neighboring nodes is much smaller compared with those non-neighboring nodes, which renders the model overemphasize the differences between the target and non-neighboring nodes, possibly leading to imprecise class boundaries. To solve this problem, we modify *Mixup* [57], an effective data augmentation that performs interpolation between samples to generate new samples, to augment neighboring nodes. Specifically, we perform *learnable interpolation* between the target node  $v_i$  and its neighboring node  $v_j \in \mathcal{N}_i$  to generate a new node, with its node representation defined as

$$\mathbf{z}'_{i,j} = g_\gamma(\beta_{i,j}\mathbf{h}_i^{(L)} + (1 - \beta_{i,j})\mathbf{h}_j^{(L)}),$$

where  $\beta_{i,j} = \text{sigmoid}(\mathbf{a}^T [\mathbf{x}_i \mathbf{W}_m \| \mathbf{x}_j \mathbf{W}_m])$  (3)

where  $\mathbf{W}_m \in \mathbb{R}^{d \times F}$  and  $\beta_{i,j}$  is defined as *learnable interpolation coefficients* with the shared attention weight  $\mathbf{a}$ . Then, we take augmented neighboring nodes as positive samples and other non-neighboring nodes as negative samples to simultaneously model the connectivity and disconnectivity between

nodes. Specifically, the learning objective of feature-level self-distillation is defined as follows:

$$\mathcal{L}_{\text{feat}} = \frac{1}{N} \sum_{i=1}^N \left( \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \|\mathbf{y}_i - \mathbf{z}'_{i,j}\|_2^2 - \frac{1}{M_i} \sum_{e_{i,k} \notin \mathcal{E}} \|\hat{\mathbf{y}}_i - \hat{\mathbf{z}}_k\|_2^2 \right) \quad (4)$$

where  $M_i = |\mathcal{E}| - |\mathcal{N}_i| - 1$  is the number of negative samples of node  $v_i$ . Besides,  $\hat{\mathbf{y}}_i = \text{softmax}(\mathbf{y}_i) \in \mathbb{R}^C$  and  $\hat{\mathbf{z}}_k = \text{softmax}(\mathbf{z}_k) \in \mathbb{R}^C$ . The loss function  $\mathcal{L}_{\text{feat}}$  defined in (4) essentially encourages positive neighboring nodes to be closer and pushes negative non-neighboring nodes away.

2) *Label-Level Self-Distillation*: Thus far, we have only discussed how to use node features  $\mathbf{X}$  and structural information  $\mathcal{E}$  for feature-level self-distillation, but have not explored how to leverage node labels  $\mathcal{Y}_L$ . A widely used solution to leverage labels is to optimize the objective on the labeled data  $\mathcal{V}_L$  as follows

$$\min_{\theta, \mathbf{W}^{(0)}, \dots, \mathbf{W}^{(L-1)}} \frac{1}{|\mathcal{V}_L|} \sum_{i \in \mathcal{V}_L} \mathcal{L}_{CE}(y_i, \hat{\mathbf{y}}_i), \quad (5)$$

where  $\mathcal{L}_{CE}(y_i, \hat{\mathbf{y}}_i)$  denotes the cross-entropy loss between  $\hat{\mathbf{y}}_i$  and ground-truth label  $y_i$ . However, (5) only considers node labels  $\mathcal{Y}_L$ , but completely ignores structural information  $\mathcal{E}$ . In practice, label propagation [58] is widely used as an effective trick to simultaneously model label and structural information and achieves promising results for various GNNs. However, label propagation involves the explicit coupling of labels with the structure, so it is heavily data-dependent with the same inference-latency problem as message passing. Earlier, we have proposed feature-level self-distillation of (4) to substitute explicit message passing in (1), and next we introduce *implicit* label-level self-distillation of (6) to substitute *explicit* label propagation to jointly exploit both label and structural information. The objective of label-level self-distillation is defined as

$$\mathcal{L}_{\text{label}} = \frac{1}{|\mathcal{V}_L|} \sum_{i \in \mathcal{V}_L} \left( \mathcal{L}_{CE}(y_i, \hat{\mathbf{y}}_i) + \sum_{j \in \mathcal{N}_i} \mathcal{L}_{CE}(y_i, \hat{\mathbf{z}}_j) \right). \quad (6)$$

### C. Training and Inferring

1) *Model Training*: In practice, directly optimizing (4)(6) faces two tricky challenges: (1) it treats all non-neighboring nodes as negative samples, which suffers from a huge computational burden; and (2) it performs the summation over the entire set of nodes, i.e., requiring a large memory space for keeping the entire graph. To address these two problems, we adopt the edge sampling strategy [59], [60] instead of feeding the entire graph into the memory for *batch-style training*. More specifically, we first sample mini-batch edges  $\mathcal{E}_b \in \mathcal{E}$  from the entire edge set. Then we randomly sample negative nodes by a pre-defined negative distribution  $P_k(v)$  for each edge  $e_{i,j} \in \mathcal{E}_b$  instead of enumerating all non-neighboring nodes as negative samples. Finally, we can rewrite (4) as

$$\mathcal{L}_{\text{feat}} = \frac{1}{B} \sum_{b=1}^B \sum_{e_{i,j} \in \mathcal{E}_b} \left( \|y_i - z'_{i,j}\|_2^2 + \|y_j - z'_{j,i}\|_2^2 - \mathbb{E}_{v_k \sim P_k(v)} \left( \|\hat{y}_i - \hat{z}_k\|_2^2 + \|\hat{y}_j - \hat{z}_k\|_2^2 \right) \right), \quad (7)$$

where  $B$  is the batch size, and  $P_k(v)$  adopts the uniform distribution by default, that is  $P_k(v_i) = \frac{1}{N}$  for each node  $v_i$ .  $P_k(v)$  can also be pre-defined based on prior knowledge, e.g., degree distribution, but in practice, we find from the experimental results in Section V-E that uniform distribution is a reasonable choice that can yield fairly good performance across various datasets. Similarly, we can rewrite the label-level self-distillation in (6) as a batch-style formulation,

$$\mathcal{L}_{\text{label}} = \frac{1}{B} \sum_{b=1}^B \frac{1}{|\mathcal{V}_b|} \sum_{i \in \mathcal{V}_b} \left( \mathcal{L}_{CE}(y_i, \hat{y}_i) + \sum_{e_{i,j} \in \mathcal{E}_b} \mathcal{L}_{CE}(y_i, \hat{z}_j) \right), \quad (8)$$

where  $\mathcal{V}_b = \{v_i, v_j | e_{i,j} \in \mathcal{E}_b\} \cap \mathcal{V}_L$  is all the sampled nodes in  $\mathcal{E}_b$ . Finally, the total training loss can be defined as,

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{label}} + \alpha \mathcal{L}_{\text{feat}}, \quad (9)$$

where  $\alpha$  is a trade-off hyperparameter.

2) *Model Inferring*: Once the model training is completed, we can directly omit the inference layer  $g_\gamma(\cdot)$  and retain the backbone MLP architecture and the inference layer  $f_\theta(\cdot)$  for label prediction. At this time, there is no data dependency for model inference, and this is attributed to the fact that we have shifted a considerable amount of work from the latency-sensitive inference stage to the latency-insensitive training stage. The pseudo-code of TGS is summarized in Algorithm 1.

### D. Discussion and Comparison

*Comparison With Message and Label Propagation*: The core of GNNs is the use of structural information, where *message passing* and *label propagation* are the two dominant schemes. The message passing models the data dependency within a local subgraph through neighborhood feature aggregation, while label

---

#### Algorithm 1: Algorithm for the Proposed TGS Framework.

---

**Input:** Features:  $\mathbf{X}$ ; Edge Set:  $\mathcal{E}$ ; # Batch:  $B$ ; # Epoch:  $E$ .

- 1: Initialize parameters  $\{\mathbf{W}^l\}_{l=0}^{L-1}$ ,  $f_\theta(\cdot)$ , and  $g_\gamma(\cdot)$ .
- 2: **for**  $epoch \in \{0, 1, \dots, E-1\}$  **do**
- 3:   **for**  $b \in \{0, 1, \dots, B-1\}$  **do**
- 4:     Sample a mini-batch of edges  $\mathcal{E}_b$  from  $\mathcal{E}$ ;
- 5:     Compute losses  $\mathcal{L}_{\text{feat}}$  and  $\mathcal{L}_{\text{label}}$  by (7)(8);
- 6:     Sum up  $\mathcal{L}_{\text{feat}}$  and  $\mathcal{L}_{\text{label}}$  as total loss  $L_{\text{total}}$ ;
- 7:     Update parameters by back-propagation of  $L_{\text{total}}$ .
- 8:   **end for**
- 9: **end for**
- 10: Predict labels  $\mathcal{Y}_U$  for those unlabeled nodes  $\mathcal{V}_U$ .
- 11: **return** Predictions  $\mathcal{Y}_U$ , parameters  $\{\mathbf{W}^l\}_{l=0}^{L-1}$  and  $f_\theta(\cdot)$ .

---

propagation focuses on diffusing label information to the neighborhood, and *they are complementary to each other*. However, both of them involve explicit coupling of features/labels with structures, leading to data dependency and inference latency. Different from the *explicit* message passing and label propagation, we use structural information as prior, as shown in Fig. 1, to *implicitly* guide bidirectional dual knowledge self-distillation: (1) feature-level self-distillation from neighborhood to the target node as in (4) and (2) label-level self-distillation from the target node to the neighborhood as in (6), where structural information is never explicitly involved in the forward propagation. Furthermore, while (4)(6) is defined on the 1-hop neighborhood, it can aggregate messages from multi-hops away by multiple cascades of self-distillation. For example, if there is knowledge self-distillation between the target node and its 1-hop neighbor and between its 1-hop neighbor and its 2-hop neighbor, then as the training proceeds, the messages from the 2-hop neighbor will first be distilled to the 1-hop neighbor, and then progressively propagated to the target node in such a “cascading” manner.

*Comparison With Graph Contrastive Learning*: Another research topic that is close to ours is graph contrastive learning, but TGS differs from it in the following two aspects: (1) learning objective, graph contrastive learning mainly aims to learn transferable knowledge from abundant unlabeled data in an *unsupervised* setting and then generalize the learned knowledge to downstream tasks. Instead, TGS works in a *semi-supervised* setting, i.e., the label information is available during training. (2) augmentation, graph contrastive learning usually requires multiple types of sophisticated augmentation to obtain different views for contrasting. However, TGS augments only positive samples by simple linear interpolation (mixup). Overall, we believe that TGS is more highly relevant to knowledge distillation than to graph contrastive learning, although we are aware that the two research topics have some aspects in common.

*Time Complexity Analysis*: The training time complexity of the proposed TGS framework is  $\mathcal{O}(|\mathcal{V}|dF + |\mathcal{E}|F)$ , which is linear with respect to the number of nodes  $|\mathcal{V}|$  and edges  $|\mathcal{E}|$ , and is in the same order of magnitude as GCNs. However, with the removal of neighborhood fetching, the inference time complexity can be reduced from  $\mathcal{O}(|\mathcal{V}|dF + |\mathcal{E}|F)$  to  $\mathcal{O}(|\mathcal{V}|dF)$ .

TABLE I  
OVERVIEW SUMMARY OF THE STATISTICAL CHARACTERISTICS OF DATASETS

Dataset	Cora	Citeseer	Amazon-Photo	Coauthor-CS	Coauthor-Phy	Amazon-Com
# Nodes	2708	3327	7650	18333	34493	13752
# Edges	5278	4614	119081	81894	247962	245861
# Features	1433	3703	745	6805	8415	767
# Classes	7	6	8	15	5	10
Label Rate	5.2%	3.6%	2.1%	1.6%	0.3%	1.5%

of MLPs. A comparison of the inference speeds of the various methods can be found in Section V-D and Table IV.

## V. EXPERIMENTS

In this section, we evaluate TGS on six real-world datasets by answering five questions. *Q1*: How does TGS compare with SOTA graph knowledge distillation methods? *Q2*: Is TGS robust under limited labeled data and label noise? *Q3*: How does TGS compare with other general GNN models and inference acceleration methods in terms of inference speed? *Q4*: How does TGS benefit from negative samples, mixup-like augmentation, and feature/label self-distillation? *Q5*: How do two key hyperparameters, loss weight  $\alpha$  and batch size  $B$ , influence the performance?

### A. Experimental Setup

1) *Datasets*: The experiments are conducted on six widely used real-world datasets, including Cora [61], Citeseer [62], Coauthor-CS, Coauthor-Physics, Amazon-Com, and Amazon-Photo [63]. For the two small-scale datasets, Cora and Citeseer, we follow the data splitting strategy in [35]. For the four large-scale datasets, Coauthor-CS, Coauthor-Physics, Amazon-Photo, and Amazon-Computers, we follow [25], [33] to select 20 nodes per class to construct a training set, 500 nodes for validation, and 1000 nodes for testing. A summary of the statistical characteristics of datasets is given in Table I.

2) *Baselines*: In this paper, we consider the following six classical baselines: Vanilla MLP, GCN [35], GAT [14], GraphSAGE [15], APPNP [36], and DAGNN [64]. In addition, we compare TGS with four types of graph knowledge distillation methods, including (1) GNN-to-GNN KD: FreeKD [42] and TinyGNN [17], (2) GNN-to-MLP KD: CPF [26], GLNN [25], and FF-G2M [27], (3) GNN Self-Distillation: RDD [31] and GNN-SD [30], and (3) MLP Self-Distillation: Graph-MLP [32] and LinkDist [33]. Moreover, with GCN as the base architecture, three classical inference acceleration methods are compared, including pruning with 50% weights (P-GCN) [38], quantization from FP32 to INT8 (Q-GCN) [39], and neighborhood sampling with fan-out 15 (NS-GCN) [40].

3) *Hyperparameter*: The hyperparameters are set the same for all datasets: Adam optimizer with learning rate  $lr = 0.01$  and weight decay  $decay = 5e-4$ ; Epoch  $E = 200$ ; Layer number  $L = 2$ . The other dataset-specific hyperparameters are determined by an AutoML toolkit NNI with the hyperparameter search spaces as hidden dimension  $F = \{256, 512, 1024\}$ ; batch size  $B = \{256, 512, 1024, 4096\}$ , trade-off weight  $\alpha = \{0.5, 0.8, 1.0\}$ .

Each set of experiments is run five times with different random seeds, and the average accuracy and standard deviation are reported as metrics. Moreover, the experiments are implemented based on the standard implementation in PyTorch 1.6.0 library with Intel(R) Xeon(R) Gold 6240R @ 2.40 GHz CPU and NVIDIA V100 GPU.

### B. Performance Comparison (*Q1*)

To answer *Q1*, we conducted experiments on six real-world datasets with comparison with state-of-the-art baselines. From the results reported in Table II, we can observe that: (1) There are some GNN-to-MLP KD and GNN Self-Distillation methods, such as RDD and FF-G2M, that can achieve comparable or even better performance than GNN-to-GNN KD, suggesting that both GNNs and teachers may not be necessary for graph knowledge distillation. (2) Considering both teacher-free and GNN-less designs, existing MLP Self-Distillation methods, such as Graph-MLP and LinkDist, lag far behind GNN-to-GNN KD and cannot even match the performance of vanilla GCNs on some datasets. (3) Regarding classification accuracy, TGS consistently achieves the best overall performance on six datasets. For example, TGS obtains the best performance on the Coauthor-Phy dataset, and more notably, our classification accuracy is 3.13% and 2.89% higher than that of GLNN and GNN-SD. (4) We additionally consider two vanilla baselines: (i) GNN-MLP, which trains a GNN and then directly copies its weights to an MLP for inference, and its performance is only close to or even poorer than that of vanilla MLP; and (ii) DGI-MLP, which performs contrastive learning with DGI but uses MLP as the backbone, and its performance is far superior to that of MLP (suggesting the potential of the MLP architecture), but still inferior to that of vanilla GCN. (5) We conduct ablation studies on two important modules, namely mixup augmentation and loss function. It can be seen that mixup augmentation plays an important role in the performance. However, even with the removal of the mixup, TGS still outperforms GCN, GAT, Graph-MLP, and LinkDist, on all six datasets. However, when mixup is considered, the performance of TGS is even comparable to those state-of-the-art graph KD methods. Besides, we find that using triple loss as the objective function slightly degrades performance on some datasets, but helps improve performance on the Amazon-Com and Amazon-Photo datasets.

### C. Evaluation on Robustness (*Q2*)

There has been some work pointing out that the performance of graph learning algorithms depends heavily on the quality and



TABLE II  
CLASSIFICATION ACCURACY  $\pm$  STD (%) ON SIX REAL-WORLD DATASETS, WHERE THE BEST AND SECOND RESULTS MARKED BY **BOLD** AND UNDERLINE, RESPECTIVELY

Type	Method	Cora	Citeseer	Coauthor-CS	Coauthor-Phy	Amazon-Com	Amazon-Photo	Avg. Rank
MLPs / GNNs	MLP	61.86 $\pm$ 0.43	59.76 $\pm$ 0.51	83.34 $\pm$ 0.64	86.24 $\pm$ 0.66	66.85 $\pm$ 1.94	78.18 $\pm$ 1.25	16.00
	GCN	81.28 $\pm$ 0.42	71.06 $\pm$ 0.44	87.76 $\pm$ 0.43	91.89 $\pm$ 0.42	77.45 $\pm$ 1.71	87.53 $\pm$ 1.64	13.33
	GAT	83.02 $\pm$ 0.45	72.56 $\pm$ 0.51	88.55 $\pm$ 0.56	92.36 $\pm$ 0.47	82.78 $\pm$ 1.89	90.19 $\pm$ 1.35	6.33
	GraphSAGE	82.22 $\pm$ 0.80	71.22 $\pm$ 0.58	88.40 $\pm$ 0.48	91.88 $\pm$ 0.53	79.23 $\pm$ 1.63	88.63 $\pm$ 1.17	10.67
	APPNP	83.28 $\pm$ 0.33	71.74 $\pm$ 0.27	88.74 $\pm$ 0.62	92.75 $\pm$ 0.60	81.28 $\pm$ 1.90	89.49 $\pm$ 1.28	6.50
	DAGNN	<u>84.30</u> $\pm$ 0.51	73.14 $\pm$ 0.62	89.32 $\pm$ 0.55	93.10 $\pm$ 0.67	80.32 $\pm$ 1.57	<b>90.72</b> $\pm$ 1.45	3.67
GNN-to-GNN	TinyGNN	82.79 $\pm$ 0.57	72.67 $\pm$ 0.72	88.72 $\pm$ 0.42	92.20 $\pm$ 0.67	79.22 $\pm$ 1.69	89.24 $\pm$ 1.24	8.67
	FreeKD	83.80 $\pm$ 0.53	73.76 $\pm$ 0.60	89.14 $\pm$ 0.61	92.63 $\pm$ 0.71	80.92 $\pm$ 1.75	89.46 $\pm$ 1.31	5.17
GNN Self-Distillation	GNN-SD	81.85 $\pm$ 0.55	71.69 $\pm$ 0.61	87.80 $\pm$ 0.50	92.07 $\pm$ 0.48	77.66 $\pm$ 1.85	87.80 $\pm$ 1.52	11.67
	RDD	83.68 $\pm$ 0.40	73.63 $\pm$ 0.50	<u>89.38</u> $\pm$ 0.44	92.74 $\pm$ 0.78	81.84 $\pm$ 1.48	89.70 $\pm$ 0.93	3.50
GNN-to-MLP	GNN-MLP	64.53 $\pm$ 0.42	62.26 $\pm$ 0.48	81.26 $\pm$ 0.87	86.47 $\pm$ 0.71	69.25 $\pm$ 1.75	76.34 $\pm$ 1.30	-
	GLNN	80.85 $\pm$ 0.60	71.21 $\pm$ 0.80	87.81 $\pm$ 0.53	91.83 $\pm$ 0.60	77.96 $\pm$ 1.70	87.98 $\pm$ 1.36	11.29
	CPF	83.65 $\pm$ 0.49	72.98 $\pm$ 0.47	89.10 $\pm$ 0.50	92.36 $\pm$ 0.63	80.90 $\pm$ 1.52	89.03 $\pm$ 1.29	6.67
	FF-G2M	84.06 $\pm$ 0.43	<u>73.85</u> $\pm$ 0.51	88.96 $\pm$ 0.45	92.83 $\pm$ 0.58	81.92 $\pm$ 1.54	89.58 $\pm$ 1.43	3.83
MLP Self-Distillation	DGI-MLP	76.39 $\pm$ 0.46	67.83 $\pm$ 0.51	85.13 $\pm$ 0.49	89.41 $\pm$ 0.45	73.25 $\pm$ 1.64	84.60 $\pm$ 1.60	-
	Graph-MLP	81.45 $\pm$ 0.52	72.87 $\pm$ 0.70	88.16 $\pm$ 0.70	91.85 $\pm$ 0.49	77.23 $\pm$ 1.76	87.64 $\pm$ 1.37	11.83
	LinkDist	76.70 $\pm$ 0.47	65.19 $\pm$ 0.55	87.89 $\pm$ 0.58	92.16 $\pm$ 0.70	76.93 $\pm$ 1.83	87.26 $\pm$ 1.42	13.67
	TGS (ours)	<b>84.90</b> $\pm$ 0.44	<b>74.08</b> $\pm$ 0.69	<b>89.62</b> $\pm$ 0.40	<b>94.96</b> $\pm$ 0.41	<u>83.44</u> $\pm$ 2.09	90.34 $\pm$ 0.85	1.17
	w/o mixup	83.18 $\pm$ 0.41	72.96 $\pm$ 0.58	89.18 $\pm$ 0.61	93.24 $\pm$ 0.56	82.94 $\pm$ 1.90	89.89 $\pm$ 0.81	-
	w/ triplet loss	84.10 $\pm$ 0.54	73.60 $\pm$ 0.63	89.12 $\pm$ 0.45	<u>93.86</u> $\pm$ 0.48	<b>83.52</b> $\pm$ 2.10	<u>90.57</u> $\pm$ 0.78	-

If not specifically mentioned, all relevant models adopt GCN as the backbone by default.

TABLE III  
CLASSIFICATION ACCURACY  $\pm$  STD (%) WITH LIMITED NODE LABELS, WHERE THE BEST RESULTS MARKED BY **BOLD**

Dataset		GCN	ANNPN	DAGNN	GNN-SD	RDD	GLNN	CPF	FF-G2M	Graph-MLP	TGS (ours)
<b>Cora</b>	5 labels	73.10 $\pm$ 0.87	76.39 $\pm$ 0.95	79.02 $\pm$ 0.94	74.32 $\pm$ 0.85	76.11 $\pm$ 0.81	73.85 $\pm$ 0.92	75.92 $\pm$ 0.90	77.62 $\pm$ 0.67	78.43 $\pm$ 0.78	<b>80.22</b> $\pm$ 0.87
	10 labels	77.52 $\pm$ 0.63	79.99 $\pm$ 0.72	81.99 $\pm$ 0.62	78.55 $\pm$ 0.64	79.68 $\pm$ 0.54	77.94 $\pm$ 0.52	79.20 $\pm$ 0.64	79.20 $\pm$ 0.44	79.60 $\pm$ 0.49	<b>82.80</b> $\pm$ 0.45
	15 labels	79.47 $\pm$ 0.56	80.70 $\pm$ 0.44	82.72 $\pm$ 0.50	79.89 $\pm$ 0.55	80.45 $\pm$ 0.46	79.16 $\pm$ 0.48	80.12 $\pm$ 0.51	80.25 $\pm$ 0.52	80.32 $\pm$ 0.57	<b>83.40</b> $\pm$ 0.53
<b>Citeseer</b>	5 labels	63.23 $\pm$ 1.04	66.28 $\pm$ 0.88	69.13 $\pm$ 0.68	63.93 $\pm$ 0.74	66.06 $\pm$ 0.57	63.10 $\pm$ 0.70	65.40 $\pm$ 0.62	68.12 $\pm$ 0.59	<b>69.64</b> $\pm$ 0.64	68.76 $\pm$ 1.16
	10 labels	67.55 $\pm$ 0.50	69.23 $\pm$ 0.64	71.74 $\pm$ 0.71	68.20 $\pm$ 0.57	69.68 $\pm$ 0.66	67.65 $\pm$ 0.62	69.14 $\pm$ 0.73	70.76 $\pm$ 0.53	70.56 $\pm$ 0.51	<b>72.66</b> $\pm$ 0.43
	15 labels	69.64 $\pm$ 0.58	70.17 $\pm$ 0.44	72.26 $\pm$ 0.53	69.86 $\pm$ 0.48	70.32 $\pm$ 0.57	69.52 $\pm$ 0.52	70.76 $\pm$ 0.46	71.59 $\pm$ 0.47	71.80 $\pm$ 0.63	<b>73.10</b> $\pm$ 0.53

quantity of the labels. To evaluate the robustness of the TGS framework, we evaluate it with extremely limited label data and label noise on the Cora and Citeseer datasets.

1) *Performance With Limited Labels*: To evaluate the effectiveness of TGS when labeled data is limited, we randomly select 5, 10, and 15 labeled samples per class for training, and the rest of the training set is considered unlabeled. From the experimental results reported in Table III, we can make three important observations (1) The performance of all methods drops as the number of labeled data is reduced, but that of TGS drops more slightly. (2) While GNN-to-MLP KD and GNN Self-Distillation methods perform well on clean data, as shown in Table II, their performance gains are reduced when labeled data is extremely limited. In contrast, the two MLP Self-Distillation methods, Graph-MLP and TGS, show a great advantage under the label-limited setting. (3) When only a limited number of labels are provided, TGS outperforms all other baselines at most label rates. For example, when trained with 5 labels per class, TGS outperforms GCN by 7.12% and 5.53% on the Cora and Citeseer datasets, respectively.

2) *Performance With Noisy Labels*: We evaluate the robustness of TGS against label noise by injecting *asymmetric* noise into class labels, where the label  $i$  ( $1 \leq i \leq C$ ) of each training sample flips independently with probability  $r$  to another class, but with probability  $1 - r$  preserved as label  $i$  [65], [66]. The performance is reported in Fig. 3(a) and (b) at various noise

ratios  $r \in \{0\%, 10\%, 20\%, \dots, 60\%\}$ . It can be seen that as noise ratio  $r$  increases, the accuracy of TGS drops more slowly than other baselines, suggesting that TGS is more robust than other baselines under various noise ratios, especially under extremely high noise ratios. For example, with  $r = 60\%$  label noise ratio, TGS outperforms RDD and FF-G2M by 6.92% and 6.41% on the Citeseer dataset, respectively.

#### D. Evaluation on Inference Speed (Q3)

Commonly used inference acceleration techniques on GNNs include Pruning [38], Quantizing [39], and Neighborhood Sampling [40]. With GCN as the base architecture, we consider its three variants: P-GCN, Q-GCN, and NS-GCN.

1) *Comparison on Different Datasets*: With the removal of neighborhood fetching, the inference time of TGS can be reduced from  $\mathcal{O}(|V|dF + |\mathcal{E}|F)$  to  $\mathcal{O}(|V|dF)$ . The inference time (*ms*) averaged over 30 sets of runs on four datasets is reported in Table IV with the acceleration multiple w.r.t the vanilla GCN marked as *green*, where all methods use  $L = 2$  layers and dimension  $F = 16$ . From Table IV, we can observe that: (1) While APPNP and DAGNN improve a lot over GCN in classification accuracy, as shown in Table III, they suffer from more severe inference latency. (2) Pruning and quantization are not very effective on GNNs, given that data dependency in GNNs has not been well resolved. Besides, the neighborhood sampling

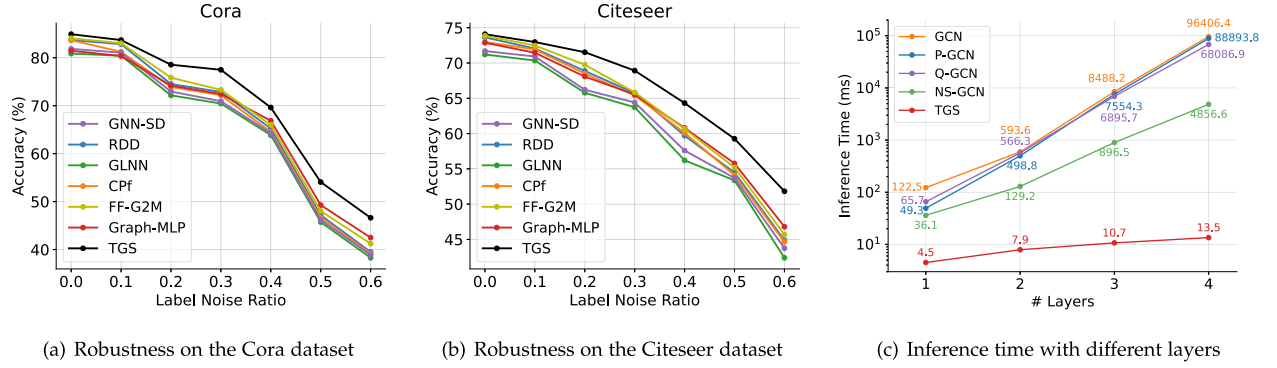


Fig. 3. (a) and (b) Classification accuracy (%) under different label noise ratios on the Cora and Citeseer datasets, respectively. (c) Inference time (ms) with different layers on the Coauthor-CS dataset.

TABLE IV  
INFERENCE TIME (*ms*) ON FOUR DATASETS, WHERE THREE COMMONLY USED INFERENCE ACCELERATION METHODS HELP TO SPEED UP GCN, BUT STILL CONSIDERABLY SLOWER THAN TGS

Method	GCN	APNP	DAGNN	P-GCN	Q-GCN	NS-GCN	TGS (ours)
<b>Cora</b>	402.7	731.8	647.6	372.9 (1.08 $\times$ )	383.5 (1.05 $\times$ )	97.7 (4.12 $\times$ )	4.6 (87.54 $\times$ )
<b>Citeseer</b>	383.4	700.5	727.5	316.9 (1.21 $\times$ )	361.4 (1.06 $\times$ )	105.6 (3.63 $\times$ )	4.3 (89.16 $\times$ )
<b>Coauthor-CS</b>	593.6	1099.1	672.4	498.8 (1.19 $\times$ )	566.3 (1.05 $\times$ )	129.2 (4.59 $\times$ )	7.9 (75.14 $\times$ )
<b>Coauthor-Phy</b>	1067.6	1467.4	708.6	922.3 (1.16 $\times$ )	997.8 (1.07 $\times$ )	269.3 (3.96 $\times$ )	12.8 (83.41 $\times$ )

Note the acceleration multiple w.r.t the vanilla GCN is marked as green.

considers but does not completely eliminate the neighborhood-fetching latency, so it infers faster than pruning and quantization, yet still lags far behind TGS which is based on MLPs. (3) TGS infers fastest across four datasets.

2) *Comparison With Different Layers*: To infer a single node with a  $L$ -layer GNN on a graph with average node degree  $R$ , it requires fetching  $\mathcal{O}(R^L)$  nodes. To compare the sensitivity of TGS to layer depth with other inference acceleration methods, we report their inference time (*ms* in log-scale) at different layer depths on the Coauthor-CS dataset in Fig. 3(c). The inference time of TGS only increases *linearly* with the layer depth, but that of other baselines increases *exponentially*. Moreover, the speed gains of GNN pruning and quantization are reduced as the layer depth increases, and they approach the vanilla GCN when the layer depth is 4. In contrast, at larger layer depths, the speed gain of the neighbor sampling gets enlarged compared to the vanilla GCN. This demonstrates that *neighborhood fetching is one major source of inference latency in GNNs*, and the linear complexity of TGS has a great advantage, especially when GNNs become deeper.

#### E. Ablation Study (Q4)

1) *Component Analysis*: To evaluate the effectiveness of negative samples in (4), mixup-like augmentation in (3), and label self-distillation in (8), we conducted four sets of experiments: the model without (A) Negative Samples (*w/o* NS); (B) mixup-like augmentation (*w/o* augment); (C) Label Self-Distillation (*w/o* LSD); and (D) the full model. Besides, to evaluate the impact of the negative distribution, we take the nodal degree  $d_i$  as a prior and preset  $P_k(v_i) = \frac{d_i}{|\mathcal{V}|}$  in place

of the default uniform distribution in this paper, denoted as (E) *w/o* uniform. After analyzing the results in Fig. 4(a), we can conclude that: (1) Negative Samples and mixup-like augmentation contribute to improving classification performance. More importantly, applying them together can further improve performance on top of each. (2) Label self-distillation helps to improve performance on top of the stand-alone feature self-distillation. (3) Even without considering any graph prior, pre-setting  $P_k(\cdot)$  as uniform distribution is sufficient to achieve comparable performance, so this paper defaults to the simplest uniform distribution without considering complex prior-based distributions.

2) *How TGS Benefit From Self-Distillation*: Previous attempts have shown that there do exist optimal MLP parameters enabling its performance to be competitive with GNNs, but it is hard to learn such parameters by only cross-entropy loss [32], [33]. The proposed TGS helps to solve this problem with two potential advantages: (1) alleviating overfitting, and (2) introducing graph topology [25].

First, we plot the training curves (with log-scale vertical coordinate) of TGS and MLPs on the Cora dataset in Fig. 4(b), from which we observe that the gap between training and validation loss is smaller for TGS than MLPs, which indicates that TGS helps to alleviate the overfitting trend of MLPs. Second, we conjecture that the absence of inductive bias, e.g., graph topology, is one of the major reasons why MLP is inferior to GNN in inference accuracy. To illustrate it, we plot in Fig. 4(c) the average cosine similarity of nodes with their 1-hop and 2-hop neighbors for MLPs, GCNs, and TGS on the Cora dataset. It can be seen that the average similarity with 1-hop neighbors is always higher than that with 2-hop neighbors throughout



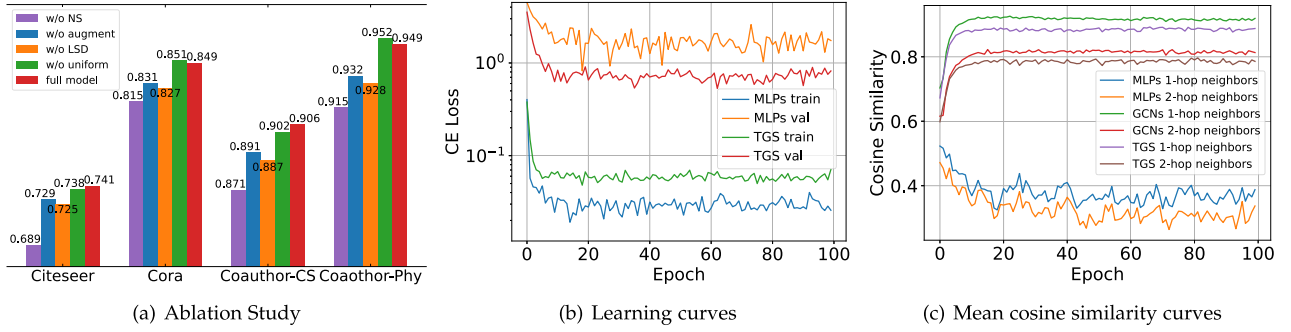


Fig. 4. (a) Ablation study on four key model components. (b) Learning curves of MLPs and TGS on the Cora dataset, showing that self-distillation helps to regularize the training. The *logarithmized* vertical coordinate is the cross-entropy loss between the predicted and ground-truth labels on the training or validation set, respectively. (c) Mean cosine similarity curves of MLPs, GCNs, and TGS between the target node with 1-hop and 2-hop neighbors on the Cora dataset.

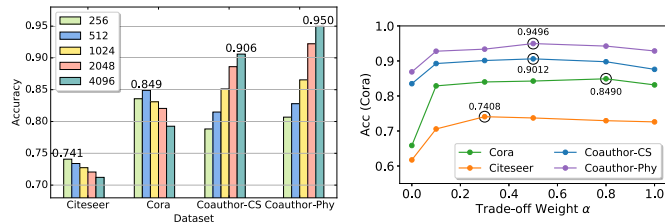


Fig. 5. Hyperparameter sensitivity analysis on the batch size  $B$  (Left) and trade-off weight  $\alpha$  (Right) on four datasets.

the training process for MLPs, GCNs, and TGS. More importantly, the average similarity of GCNs and TGS gradually increases with training, while that of MLPs gradually decreases, which indicates that TGS has introduced graph topology as an inductive bias (as GCN has done), while MLP does not. As a result, our TGS enjoys the benefits of topology-awareness in training but without neighborhood-fetching latency in inference.

#### F. Hyperparameter Sensitivity Analysis (Q5)

To answer Q5, we evaluate the hyperparameter sensitivity w.r.t two key hyperparameters: trade-off weight  $\alpha \in \{0.0, 0.1, 0.3, 0.5, 0.8, 1.0\}$  and batch size  $B \in \{256, 512, 1024, 2048, 4096\}$  in Fig. 5, from which we can make two key observations that (1) batch size  $B$  is a dataset-specific hyperparameter. For simple graphs with few nodes and edges, such as Cora, a small batch size,  $B = 256$ , can yield fairly good performance. However, for large-scale graphs with more nodes and edges, such as Coauthor-Phy, the model performance usually improves with the increase of batch size  $B$ . (2) When  $\alpha$  is set to 0, i.e., the feature-level self-distillation is completely removed, the performance of TGS degrades to be close to that of MLPs. In contrast, when  $\alpha$  takes a non-zero value, the performance of TGS improves as  $\alpha$  increases. However, when  $\alpha$  becomes too large, it weakens the benefit of label information, yielding lower performance improvements. In practice, we can usually determine  $B$  and  $\alpha$  by selecting the model with the highest accuracy on the validation set through the grid search.

## VI. CONCLUSION

Motivated by the complementary strengths and weaknesses of GNNs and MLPs, we propose *Teacher-Free Graph Self-Distillation* (TGS) framework that does not require both teacher models and GNNs during training and inference. The proposed TGS framework is purely based on MLPs, where structural information is only implicitly used to guide dual knowledge self-distillation between the target node and its neighborhood. More importantly, we study TGS comprehensively by investigating how they benefit from neighborhood self-distillation and how they are different from existing works. Extensive experiments have shown the advantages of TGS over existing methods in terms of both inference accuracy and inference efficiency. Despite the great progress, limitations still exist and a major concern is that TGS is based on the neighborhood smoothing assumption (a common assumption adopted by various models such as GCN, GAT, etc.), so how to extend TGS to heterophily graphs may be a promising direction for future work.

## REFERENCES

- [1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.
- [2] J. Zhou et al., "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.
- [3] L. Wu, H. Lin, Z. Liu, Z. Liu, Y. Huang, and S. Z. Li, "Homophily-enhanced self-supervision for graph structure learning: Insights and directions," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, 2023, doi: [10.1109/TNNLS.2023.3257325](https://doi.org/10.1109/TNNLS.2023.3257325).
- [4] L. Wu, J. Xia, Z. Gao, H. Lin, C. Tan, and S. Z. Li, "GraphMixup: Improving class-imbalanced node classification by reinforcement mixup and self-supervised context prediction," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, Springer, 2022, pp. 519–535.
- [5] L. Wu et al., "Beyond homophily and homogeneity assumption: Relation-based frequency adaptive graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, 2023, doi: [10.1109/TNNLS.2023.3230417](https://doi.org/10.1109/TNNLS.2023.3230417).
- [6] H. Lin, Z. Gao, Y. Xu, L. Wu, L. Li, and S. Z. Li, "Conditional local convolution for spatio-temporal meteorological forecasting," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 7470–7478.
- [7] M. Ma et al., "HiSTGNN: Hierarchical spatio-temporal graph neural network for weather forecasting," *Inf. Sci.*, vol. 648, 2023, Art. no. 119580.
- [8] L. Wu et al., "MAPE-PPI: Towards effective and efficient protein-protein interaction prediction via microenvironment-aware protein embedding," in *Proc. 12th Int. Conf. Learn. Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=itGkF993gz>

- [9] L. Wu et al., "PSC-CPI: Multi-scale protein sequence-structure contrasting for efficient and generalizable compound-protein interaction prediction," in *Proc. AAAI Conf. Artif. Intell.*, 2024, pp. 9342–9352.
- [10] F. Zhou, Q. Yang, T. Zhong, D. Chen, and N. Zhang, "Variational graph neural networks for road traffic prediction in intelligent transportation systems," *IEEE Trans. Ind. Inform.*, vol. 17, no. 4, pp. 2802–2812, Apr. 2021.
- [11] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Expert Syst. Appl.*, vol. 207, 2022, Art. no. 117921.
- [12] Z. Jia, S. Lin, R. Ying, J. You, J. Leskovec, and A. Aiken, "Redundancy-free computation for graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 997–1005.
- [13] T. N. Kipf and M. Welling, "Variational graph auto-encoders," 2016, *arXiv:1611.07308*.
- [14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," 2017, *arXiv: 1710.10903*.
- [15] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.
- [16] G. Li, M. Müller, B. Ghanem, and V. Koltun, "Training graph neural networks with 1000 layers," 2021, *arXiv:2106.07476*.
- [17] B. Yan, C. Wang, G. Guo, and Y. Lou, "TinyGNN: Learning efficient graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 1848–1856.
- [18] H. He, J. Wang, Z. Zhang, and F. Wu, "Compressing deep graph neural networks via adversarial knowledge distillation," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2022, pp. 534–544.
- [19] H. Zhang et al., "Iterative graph self-distillation," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 3, pp. 1161–1169, Mar. 2024.
- [20] F. M. Nardini, C. Rulli, S. Trani, and R. Venturini, "Distilled neural networks for efficient learning to rank," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 5, pp. 4695–4712, May 2023.
- [21] W. Liu, D. Gong, M. Tan, J. Q. Shi, Y. Yang, and A. G. Hauptmann, "Learning distilled graph for large-scale social network data clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 7, pp. 1393–1404, Jul. 2020.
- [22] Y. Yang, J. Qiu, M. Song, D. Tao, and X. Wang, "Distilling knowledge from graph convolutional networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 7074–7083.
- [23] Y. Ren, J. Ji, L. Niu, and M. Lei, "Multi-task self-distillation for graph-based semi-supervised learning," 2021, *arXiv:2112.01174*.
- [24] L. Wu, H. Lin, Y. Huang, and S. Z. Li, "Knowledge distillation improves graph structure augmentation for graph neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2022, pp. 11815–11827.
- [25] S. Zhang, Y. Liu, Y. Sun, and N. Shah, "Graph-less neural networks: Teaching old MLPs new tricks via distillation," 2021, *arXiv:2110.08727*.
- [26] C. Yang, J. Liu, and C. Shi, "Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework," in *Proc. Web Conf.*, 2021, pp. 1227–1237.
- [27] L. Wu, H. Lin, Y. Huang, T. Fan, and S. Z. Li, "Extracting low-/high-frequency knowledge from graph neural networks and injecting it into MLPs: An effective GNN-to-MLP distillation framework," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 10351–10360.
- [28] Anonymous, "Double wins: Boosting accuracy and efficiency of graph neural networks by reliable knowledge distillation," in *Proc. 11th Int. Conf. Learn. Representations*, 2023. [Online]. Available: <https://openreview.net/forum?id=NGIFt6BNvLe>
- [29] L. Wu, H. Lin, Y. Huang, and S. Z. Li, "Quantifying the knowledge in GNNs for reliable distillation into MLPs," 2023, *arXiv:2306.05628*.
- [30] Y. Chen, Y. Bian, X. Xiao, Y. Rong, T. Xu, and J. Huang, "On self-distilling graph neural network," 2020, *arXiv: 2011.02255*.
- [31] W. Zhang et al., "Reliable data distillation on graph convolutional network," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 1399–1414.
- [32] Y. Hu, H. You, Z. Wang, Z. Wang, E. Zhou, and Y. Gao, "Graph-MLP: Node classification without message passing in graph," 2021, *arXiv:2106.04051*.
- [33] Y. Luo, A. Chen, K. Yan, and L. Tian, "Distilling self-knowledge from contrastive links to classify graph nodes without passing messages," 2021, *arXiv:2106.08541*.
- [34] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," 2016, *arXiv:1606.09375*.
- [35] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
- [36] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized PageRank," 2018, *arXiv: 1810.05997*.
- [37] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 249–270, Jan. 2022.
- [38] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, *arXiv:1506.02626*.
- [39] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.
- [40] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," 2018, *arXiv: 1801.10247*.
- [41] X. Xu, F. Zhou, K. Zhang, and S. Liu, "CCGL: Contrastive cascade graph learning," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 5, pp. 4539–4554, May 2023.
- [42] K. Feng, C. Li, Y. Yuan, and G. Wang, "FreeKD: Free-direction knowledge distillation for graph neural networks," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2022, pp. 357–366.
- [43] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, "Label propagation for deep semi-supervised learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 5070–5079.
- [44] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, "Be your own teacher: Improve the performance of convolutional neural networks via self distillation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 3713–3722.
- [45] Y. Tian, S. Pei, X. Zhang, C. Zhang, and N. V. Chawla, "Knowledge distillation on graphs: A survey," 2023, *arXiv:2302.00219*.
- [46] L. Wu, H. Lin, C. Tan, Z. Gao, and S. Z. Li, "Self-supervised learning on graphs: Contrastive, generative, or predictive," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 4216–4235, Apr. 2023.
- [47] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *Proc. 7th Int. Conf. Learn. Representations*, 2019, pp. 4856–4869.
- [48] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 5812–5823.
- [49] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Graph contrastive learning with adaptive augmentation," 2020, *arXiv: 2010.14945*.
- [50] D. Xu, W. Cheng, D. Luo, H. Chen, and X. Zhang, "InfoGCL: Information-aware graph contrastive learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 30 414–30 425.
- [51] J.-B. Grill et al., "Bootstrap your own latent-A new approach to self-supervised learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 21 271–21 284.
- [52] S. Thakoor, C. Tallec, M. G. Azar, R. Munos, P. Veličković, and M. Valko, "Bootstrapped representation learning on graphs," in *Proc. Workshop Geometrical Topological Representation Learn.*, 2021, pp. 142–151.
- [53] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 297–304.
- [54] Z. T. Kefato and S. Girdzijauskas, "Self-supervised graph neural networks without explicit negative sampling," 2021, *arXiv:2103.14958*.
- [55] C. Park, D. Kim, J. Han, and H. Yu, "Unsupervised attributed multiplex network embedding," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 5371–5378.
- [56] X. Yang, Z. Song, I. King, and Z. Xu, "A survey on deep semi-supervised learning," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 9, pp. 8934–8954, Sep. 2023.
- [57] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," 2017, *arXiv: 1710.09412*.
- [58] A. Iscen et al., "Label propagation for deep semi-supervised learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 5070–5079.
- [59] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.
- [60] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 1067–1077.
- [61] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Mag.*, vol. 29, no. 3, pp. 93–93, 2008.
- [62] C. L. Giles, K. D. Bollacker, and S. Lawrence, "CiteSeer: An automatic citation indexing system," in *Proc. 3rd ACM Conf. Digit. Libraries*, 1998, pp. 89–98.

- [63] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," 2018, *arXiv: 1811.05868*.
- [64] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 338–348.
- [65] N. Yin, L. Shen, M. Wang, X. Luo, Z. Luo, and D. Tao, "OMG: Towards effective graph classification against label noise," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 12, pp. 12873–12886, Dec. 2023.
- [66] J. Xia et al., "GNN cleaner: Label cleaner for graph structured data," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 2, pp. 640–651, Feb. 2024.



**Lirong Wu** received the BS degree from the Department of Information Science & Electronic Engineering, Zhejiang University, Hangzhou, China, in 2020. He is currently working toward the PhD degree with the School of Engineering, Westlake University, Hangzhou, China. His main research interests include low-level vision, deep clustering, graph learning, self-supervised learning. He has published several papers in top-tier conferences and journals such as *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Neural Networks and Learning Systems*, *IEEE Transactions on Circuits and Systems for Video Technology*, *WWW*, *AAAI*, *CVPR*, etc.



**Haitao Lin** received the BS degree in material physics and statistics from Sichuan University, Chengdu, China. He is currently working toward the PhD degree with the School of Engineering, Westlake University, Hangzhou, China. His main research interests include spatio-temporal model, multivariate time series forecasting with relational inference, and dynamic system in biology and physics. He has published several papers in top-tier conferences and journals such as *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Neural Networks and Learning Systems*, *WWW*, *AAAI*, etc.



**Zhangyang Gao** received the BS degree from the School of Automation, Central South University, Changsha, China, in 2020. He is currently working toward the PhD degree with the School of Engineering, Westlake University, Hangzhou, China. His main research interests include clustering, unsupervised video tasks, and deep learning.



**Guojiang Zhao** received the BS degree in information and computing science from the Xi'an University of Post & Telecommunications, Xi'an, China. He is currently working toward the MS degree in information systems management from Carnegie Mellon University, Pittsburgh, America. His main research interests include spatio-temporal model and recommendation system.



**Stan Z. Li** (Fellow, IEEE) received the BEng degree from Hunan University, Changsha, China, in 1982, the MEng degree from the National University of Defense Technology, Changsha, in 1985, and the PhD degree from the University of Surrey, Guildford, U.K., in 1991. He was the director of the Center for Biometrics and Security Research (CBSR), Chinese Academy of Sciences, 2004–2019. He worked with Microsoft Research Asia as a research lead, 2000–2004. Prior to that, he was an associate professor (tenure) with Nanyang Technological University, Singapore. He joined Westlake University as a chair professor of artificial intelligence in 2019. His main research interests include fundamental research in machine learning, data science, and applied research in multiple AI-related interdisciplinary fields (computer vision, smart sensors, life science, material science, and environmental science). He has published more than 400 papers in international journals and conferences, and authored and edited 10 books, with more than 50,000 Google Scholar citations. Among these are *Markov Random Field Models in Image Analysis* (Springer), *Handbook of Face Recognition* (Springer) and *Encyclopedia of Biometrics* (Springer). He served as an associate editor of *IEEE Transactions on Pattern Analysis and Machine Intelligence* and organized more than 100 international conferences or workshops.