ProGMLP: A Progressive Framework for GNN-to-MLP Knowledge Distillation with Efficient Trade-offs

Weigang Lu[1], Ziyu Guan[2*], Wei Zhao[2], Yaming Yang[2], Yujie Sun[2], Zheng Liang[1], Yibing Zhan[3], Dapeng Tao[4]

[1]Department of Civil and Environmental Engineering, The Hong Kong University of Science and Technology, Hong Kong SAR
[2]School of Computer Science and Technology, Xidian University, Xi'an, China
[3]JD Explore Academy, Beijing, China
[4]School of Information Science and Engineering, Yunnan University, Kunming, China
weiganglu@ust.hk, {zyguan@, ywzhao@mail., yym@, yujsun@stu.}xidian.edu.cn, liangz@ust.hk, zhanyibing@jd.com, dapeng.tao@gmail.com

Abstract

GNN-to-MLP (G2M) methods have emerged as a promising approach to accelerate Graph Neural Networks (GNNs) by distilling their knowledge into simpler Multi-Layer Perceptrons (MLPs). These methods bridge the gap between the expressive power of GNNs and the computational efficiency of MLPs, making them well-suited for resource-constrained environments. However, existing G2M methods are limited by their inability to flexibly adjust inference cost and accuracy dynamically, a critical requirement for real-world applications where computational resources and time constraints can vary significantly. To address this, we introduce a Progressive framework designed to offer flexible and on-demand trade-offs between inference cost and accuracy for GNN-to-MLP knowledge distillation (ProGMLP). ProGMLP employs a Progressive Training Structure (PTS), where multiple MLP students are trained in sequence, each building on the previous one. Furthermore, ProGMLP incorporates Progressive Knowledge Distillation (PKD) to iteratively refine the distillation process from GNNs to MLPs, and Progressive Mixup Augmentation (PMA) to enhance generalization by progressively generating harder mixed samples. Our approach is validated through comprehensive experiments on eight real-world graph datasets, demonstrating that ProGMLP maintains high accuracy while dynamically adapting to varying runtime scenarios, making it highly effective for deployment in diverse application settings.

Code — https://github.com/WeigangLu/ProGMLP-main

Introduction

GNN-to-MLP (**G2M**) methods have recently gained attention as an effective approach for accelerating the inference of Graph Neural Networks (GNNs) (Kipf and Welling 2016; Veličković et al. 2017; Hamilton, Ying, and Leskovec 2017; Wu et al. 2019; Xu et al. 2018a; Klicpera, Bojchevski, and Günnemann 2018; Yang et al. 2022a; Lu et al. 2024c) by distilling their knowledge into simpler Multi-Layer Perceptrons (MLPs). These methods typically involve training a single MLPs (Zhang et al. 2021; Tian et al. 2022; Wu et al. 2023b,a), or a set of MLPs (Lu et al. 2024a), to mimic GNN predictions. They aim to bridge the gap between the expressive power of GNNs and the computational efficiency of MLPs, making them promising for deployment in resource-constrained environments or latency-sensitive applications. By pre-compiling graph knowledge into MLPs, G2M techniques significantly reduce the need for explicit graph traversal and neighborhood aggregation during inference, leading to faster prediction times.

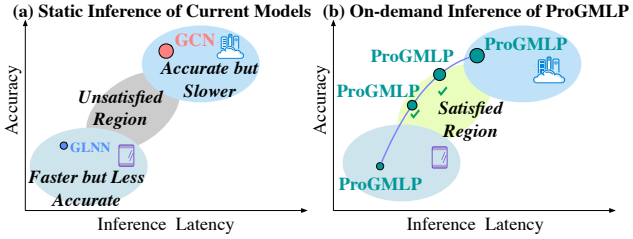


Figure 1: Figure 1: ProGMLP Motivation. (a) Existing static models offer only a single, fixed accuracy-latency trade-off, failing to meet dynamic application needs. (b) ProGMLP provides a single, adaptive framework that enables flexible, on-demand trade-offs, offering a spectrum of operating points to satisfy diverse requirements.

However, a critical limitation of existing G2M approaches is their lack of flexibility in balancing inference cost and accuracy. Current methods are designed to operate within fixed computational budgets, which means they cannot dynamically adjust to the varying demands of different application scenarios. As illustrated in Figure 1 (a), this creates a fundamental mismatch. System operators are forced to make a preemptive choice: deploy a fast, low-accuracy model suitable for edge devices, or a slow, high-accuracy model for server-side analytics. In real-world settings, the ability to control the trade-off between inference accuracy and computational cost on-demand during execution is crucial. For instance, in edge computing (Cao et al. 2020; Mao et al. 2017; Chen and Ran 2019; Shi et al. 2016) or mobile environments (Hoehle

*Corresponding Author

and Venkatesh 2015; Islam, Islam, and Mazumder 2010), where computational power and energy resources can fluctuate, a model that can adaptively tune its inference process to optimize for either speed or accuracy would be highly beneficial. During peak loads, it may be necessary to serve predictions with lower latency at the cost of a slight accuracy drop to meet throughput demands. This reality necessitates the use of early-exit (Bolukbasi et al. 2017; Dennis et al. 2018) or anytime inference (Ruiz and Verbeek 2021; Huang et al. 2017; Dennis et al. 2023) applications, where the inference process may be interrupted due to changing resource availability. Unfortunately, this aspect of flexibility has been largely overlooked in existing G2M methods, limiting their practical utility.

To address this gap, we propose **ProGMLP**, a progressive framework designed to offer flexible and on-demand trade-offs between inference cost and accuracy in the context of G2M. The motivation behind ProGMLP is to enable users to dynamically adjust the inference process based on the specific needs of their applications, whether it be maximizing accuracy under loose time constraints or minimizing inference time when computational resources are scarce. As visualized in Figure 1 (b), ProGMLP is not a single-point solution but an adaptive framework that spans a spectrum of operating points on the accuracy-latency curve. ProGMLP achieves this through a sequence of progressively trained MLPs, each building on the knowledge of the previous one, and incorporates a performance-time budgeting mechanism that intelligently manages when to stop inference based on real-time performance metrics.

Our contributions can be summarized as follows:

- We propose ProGMLP, the first framework to address flexible trade-offs between accuracy and inference cost in the G2M paradigm, allowing dynamic runtime adjustments based on application needs, such as computational resources or time constraints.
- ProGMLP features a novel design with Progressive Training, Progressive Knowledge Distillation, and Progressive Mixup Augmentation, refining each student model for improved task performance.
- We evaluate ProGMLP on eight real-world graphs, demonstrating that it delivers high accuracy while enabling flexible control over inference costs, making it practical for diverse runtime scenarios.

Preliminaries

Notations

We define a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, Y\}$, where $\mathcal{V}$ is the set of N nodes and $\mathcal{E}$ is the set of edges. The node features are represented by the matrix $X \in \mathbb{R}^{N \times d}$, where x_i is the d-dimensional feature vector for node i. The one-hot label matrix for C classes is $Y \in \mathbb{R}^{N \times C}$, and the graph structure is encoded by the adjacency matrix $A \in \mathbb{R}^{N \times N}$, where $A_{ij} = 1$ if an edge exists between nodes i and j, and 0 otherwise. For the node classification task, we divide the nodes into labeled ($\mathcal{V}^L$) and unlabeled ($\mathcal{V}^U$) subsets. In this paper, uppercase letters (e.g., X) denote matrices, while lowercase letters (e.g., x_i) denote row vectors from these matrices.

Related Works

Graph Neural Networks. Graph Neural Networks (GNNs) (Kipf and Welling 2016; Veličković et al. 2017; Hamilton, Ying, and Leskovec 2017; Wu et al. 2019; Xu et al. 2018a; Klicpera, Bojchevski, and Günnemann 2018; Yang et al. 2022a; Chen et al. 2020a; Xu et al. 2018b; Du et al. 2024) are a class of models that leverage the graph structure to learn node representations. A GNN typically follows an iterative message-passing paradigm, where each node in the graph aggregates information from its neighboring nodes to update its own representation. Through multiple layers, GNNs enable the learning of node embeddings that incorporate not only the node's features but also the structural information from its local and extended neighborhood. This mechanism allows GNNs to generalize well to diverse tasks such as node classification (Lu et al. 2024c, 2023, 2024b, 2025), link prediction (Shomer et al. 2024), and graph classification (Yang et al. 2022b). However, despite their efficacy, GNNs face several challenges, particularly in terms of scalability and inference efficiency. As the size and complexity of real-world graphs grow, GNNs become increasingly expensive to train and deploy due to the high computational and memory costs associated with processing large graphs and multiple layers of message passing. This issue is further exacerbated when GNNs are deployed in resource-constrained environments, such as edge devices or mobile platforms, where both computation and memory are limited.

GNN-to-GNN Knowledge Distillation. To address the efficacy issue in GNNs, GNN-to-GNN Knowledge Distillation (G2G KD) (Lassance et al. 2020; Zhang et al. 2023; Ren et al. 2021; Joshi et al. 2022; Wu et al. 2022a; Zhang et al. 2019; Ren et al. 2021; Chen et al. 2020b) has been extensively studied. They aim to compress large, complex Graph Neural Networks (GNNs) into smaller, more efficient GNN models. These approaches leverage KD techniques (Hinton, Vinyals, and Dean 2015; Ba and Caruana 2014) to transfer the knowledge embedded in a large teacher GNN to a smaller student GNN. For example, methods like LSP (Yang et al. 2020) and TinyGNN (Yan et al. 2020) facilitate the transfer of localized structural information from teacher GNNs to their student counterparts. Similarly, RDD (Zhang et al. 2020) enhances G2G KD by considering the reliability of nodes and edges. Although effective, these methods often require neighbor fetching during inference, which can introduce latency and make them less practical for real-time or resource-constrained applications.

GNN-to-MLP Knowledge Distillation. To address the inference efficacy issue in G2G KD, GNN-to-MLP (G2M) KD has emerged as a promising alternative. This approach transfers the knowledge of a GNN to a simpler MLPs model, which eliminates the need for message passing during inference and thus significantly reduces latency. Early work such as GLNN (Zhang et al. 2021) introduces a general G2M framework where an MLPs student is trained using both ground-truth and soft labels from a GNN teacher. Subsequent methods like KRD (Wu et al. 2023b) introduces a reliable sampling strategy to improve the quality of the knowledge transferred, while NOSMOG (Tian et al. 2022)

and GSDN (Wu et al. 2022b) incorporates structural information to further enhance the MLP student's performance. VQGraph (Yang et al. 2024) proposes to learn a codebook that represents informative local structures, and uses these local structures as additional information for distillation. The most related work is AdaGMLP (Lu et al. 2024a) which adopts an AdaBoosting ensemble framework to train multiple MLPs, collecting all the knowledge from each student to make predictions.

Motivation

In resource-constrained environments, it is often crucial to balance the trade-off between computational cost and model accuracy. For example, early exit strategies (Bolukbasi et al. 2017; Dennis et al. 2018; Laskaridis, Kouris, and Lane 2021; Teerapittayanon, McDanel, and Kung 2016) have been extensively studied to allow models to terminate inference early if a satisfactory prediction can be made to improve energy efficiency. This is particularly useful in scenarios where the computational budget can be variable, but still allows the model to make high-confidence predictions in cases where the input is easy to classify. Early exit mechanisms are proactive in reducing computation by identifying simpler cases early in the inference process, improving efficiency without significantly sacrificing accuracy.

On the other hand, anytime inference strategies (Ruiz and Verbeek 2021; Huang et al. 2017; Dennis et al. 2023) have also been proposed to provide flexible, interruptible inference, where a model can yield a prediction even when inference is interrupted, e.g., due to the time limitation. In this scenario, the model is expected to provide a prediction, even if it has not completed the entire inference process. The main focus here is ensuring the model can yield a usable (though potentially suboptimal) prediction at any moment, allowing for adaptive responses in real-time systems where the computation time is constrained or unpredictable.

However, to our knowledge, current G2M methods ignore the urgent need for a more flexible and adaptive approach. Our ProGMLP is designed to fill this gap by offering a progressive training and inference mechanism that enables dynamic adjustment of the trade-off between accuracy and inference cost. This makes ProGMLP particularly suitable for real-world applications where computational resources are limited, and the ability to control inference time is critical.

Methodology

Overview

ProGMLP is an ensemble framework designed to distill knowledge from a pre-trained GNN into a sequence of progressively trained MLPs, as illustrated in Figure 2. The progressive training structure is established by initializing each student MLP f_{k+1} with the parameters θ_k of the previously trained student f_k. A pre-trained GNN serves as the teacher, guiding the learning of this MLP sequence. For each student MLP f_{k+1}, the input comprises both the original node features X and the hidden representation H_k output by the previous MLP. The initial hidden representation H_0 is a zero matrix. Each MLP is trained using a combination of three

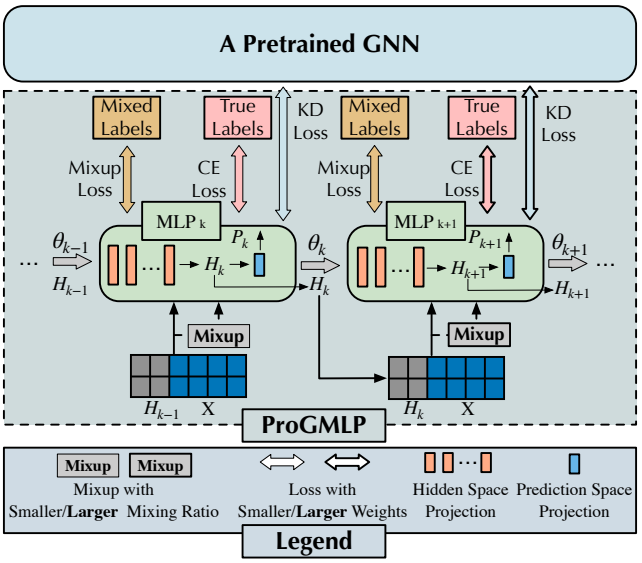


Figure 2: The training architecture of ProGMLP.

loss functions: Mixup Loss (from interpolated samples and labels), Cross-Entropy Loss (from predictions and ground-truth labels), and Knowledge Distillation (KD) Loss (from the discrepancy between the teacher GNN and the student MLP outputs). This training process proceeds iteratively, with each subsequent MLP student refining the knowledge distilled by its predecessor. The mixup strategy employs an increasingly stronger mixing ratio over time, while the loss weights for KD and mixup components are progressively increased for later MLPs. This encourages deeper and more robust learning.

Progressive Training Structure

The progressive training structure is at the heart of ProGMLP. Unlike conventional G2M methods that train a single student model, ProGMLP trains a sequence of student MLPs, each one initialized with the parameters of the previously trained model. This progressive approach ensures that the later models in the sequence start from a more advanced state and are capable of tackling more complex tasks. A L-layer MLPs student f_k, consisting of a $(L-1)$-layer fully connected network (FCNs) with the same hidden dimensionality for latent space projection and 1-layer FCN for prediction space projection, can be described as:

$$H_k, P_k = f_k(X_k), \quad H_k \in \mathbb{R}^{N\times d'}, P_k \in \mathbb{R}^{N\times C}, \qquad (1)$$

where $X_k = \mathrm{CONCAT}(X, H_{k-1})$ is the input consisting of raw features X and hidden representations H_{k-1} from the previous student. Here, $H_0 = \mathbf{O} \in \mathbb{R}^{N\times d'}$ is a zero matrix. First, f_k projects input X_k into the hidden space within the first $(L-1)$-layer FCNs and then maps the hidden representations H_k into the prediction space to obtain the predictions P_k via the last FCN. The first MLPs student f_1 is trained with random initialization. Then, after training f_k, its parameters are used to initialize the next student f_{k+1}. The process is repeated for each subsequent student model, progressively refining the learning process. The relationship

between these students can be expressed as:

$$\theta_k \leftarrow \mathbf{Train}(f_{k-1}|\theta_{k-1}), \tag{2}$$

where θ_{k-1} represents the parameters from the $(k-1)$-th student . For the k-th MLPs student, the parameters are optimized guided by the following loss function:

$$\mathcal{L}_k = \mathcal{L}_k^{PKD} + \mathcal{L}_k^{PMA}, \tag{3}$$

where $\mathcal{L}_k^{PKD}$ is the progressive knowledge distillation loss defined by Eq. (4) and $\mathcal{L}_k^{PMA}$ is the progressive mixup loss defined by Eq. (6). We will introduce them in the following sections.

The idea behind this approach is to incrementally build on the learned knowledge, much like how a Recurrent Neural Network (RNN) propagates hidden states through time steps. By initializing each student model with the parameters of its predecessor, ProGMLP allows the knowledge to be gradually refined, leading to models that are both more accurate and more capable of handling diverse inputs.

Progressive Knowledge Distillation

Progressive Knowledge Distillation is a combination of the cross-entropy loss and the Kullback-Leibler (KL) divergence (Hinton, Vinyals, and Dean 2015; Ba and Caruana 2014), allowing each student model to refine its predictions based on both the true labels and the predictions of the previous model. In this context, each student in the sequence is expected to improve upon the predictions of the previous one. The loss function for student k can be defined as:

$$\begin{aligned}\mathcal{L}_k^{PKD} &= k^\beta \left(\alpha \mathcal{L}_k^{CE} + (1-\alpha)\mathcal{L}_k^{KD}\right) \\ &= k^\beta \Big(\frac{\alpha}{|\mathcal{V}^L|}\sum_{i\in\mathcal{V}^L} \ell^{CE}\left(f_k(x_i), y_i\right) \\ &\quad + \frac{1-\alpha}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} \ell^{KD}\left(z_i^g, f_k(x_i)\right)\Big),\end{aligned} \tag{4}$$

where ℓ^{CE} is the cross-entropy loss between the predictions and true labels and ℓ^{KD} is the KL divergence between the current student predictions $f_k(x)$ and the GNNs' predictions z^g. β is a factor that increases the weight of the loss for later student in the sequence, encouraging more accurate predictions and α is the weighting factor that balances the distillation loss and the standard cross-entropy loss. The loss function is applied to each student output, but with different weights k^β. The later students in the sequence are assigned higher weights, incentivizing them to make more accurate predictions.

Progressive Mixup Augmentation

The Progressive Mixup Augmentation in ProGMLP is designed to gradually increase the difficulty of the learning tasks assigned to each student model. Mixup is a data augmentation technique that generates new training examples by linearly interpolating between pairs of examples. In ProGMLP, we produce the progressive hard examples for student $k \geq 1$ as follows:

$$\begin{aligned}x_{ij} &= \lambda\,\mathrm{CONCAT}(x_i, H_{k-1}[i,:]) \\ &\quad + (1-\lambda)\,\mathrm{CONCAT}(x_j, H_{k-1}[j,:]) \\ y_{ij} &= \lambda y_i + (1-\lambda) y_j,\end{aligned} \tag{5}$$

where λ is the mixing rate and $H_{k-1}[i,:]$ is the i-th row vector of hidden representations H_{k-1}. Then, the progressive mixup loss $\mathcal{L}^{PMA}$ for mixed samples is:

$$\mathcal{L}_k^{PMA} = \frac{1}{|\mathcal{V}^L|}\sum_{(i,j)\in\mathcal{D}_k^m} \ell^{CE}\left(f_k(x_{ij}), y_{ij}\right), \tag{6}$$

where $\mathcal{D}_k^m$ is the mixup pair set for the k-th student MLPs.

From Eq. (5), *we can see that a λ close to 0 results in examples that are almost identical to one of the original examples, while a λ close to 0.5 results in examples that are more challenging and significantly different from either of the original examples.* By adjusting λ over time, we can control the difficulty of the examples presented to each MLPs student in the ProGMLP sequence:

$$\lambda_k \leftarrow \min\left(\max\left(\lambda_{k-1} + \gamma(\bar{\ell} - \tau), 0\right), 0.5\right) \tag{7}$$

Here, λ_k is the mixing factor for the k-th MLPs student, $\gamma > 0$ is an adjustment rate which determines how sensitively λ responds to changes in the model's learning process, $\bar{\ell}$ is the moving average of the mixup loss for the current student, and τ is a predefined threshold that acts as a reference point for adjusting λ. When the moving average loss is close to τ, λ remains stable; when the loss decreases (indicating better performance), λ increases. In this paper, we set $\tau = 0.1$.

$\bar{\ell}$ is typically calculated using an exponential moving average (EMA), which gives more weight to recent losses while still considering past losses. The formula for calculating the exponential moving average at time step t (iteration) is:

$$\bar{\ell}_t = \sigma\bar{\ell}_{t-1} + (1-\sigma)\bar{\ell}_t, \tag{8}$$

where $\bar{\ell}_t$ is the moving average loss at the iteration t and σ is a smoothing factor between 0 and 1, controlling how much weight is given to past losses. A higher σ value means $\bar{\ell}$ changes more slowly, giving more weight to past losses. $\bar{\ell}_0$ is initialized at the current mixup loss calculated by Eq. (4). In this paper, we set $\sigma = 0.1$.

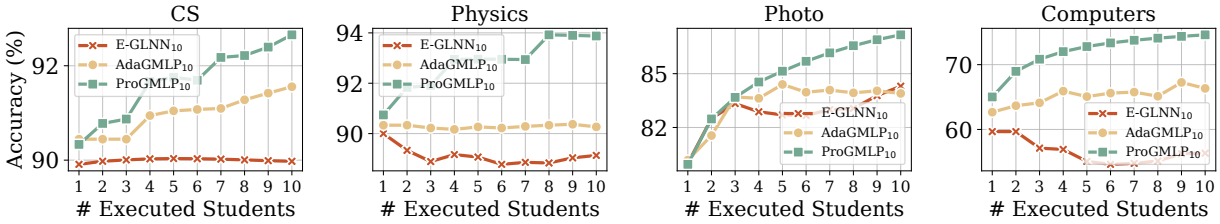
As the student models become more capable, they are exposed to more challenging examples, which pushes them to learn more complex patterns and generalize better. This approach aligns with the progressive nature of ProGMLP, where each model is expected to handle more difficult tasks than its predecessor.

Train and Inference

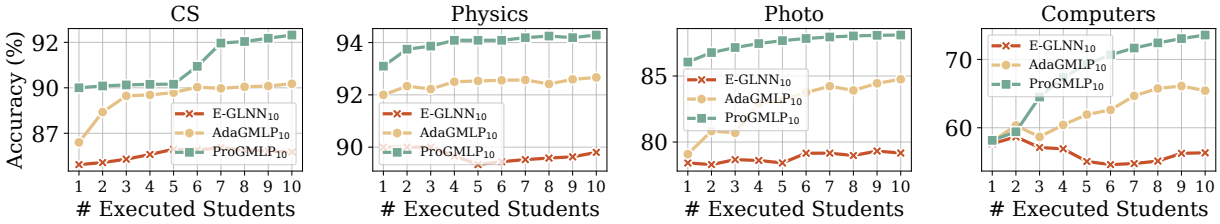
Training. ProGMLP consists of K student MLPs, where the k-th student f_k is trained using the loss function $\mathcal{L}_k$, as defined in Eq. (3). Each student is trained for up to E_1 epochs, with early stopping applied based on a patience criterion of E_2 epochs, where $E_2 < E_1$. This ensures that the training process is efficient and prevents overfitting.

Inference. ProGMLP evaluates student models sequentially during inference, stopping early when a confidence threshold is met to balance speed and accuracy. The process stops when the confidence of the k-th student, denoted as c_k, exceeds a predefined threshold τ^{conf}. The confidence is computed as the mean of the highest softmax probabilities across all unlabeled nodes:

$$c_k = \underset{i\in\mathcal{V}^U}{\mathrm{mean}}\left(\max_{j\in 1,\ldots,C} \mathrm{SOFTMAX}\left(f_k(x_i)\right)_j\right). \tag{9}$$



(a) Teacher = GCN



(b) Teacher = GraphSAGE

Figure 3: Accuracy vs. Inference Cost (# Number of Executed Students) for ProGMLP, AdaGMLP, and GLNN.

If $c_k < \tau^{conf}$, ProGMLP proceeds to the next student f_{k+1}. The output P is a weighted sum of predictions from the evaluated models:

$$P = \sum_{j=1}^{k} w_j P_j, \quad w_k = \text{SOFTMAX}(\{c_1, c_2, \cdots, c_k\})_k, \tag{10}$$

This ensures efficient and adaptive inference for varying conditions.

Complexity

The time complexity of the ProGMLP framework is primarily determined by the number of MLPs students K. The time complexity of training each MLPs student mainly comes from: (1) feature forward $O(Nd'(d + d' + C))$; (2) knowledge distillation $O(NC)$; (3) mixup augmentation $O(|\mathcal{V}^L|(d + d'))$, where $|\mathcal{V}^L|$ and d' are largely small compared to N and d, respectively. Therefore, the overall training complexity of ProGMLP can be approximated as $O\left(KN(d'(d + d' + C) + C)\right)$. During inference, ProGMLP evaluates each MLPs student sequentially, stopping the process based on the confidence-time budgeting mechanism. *In the worst-case scenario*, all K MLPs are evaluated. The inference time complexity for a single sample is therefore $O\left(Kd'(d + d' + C)\right)$.

Experiments

In this section, we present a comprehensive set of experiments to evaluate our ProGMLP.

Experimental Setup

Hardware and Software. ProGMLP is implemented based on the Torch Geometric library (Fey and Lenssen 2019) and PyTorch 3.7.1 with Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz and one NVIDIA A100 GPUs with 40GB memory.

Dataset	# Nodes	# Edges	# Features	# Classes
Cora	2,708	5,278	1,433	7
Pubmed	19,717	44,324	500	3
Amazon Photo	7,650	119,081	745	8
Amazon Computers	13,381	245,778	767	10
Coauthor CS	18,333	81,894	6,805	15
Coauthor Physics	34,493	247,962	8,415	5
ogbn-arxiv	169,343	1,166,243	128	40
ogbn-products	2,449,029	61,859,140	128	47

Table 1: Datasets Statics.

Datasets. To comprehensively evaluate the performance, generalizability, and scalability of ProGMLP, we have selected *eight* widely-adopted real-world graph datasets: Cora, Pubmed (Sen et al. 2008), Amazon Photo, Amazon Computers, Coauthor CS, Coauthor Physics (Shchur et al. 2018), ogbn-arxiv, and ogbn-products (Hu et al. 2020). These datasets exhibit diversity in node features, graph structures, and task complexities, offering a comprehensive benchmark for evaluating the generalizability of our approach. We provide the statistical summaries in Table 1.

Teacher Models. For the teacher GNN models, we select three of the most representative architectures: GCN (Kipf and Welling 2016), GAT (Veličković et al. 2017), and GraphSAGE (Hamilton, Ying, and Leskovec 2017). Each GNN is implemented with a standard 2-layer structure to ensure a fair and consistent comparison across experiments.

Ensemble Student Models. To compare ProGMLP in settings that require early-exit or anytime inference, we adapt two representative G2M methods, GLNN (Zhang et al. 2021) and AdaGMLP (Lu et al. 2024a) , which normally only produce output upon complete execution. The adapted versions are as follows:

- E-GLNN$_K$: This is an ensemble of K students based on the GLNN method. For an early exit, it averages the

Teacher +Student	Cora	Pubmed	CS	Physics	Photo	Computers	***Impro.***
GCN	$79.03_{\pm 0.37}$	$76.66_{\pm 0.35}$	$90.68_{\pm 0.17}$	$93.59_{\pm 0.59}$	$88.57_{\pm 0.83}$	$77.82_{\pm 0.88}$	0.00%
+E-GLNN$_2$	$79.28_{\pm 0.62}$	$76.60_{\pm 0.48}$	$89.88_{\pm 0.16}$	$92.11_{\pm 0.82}$	$84.42_{\pm 1.18}$	$75.39_{\pm 1.17}$	-1.67%
+E-GLNN$_4$	$79.30_{\pm 0.83}$	$76.98_{\pm 0.53}$	$89.79_{\pm 0.19}$	$92.65_{\pm 0.33}$	$84.73_{\pm 1.64}$	$75.82_{\pm 1.12}$	-1.36%
+AdaGMLP$_2$	$79.15_{\pm 0.19}$	$76.84_{\pm 0.64}$	$90.53_{\pm 0.12}$	$92.58_{\pm 0.26}$	$84.49_{\pm 1.53}$	$76.34_{\pm 1.26}$	-1.23%
+AdaGMLP$_4$	$79.36_{\pm 1.12}$	$77.21_{\pm 0.66}$	$91.69_{\pm 0.37}$	$92.71_{\pm 0.29}$	$85.26_{\pm 1.61}$	$77.11_{\pm 1.19}$	-0.56%
+ProGMLP	$\mathbf{80.19}_{\pm 0.39}$	$\mathbf{77.42}_{\pm 0.35}$	$\mathbf{92.43}_{\pm 0.12}$	$\mathbf{94.23}_{\pm 0.04}$	$\mathbf{90.91}_{\pm 0.77}$	$\mathbf{78.82}_{\pm 0.44}$	**+1.50%**
GAT	$78.47_{\pm 2.77}$	$75.71_{\pm 0.46}$	$90.42_{\pm 0.66}$	$92.88_{\pm 0.24}$	$86.48_{\pm 1.51}$	$76.82_{\pm 1.23}$	0.00%
+E-GLNN$_2$	$78.94_{\pm 2.05}$	$76.84_{\pm 0.76}$	$89.72_{\pm 0.70}$	$90.63_{\pm 0.95}$	$82.66_{\pm 1.69}$	$74.78_{\pm 2.16}$	-1.36%
+E-GLNN$_4$	$79.04_{\pm 2.12}$	$77.08_{\pm 0.44}$	$90.66_{\pm 0.41}$	$90.98_{\pm 0.48}$	$83.01_{\pm 1.63}$	$74.54_{\pm 2.34}$	-1.04%
+AdaGMLP$_2$	$78.95_{\pm 2.44}$	$76.92_{\pm 0.79}$	$90.42_{\pm 0.25}$	$92.94_{\pm 0.18}$	$84.39_{\pm 1.86}$	$76.85_{\pm 1.17}$	-0.02%
+AdaGMLP$_4$	$79.95_{\pm 2.50}$	$77.25_{\pm 0.46}$	$90.77_{\pm 0.14}$	$93.07_{\pm 0.29}$	$85.64_{\pm 1.69}$	$77.31_{\pm 1.38}$	+0.70%
+ProGMLP	$\mathbf{80.50}_{\pm 2.08}$	$\mathbf{77.63}_{\pm 0.82}$	$\mathbf{91.83}_{\pm 0.44}$	$\mathbf{94.16}_{\pm 0.14}$	$\mathbf{88.94}_{\pm 1.64}$	$\mathbf{79.16}_{\pm 1.29}$	**+2.33%**
GraphSAGE	$78.56_{\pm 0.64}$	$75.39_{\pm 0.49}$	$91.84_{\pm 0.46}$	$92.37_{\pm 1.44}$	$86.54_{\pm 0.69}$	$79.32_{\pm 0.31}$	0.00%
+E-GLNN$_2$	$78.37_{\pm 0.80}$	$76.09_{\pm 0.65}$	$91.11_{\pm 0.13}$	$92.24_{\pm 0.41}$	$84.55_{\pm 1.16}$	$76.28_{\pm 0.29}$	-1.06%
+E-GLNN$_4$	$78.32_{\pm 0.97}$	$76.95_{\pm 0.79}$	$90.62_{\pm 0.36}$	$92.41_{\pm 0.39}$	$84.78_{\pm 2.38}$	$76.14_{\pm 0.32}$	-0.93%
+AdaGMLP$_2$	$78.39_{\pm 0.76}$	$77.01_{\pm 0.84}$	$92.41_{\pm 0.20}$	$92.67_{\pm 0.29}$	$85.16_{\pm 0.84}$	$77.94_{\pm 1.41}$	-0.08%
+AdaGMLP$_4$	$78.84_{\pm 0.98}$	$77.31_{\pm 0.29}$	$92.79_{\pm 0.21}$	$\mathbf{94.20}_{\pm 0.30}$	$86.98_{\pm 0.72}$	$79.23_{\pm 1.06}$	+1.05%
+ProGMLP	$\mathbf{79.28}_{\pm 0.86}$	$\mathbf{77.84}_{\pm 1.05}$	$\mathbf{93.13}_{\pm 0.10}$	$93.70_{\pm 0.16}$	$\mathbf{87.48}_{\pm 1.74}$	$\mathbf{80.21}_{\pm 1.21}$	**+1.54%**

Table 2: Comparison with Ensemble G2M Methods.

predictions from all students that have finished executing.

- AdaGMLP$_K$: This is an ensemble framework that uses the AdaBoost algorithm. If stopped early, its prediction is the re-normalized, weighted sum of the outputs from the already-executed students

Non-Ensemble G2M Methods. Additionally, we choose three state-of-the-art (SOTA) non-ensemble G2M methods:

- NOSMOG (Tian et al. 2022): It aims to enhance the MLP student's performance by incorporating structural information from the graph.
- KRD (Wu et al. 2023b): It focuses on improving the quality of the knowledge transferred from the GNN teacher to the MLP student.
- HGMD (Wu et al. 2024): It decouples and estimates two types of distillation hardness and knowledge hardness to better transfer knowledge from GNNs to MLPs.

Hyperparameters. We search learning rate in $\{0.001, 0.002, 0.005, 0.01, 0.02, 0.05\}$, hidden dimensionality in $\{16, 32, 64, 128, 256, 512\}$, weight decay rate in $\{5e-4, 5e-5, 5e-7, 5e-9\}$, dropout in $\{0.1, 0.2, \cdots, 0.9\}$, teacher model depth in $\{2, 3\}$, and number of students in $\{2, 3, \cdots, 6\}$, for all the methods.

Main Results

Accuracy-Cost Trade-off. Our central claim is that ProGMLP excels across the entire accuracy-cost spectrum. Figure 3 visualizes the performance as a function of inference cost (number of executed students). The ProGMLP curve (green) consistently dominates the baselines. Notably, it delivers strong accuracy in the low-cost regime (e.g., with 1-3 students) and continues to improve steadily as more computation is permitted. In contrast, E-GLNN plateaus quickly, while AdaGMLP's gains are inefficient. This demonstrates ProGMLP's ability to provide a meaningful, on-demand trade-off, making it highly adaptable to diverse computational budgets.

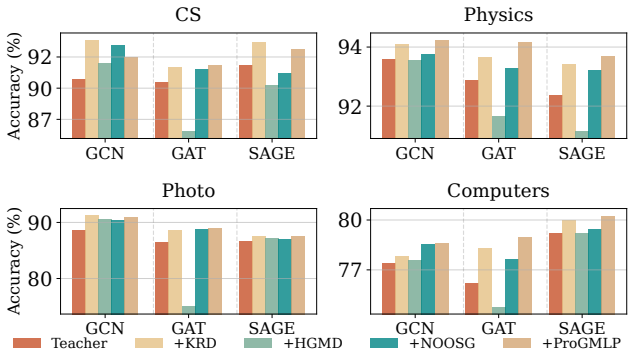


Figure 4: Performance Comparison against G2M Methods.

Peak Performance. Beyond the trade-off curve, ProGMLP also achieves state-of-the-art peak accuracy. Tables 2 and Figure 4 show that ProGMLP's final performance is consistently superior or highly competitive against both ensemble and single-student baselines across all datasets and teacher models. This confirms that our progressive approach does not sacrifice peak performance for flexibility.

	ogbn-arxiv		ogbn-products	
	Acc.	Inf. Time	Acc.	Inf. Time
GCN	$68.42_{\pm 0.67}$	84.13	$72.94_{\pm 0.14}$	274.77
+ProGMLP	$\mathbf{70.11}_{\pm 0.74}$	3.95	$\mathbf{73.00}_{\pm 0.12}$	52.21
Impro.	+2.47%	↑ 21×	+0.08%	↑ 5×
GAT	$69.25_{\pm 1.26}$	97.81	OOM	-
+ProGMLP	$\mathbf{70.29}_{\pm 1.94}$	12.80	-	-
Impro.	+1.50%	↑ 8×	-	-
GraphSAGE	$69.70_{\pm 0.21}$	27.99	$75.59_{\pm 0.10}$	139.21
+ProGMLP	$\mathbf{70.97}_{\pm 0.85}$	6.93	$\mathbf{75.68}_{\pm 0.07}$	59.06
Impro.	+1.80%	↑ 4×	+0.01%	↑ 2×

Table 3: Node Classification Accuracy (%) and Inference Time (ms) on Large-scale Graphs. "↑ m ×" indicates ProGMLP is m times faster than the teacher at the inference stage.

Scalability and Generalization

We further test ProGMLP on large-scale graphs and in an inductive setting to verify its practicality.

Large-scale Graphs. On the OGB datasets (Table 3), ProGMLP not only improves accuracy over strong GNN teachers but also delivers massive inference speedups: up to 21× faster than GCN on ogbn-arxiv. This result highlights its suitability for real-world, large-scale applications where inference latency is a critical bottleneck.

Inductive Setting. When required to generalize to unseen nodes (Table 4), ProGMLP again demonstrates substantial improvements over its GNN teachers (e.g., +9.85% over GCN on Pubmed). This suggests our proposed PKD and PMA components effectively distill robust, generalizable knowledge, not just memorizing patterns in a transductive setting.

	Cora	Pubmed	CS	Physics
GCN	$70.31_{\pm 0.54}$	$80.06_{\pm 0.37}$	$90.93_{\pm 0.35}$	$93.27_{\pm 0.26}$
+ProGMLP	$\mathbf{73.07}_{\pm 0.66}$	$\mathbf{87.95}_{\pm 0.36}$	$\mathbf{93.38}_{\pm 0.31}$	$\mathbf{95.44}_{\pm 0.18}$
Impro.	+3.92%	+9.85%	+2.69%	+2.32%
GAT	$71.47_{\pm 1.35}$	$82.67_{\pm 0.88}$	$90.51_{\pm 0.30}$	$93.41_{\pm 0.18}$
+ProGMLP	$\mathbf{72.50}_{\pm 0.91}$	$\mathbf{88.01}_{\pm 0.42}$	$\mathbf{94.02}_{\pm 0.21}$	$\mathbf{95.99}_{\pm 0.20}$
Impro.	+1.44%	+6.46%	+3.88%	+2.76%
GraphSAGE	$69.71_{\pm 1.13}$	$85.56_{\pm 0.54}$	$90.50_{\pm 0.53}$	$93.95_{\pm 0.49}$
+ProGMLP	$\mathbf{71.91}_{\pm 0.71}$	$\mathbf{88.60}_{\pm 0.30}$	$\mathbf{94.80}_{\pm 0.21}$	$\mathbf{95.26}_{\pm 0.15}$
Impro.	+3.16%	+3.55%	+4.75%	+1.39%

Table 4: Inductive Node Classification Accuracy (%).

Ablation and Hyperparameter Analysis

To understand the source of ProGMLP's effectiveness, we ablate its components and analyze its hyperparameters.

Component Ablation. Figure 5 shows that all components are integral to our method's success. The Progressive Training Structure (PTS) is most critical; its removal causes a drastic accuracy drop (4% on CS), confirming that our core idea of sequential knowledge refinement is essential. Progressive Knowledge Distillation (PKD) and Progressive Mixup Augmentation (PMA) provide significant and complementary gains, aiding in effective knowledge transfer and generalization, respectively.

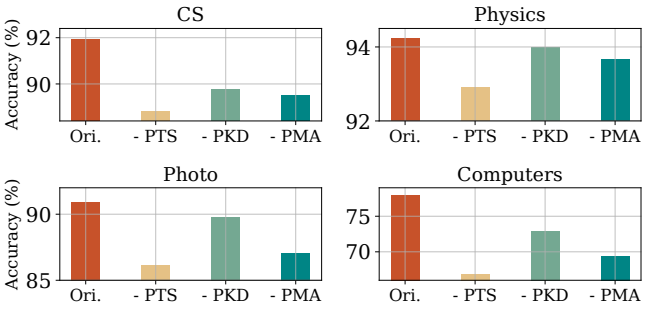


Figure 5: Ablation Study. Each figure compares the accuracy of the original ProGMLP (Ori.) with three ablated versions: without PTS (- PTS) / PKD (- PKD) / PMA (- PMA).

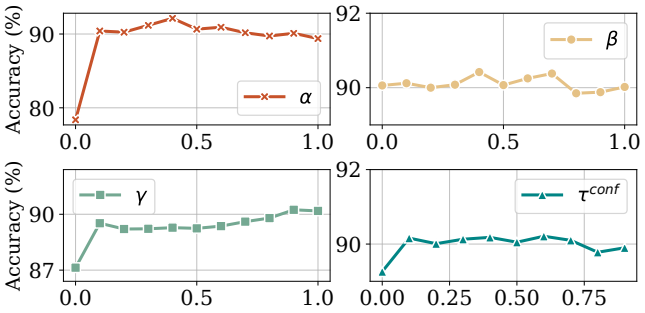


Figure 6: Hyperparameter Sensitivity Analysis.

Hyperparameter Sensitivity. Our analysis in Figure 6 reveals that ProGMLP is robust to most hyperparameter choices. The model's performance is largely stable across different values for β, γ, and the early-exit threshold τ^{conf}. The most sensitive parameter is α, which balances the distillation and ground-truth losses, underscoring the importance of this balance in the distillation process.

Conclusion

In this paper, we present ProGMLP, a novel framework that bridges the gap between the high expressiveness of GNNs and the computational efficiency of MLPs. ProGMLP introduces a progressive learning mechanism that allows for flexible and adaptive trade-offs between inference cost and accuracy. Through comprehensive evaluations on multiple datasets, including large-scale graphs, ProGMLP demonstrates significant improvements in accuracy over existing methods while drastically reducing inference times. The results highlight ProGMLP's suitability for real-world applications where computational resources and time are limited. The framework's ability to scale to large datasets further shows its effectiveness in balancing performance with efficiency.

Acknowledgments

The authors gratefully acknowledge the support from the GREAT Smart City Institute at The Hong Kong University of Science and Technology. This work was supported in part by the National Natural Science Foundation of China under Grants 62425605, 62133012, 62572375 and 62303366, in part by the Key Research and Development Program of Shaanxi under Grants 2025CY-YBXM-041, 2024GXYBXM-122, 2022ZDLGY01-10, and 2024CY2-GJHX-15, and in part by Natural Science Basic Research Program of Shaanxi (Program No. 2025JC-QYXQ-040), and in part by Xidian University Specially Funded Project for Interdisciplinary Exploration (TZJHF202506).

References

Ba, J.; and Caruana, R. 2014. Do deep nets really need to be deep? *Advances in neural information processing systems*, 27.

Bolukbasi, T.; Wang, J.; Dekel, O.; and Saligrama, V. 2017. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, 527–536. PMLR.

Cao, K.; Liu, Y.; Meng, G.; and Sun, Q. 2020. An overview on edge computing research. *IEEE access*, 8: 85714–85728.

Chen, J.; and Ran, X. 2019. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8): 1655–1674.

Chen, M.; Wei, Z.; Huang, Z.; Ding, B.; and Li, Y. 2020a. Simple and deep graph convolutional networks. In *International conference on machine learning*, 1725–1735. PMLR.

Chen, Y.; Bian, Y.; Xiao, X.; Rong, Y.; Xu, T.; and Huang, J. 2020b. On self-distilling graph neural network. *arXiv preprint arXiv:2011.02255*.

Dennis, D.; Pabbaraju, C.; Simhadri, H. V.; and Jain, P. 2018. Multiple instance learning for efficient sequential data classification on resource-constrained devices. *Advances in Neural Information Processing Systems*, 31.

Dennis, D.; Shetty, A.; Sevekari, A. P.; Koishida, K.; and Smith, V. 2023. Progressive ensemble distillation: building ensembles for efficient inference. *Advances in Neural Information Processing Systems*, 36: 43525–43543.

Du, H.; Shi, L.; Chen, X.; Zhao, Y.; Zhang, H.; Yang, C.; Zhuang, F.; and Kou, G. 2024. Representation Learning of Temporal Graphs with Structural Roles. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 654–665.

Fey, M.; and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, 1024–1034.

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

Hoehle, H.; and Venkatesh, V. 2015. Mobile application usability. *MIS quarterly*, 39(2): 435–472.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*.

Huang, G.; Chen, D.; Li, T.; Wu, F.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*.

Islam, R.; Islam, R.; and Mazumder, T. 2010. Mobile application and its global impact. *International Journal of Engineering & Technology*, 10(6): 72–78.

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

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*.

Laskaridis, S.; Kouris, A.; and Lane, N. D. 2021. Adaptive inference through early-exit networks: Design, challenges and directions. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, 1–6.

Lassance, C.; Bontonou, M.; Hacene, G. B.; Gripon, V.; Tang, J.; and Ortega, A. 2020. Deep geometric knowledge distillation with graphs. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8484–8488. IEEE.

Lu, W.; Guan, Z.; Zhao, W.; and Yang, Y. 2024a. AdaGMLP: AdaBoosting GNN-to-MLP Knowledge Distillation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2060–2071.

Lu, W.; Guan, Z.; Zhao, W.; Yang, Y.; and Jin, L. 2024b. Nodemixup: Tackling under-reaching for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 14175–14183.

Lu, W.; Guan, Z.; Zhao, W.; Yang, Y.; Lv, Y.; Xing, L.; Yu, B.; and Tao, D. 2023. Pseudo contrastive learning for graph-based semi-supervised learning. *arXiv preprint arXiv:2302.09532*.

Lu, W.; Guan, Z.; Zhao, W.; Yang, Y.; Zhan, Y.; Lu, Y.; and Tao, D. 2025. Agmixup: Adaptive graph mixup for semi-supervised node classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 19143–19151.

Lu, W.; Zhan, Y.; Lin, B.; Guan, Z.; Liu, L.; Yu, B.; Zhao, W.; Yang, Y.; and Tao, D. 2024c. SkipNode: On Alleviating Performance Degradation for Deep Graph Convolutional Networks. *IEEE Transactions on Knowledge and Data Engineering*, 1–14.

Mao, Y.; You, C.; Zhang, J.; Huang, K.; and Letaief, K. B. 2017. A survey on mobile edge computing: The communication perspective. *IEEE communications surveys & tutorials*, 19(4): 2322–2358.

Ren, Y.; Ji, J.; Niu, L.; and Lei, M. 2021. Multi-task Self-distillation for Graph-based Semi-Supervised Learning. *arXiv preprint arXiv:2112.01174*.

Ruiz, A.; and Verbeek, J. 2021. Anytime inference with distilled hierarchical neural ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 9463–9471.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine*, 29(3): 93–93.

Shchur, O.; Mumme, M.; Bojchevski, A.; and Günnemann, S. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*.

Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; and Xu, L. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5): 637–646.

Shomer, H.; Ma, Y.; Mao, H.; Li, J.; Wu, B.; and Tang, J. 2024. LPFormer: An Adaptive Graph Transformer for Link Prediction. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2686–2698.

Teerapittayanon, S.; McDanel, B.; and Kung, H.-T. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, 2464–2469. IEEE.

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

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*, 6861–6871. PMLR.

Wu, L.; Lin, H.; Huang, Y.; Fan, T.; and Li, S. Z. 2023a. Extracting Low-/High-Frequency Knowledge from Graph Neural Networks and Injecting it into MLPs: An Effective GNN-to-MLP Distillation Framework. *arXiv preprint arXiv:2305.10758*.

Wu, L.; Lin, H.; Huang, Y.; and Li, S. Z. 2022a. Knowledge distillation improves graph structure augmentation for graph neural networks. *Advances in Neural Information Processing Systems*, 35: 11815–11827.

Wu, L.; Lin, H.; Huang, Y.; and Li, S. Z. 2023b. Quantifying the Knowledge in GNNs for Reliable Distillation into MLPs. *arXiv preprint arXiv:2306.05628*.

Wu, L.; Liu, Y.; Lin, H.; Huang, Y.; and Li, S. Z. 2024. Teach Harder, Learn Poorer: Rethinking Hard Sample Distillation for GNN-to-MLP Knowledge Distillation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, 2554–2563. New York, NY, USA: Association for Computing Machinery. ISBN 9798400704369.

Wu, L.; Xia, J.; Lin, H.; Gao, Z.; Liu, Z.; Zhao, G.; and Li, S. Z. 2022b. Teaching Yourself: Graph Self-Distillation on Neighborhood for Node Classification. *arXiv preprint arXiv:2210.02097*.

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

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018b. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, 5453–5462. PMLR.

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

Yang, L.; Tian, Y.; Xu, M.; Liu, Z.; Hong, S.; Qu, W.; Zhang, W.; Bin, C.; Zhang, M.; and Leskovec, J. 2024. VQGraph: Rethinking graph representation space for bridging GNNs and MLPs. In *The Twelfth International Conference on Learning Representations*.

Yang, Y.; Guan, Z.; Zhao, W.; Lu, W.; and Zong, B. 2022a. Graph substructure assembling network with soft sequence and context attention. *IEEE Transactions on Knowledge and Data Engineering*, 35(5): 4894–4907.

Yang, Y.; Guan, Z.; Zhao, W.; Lu, W.; and Zong, B. 2022b. Graph substructure assembling network with soft sequence and context attention. *IEEE Transactions on Knowledge and Data Engineering*, 35(5): 4894–4907.

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

Zhang, H.; Lin, S.; Liu, W.; Zhou, P.; Tang, J.; Liang, X.; and Xing, E. P. 2023. Iterative graph self-distillation. *IEEE Transactions on Knowledge and Data Engineering*.

Zhang, L.; Song, J.; Gao, A.; Chen, J.; Bao, C.; and Ma, K. 2019. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 3713–3722.

Zhang, S.; Liu, Y.; Sun, Y.; and Shah, N. 2021. Graph-less Neural Networks: Teaching Old MLPs New Tricks Via Distillation. In *International Conference on Learning Representations*.

Zhang, W.; Miao, X.; Shao, Y.; Jiang, J.; Chen, L.; Ruas, O.; and Cui, B. 2020. Reliable Data Distillation on Graph Convolutional Network. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, 1399–1414. New York, NY, USA: Association for Computing Machinery. ISBN 9781450367356.