AdapterGNN: Parameter-Efficient Fine-Tuning Improves Generalization in GNNs

Shengrui Li[1,2*], **Xueting Han**[2†], **Jing Bai**[2]

[1]Tsinghua University
[2]Microsoft Research Asia
lsr22@mails.tsinghua.edu.cn, {chrihan, jbai}@microsoft.com

Abstract

Fine-tuning pre-trained models has recently yielded remarkable performance gains in graph neural networks (GNNs). In addition to pre-training techniques, inspired by the latest work in the natural language fields, more recent work has shifted towards applying effective fine-tuning approaches, such as parameter-efficient fine-tuning (PEFT). However, given the substantial differences between GNNs and transformer-based models, applying such approaches directly to GNNs proved to be less effective. In this paper, we present a comprehensive comparison of PEFT techniques for GNNs and propose a novel PEFT method specifically designed for GNNs, called AdapterGNN. AdapterGNN preserves the knowledge of the large pre-trained model and leverages highly expressive adapters for GNNs, which can adapt to downstream tasks effectively with only a few parameters, while also improving the model's generalization ability. Extensive experiments show that AdapterGNN achieves higher performance than other PEFT methods and is the only one consistently surpassing full fine-tuning (outperforming it by 1.6% and 5.7% in the chemistry and biology domains respectively, with only 5% and 4% of its parameters tuned) with lower generalization gaps. Moreover, we empirically show that a larger GNN model can have a worse generalization ability, which differs from the trend observed in large transformer-based models. Building upon this, we provide a theoretical justification for PEFT can improve generalization of GNNs by applying generalization bounds. Our code is available at https://github.com/Lucius-lsr/AdapterGNN.

Introduction

Graph neural networks (GNNs) (Scarselli et al. 2008; Wu et al. 2020) have achieved remarkable success in analyzing graph-structured data (Hamilton, Ying, and Leskovec 2017; Velickovic et al. 2017; Xu et al. 2018) but face challenges such as the scarcity of labeled data and low out-of-distribution generalization ability. To overcome these challenges, recent efforts have focused on designing GNNs pretraining approaches (Hu et al. 2019; Xia et al. 2022a; You et al. 2020) that leverage abundant unlabeled data to capture transferable intrinsic graph properties and generalize them

---

to different downstream tasks by fine-tuning (Zhou et al. 2019a,b; Wang et al. 2022). While fine-tuning all parameters from a pre-trained model can improve performance (Peng et al. 2019; Hu et al. 2019), it usually requires a relatively large model architecture to effectively glean knowledge from pre-training tasks (Devlin et al. 2018; Brown et al. 2020). This becomes challenging when the downstream task has limited data, as optimizing a large number of parameters can lead to overfitting (LeCun, Bengio, and Hinton 2015). Moreover, training and maintaining a separate large-scale model for each task can prove to be inefficient as the number of tasks grows. To address these challenges, recent research has focused on developing parameter-efficient finetuning (PEFT) techniques that can effectively adapt pretrained models to new tasks (Ding et al. 2022), such as adapter tuning (Houlsby et al. 2019), LoRA (Hu et al. 2021), BitFit (Zaken, Ravfogel, and Goldberg 2021), prefix-tuning (Li and Liang 2021), and the prompt tuning (Lester, Al-Rfou, and Constant 2021). PEFT seeks to tune a small portion of parameters and keep the left parameters frozen. This approach reduces training costs and allows for use in lowdata scenarios. PEFT could be applied to the GNNs (Xia et al. 2022b). In particular, the idea of prompt tuning has been widely adopted to GNNs (Wu et al. 2023). This adoption involves either manual prompt engineering (Sun et al. 2022) or soft prompt tuning techniques (Liu et al. 2023; Diao et al. 2022; Fang et al. 2022). However, prompt-based methods involve modifying only the raw input and not the inner architecture and thus struggling to match full fine-tuning performance (Fang et al. 2022), unless in special few-shot setting (Liu et al. 2023). Existing works lack study and comparison with other PEFT methods. And due to the inherent difference between transformer-based models and GNNs, not all NLP solutions can be directly applied to GNNs.

To address this issue, we propose an effective method, AdapterGNN, that combines task-specific knowledge in tunable adapters with task-agnostic knowledge in the frozen pre-trained model. Unlike the vanilla adapter (Houlsby et al. 2019) which exhibits poor performance when directly applied to GNNs, AdapterGNN is specifically designed to cater to the non-transformer GNN architecture employing novel techniques: (1) Dual adapter modules; (2) Batch normalization (BN); (3) Learnable scaling.

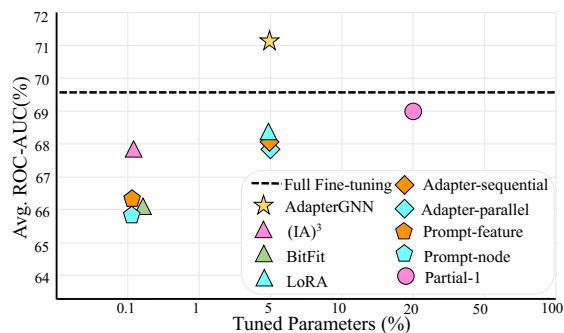As illustrated in Fig. 1, we conduct a comprehensive com-

Figure 1: Comparison among various PEFT methods of GNNs in six small molecular datasets. Detailed comparisons are shown in Table 1 and explanations are in Appendix D.2

parison of various PEFT approaches in GNNs. Applying them on GNNs is non-trivial since most of them were implemented on transformer-based models. AdapterGNN, with only 5% tuned parameters, achieves higher evaluation performance than other PEFT methods and is the only one consistently surpassing full fine-tuning.

This improvement can be attributed to AdapterGNN maximally utilizing advantages of PEFT with special designs. PEFT addresses two drawbacks of full fine-tuning to improve generalization. First, catastrophic forgetting of pre-trained knowledge is disastrous during generalizing (Kirkpatrick et al. 2017). Since PEFT keeps most parameters fixed, catastrophic forgetting can be potentially mitigated, resulting in improved model transferability and generalization (Ding et al. 2022). Second, overfitting is severe when tuning on a small dataset with large parameters (Aghajanyan, Zettlemoyer, and Gupta 2020; Arora et al. 2018), particularly in the OOD case (Kuhn et al. 2013; Hu et al. 2019).

For the first time, we provide a detailed theoretical justification from the perspective of generalization bounds (Mohri, Rostamizadeh, and Talwalkar 2018; Shalev-Shwartz and Ben-David 2014) to explain how PEFT in GNNs mitigates overfitting and promotes generalization ability.

To conclude, our work makes the following contributions:

- We are the first to apply the adapter to GNNs. To cater to the non-transformer GNN architecture, we integrate special techniques (dual adapter modules, BN, learnable scaling), which are essential in improving performance.

- We are the first to provide a theoretical justification of how PEFT improves generalization for non-transformer models. While there are works to study the theoretical support of PEFT in large transformer models, there is a lack of theoretical study for non-transformer models, e.g., GNNs. Although numerous empirical studies have demonstrated the efficacy of PEFT, we contribute to filling the gap between theory and empirical results.

- We are the first to apply all prevalent PEFT methods on GNNs and provide a detailed comparison. Applying such PEFT methods on GNNs is non-trivial since most PEFT methods were implemented on transformer-based mod-

els. We make special variations and implement them in GNNs, which fills the void in this field.

Related Work

Parameter-efficient fine-tuning methods. Full fine-tuning tunes all the model parameters and adapts them to downstream tasks, but this becomes inefficient with the growth of model size and task count. Recent NLP work has explored PEFT techniques that tune only a small portion of parameters for efficiency (Ding et al. 2022). Prompt tuning (Lester, Al-Rfou, and Constant 2021) aims to modify model inputs rather than model architecture. Prefix-tuning (Li and Liang 2021) only updates task-specific trainable parameters in each layer. Adapter tuning (Houlsby et al. 2019; Chen et al. 2022) inserts adapter modules with bottleneck architecture between layers. BitFit (Zaken, Ravfogel, and Goldberg 2021) only updates the bias terms while freezing the remaining. LoRA (Hu et al. 2021) decomposes the weight matrix into low-rank matrices to reduce the number of trainable parameters. As for the GNNs field, the idea of prompt tuning has gained widespread acceptance (Wu et al. 2023). GPPT (Sun et al. 2022) specially designs a framework for GNNs but is limited to node-level tasks. Limited to the molecular field, MolCPT (Diao et al. 2022) encodes additional molecular motif information to enhance graph embedding. GPF (Fang et al. 2022) and GraphPrompt (Liu et al. 2023) are parameter-efficient but struggle to match the full fine-tuning baseline in the non-few-shot setting.

Generalization error bounds. Generalization error bounds, also known as generalization bounds, provide insights into the predictive performance of learning algorithms in statistical machine learning. In the classical regime, bias–variance tradeoff states that the test error as a function of model complexity follows the U-shaped behavior (Mohri, Rostamizadeh, and Talwalkar 2018; Shalev-Shwartz and Ben-David 2014). However, in the over-parameterized regime, increasing complexity decreases test error, following the modern intuition of larger model generalizes better (Belkin et al. 2019; Nakkiran et al. 2021; Zhang et al. 2021; Sun et al. 2016; Hardt, Recht, and Singer 2016; Mou et al. 2018). (Aghajanyan, Zettlemoyer, and Gupta 2020) has theoretically analyzed generalization on over-parameterized large language models, it explains the empirical results that larger pre-trained models generalize better by applying intrinsic-dimension-based generalization bounds. While our work utilizes conventional generalization bounds to analyze the generalization ability of PEFT on GNNs.

Parameter-Efficient Fine-Tuning Improving Generalization Ability in GNNs

Several works have empirically shown that transformer-based large models in NLP/CV are over-parameterized and larger pre-trained models generalize better, aligning with the modern regime in Fig. 2. They use generalization bounds which are independent of the parameter count for large pre-trained models (Aghajanyan, Zettlemoyer, and Gupta 2020).
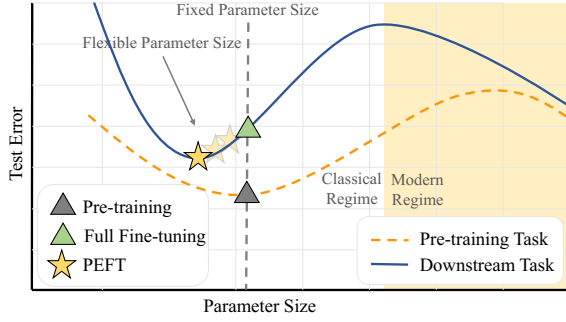
Figure 2: A large model is often employed for pre-training when sufficient data is available. However, for downstream tasks with limited data, a smaller model is optimal in the classical regime. Compared with full fine-tuning, PEFT preserves expressivity while reducing the size of parameter space, leading to lower test error.

However, theoretical study for PEFT in models aligning with the classical regime is still lacking. Our empirical findings in Appendix C.1 (Li, Han, and Bai 2023) find that larger GNNs generalize worse, satisfying the classical regime in Fig. 2. Therefore, we explore how PEFT benefits these models. We theoretically demonstrate PEFT can lower the bounds of test error (generalization bounds) and improve generalization ability in GNNs compared to full fine-tuning, as illustrated in Fig. 2. Our justification begins with this premise:

Premise 1. *GNNs satisfy the classical regime of generalization bounds theory.*

Within this regime, we apply a widely used parameter counting based generalization bounds theorem (Aghajanyan, Zettlemoyer, and Gupta 2020; Arora et al. 2018) and the detailed proof of this theorem can be found in Appendix B.1:

Theorem 1. *Generalization bounds for finite hypothesis space in classical regime. Training data $\mathcal{D}_n$ and the trained parameters $\mathcal{P}$ are variables of training error $\hat{\mathcal{E}}$. The number of training samples n and size of parameter space $|\mathcal{P}|$ are variables of generalization gap bounds. Then, statistically, the upper bound $\mathcal{U}(\mathcal{E})$ of the test error $\mathcal{E}$ of a model in finite hypothesis space is determined as follows:*

$$\mathcal{E} \leq \mathcal{U}(\mathcal{E}) = \hat{\mathcal{E}}(\mathcal{D}_n, \mathcal{P}) + O\left(\sqrt{|\mathcal{P}|/n}\right). \quad (1)$$

Before introducing PEFT, we first compare the error bounds of two paradigms: "supervised training from scratch" and "pre-train, fine-tune". For supervised training from scratch, we denote the task as T and training data as $\mathcal{D}_{n_T}^T$. With the increase of $|\mathcal{P}_T|$, training error $\hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \mathcal{P}_T)$ decreases due to stronger optimization capability and generalization gap bounds $O(\sqrt{|\mathcal{P}_T|/n_T})$ increases. Therefore, following the U-shaped behavior, the following corollary is obtained and the detailed proof can be found in Appendix B.2:

Corollary 1. *Bounds of supervised training from scratch. There is an optimal $|\mathcal{P}_T|$ to get the tightest upper bound $min(\mathcal{U}(\mathcal{E}_T)) = \hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_T) + O\left(\sqrt{|\bar{\mathcal{P}}_T|/n_T}\right).$*

For pre-training task S, data is more abundant: $n_S > n_T$. Many previous works discover that with abundant data to pre-train, we should employ a larger model to capture sufficient knowledge (Han et al. 2021; Ding et al. 2022; Wei et al. 2022). Therefore, we have the following corollary and the proof can be found in Appendix B.3:

Corollary 2. *Bounds of pre-training. For the pre-training task S satisfying $n_S > n_T$, there is also an optimal $|\bar{\mathcal{P}}_S|$ and $|\bar{\mathcal{P}}_S| > |\bar{\mathcal{P}}_T|$ to get the tightest upper bound $min(\mathcal{U}(\mathcal{E}_S)) = \hat{\mathcal{E}}(\mathcal{D}_{n_S}^S, \bar{\mathcal{P}}_S) + O(\sqrt{|\bar{\mathcal{P}}_S|/n_S}).$*

In the "pre-train, fine-tune" paradigm, a model that has been pre-trained on task S is used as initialization to improve the performance of a supervised task. To account for the effect of the initial parameter values, we use $\hat{\mathcal{E}}_S(D, \mathcal{P})$ to denote the training error of the pre-trained model.

Compared to training from scratch with the same model as pre-training: $\mathcal{U}(\mathcal{E}_T) = \hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_S) + O(\sqrt{|\bar{\mathcal{P}}_S|/n_T})$, fine-tuning benefits from a better initialization, which leads to a lower error bound: $\mathcal{U}(\mathcal{E}_F) = \hat{\mathcal{E}}_S(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_S) + O(\sqrt{|\bar{\mathcal{P}}_S|/n_T})$. Based on this, we make the following assumption and corollary:

Assumption 1. *The Transfer Gain $\mathbf{TG}$ from the pre-trained model can be quantified. It is solely determined by the properties of S, T and can be calculated as: $\mathbf{TG} = \hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{P}_S) - \hat{\mathcal{E}}_S(\mathcal{D}_{n_T}^T, \bar{P}_S)$*

Corollary 3. *Bounds of full fine-tuning. Full fine-tuning of a pre-trained model can result in a lower error bound, which can be measured by the transfer gain denoted by $\mathbf{TG}$:*

$$\mathcal{U}(\mathcal{E}_F) = \hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_S) + O\left(\sqrt{|\bar{\mathcal{P}}_S|/n_T}\right) - \mathbf{TG} \quad (2)$$

In regard to PEFT, the PEFT model is initialized with the same pre-trained parameters as fine-tuning, so $\mathbf{TG}$ is inherited similarly to fine-tuning, leading to the following corollary:

Corollary 4. *Bounds of PEFT. Bounds of PEFT $\mathcal{U}(\mathcal{E}_E)$ share a similar form as full fine-tuning:*

$$\mathcal{U}(\mathcal{E}_E) = \hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \mathcal{P}_E) + O\left(\sqrt{|\mathcal{P}_E|/n_T}\right) - \mathbf{TG}. \quad (3)$$

For PEFT, we propose a prerequisite: the preservation of the expressivity of full fine-tuning GNNs. Our AdapterGNN is specifically designed to maintain the capacity of the original GNNs maximally. With comparable parameters, it can be almost as powerful as full fine-tuning. We validate this in Appendix C.2. In contrast, several PEFT techniques have failed to yield satisfactory results in GNNs, such as prompt tuning (Fang et al. 2022; Liu et al. 2023). This can be attributed to not meeting this prerequisite.

With this prerequisite, when training from scratch on T, the trainable structure of PEFT and fine-tuning yields similar minimal generalization errors (under $\bar{\mathcal{P}}_T$ and $\bar{\mathcal{P}}_E$, respectively) as follows:
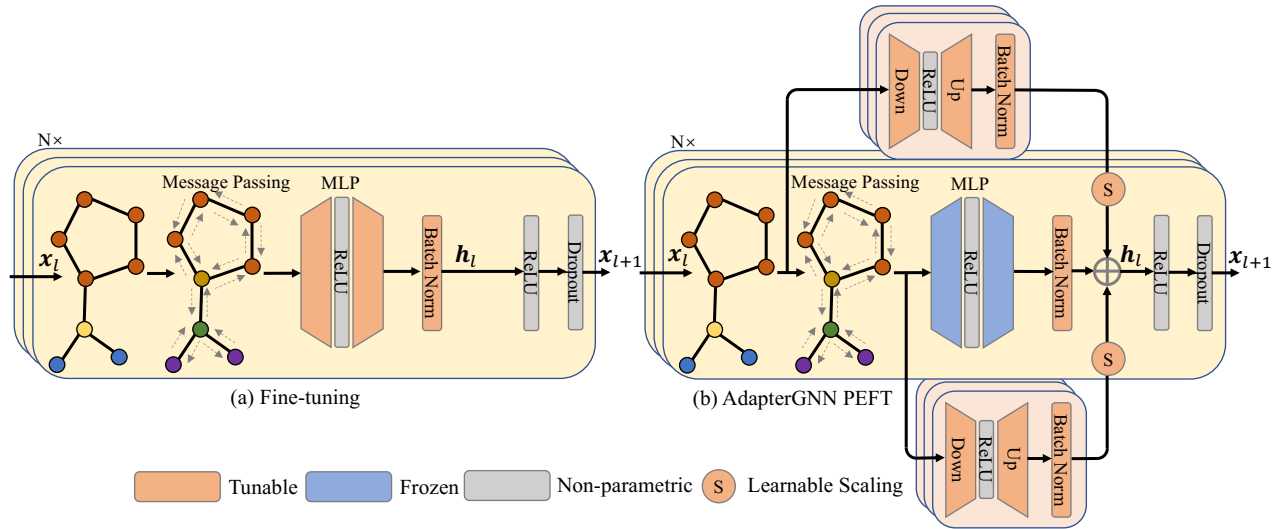
Figure 3: Comparison between full fine-tuning and AdapterGNN PEFT on GIN. a) The full fine-tuning updates all parameters of each pre-trained GNN layer. (b) AdapterGNN includes two parallel adapters taking input before and after the message passing. Their outputs are added to the original output of batch normalization with learnable scaling. During tuning, the original MLP of each GNN layer, which comprises the majority of the parameters, is frozen.

$$\left| \left(\hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_T) + O(\sqrt{\frac{|\bar{\mathcal{P}}_T|}{n_T}}) \right) - \left(\hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_E) + O(\sqrt{\frac{|\bar{\mathcal{P}}_E|}{n_T}}) \right) \right| < \varepsilon. \tag{4}$$

And for downstream task T, the optimal size is $|\bar{\mathcal{P}}_T|$ as in Cor. 1. But Cor. 2 gives $|\bar{\mathcal{P}}_S| > |\bar{\mathcal{P}}_T|$. Therefore, error bound $\mathcal{U}(\mathcal{E}_T)$ under $|\bar{\mathcal{P}}_S|$ is larger than that under the optimal $|\bar{\mathcal{P}}_T|$:

$$\begin{aligned} min(\mathcal{U}(\mathcal{E}_T)) &= \hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_T) + O\left(\sqrt{|\bar{\mathcal{P}}_T|/n_T}\right) \\ &< \hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_S) + O\left(\sqrt{|\bar{\mathcal{P}}_S|/n_T}\right) \end{aligned} \tag{5}$$

This means reducing the size of parameter space leads to a decrease in test error, which is aligned with the phenomena in the classical regime. We also empirically validate this trend in Appendix C.1. We define this **O**verfitting mitigation **G**ain as: $\mathbf{OG} = \hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_S) + O(\sqrt{|\bar{\mathcal{P}}_S|/n_T}) - (\hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_T) + O(\sqrt{|\bar{\mathcal{P}}_T|/n_T}))$.

Combining Equation 2,3, Inequality 4, and definition of **OG**, we obtain the following proposition:

Proposition 1. *Conditioned on $\varepsilon < \mathbf{OG}$, PEFT has tighter bounds than fine-tuning. Compare the tightest upper bound of PEFT on $|\bar{\mathcal{P}}_E|$ (where $|\bar{\mathcal{P}}_E| < |\bar{\mathcal{P}}_S|$) with the bounds of fine-tuning:*

$$\begin{aligned} &\mathcal{U}(\mathcal{E}_F) - min(\mathcal{U}(\mathcal{E}_E)) \\ &> \hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_S) + O(\sqrt{\frac{|\bar{\mathcal{P}}_S|}{n_T}}) - \left(\hat{\mathcal{E}}(\mathcal{D}_{n_T}^T, \bar{\mathcal{P}}_T) + O(\sqrt{\frac{|\bar{\mathcal{P}}_T|}{n_T}}) \right) - \varepsilon \\ &= \mathbf{OG} - \varepsilon \end{aligned}$$

$$\tag{6}$$

If PEFT preserves enough expressivity and $\mathbf{OG}$ is large enough, the condition $\varepsilon < \mathbf{OG}$ is satisfied. Therefore, PEFT

provides tighter bounds than full fine-tuning. The best approach is pre-training on a larger model and utilizing PEFT with comparable expressivity and much fewer parameters.

Methodology

AdapterGNN

We propose a GNN PEFT framework called AdapterGNN. The framework is demonstrated in Fig. 3. It adds trainable adapters in parallel to GNN MLPs, combining task-specific knowledge in adapters with task-agnostic knowledge from the pre-trained model. The multi-layer perception (MLP) module contains the majority of the learnable parameters and is important for GNNs. Therefore, we introduce adapters as parallel computations to the GNN MLPs. It utilize bottleneck architecture which includes a down-projection $\mathbf{W}_{\text{down}} : \mathbb{R}^{n_{\text{in}}} \rightarrow \mathbb{R}^{n_{\text{mid}}}$, a ReLU activation, and an up-projection $\mathbf{W}_{\text{up}} : \mathbb{R}^{n_{\text{mid}}} \rightarrow \mathbb{R}^{n_{\text{out}}}$. Unlike the original GNN MLP, the middle dimension n_{mid} is greatly reduced (e.g., $20\times$) as a bottleneck, resulting in a significant reduction in the size of tunable parameters space.

AdapterGNN utilizes ideas adopted in adapters (Houlsby et al. 2019). However, different from transformer-based models, GNNs possess distinctive characteristics. It needs to aggregate information from the neighborhood. The prerequisite in the justification Section gives that preserving the expressivity of the full fine-tuning GNNs is crucial for PEFT to generalize better than full fine-tuning. To enhance the expressivity in non-transformer-based GNNs, AdapterGNN employs several novel techniques specially designed:

Dual adapter modules. In each GNN layer, message passing (MP) aggregates information from the neighborhood. Node embeddings before and after MP provides complementary and informative information. To sufficiently cap-

ture the complementary information, we adopt dual parallel adapters. The first adapter takes input before MP, able to deal with the original information. And the other adapter takes input after the non-parametric MP can effectively adapt the MP-aggregated information.

Batch normalization (BN). Vanilla adapters did not utilize BN. However, data distribution may shift in GNNs. It is essential to ensure that affine parameters in BN are tunable during training. To maintain consistency with the backbone output, we incorporate tunable BN within each adapter.

Learnable scaling. The vanilla adapter simply adds the output of an adapter to the original output without scaling. This often causes catastrophic forgetting. On the other hand, manually tuning (Hu et al. 2021; Chen et al. 2022) the scaling factors requires extensive effort and time, as the optimal scaling factors may vary significantly across different datasets. This goes against our objective of achieving efficient training. In AdapterGNN, we use learnable scaling factors s_1, s_2 with a small start value. They can be dynamically adjusted during training.

Our experiments demonstrate that AdapterGNN can achieve stable and superior performance with these novel techniques. Formally, the output of the adapter is:

$$\mathbf{A}(\mathbf{x}) = \mathrm{BN}(\mathbf{W}_{\mathrm{up}}(\mathrm{ReLU}(\mathbf{W}_{\mathrm{down}}(\mathbf{x})))). \quad (7)$$

The output of adapters and the original embedding are combined by element-wise addition:

$$\mathbf{h}_l = \mathrm{BN}(\mathrm{MLP}(\mathrm{MP}(\mathbf{x}_l))) + s_1 \cdot \mathbf{A}_1(\mathbf{x}_l) + s_2 \cdot \mathbf{A}_2(\mathrm{MP}(\mathbf{x}_l)). \quad (8)$$

where $\mathbf{x}_l$ is the input of the lth layer and $\mathbf{h}_l$ is the embedding before GNN ReLU and Dropout, i.e., $\mathbf{x}_{l+1} = \mathrm{Dropout}(\mathrm{ReLU}(\mathbf{h}_l))$.

Discussion

Our framework provides several advantages. Firstly, the issue of catastrophic forgetting of pre-trained knowledge, is effectively mitigated through our parallel adapter design and novel learnable scaling strategy. This strategy automatically controls the proportion of newly-tuned task-specific knowledge, while persistently preserving the pre-trained knowledge. Secondly, to address overfitting when tuning on a small dataset with large parameters. Our framework introduces a bottleneck that significantly reduces the size of tunable parameter space to alleviate overfitting and improve generalization, while also being highly efficient. Additionally, AdapterGNN is specifically designed for GNNs and takes input both before and after message passing. This dual structure along with BN maximally preserves expressivity to satisfy the prerequisite to fully exploit the benefits of PEFT. Although our architecture is specifically designed for GIN (Xu et al. 2018), it can be transferred to any existing GNN architecture such as GAT (Velickovic et al. 2017) and GraphSAGE (Hamilton, Ying, and Leskovec 2017).

Experiments

Experimental setup. We evaluate the effectiveness of AdapterGNN by conducting extensive graph-level classifi-

cation experiments on eight molecular datasets and one biology dataset. We employ prevalent pre-training methods, all based on a GIN backbone. Details of datasets and pre-trained models are in Appendix D.3 and D.4. Implementations can be found in Appendix D.1.

Baseline methods. We mainly compare our method with full fine-tuning. Besides, we make special variations for PEFT methods and implement them in GNNs. Comprehensive comparisons of them are presented in Fig. 1, measured by the average ROC-AUC over six small molecular datasets and an AttrMasking pre-trained model. AdapterGNN is the only one surpassing full fine-tuning. Implementations of these PEFT methods in GNNs are in Appendix D.2. We also compare existing PEFT methods in GNNs. GPF (Fang et al. 2022) only modifies the input by adding a prompt feature to the input embeddings. MolCPT (Diao et al. 2022) (backbone frozen) leverages molecular motifs to provide additional information to graph embedding. We omit other related works such as GPPT (Sun et al. 2022) and GraphPrompt (Liu et al. 2023) because they are either not efficient or cannot yield satisfying performance without a few-shot setting.

Main Results

We **bold** the higher one in Table 1 and Table 2 and the results suggests the following:

(1) AdapterGNN is the only method consistently outperforming full fine-tuning. On molecular benchmarks, among eight datasets with five pre-training methods, AdapterGNN achieves higher average performances in all pre-training methods. It is the only method that outperforms full fine-tuning. The overall average ROC-AUC is 71.2%, which is 1.6% relatively higher than 70.1% of full fine-tuning. On the PPI benchmark, AdapterGNN is the only one that consistently outperforms full fine-tuning. The overall average ROC-AUC is 68.9%, which is 5.7% higher. In detail, AdapterGNN achieves higher performances in 32/46 (70%) experiments. It achieves these with only 5.2% and 4.0% tunable parameters, respectively. It also achieves training efficiency (both FLOPs and latency) as detailed in Appendix C.3.

(2) AdapterGNN outperforms state-of-the-art PEFT methods. In addition to comparing AdapterGNN with full fine-tuning, we implement variants of parallel adapter (He et al. 2021) and LoRA (Hu et al. 2021) in GNNs for the first time, which are prevalent PEFT methods in transformer-based models. They are compared under similar proportions of tuned parameters in detail. AdaptGNN has demonstrated consistent and conspicuous improvements on molecular benchmarks. On the PPI benchmark, overfitting is severe for full fine-tuning. Our implemented PEFT methods all achieve superior performance than it. Among them, ApdaterGNN is the best with the highest average. When compared with existing GNN PEFT methods, the advantage is more significant. Though GPF has a parameter efficiency of only 0.1%, its performance is limited. And MolCPT tunes much more parameters. On molecular benchmarks, AdapterGNN outperforms GPF and MolCPT by an average of 8.9% in GraphCL. And on the PPI benchmark, com-

Tuning Method	Pre-training Method	Dataset								
		BACE	BBBP	ClinTox	HIV	SIDER	Tox21	MUV	ToxCast	Avg.
Full Fine-tune (100%)	EdgePred	$79.9_{\pm0.9}$	$67.3_{\pm2.4}$	$64.1_{\pm3.7}$	$76.3_{\pm1.0}$	$60.4_{\pm0.7}$	$\mathbf{76.0}_{\pm0.6}$	$74.1_{\pm2.1}$	$64.1_{\pm0.6}$	70.3
	ContextPred	$\mathbf{79.6}_{\pm1.2}$	$68.0_{\pm2.0}$	$65.9_{\pm3.8}$	$\mathbf{77.3}_{\pm1.0}$	$60.9_{\pm0.6}$	$75.7_{\pm0.7}$	$75.8_{\pm1.7}$	$\mathbf{63.9}_{\pm0.6}$	70.9
	AttrMasking	$79.3_{\pm1.6}$	$64.3_{\pm2.8}$	$71.8_{\pm4.1}$	$\mathbf{77.2}_{\pm1.1}$	$61.0_{\pm0.7}$	$\mathbf{76.7}_{\pm0.4}$	$74.7_{\pm1.4}$	$\mathbf{64.2}_{\pm0.5}$	71.1
	GraphCL	$74.6_{\pm2.2}$	$68.6_{\pm2.3}$	$69.8_{\pm7.2}$	$78.5_{\pm1.2}$	$59.6_{\pm0.7}$	$74.4_{\pm0.5}$	$73.7_{\pm2.7}$	$62.9_{\pm0.4}$	70.3
	SimGRACE	$74.7_{\pm1.0}$	$\mathbf{69.0}_{\pm1.0}$	$59.9_{\pm2.3}$	$74.6_{\pm1.2}$	$59.1_{\pm0.6}$	$73.9_{\pm0.4}$	$71.0_{\pm1.9}$	$61.8_{\pm0.4}$	68.0
Adapter (5.2%)	EdgePred	$78.5_{\pm1.7}$	$65.9_{\pm2.8}$	$66.6_{\pm5.4}$	$73.5_{\pm0.2}$	$60.9_{\pm1.3}$	$75.4_{\pm0.5}$	$73.0_{\pm1.0}$	$63.0_{\pm0.7}$	69.6
	ContextPred	$75.0_{\pm3.3}$	$68.2_{\pm3.0}$	$57.6_{\pm3.6}$	$75.4_{\pm0.6}$	$62.4_{\pm1.2}$	$74.7_{\pm0.7}$	$73.3_{\pm0.8}$	$62.2_{\pm0.4}$	68.6
	AttrMasking	$76.1_{\pm1.4}$	$68.7_{\pm1.7}$	$65.8_{\pm4.4}$	$75.6_{\pm0.7}$	$59.8_{\pm1.7}$	$74.4_{\pm0.9}$	$75.8_{\pm2.4}$	$62.6_{\pm0.8}$	69.9
	GraphCL	$72.5_{\pm3.0}$	$\mathbf{69.3}_{\pm0.6}$	$67.3_{\pm7.4}$	$75.0_{\pm0.4}$	$\mathbf{59.7}_{\pm1.2}$	$74.7_{\pm0.4}$	$72.9_{\pm1.7}$	$62.9_{\pm0.4}$	69.3
	SimGRACE	$73.4_{\pm1.1}$	$64.8_{\pm0.7}$	$63.5_{\pm4.4}$	$73.9_{\pm1.0}$	$\mathbf{59.9}_{\pm0.9}$	$73.1_{\pm0.9}$	$70.1_{\pm4.6}$	$61.7_{\pm0.8}$	67.6
LoRA (5.0%)	EdgePred	$\mathbf{81.0}_{\pm0.8}$	$64.8_{\pm1.6}$	$\mathbf{67.7}_{\pm1.2}$	$74.8_{\pm1.2}$	$60.8_{\pm1.1}$	$74.6_{\pm0.4}$	$75.0_{\pm1.5}$	$62.2_{\pm1.0}$	70.1
	ContextPred	$78.5_{\pm1.1}$	$65.3_{\pm2.4}$	$61.3_{\pm1.9}$	$74.7_{\pm1.6}$	$60.8_{\pm0.4}$	$72.9_{\pm0.4}$	$75.4_{\pm0.9}$	$63.4_{\pm0.2}$	69.0
	AttrMasking	$\mathbf{79.8}_{\pm0.7}$	$64.2_{\pm1.1}$	$70.1_{\pm2.9}$	$76.1_{\pm1.4}$	$59.7_{\pm0.5}$	$74.6_{\pm0.5}$	$76.6_{\pm1.6}$	$61.7_{\pm0.4}$	70.4
	GraphCL	$75.1_{\pm0.7}$	$67.8_{\pm1.1}$	$65.1_{\pm3.5}$	$\mathbf{78.9}_{\pm0.6}$	$57.6_{\pm0.7}$	$73.9_{\pm0.9}$	$72.8_{\pm1.2}$	$62.7_{\pm0.6}$	69.2
	SimGRACE	$73.2_{\pm1.0}$	$67.5_{\pm0.4}$	$60.7_{\pm0.4}$	$74.1_{\pm0.5}$	$57.6_{\pm2.6}$	$72.2_{\pm0.2}$	$67.9_{\pm0.9}$	$61.8_{\pm0.2}$	66.9
GPF (0.1%)	EdgePred	$68.0_{\pm0.4}$	$55.9_{\pm0.2}$	$50.8_{\pm0.1}$	$66.0_{\pm0.7}$	$51.5_{\pm0.7}$	$63.1_{\pm0.5}$	$63.1_{\pm0.1}$	$55.7_{\pm0.5}$	59.3
	ContextPred	$58.7_{\pm0.6}$	$58.6_{\pm0.6}$	$39.8_{\pm0.8}$	$68.0_{\pm0.4}$	$59.4_{\pm0.2}$	$67.8_{\pm0.9}$	$71.8_{\pm0.8}$	$58.8_{\pm0.5}$	60.4
	AttrMasking	$61.7_{\pm0.3}$	$54.3_{\pm0.3}$	$56.4_{\pm0.2}$	$64.0_{\pm0.2}$	$52.0_{\pm0.2}$	$69.2_{\pm0.3}$	$62.9_{\pm0.9}$	$58.1_{\pm0.3}$	59.8
	GraphCL	$71.5_{\pm0.6}$	$63.7_{\pm0.4}$	$64.5_{\pm0.6}$	$70.3_{\pm0.5}$	$55.3_{\pm0.6}$	$65.5_{\pm0.5}$	$70.1_{\pm0.7}$	$58.5_{\pm0.5}$	64.9
MolCPT (40.0%)	GraphCL	$74.1_{\pm0.5}$	$60.5_{\pm0.8}$	$73.4_{\pm0.8}$	$64.5_{\pm0.8}$	$55.9_{\pm0.3}$	$67.4_{\pm0.7}$	$65.7_{\pm2.2}$	$57.9_{\pm0.3}$	64.9
AdapterGNN (ours) (5.2%)	EdgePred	$79.0_{\pm1.5}$	$\mathbf{69.7}_{\pm1.4}$	$67.7_{\pm3.0}$	$76.4_{\pm0.7}$	$61.2_{\pm0.9}$	$75.9_{\pm0.9}$	$\mathbf{75.8}_{\pm2.1}$	$\mathbf{64.2}_{\pm0.5}$	**71.2**
	ContextPred	$78.7_{\pm2.0}$	$68.2_{\pm2.9}$	$\mathbf{68.7}_{\pm5.3}$	$76.1_{\pm0.5}$	$61.1_{\pm1.0}$	$75.4_{\pm0.6}$	$\mathbf{76.3}_{\pm1.0}$	$63.2_{\pm0.3}$	**71.0**
	AttrMasking	$79.7_{\pm1.3}$	$67.5_{\pm2.2}$	$\mathbf{78.3}_{\pm2.6}$	$76.7_{\pm1.2}$	$\mathbf{61.3}_{\pm1.1}$	$76.6_{\pm0.5}$	$\mathbf{78.4}_{\pm0.7}$	$63.6_{\pm0.5}$	**72.8**
	GraphCL	$\mathbf{76.1}_{\pm2.2}$	$67.8_{\pm1.4}$	$72.0_{\pm3.8}$	$77.8_{\pm1.3}$	$59.6_{\pm1.3}$	$74.9_{\pm0.9}$	$75.1_{\pm2.1}$	$63.1_{\pm0.4}$	**70.7**
	SimGRACE	$\mathbf{77.7}_{\pm1.7}$	$68.1_{\pm1.3}$	$73.9_{\pm7.0}$	$75.1_{\pm1.2}$	$58.9_{\pm0.9}$	$74.4_{\pm0.6}$	$71.8_{\pm1.4}$	$62.6_{\pm0.6}$	**70.3**

Table 1: Test ROC-AUC (%) on molecular prediction benchmarks with different tuning and pre-training methods.

Tuning Method	-	EdgePred	ContextPred	AttrMasking	GraphCL	SimGRACE	Avg.
Full Fine-tune (100%)	$65.2_{\pm1.2}$	$65.6_{\pm0.9}$	$63.5_{\pm1.1}$	$63.2_{\pm1.2}$	$65.5_{\pm0.8}$	$68.2_{\pm1.2}$	65.2
GPF (0.1%)	$65.9_{\pm1.9}$	$51.2_{\pm1.3}$	$67.1_{\pm0.6}$	$69.0_{\pm0.3}$	$62.3_{\pm0.5}$	$50.0_{\pm0.9}$	60.9
Adapter (4.0%)	$65.6_{\pm1.1}$	$69.8_{\pm0.5}$	$68.2_{\pm1.5}$	$\mathbf{70.9}_{\pm1.0}$	$69.0_{\pm0.8}$	$64.5_{\pm2.0}$	68.0
LoRA (4.0%)	$63.0_{\pm0.4}$	$68.0_{\pm1.0}$	$68.0_{\pm1.1}$	$69.2_{\pm0.8}$	$\mathbf{69.4}_{\pm0.6}$	$63.0_{\pm0.3}$	66.8
AdpaterGNN (4.0%)	$\mathbf{66.3}_{\pm0.9}$	$\mathbf{70.6}_{\pm1.1}$	$\mathbf{68.3}_{\pm1.5}$	$69.7_{\pm1.1}$	$68.1_{\pm1.5}$	$\mathbf{70.1}_{\pm1.2}$	**68.9**

Table 2: Test ROC-AUC (%) on PPI benchmark with different tuning and pre-training methods. ”-”: no pre-training.

pared with GPF, AdapterGNN outperforms it by an average of 13.1%.

(3) AdapterGNN avoids negative transfer and retains the stable improvements of pre-training. On the PPI benchmark, negative transfer frequently occurs in full fine-tuning and GPF, where the pre-trained model often yields inferior performance. On the contrary, the benefits of pre-training remain stable in AdapterGNN.

Ablation Studies

We investigate the properties that make for a good AdapterGNN. And by ablating on model size and training data size, we validate our theoretical justification for the generalization bounds. All models are pre-trained by AttrMasking. Unless otherwise specified, the performance represents the average ROC-AUC over six small molecular datasets.

Insertion form and BN. AdapterGNN includes dual adapters parallel to GNN MLP taking input before and after the message passing. To demonstrate the effectiveness of this design, we compare its performance with those of a single parallel adapter and a sequential adapter inserted after GNN MLP. We also compare with the variant without batch normalization (BN). The count of adapter parameters is the same across all forms. Table 3 shows that adopting a single adapter already achieves superior performance over full fine-tuning. Combining two parallel adapters further improves the expressivity of PEFT, yielding the best performance. But without BN, performance drops by a large margin.

Scaling strategy. We compared our novel learnable scaling strategy with various fixed scaling ranging from 0.01 to 5. Table 4 shows that in five out of six datasets, as well as on average, our learnable scaling strategy achieved the highest performance. Among the fixed scalings, smaller scaling is superior. As the scaling factor increased, the performance deteriorated due to catastrophic forgetting of pre-trained knowledge.

Form	Position	BN	Avg.
Full Fine-tune			69.6
Sequential	After MLP	✓	70.2
Parallel	Before MP	✓	70.4
Parallel	After MP	✓	70.3
Parallel	Dual	×	68.0
Parallel	Dual	✓	**71.2**

Table 3: Comparison of insertion forms and BN.

	BACE	BBBP	ClinTox	SIDER	Tox21	ToxCast	Avg.
0.01	$77.8_{\pm1.9}$	$67.6_{\pm1.4}$	$76.3_{\pm2.8}$	$59.9_{\pm1.1}$	$74.8_{\pm0.4}$	$62.5_{\pm0.4}$	69.8
0.1	$78.5_{\pm1.1}$	$67.6_{\pm2.1}$	$72.6_{\pm7.0}$	$61.0_{\pm0.7}$	$76.2_{\pm0.5}$	$63.3_{\pm0.3}$	69.9
0.5	$78.7_{\pm1.1}$	$67.6_{\pm3.2}$	$72.3_{\pm6.0}$	$60.9_{\pm1.0}$	$76.2_{\pm0.7}$	$63.3_{\pm0.3}$	69.8
1	$78.6_{\pm1.7}$	$\mathbf{68.7}_{\pm3.0}$	$66.3_{\pm7.2}$	$\mathbf{61.3}_{\pm0.8}$	$75.7_{\pm0.7}$	$63.3_{\pm0.6}$	69.0
5	$73.7_{\pm2.4}$	$66.6_{\pm2.1}$	$55.9_{\pm6.6}$	$60.8_{\pm1.5}$	$75.3_{\pm0.8}$	$62.9_{\pm0.5}$	65.9
Ours	$\mathbf{79.7}_{\pm1.3}$	$67.5_{\pm2.2}$	$\mathbf{78.3}_{\pm2.6}$	$\mathbf{61.3}_{\pm1.1}$	$\mathbf{76.6}_{\pm0.5}$	$\mathbf{63.6}_{\pm0.5}$	**71.2**

Table 4: Comparison of our learnable scaling and fixed scaling.

Model size. We report performances and errors across model sizes by varying embedding dimensions. **(1)** As shown in Fig. 4(a), average performance initially increases and then decreases, indicating a larger model can be worse, which is consistent with the classical regime. AdapterGNN consistently outperforms full fine-tuning across all model sizes. Specifically, in the BBBP dataset, AdapterGNN may not surpass full fine-tuning with small model sizes, but it achieves superior performance in larger models. **(2)** Fig. 4(b) displays the errors in the Tox21 dataset. The U-shaped test curve also validates the classical regime in our tasks. Although increasing the model size leads to a significant increase in the generalization gap in full fine-tuning, the gap is well-controlled with AdapterGNN. It demonstrates AdapterGNN's superior generalization ability, especially in larger models.

Training data size. Reducing the size of the training samples results in inferior generalization, while AdapterGNN can mitigate this overfitting issue. We compare the performance of full fine-tuning with two AdapterGNN settings, with bottleneck dimensions of 15 (default) and 5, respectively. Fig. 5 demonstrates that when data becomes scarce, the performance of AdapterGNN with fewer tunable parameters decreases slower and obtains superior results. And fewer parameters yield better results.

Bottleneck dimension. Fig.6 demonstrates that reducing the bottleneck dimension to limit the size of tunable parameter space can improve the generalization ability of the model. But when the size is too small, the model may suffer from underfitting, which can restrict its performance. Therefore, selecting a bottleneck dimension of 15, which represents 5.2% of all parameters, yields the best average performance. Meanwhile, a dimension of 5, which accounts for only 2.2% of all parameters, can surpass the results of full fine-tuning.
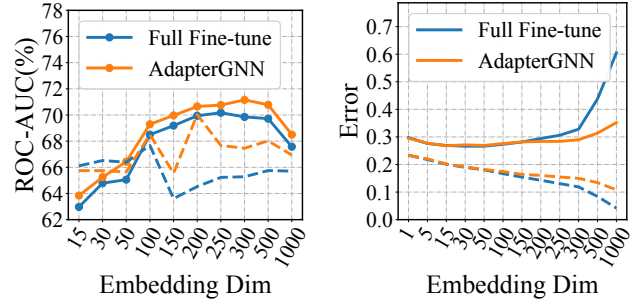
(a) Average (solid) and BBBP performance (dashed).

(b) The generalization gap in Tox21. Test error (solid) and training error (dashed).

Figure 4: Test ROC-AUC (%) performances and generalization gap with different model sizes.

(a) SIDER (1142 training samples).
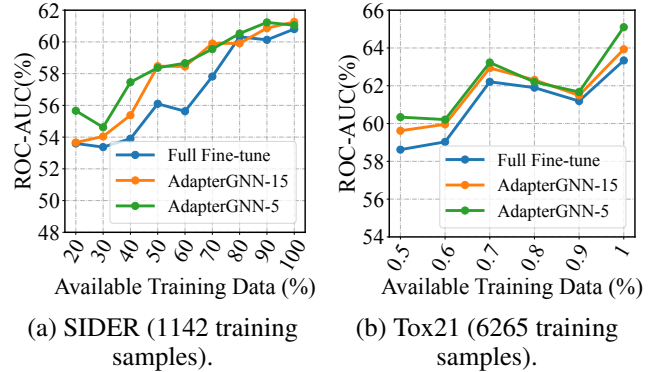
(b) Tox21 (6265 training samples).

Figure 5: Test ROC-AUC (%) performances with different training data sizes.
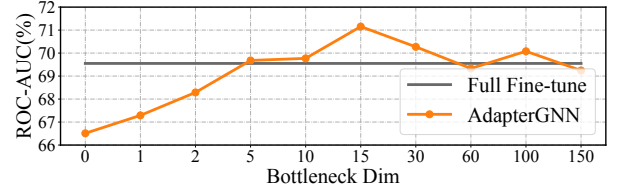

Figure 6: Performances with different bottleneck dimensions. 0 represents identical mapping.

Conclusion

We present an effective PEFT framework specially designed for GNNs called AdapterGNN. It is the only PEFT method that outperforms full fine-tuning with much fewer tunable parameters, improving both efficiency and effectiveness. We provide a theoretical justification for this improvement and find that our tasks fall within the classical regime of the generalization error, where larger GNNs can be worse, and reducing the size of the parameter space during tuning can result in lower test error and superior performance. We focus on graph-level classification tasks on GINs and leave other tasks and GNN models for future exploration.

References

Aghajanyan, A.; Zettlemoyer, L.; and Gupta, S. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Arora, S.; Ge, R.; Neyshabur, B.; and Zhang, Y. 2018. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning*, 254–263. PMLR.

Belkin, M.; Hsu, D.; Ma, S.; and Mandal, S. 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32): 15849–15854.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Chen, S.; Ge, C.; Tong, Z.; Wang, J.; Song, Y.; Wang, J.; and Luo, P. 2022. Adaptformer: Adapting vision transformers for scalable visual recognition. *arXiv preprint arXiv:2205.13535*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Diao, C.; Zhou, K.; Huang, X.; and Hu, X. 2022. Mol-CPT: Molecule Continuous Prompt Tuning to Generalize Molecular Representation Learning. *arXiv preprint arXiv:2212.10614*.

Ding, N.; Qin, Y.; Yang, G.; Wei, F.; Yang, Z.; Su, Y.; Hu, S.; Chen, Y.; Chan, C.-M.; Chen, W.; et al. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*.

Fang, T.; Zhang, Y.; Yang, Y.; and Wang, C. 2022. Prompt Tuning for Graph Neural Networks. *arXiv preprint arXiv:2209.15240*.

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

Han, X.; Zhang, Z.; Ding, N.; Gu, Y.; Liu, X.; Huo, Y.; Qiu, J.; Yao, Y.; Zhang, A.; Zhang, L.; et al. 2021. Pre-trained models: Past, present and future. *AI Open*, 2: 225–250.

Hardt, M.; Recht, B.; and Singer, Y. 2016. Train faster, generalize better: Stability of stochastic gradient descent. In *International conference on machine learning*, 1225–1234. PMLR.

He, J.; Zhou, C.; Ma, X.; Berg-Kirkpatrick, T.; and Neubig, G. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; and Gelly, S. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, 2790–2799. PMLR.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Hu, W.; Liu, B.; Gomes, J.; Zitnik, M.; Liang, P.; Pande, V.; and Leskovec, J. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13): 3521–3526.

Kuhn, M.; Johnson, K.; Kuhn, M.; and Johnson, K. 2013. Over-fitting and model tuning. *Applied predictive modeling*, 61–92.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature*, 521(7553): 436–444.

Lester, B.; Al-Rfou, R.; and Constant, N. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Li, S.; Han, X.; and Bai, J. 2023. AdapterGNN: Efficient Delta Tuning Improves Generalization Ability in Graph Neural Networks. *arXiv preprint arXiv:2304.09595*.

Li, X. L.; and Liang, P. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Liu, Z.; Yu, X.; Fang, Y.; and Zhang, X. 2023. Graph-Prompt: Unifying Pre-Training and Downstream Tasks for Graph Neural Networks. *arXiv preprint arXiv:2302.08043*.

Mohri, M.; Rostamizadeh, A.; and Talwalkar, A. 2018. *Foundations of machine learning*. MIT press.

Mou, W.; Wang, L.; Zhai, X.; and Zheng, K. 2018. Generalization bounds of sgld for non-convex learning: Two theoretical viewpoints. In *Conference on Learning Theory*, 605–638. PMLR.

Nakkiran, P.; Kaplun, G.; Bansal, Y.; Yang, T.; Barak, B.; and Sutskever, I. 2021. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12): 124003.

Peng, A. Y.; Sing Koh, Y.; Riddle, P.; and Pfahringer, B. 2019. Using supervised pretraining to improve generalization of neural networks on binary classification problems. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part I 18*, 410–425. Springer.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.

Shalev-Shwartz, S.; and Ben-David, S. 2014. *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Sun, M.; Zhou, K.; He, X.; Wang, Y.; and Wang, X. 2022. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 1717–1727.

Sun, S.; Chen, W.; Wang, L.; Liu, X.; and Liu, T.-Y. 2016. On the depth of deep neural networks: A theoretical view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y.; et al. 2017. Graph attention networks. *stat*, 1050(20): 10–48550.

Wang, Z.; Liu, M.; Luo, Y.; Xu, Z.; Xie, Y.; Wang, L.; Cai, L.; Qi, Q.; Yuan, Z.; Yang, T.; et al. 2022. Advanced graph and sequence neural networks for molecular property prediction and drug discovery. *Bioinformatics*, 38(9): 2579–2586.

Wei, J.; Tay, Y.; Bommasani, R.; Raffel, C.; Zoph, B.; Borgeaud, S.; Yogatama, D.; Bosma, M.; Zhou, D.; Metzler, D.; et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Wu, X.; Zhou, K.; Sun, M.; Wang, X.; and Liu, N. 2023. A Survey of Graph Prompting Methods: Techniques, Applications, and Challenges. *arXiv preprint arXiv:2303.07275*.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1): 4–24.

Xia, J.; Wu, L.; Chen, J.; Hu, B.; and Li, S. Z. 2022a. Simgrace: A simple framework for graph contrastive learning without data augmentation. In *Proceedings of the ACM Web Conference 2022*, 1070–1079.

Xia, J.; Zhu, Y.; Du, Y.; and Li, S. Z. 2022b. A survey of pretraining on graphs: Taxonomy, methods, and applications. *arXiv preprint arXiv:2202.07893*.

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

You, Y.; Chen, T.; Sui, Y.; Chen, T.; Wang, Z.; and Shen, Y. 2020. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33: 5812–5823.

Zaken, E. B.; Ravfogel, S.; and Goldberg, Y. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; and Vinyals, O. 2021. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3): 107–115.

Zhou, F.; Cao, C.; Zhang, K.; Trajcevski, G.; Zhong, T.; and Geng, J. 2019a. Meta-gnn: On few-shot node classification in graph meta-learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2357–2360.

Zhou, K.; Song, Q.; Huang, X.; and Hu, X. 2019b. Autognn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184*.