G-Adapter: Towards Structure-Aware Parameter-Efficient Transfer Learning for Graph Transformer Networks

Anchun Gui, Jinqiang Ye and Han Xiao[*]
Department of Artificial Intelligence
School of Informatics, Xiamen University
{anchungui,jinqiangye}@stu.xmu.edu.cn, bookman@xmu.edu.cn

Abstract

It has become a popular paradigm to transfer the knowledge of large-scale pre-trained models to various downstream tasks via fine-tuning the entire model parameters. However, with the growth of model scale and the rising number of downstream tasks, this paradigm inevitably meets the challenges in terms of computation consumption and memory footprint issues. Recently, Parameter-Efficient Fine-Tuning (PEFT) (e.g., Adapter, LoRA, BitFit) shows a promising paradigm to alleviate these concerns by updating only a portion of parameters. Despite these PEFTs having demonstrated satisfactory performance in natural language processing, it remains under-explored for the question of whether these techniques could be transferred to graph-based tasks with Graph Transformer Networks (GTNs). Therefore, in this paper, we fill this gap by providing extensive benchmarks with traditional PEFTs on a range of graph-based downstream tasks. Our empirical study shows that it is sub-optimal to directly transfer existing PEFTs to graph-based tasks due to the issue of *feature distribution shift*. To address this issue, we propose a novel structure-aware PEFT approach, named G-Adapter, which leverages graph convolution operation to introduce graph structure (e.g., graph adjacent matrix) as an inductive bias to guide the updating process. Besides, we propose Bregman proximal point optimization to further alleviate feature distribution shift by preventing the model from aggressive update. Extensive experiments demonstrate that G-Adapter obtains the state-of-the-art performance compared to the counterparts on nine graph benchmark datasets based on two pre-trained GTNs, and delivers tremendous memory footprint efficiency compared to the conventional paradigm.

1 Introduction

Pre-training then fine-tuning has become a prevalent training paradigm with the remarkable success of large-scale pre-trained models in Natural Language Processing (NLP) [5, 9, 32, 37, 48, 53]. Recently, more researchers are striving to apply this paradigm to graph-based tasks with Graph Transformer Networks (GTNs) [4, 6, 7, 10, 25, 28, 30, 40, 41, 62, 64, 65, 68]. For instance, based on multi-layer Transformer encoders [54], Graphormer [62] first performs the well-designed unsupervised tasks on large-scale molecular datasets, and then fine-tunes the entire pre-trained parameters of the model on downstream molecular tasks of interest, which is also known as *full fine-tuning*. However, full fine-tuning poses several issues in practice: (i) Given that the labels of graph data from some domains (e.g., chemistry, biology) are inaccessible without the expertise and labor-heavy annotations [60], it is common that there are insufficient labeled samples in downstream tasks of interest. Hence, full fine-tuning would incur serious over-fitting and catastrophic forgetting issues [15, 56]. (ii) When

---

[*]Corresponding author.

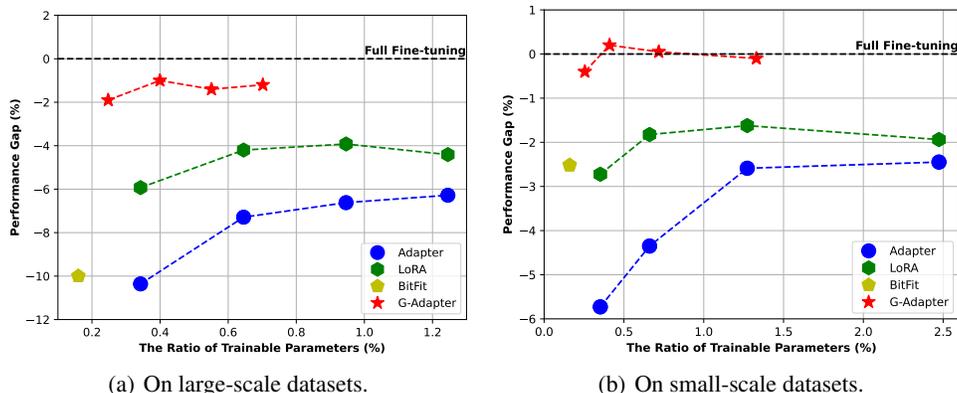|	(a) On large-scale datasets.	(b) On small-scale datasets.

Figure 1: The comparison between PEFTs (Adapter, LoRA, BitFit, and G-Adapter) and full fine-tuning on large- and small-scale datasets. (a) Based on the pre-trained Graphormer, we first average the results of each PEFT on two large-scale datasets, and then compute the performance gap compared to full fine-tuning. (b) Similarly, we calculate the performance gap of each PEFT on seven small-scale datasets, based on another pre-trained model MAT. Refer to Sec. 3.1 for more descriptions.

handling multiple diverse downstream tasks, full fine-tuning has to duplicate a modified copy of all parameters per task, which hinders the flexibility and applicability of large-scale models, especially in scenarios with constrained storage resources (e.g., mobile detection devices).

Recently, Parameter-Efficient Fine-Tuning (PEFT), as an alternative to full fine-tuning, has been proposed and widely investigated in NLP [3, 11, 14–16]. PEFT aims to achieve competitive performance with full fine-tuning while consuming computation and storage resources as few as possible. Instead of updating the entire parameters during the fine-tuning phase, PEFT only updates a small fraction of parameters within the original model or additionally introduced modules, while freezing the remaining parameters. For example, Adapter [15] inserts two compact modules in each encoder of Transformer, while BitFit [3] only updates the bias terms in the model parameters, as shown in Fig. 3. Despite the remarkable achievements of traditional PEFTs in natural language understanding tasks, the question is still under-explored whether these PEFTs from the language domain are feasible for various GTNs under graph-based tasks, given that the intrinsic discrepancy between graph and text modalities (e.g., the graph has rich structure information). Therefore, in this paper, we shall fill this gap by answering the following questions: **Can PEFTs from the language domain be transferred directly to graph-based tasks? If not, how to design a graph-specific PEFT method?**

To start with, we comprehensively examine the performance of mainstream PEFTs (Adapter [15], LoRA [16], and BitFit [3]) on popular molecular graph datasets based on two pre-trained GTNs (Graphormer [62] and MAT [40]). The overall comparison is shown in Fig. 1, in which we unfortunately observe a significant gap between traditional PEFTs and full fine-tuning, especially on large-scale datasets. Further, our exploration reveals the *feature distribution shift* issue due to the absence of graph structure in the fine-tuning process (see Fig. 2 and Sec. 3.1 for more discussions). To alleviate these concerns, we propose a novel structure-aware PEFT method, G-Adapter, which leverages graph convolution operation to introduce graph structure as the inductive bias to guide the updating process. Moreover, we apply the low-rank decomposition to the learnable weights, which makes G-Adapter highly lightweight. Besides, we propose Bregman proximal point optimization to further ease the feature distribution shift by preventing the model from aggressive update.

To verify the effectiveness of our approach, we conduct extensive experiments on a variety of graph-based downstream tasks based on pre-trained GTNs. The results demonstrate that our proposed G-Adapter can effectively address the feature distribution shift issue and significantly enhance the performance. Specifically, (i) G-Adapter obtains the state-of-the-art performance than baselines on both large- and small-scale datasets. Even compared to full fine-tuning, G-Adapter could achieve competitive (or superior) results with fewer trainable parameters. For example, full fine-tuning achieves 0.804 AUC with 100% trainable parameters on MolHIV, while G-Adapter gains 0.790 AUC with only 0.24% trainable parameters. (ii) G-Adapter enjoys remarkable advantages over full

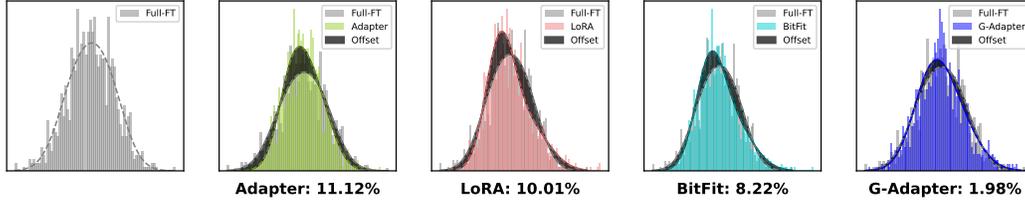| Adapter: 11.12% | LoRA: 10.01% | BitFit: 8.22% | G-Adapter: 1.98% |

Figure 2: The illustration of feature distribution shift, where Full-FT denotes full fine-tuning. For the identical input, the feature distribution of traditional PEFTs (Adapter, LoRA, and BitFit) has a significant offset (dark region) compared to full fine-tuning. In contrast, our proposed G-Adapter has a highly similar behavior to full fine-tuning. Here, Jensen-Shannon divergence is utilized to measure the discrepancy between two distributions. Refer to Sec. 3.1 for more discussions.

fine-tuning in terms of memory footprint. For instance, full fine-tuning stores 161MB checkpoint per task while G-Adapter merely requests $0.4MB^2$ checkpoint for each downstream task. Additionally, the introduced G-Adapter modules barely degrade the training efficiency and inference speed, and extensive ablation experiments also confirm the rationality of each component in our design.

To summarize, our contributions are as follows:

- To the best of our knowledge, this is the first work formally to investigate the parameter-efficient fine-tuning of graph-based tasks and models. And, we benchmark several widely used PEFTs from the language domain on a range of graph-based downstream tasks.

- We exhibit the phenomenon of feature distribution shift when directly applying existing PEFTs to graph-based tasks. Further, our study empirically shows that the graph structure and Bregman proximal point optimization could alleviate this concern well.

- We propose a structure-aware parameter-efficient method (G-Adapter) for adapting pre-trained GTNs to various graph-based downstream tasks, in which G-Adapter introduces graph structure as an inductive bias to guide the updating process.

- Extensive experiments demonstrate that G-Adapter outperforms the counterparts by a significant margin. Furthermore, compared to full fine-tuning, our method yields tremendous memory footprint benefits almost without sacrificing the efficiency of training and inference.

2 Related Work

Graph Transformer Networks.   Transformer [54], as one of the most popular network architectures so far, has demonstrated remarkable success in NLP [5, 9, 32, 37, 48, 53], which spurs extensive research on transferring Transformer to graph representation learning [42, 60]. Considering the intrinsic discrepancy between graph and text modalities, current efforts mainly focus on two aspects: *the design of pre-training tasks* and *the encoding of nodes and edges*. For the first aspect, there are generally three folds: (i) Supervised learning: the preset supervised task is constructed by measuring the labels of graph data using professional tools [20, 49, 67]. (ii) Graph autoregressive modeling: similar to the GPT-style pre-training tasks [5, 46, 47] in NLP, some nodes and edges in the graph are randomly masked first, and then the masked elements are recovered in a step-by-step manner [21, 67]. (iii) Masked components modeling: this approach is analogous to the MLM task in BERT [9], where all masked elements in the graph are predicted simultaneously [20, 49]. For the second aspect, each node (e.g., an atom in the molecular graph) is regarded as a "token" in text sequence, and then the hidden representation of the node is learned similar to Transformers in NLP [7]. Compared to the simple sequential relationship between tokens in text sequence, the relationship between edges in the graph could be more complex and essential [42, 60]. Therefore, substantial works focus on modeling graph structures [4, 6, 10, 25, 28, 30, 41, 62, 64, 68]. For example, Graphormer [62] leverages the centrality and spatial encoding as the graph structural signal, and MAT [40] augments the attention mechanism in Transformer using inter-atomic distances and the molecular graph structure. Kreuzer

---

^{2}Here, the bottleneck size $r = 4$, based on the pre-trained MAT. See Tab. 3 for more comparisons.
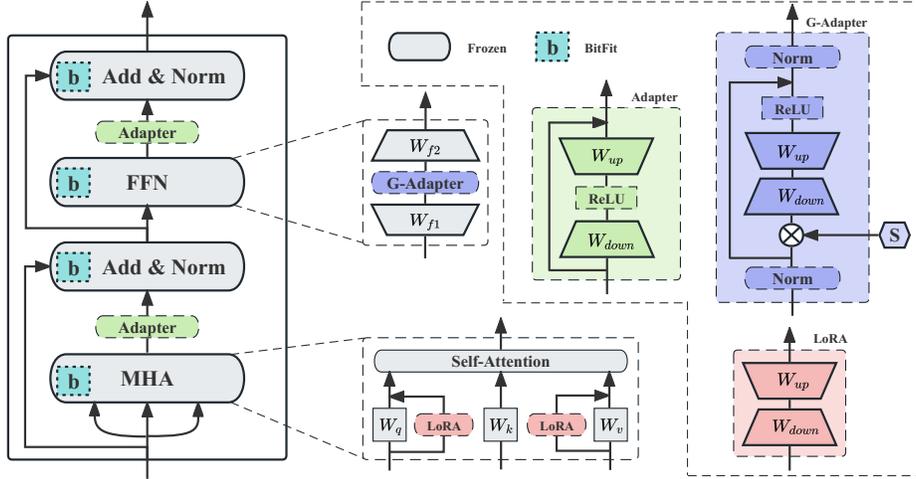
Figure 3: An overview of existing popular PEFTs (Adapter, LoRA, and BitFit) and our proposed G-Adapter. In the left part, we demonstrate the insertion position of PEFT blocks in a standard encoder of Transformer. In the right part, we depict the architecture of each PEFT method. Here, each color represents an approach, where the grey blocks are frozen during the fine-tuning process. Our proposed G-Adapter is marked and demonstrated in purple, in which **S** indicates the introduced graph structure information (e.g., the graph adjacent matrix).

et al. [30] propose the learnable structural encoding via Laplacian spectrum, which can learn the position of each node in the graph. Moreover, Zhao et al. [68] proposes a proximity-enhanced multi-head attention to capture the multi-hop graph structure, and Khoo et al. [27] design a structure-aware self-attention for modeling the tree-structured graphs. Additionally, Min et al. [42] systematically investigate the effectiveness and application of Transformers in the graph domain.

Parameter-Efficient Transfer Learning. Parameter-Efficient Fine-Tuning (PEFT) is receiving considerably growing attention in diverse domains [11, 14, 44, 63]. Adapter [15], as the representative work of PEFT, is proposed to tackle natural language understanding tasks by inserting the compact blocks into Multi-Head Attention (MHA) and Feed-Forward Networks (FFN) in Transformer. Following this work, a series of subsequent efforts are proposed to improve the performance of Adapter. For instance, AdapterDrop [50] removes Adapter blocks from the lower layers, and Compacter/Compacter++ [38] introduce Kronecker product and weights sharing tricks to further reduce the proportion of trainable parameters. More similar works are included [12, 26, 43, 52, 56, 57]. Based on the hypothesis of *low intrinsic rank* [1], LoRA [16] tunes two low-rank learnable matrices to approximate the updating of query and value weights in MHA. Moreover, Zhang et al. [66] enhance LoRA by adaptively allocating the trainable parameters budget at each layer, and FacT [24] extends LoRA by introducing a new tensorization-decomposition framework. Instead of introducing extra parameters, BitFit [3], a simple heuristic strategy, only fine-tunes the bias terms of the model. In addition, prompt-based tuning [31, 33, 36, 55, 58] is also an interesting direction, but we do not involve these methods here given that *the training curse of prompt-based methods* [11]. In addition, substantial works attempt to combine different PEFTs together through tailored mechanisms [8, 17, 23, 39, 69]. Finally, He et al. [14] provide a unified view of existing PEFTs, and more detailed descriptions of PEFTs are discussed in the survey literature [11].

3 Methodology

3.1 Pilot Experiments

To answer the first question: Can PEFTs from the language domain be transferred directly to graph-based tasks? We evaluate the performance of three mainstream PEFTs (Adapter [15], LoRA [16], and BitFit [3]) on large- and small-scale graph-based downstream tasks, respectively. To be specific, on the large-scale datasets, i.e., MolHIV (41K) and MolPCBA (437K) [19], we first average the

results of each PEFT based on the pre-trained Graphormer [62], and then subtract the average result of full fine-tuning. Here, we refer to the final result as *Performance Gap*, as shown in Fig. 1. Similar operations are also conducted on seven small-scale datasets ($0.6 \sim 2.4$K), i.e., FreeSolv, ESOL, BBBP, Estrogen-α, Estrogen-β, MetStab$_{low}$, and MetStab$_{high}$ [13, 45, 59], based on another pre-trained model MAT [40]. From the comparison in Fig. 1, we can observe that the performance of traditional PEFTs is far from full fine-tuning on graph-based tasks, especially on large-scale datasets, across varying degrees of the ratio of trainable parameters.

To shed light on why there is such a significant gap between traditional PEFTs and full fine-tuning, we investigate the feature distribution of different methods inspired by Lian et al. [34]. Specifically, based on BBBP and pre-trained MAT, we first take the hidden representation of a virtual node (similar to the [CLS] token in NLP [9]) in the last layer as the entire graph representation. Then, for the identical input, the graph feature representations from diverse methods are visualized in Fig. 2. More results are shown in Appendix A.4. Given that full fine-tuning updates all parameters of the model, its performance can be seen as an "upper bound" for PEFT[3]. Therefore, a good PEFT is believed that it should have similar behavior with full fine-tuning, such as the encoding of features. However, from the comparison in Fig. 2, we can observe that the feature distributions encoded by traditional PEFTs are shifted compared to full fine-tuning, which here is called *feature distribution shift*.

To understand the reason underlying this phenomenon, we revisit the relationship between GTNs and vanilla Transformers. For the encoding of node/token, they have highly similar operations, e.g., encoding the representation of node/token through an embedding layer. However, there are significant discrepancies in terms of the encoding of position (or edge in the graph). Specifically, only the position embedding layer is utilized within vanilla Transformers in NLP, while most existing GTNs extract diverse graph structure information as the inductive bias and then inject them into the model [40, 49, 62, 65, 68], since the graph structure contains rich edge semantic information. In addition, recent researches also demonstrate the significant effectiveness of graph structure in learning graph representation [6, 10, 30]. Motivated by these observations, in this paper, we attempt to introduce graph structure as the inductive bias to alleviate the feature distribution shift issue.

3.2 Structure-Aware Parameter-Efficient Fine-Tuning

For the parameter-efficient module, we believe that the following principles should be taken into consideration: (i) it can explicitly encode graph structure during the fine-tuning process; (ii) it should satisfy the main property of PEFT — lightweight [15, 16, 38]; (iii) it should be easy to implement and can be integrated into diverse GTNs.

The design of parameter-efficient module. Inspired by the design of Graph Convolutional Networks (GCN) [29, 35], which can model both graph structure and node representation simultaneously, we leverage this operation to explicitly introduce graph structure into the model. Here, we give the following definition:

$$X' = \text{GraphConv}(S, X; W) = \sigma(SXW) \qquad (1)$$

where $X, X' \in \mathbb{R}^{n \times d}$ (n: the sequence length, d: the hidden representation dimension) refer to the input, output of the module, respectively. $S \in \mathbb{R}^{n \times n}$ indicates the introduced graph structure information (e.g., the adjacency matrix of graph), $W \in \mathbb{R}^{d \times d}$ is the learnable weight, and $\sigma(\cdot)$ indicates the nonlinear activation function. Further, following the lightweight principle, we decompose W into two low-rank matrices to reduce the number of learnable parameters, i.e., $W = W_{down} W_{up}$, where $W_{down} \in \mathbb{R}^{d \times r}, W_{up} \in \mathbb{R}^{r \times d}$ and r is called the bottleneck size. Moreover, to stabilize the training process of PEFT, we insert two LayerNorm layers [2] before and after $\text{GraphConv}(\cdot)$, respectively, as depicted in Fig. 3.

Overall, the pipeline of our PEFT module is as follows: firstly, the input (X) goes through the first LayerNorm layer, then passes $\text{GraphConv}(\cdot)$ by absorbing the graph structure information (S). Next, we construct a skip connection between the output of $\text{GraphConv}(\cdot)$ and the normalized input (X'). Lastly, the final output (X'') is obtained through the second LayerNorm layer, i.e.:

$$X' = \text{LN}(X), X'' = \text{LN}\big(X' + \sigma(SX'W_{down}W_{up})\big) \qquad (2)$$

where $\text{LN}(\cdot)$ represents the LayerNorm layer. In Fig. 3, we describe in detail the architecture of our proposed PEFT (G-Adapter) and compare it with traditional PEFTs. In addition, thanks to the

---

[3]Note that this claim is not rigorous, since PEFTs might outperform full fine-tuning on small-scale datasets.

modular and lightweight properties, G-Adapter can be seamlessly integrated into diverse GTNs. We also provide some general pseudo-code to execute our approach in Appendix A.3.

The selection of graph structure information. To start with, we consider the adjacency matrix (with self-connections) of the graph: $S_1 = A + I_n$, where $A, I_n \in \mathbb{R}^{n \times n}$ refer to the adjacency matrix and identity matrix, respectively. Then, following Kipf and Welling [29], we introduce the degree matrix of nodes to normalize the adjacency matrix: $S_2 = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, where $\tilde{A} = S_1$, $\tilde{D}$ is the diagonal matrix and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. In addition, we propose a distance-based graph structure information: $S_3 = [\mathrm{dis}(v_i, v_j)]_{n \times n}$, where $\mathrm{dis}(v_i, v_j)$ refers to the distance of the shortest path (or the inter-atomic distance in the molecular graph) between two nodes v_i and v_j. Last, we combine the adjacency and distance to construct a hybrid structure information: $S_4 = \alpha \cdot \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} + \beta \cdot [\mathrm{dis}(v_i, v_j)]_{n \times n}$, where α, β are scalar hyper-parameters to balance the impacts of the adjacency and distance terms. We evaluate the proposed graph structure information (S_1, S_2, S_3, S_4) on a range of graph-based tasks, and the detailed comparisons are presented in Sec. 4.2.

3.3 Bregman Proximal Point Optimization

It is expected that the feature distribution encoded by PEFT should be aligned with full fine-tuning as much as possible, as discussed in Sec. 3.1. However, the feature distribution of full fine-tuning is unavailable during the training process of PEFT. Interestingly, we observe that the feature distribution encoded by the original model parameters has a high similarity with full fine-tuning, as shown in Fig. 4. It indicates that full fine-tuning only slightly modulates the value of model parameters but does not change the ability of the model to encode features.

Figure 4: Comparison of feature distribution between full fine-tuning and the original model parameters, where Jensen-Shannon divergence is 0.27%.

Therefore, to maintain consistency with the feature distribution of the original parameters, we propose Bregman proximal point optimization strategy [22] to prevent the model from aggressive update. Specifically, for the pre-trained model $f(\cdot; \theta)$ with trainable parameters θ, at the $(t+1)$-th iteration, we have

$$\theta_{t+1} = \arg\min_{\theta} \ (1 - \mu) \cdot \mathcal{L}_{\mathrm{vanilla}}(\theta) + \mu \cdot \mathcal{L}_{\mathrm{bregman}}(\theta, \theta_t) \tag{3}$$

where $\mu > 0$ is a hyper-parameter, $\mathcal{L}_{\mathrm{vanilla}}$ is a common classification or regression loss function, and $\mathcal{L}_{\mathrm{bregman}}$ is the Bregman divergence defined as:

$$\mathcal{L}_{\mathrm{bregman}}(\theta, \theta_t) = \mathbb{E}_{x \sim \mathcal{D}} \Big[\ell \big(f(x; \theta), f(x; \theta_t) \big) \Big] \tag{4}$$

where the input x is derived from the training set $\mathcal{D}$, and here we leverage the symmetric KL-divergence, i.e., $\ell(p, q) = \mathrm{KL}(p||q) + \mathrm{KL}(q||p)$. Intuitively, $\mathcal{L}_{\mathrm{bregman}}$ serves as a regularizer and prevents θ_{t+1} from deviating too much from the previous iteration θ_t, therefore can effectively retain the capacity of encoding feature in the pre-trained model $f(\cdot; \theta)$.

4 Experiments

4.1 Setup

Datasets & Evaluation Protocols. We conduct our experiments on nine benchmark datasets: MolHIV, MolPCBA, FreeSolv, ESOL, BBBP, Estrogen-α, Estrogen-β, MetStab$_{\mathrm{low}}$ and MetStab$_{\mathrm{high}}$ [13, 19, 45, 59], where MolHIV (41K) and MolPCBA (437K) are two large-scale molecular property prediction datasets and the others are small-scale molecular datasets ($0.6 \sim 2.4\mathrm{K}$). We provide more descriptions of datasets in Appendix A.2. Following the previous settings [40, 62], we employ the scaffold split on MolHIV, MolPCBA, BBBP, and Estrogen-α/β, and then the random split on the rest of datasets. For the evaluation protocols, MolPCBA is evaluated by Accuracy Precision (AP), FreeSolv and ESOL are evaluated by RMSE, and the others are evaluated by AUC.
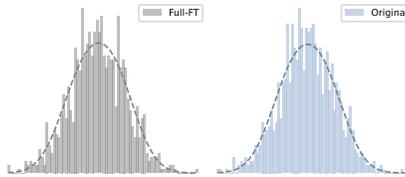
Table 1: Comparison of PEFTs and full fine-tuning on small-scale datasets. The results are averaged from six seeds, and the subscript is the standard deviation, where **bold** indicates the best results in PEFTs. ∗ represents the mean ratio of trainable parameters over seven datasets.

Method	Ratio* (γ)	RMSE $(\downarrow)$		AUC $(\uparrow)$				
		FreeSolv	ESOL	BBBP	Estrogen-α	Estrogen-β	MetStab$_{low}$	MetStab$_{high}$
Full Finetunig	100%	$0.286_{\pm 0.035}$	$0.270_{\pm 0.037}$	$0.764_{\pm 0.008}$	$0.979_{\pm 0.002}$	$0.778_{\pm 0.005}$	$0.863_{\pm 0.025}$	$0.878_{\pm 0.032}$
Adapter	2.52%	$0.327_{\pm 0.011}$	$0.320_{\pm 0.072}$	$0.724_{\pm 0.009}$	$0.978_{\pm 0.024}$	$0.768_{\pm 0.021}$	$0.846_{\pm 0.034}$	$0.859_{\pm 0.028}$
Hyperformer	2.43%	$0.310_{\pm 0.020}$	$0.321_{\pm 0.045}$	$0.727_{\pm 0.012}$	$0.977_{\pm 0.027}$	$0.770_{\pm 0.013}$	$0.842_{\pm 0.022}$	$0.853_{\pm 0.023}$
Compacter	1.56%	$0.314_{\pm 0.028}$	$0.316_{\pm 0.038}$	$0.730_{\pm 0.022}$	$0.971_{\pm 0.034}$	$0.764_{\pm 0.027}$	$0.832_{\pm 0.019}$	$0.860_{\pm 0.046}$
MAM	1.28%	$0.302_{\pm 0.019}$	$0.292_{\pm 0.022}$	$0.743_{\pm 0.014}$	$0.980_{\pm 0.011}$	$0.776_{\pm 0.022}$	$0.851_{\pm 0.023}$	$0.872_{\pm 0.054}$
LoRA	1.01%	$0.309_{\pm 0.032}$	$0.284_{\pm 0.054}$	$0.726_{\pm 0.012}$	$0.979_{\pm 0.007}$	$0.781_{\pm 0.039}$	$0.839_{\pm 0.022}$	$0.878_{\pm 0.027}$
BitFit	0.10%	$0.321_{\pm 0.048}$	$0.314_{\pm 0.031}$	$0.739_{\pm 0.005}$	$0.977_{\pm 0.019}$	$0.770_{\pm 0.035}$	$0.848_{\pm 0.031}$	$0.805_{\pm 0.045}$
G-Adapter (S_1)	0.71%	$\mathbf{0.280}_{\pm 0.012}$	$\mathbf{0.279}_{\pm 0.018}$	$0.750_{\pm 0.012}$	$0.976_{\pm 0.033}$	$\mathbf{0.791}_{\pm 0.022}$	$0.865_{\pm 0.036}$	$\mathbf{0.881}_{\pm 0.023}$
G-Adapter (S_2)	0.71%	$0.282_{\pm 0.014}$	$0.286_{\pm 0.022}$	$\mathbf{0.751}_{\pm 0.009}$	$\mathbf{0.981}_{\pm 0.017}$	$0.788_{\pm 0.031}$	$\mathbf{0.870}_{\pm 0.013}$	$0.874_{\pm 0.024}$
G-Adapter (S_3)	0.71%	$0.291_{\pm 0.008}$	$0.289_{\pm 0.017}$	$0.744_{\pm 0.011}$	$0.973_{\pm 0.015}$	$0.786_{\pm 0.034}$	$0.860_{\pm 0.031}$	$0.861_{\pm 0.018}$
G-Adapter (S_4)	0.71%	$0.298_{\pm 0.011}$	$0.282_{\pm 0.019}$	$0.747_{\pm 0.006}$	$0.975_{\pm 0.011}$	$0.775_{\pm 0.024}$	$0.858_{\pm 0.025}$	$0.869_{\pm 0.037}$

Pre-trained Models & Baselines. Two widely used pre-trained GTNs are leveraged as our backbones: Graphormer [62] and MAT [40]. In our experiments, we employ the base version of Graphormer, which has 12 layers Transformer encoders and is pre-trained on large-scale molecular dataset PCQM4M-LSC [18]. MAT is built on 8 encoders of Transformer, where the dimension of hidden representation is set to 1024. And, the node-level self-supervised learning serves as a pre-training task for MAT on ZINC15 [51]. For the baselines, we include full fine-tuning as a strong counterpart and six popular traditional PEFTs: Adapter [15], LoRA [16], BitFit [3], Hyperformer [26], Compacter [38], and MAM [14]. More descriptions per baseline are presented in Appendix A.1.

Implementation. Before fine-tuning, we begin by reusing the official released pre-trained checkpoints[4] to initialize our backbones, while the introduced modules are randomly initialized, and then use AdamW optimizer to fine-tune the models. We set fair hyperparametric search budgets for various PEFTs, and the detailed configurations per method on diverse datasets are shown in Appendix A.3.

4.2 Main Results

We report the comparison results on Mol-HIV and MolPCBA based on the pre-trained Graphormer in Tab. 2, and more results are shown in Tab. 1 on small-scale datasets based on the pre-trained MAT. From these comparisons, we can draw the following observations:

Observation I: Simple graph structure could deliver significant performance benefits. For instance, based on graph adjacent information, G-Adapters (S_1, S_2) obtain better results compared to G-Adapters (S_3, S_4) with distance-based structure information in Tab. 1. Moreover, in Tab. 2, G-Adapter (S_1) achieves the optimal performance among all PEFTs. We speculate that this may be because the graph adjacency matrix (S_1) is the complete information of graph structures, that is, the distance-based structure (S_3) can be derived from S_1. Therefore, this suggests that our model is not only remarkably expressive but also insensitive to the graph structure, which means that we do not have to design tailored graph structures except for graph adjacency information. In the following statement, we take G-Adapter (S_1) as our baseline unless otherwise specified.

Table 2: The comparison on two large-scale datasets MolHIV and MolPCBA.

Method	MolHIV		MolPCBA	
	Ratio (γ)	AUC $(\uparrow)$	Ratio (γ)	AP $(\uparrow)$
Full Finetunig	100%	$0.804_{\pm 0.006}$	100%	$0.272_{\pm 0.013}$
Adapter	1.24%	$0.743_{\pm 0.010}$	4.69%	$0.235_{\pm 0.009}$
Hyperformer	1.13%	$0.740_{\pm 0.012}$	4.37%	$0.246_{\pm 0.012}$
Compacter	0.64%	$0.752_{\pm 0.023}$	3.42%	$0.230_{\pm 0.023}$
MAM	0.57%	$0.758_{\pm 0.017}$	2.66%	$0.251_{\pm 0.016}$
LoRA	0.34%	$0.763_{\pm 0.014}$	2.42%	$0.246_{\pm 0.012}$
BitFit	0.16%	$0.709_{\pm 0.008}$	0.16%	$0.184_{\pm 0.011}$
G-Adapter (S_1)	0.24%	$\mathbf{0.790}_{\pm 0.011}$	1.89%	$\mathbf{0.269}_{\pm 0.008}$
G-Adapter (S_2)	0.24%	$0.788_{\pm 0.006}$	1.89%	$0.264_{\pm 0.012}$
G-Adapter (S_3)	0.24%	$0.772_{\pm 0.008}$	1.89%	$0.250_{\pm 0.011}$
G-Adapter (S_4)	0.24%	$0.781_{\pm 0.012}$	1.89%	$0.262_{\pm 0.011}$

Observation II: Our proposed G-Adapter consistently outperforms traditional PEFTs and offers a better trade-off between the ratio of trainable parameters (γ) and performance. For instance, Adapter, LoRA, and BitFit lag far behind G-Adapter on MolHIV and MolPCBA in Tab. 2. Although BitFit updates the fewest number of parameters ($\gamma = 0.16\%$), it also yields the worst performances (0.709 AUC, 0.184 AP). In comparison, our proposed G-Adapter achieves 79.0 AUC, 0.269 AP with $\gamma = 0.24\%, \gamma = 1.89\%$, respectively, which is also the optimal solution compared to other PEFTs.

---

[4] https://github.com/microsoft/Graphormer; https://github.com/ardigen/MAT

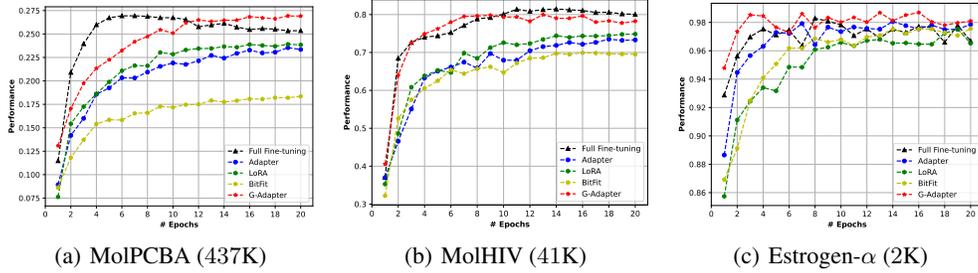| (a) MolPCBA (437K) | (b) MolHIV (41K) | (c) Estrogen-α (2K) |

Figure 5: The comparison of training efficiency between PEFTs (Adapter, LoRA, BitFit, and our proposed G-Adapter) and full fine-tuning on diverse scale datasets.

Table 3: The comparison of PEFTs and full fine-tuning in terms of inference speed (the millisecond per sample) and memory footprint across different bottleneck sizes.

Method	MolPCBA$_{(r=32)}$		MolHIV$_{(r=16)}$		FreeSolv$_{(r=8)}$		Estrogen-$\alpha_{(r=4)}$	
	Mem. (MB)	Infer (ms)	Mem. (MB)	Infer (ms)	Mem. (MB)	Infer (ms)	Mem. (MB)	Infer (ms)
Full Finetunig	185	0.81	185	1.39	161	0.42	161	1.11
Adapter	5.0	0.99	2.4	1.46	1.1	0.46	0.7	1.15
LoRA	5.0	0.97	2.4	1.43	1.1	0.47	0.7	1.13
BitFit	0.3	0.81	0.3	1.39	0.2	0.42	0.2	1.11
G-Adapter	2.9	0.93	1.4	1.41	0.7	0.44	0.4	1.12

Observation III: Compared to full fine-tuning, G-Adapter could achieve competitive and (most) superior performances on large- and small-scale datasets, respectively. For example, G-Adapter outperforms full fine-tuning by a significant margin (1.3% AUC) on Estrogen-β. We believe that the improvement on small-scale datasets is understandable, there are several reasons: (i) With decreasing the scale of the training set, it might meet serious over-fitting and catastrophic forgetting issues if the entire parameters are updated in full fine-tuning, whereas our method eases these concerns by only tuning G-Adapter blocks while freezing the original parameters. (ii) G-Adapter restricts the drastic updating of parameters via Bregman proximal point optimization strategy, which acts as a regularizer during the training process and therefore boosts the generalization capacity.

5 Analysis

5.1 Efficiency of Training, Inference and Memory Footprint

In this subsection, we mainly investigate the following questions: Does G-Adapter seriously affect the training (or convergence) efficiency and inference speed compared to full fine-tuning? And, can G-Adapter bring significant benefits to the storage of model weights, as we claimed before? Specifically, for the training efficiency, we evaluate PEFTs and full fine-tuning on three datasets with different scales[5]: MolPCBA (437K), MolHIV (41K), and Estrogen-α (2K). The experimental results are shown in Fig. 5, in which we observe that: (i) For the large-scale dataset, the convergence of PEFTs lags behind full fine-tuning by about $4 \sim 5$ epochs. However, this gap is significantly narrowed as the amount of training data decreases. (ii) Compared to traditional PEFTs, G-Adapter not only achieves faster convergence but also higher performance over datasets of varying scales.

For the inference efficiency and memory footprint, we adopt different bottleneck sizes ($r = 32, 16, 8, 4$) on MolPCBA, MolHIV, FreeSolv, and Estrogen-α, respectively. The experimental results are shown in Tab. 3, where we observe that: (i) Compared to full fine-tuning, the extra introduced modules result in a trivial inference delay, which is almost negligible with the bottleneck size decreasing. Note that BitFit does not introduce additional modules but merely tunes the bias terms, therefore it theoretically has the same inference efficiency as full fine-tuning. (ii) For the storage requirements, the distinction between PEFTs and full fine-tuning is remarkably significant. For an example on Estrogen-α ($r = 4$), full fine-tuning requires storing a complete checkpoint (161MB)

---

[5]The experiments are conducted on single NVIDIA GeForce RTX 3090 GPU (24G), where CPU is AMD EPYC 7763 64-Core Processor.

for each downstream task, while Adapter (LoRA), BitFit and G-Adapter only need to store 0.7MB, 0.2MB and 0.4MB checkpoint per task, respectively, which greatly reduces the storage requirements.

5.2 The Impact of Insertion Position and Components

We discuss the impact of potential designs on performance from two perspectives: (i) The insertion position. First, we insert G-Adapter block into the front and back of MHA, denoted as **pre_mha** and **post_mha**, respectively. Similarly, G-Adapter block is plugged before and after FFN, denoted as **pre_ffn** and **post_ffn**, respectively. Note that our baseline can be regarded as inserting the G-Adapter block in the middle of FFN. Finally, like the insertion position of Adapter in Fig. 3, we insert two G-Adapter blocks into MHA and FFN, denoted as **mha + ffn**. (ii) Importance of each component. First, we remove the adjacency matrix (denoted as **w/o. S**) to explore the importance of graph structure. Then, we separately remove the first, second, and both LayerNorm layers to explore individual effects denoted as **w/o. pre_ln**, **w/o. post_ln** and **w/o. ln**. We also explore the role of nonlinear activation function by removing it, denoted as **w/o. act_fn**. In addition, the effect of Bregman proximal point optimization is evaluated by only using the vanilla loss function, denoted as **w/o. breg**.

The experimental results are shown in Tab. 4, in which we can obtain that: (i) Plugging G-Adapter block into the front, middle, or back of FFN could yield better performance than MHA, and more blocks seem not to give better results, which is also consistent with the previous conclusion in NLP [14]. An intuitive explanation is that, in each encoder of Transformer, FFN concentrates most of the parameters ($\sim 67\%$), while MHA only accounts for $\sim 33\%$. Therefore, tweaking the weights of FFN may be a more efficient way for fine-tuning. (ii) Removing any of LayerNorm layers or the nonlinear activation function will hurt the performance. Moreover, removing the graph structure or Bregman proximal point optimization strategy would also significantly degrade the performance.

Table 4: The impact of insertion position and components on performance.

Method	MolHIV	MolPCBA
G-Adapter	0.790	0.269
G-Adapter (pre_mha)	0.752	0.246
G-Adapter (post_mha)	0.747	0.232
G-Adapter (pre_ffn)	0.781	0.258
G-Adapter (post_ffn)	0.770	0.260
G-Adapter (mha + ffn)	0.763	0.245
G-Adapter (w/o. S)	0.728	0.214
G-Adapter (w/o. pre_ln)	0.762	0.247
G-Adapter (w/o. post_ln)	0.755	0.250
G-Adapter (w/o. ln)	0.745	0.237
G-Adapter (w/o. act_fn)	0.766	0.240
G-Adapter (w/o. breg)	0.754	0.234

5.3 Can Graph Structure Information Benefit Traditional PEFTs?

One of the major contributions of G-Adapter is the introduction of graph structure, therefore a natural question is: can the graph structure enhance the traditional PEFTs as well? Given that the adjacency matrix (S) has performed well as graph structure information in previous experiments, we directly introduce S into Adapter and LoRA.

Table 5: The impact of graph structure information on traditional PEFTs.

Method	MolHIV	MolPCBA	BBBP	MetStab$_{low}$
Adapter	0.743	0.235	0.724	0.839
Adapter + S	0.749	0.242	0.733	0.844
LoRA	0.763	0.246	0.739	0.858
LoRA + S	0.766	0.252	0.742	0.861

Their modified updating formulas are presented in Appendix A.1. We conduct the experiments on four datasets: MolHIV, MolPCBA, BBBP, and MetStab$_{low}$. The results are reported in Tab. 5, in which we could observe a slight improvement in the modified methods compared to the original Adapter and LoRA. However, there is still a significant gap with our proposed G-Adapter, which further justifies that the traditional PEFT architectures are not suitable for handling graph-based tasks.

6 Conclusion & Limitations

In this paper, we propose a novel structure-aware PEFT method, G-Adapter, for graph-based tasks based on pre-trained GTNs. Unlike the traditional PEFTs, which lead to the issue of feature distribution shift, G-Adapter leverages the graph structure and Bregman proximal point optimization strategy to mitigate this concern. Extensive experiments on a variety of graph-based downstream tasks demonstrate the effectiveness of our proposed method. Although our approach demonstrates satisfactory performance, there are still some limitations: (i) Considering that the applicable scenarios of PEFT are large-scale models, our method is not tested on conventional graph network architectures (e.g., GCN [29], GIN [61]). Because these models are already quite lightweight, resulting in the advantages of PEFT not being sufficiently exploited. (ii) Limited by computational resources, we only evaluate two pre-trained GTNs (Graphormer and MAT). Nevertheless, thanks to the simplicity and generality of our proposed method, it can be applied to various graph Transformer-based models.

References

[1] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 7319–7328. `https://doi.org/10.18653/v1/2021.acl-long.568`

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *arXiv preprint arXiv:1607.06450*.

[3] Elad Ben-Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 1–9. `https://doi.org/10.18653/v1/2022.acl-short.1`

[4] Deyu Bo, Chuan Shi, Lele Wang, and Renjie Liao. 2023. Specformer: Spectral Graph Neural Networks Meet Transformers. In *The Eleventh International Conference on Learning Representations*.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models Are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. 1877–1901.

[6] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. 2022. Structure-Aware Transformer for Graph Representation Learning. In *Proceedings of the 39th International Conference on Machine Learning*. 3469–3489.

[7] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. 2023. NAGphormer: A Tokenized Graph Transformer for Node Classification in Large Graphs. In *The Eleventh International Conference on Learning Representations*.

[8] Jiaao Chen, Aston Zhang, Xingjian Shi, Mu Li, Alex Smola, and Diyi Yang. 2023. Parameter-Efficient Fine-Tuning Design Spaces. In *The Eleventh International Conference on Learning Representations*.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186. `https://doi.org/10/ggbwf6`

[10] Cameron Diao and Ricky Loynd. 2023. Relational Attention: Generalizing Transformers for Graph-Structured Tasks. In *The Eleventh International Conference on Learning Representations*.

[11] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2022. Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models. *arXiv preprint arXiv:2203.06904*.

[12] Chin-Lun Fu, Zih-Ching Chen, Yun-Ru Lee, and Hung-yi Lee. 2022. AdapterBias: Parameter-efficient Token-dependent Representation Shift for Adapters in NLP Tasks. *arXiv preprint arXiv:2205.00305*.

[13] A. Gaulton, L. J. Bellis, A. P. Bento, J. Chambers, M. Davies, A. Hersey, Y. Light, S. McGlinchey, D. Michalovich, B. Al-Lazikani, and J. P. Overington. 2012. ChEMBL: A Large-Scale Bioactivity Database for Drug Discovery. *Nucleic Acids Research* 40, D1 (2012), D1100–D1107. https://doi.org/10.1093/nar/gkr777

[14] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a Unified View of Parameter-Efficient Transfer Learning. In *International Conference on Learning Representations*.

[15] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*. 2790–2799.

[16] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

[17] Shengding Hu, Zhen Zhang, Ning Ding, Yadao Wang, Yasheng Wang, Zhiyuan Liu, and Maosong Sun. 2022. Sparse Structure Search for Delta Tuning. In *Advances in Neural Information Processing Systems*.

[18] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs. *arXiv preprint arXiv:2103.09430*.

[19] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2021. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687*.

[20] Weihua Hu*, Bowen Liu*, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2020. Strategies for Pre-training Graph Neural Networks. In *International Conference on Learning Representations*.

[21] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. GPT-GNN: Generative Pre-Training of Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '20)*. 1857–1867. https://doi.org/10.1145/3394486.3403237

[22] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2177–2190. https://doi.org/10.18653/v1/2020.acl-main.197

[23] Zeyinzi Jiang, Chaojie Mao, Ziyuan Huang, Yiliang Lv, Deli Zhao, and Jingren Zhou. 2023. Rethinking Efficient Tuning Methods from a Unified Perspective. *arXiv preprint arXiv:2303.00690*.

[24] Shibo Jie and Zhi-Hong Deng. 2022. FacT: Factor-Tuning for Lightweight Adaptation on Vision Transformer. *arXiv preprint arXiv:2212.03145*.

[25] Bowen Jin, Yu Zhang, Yu Meng, and Jiawei Han. 2023. Edgeformers: Graph-Empowered Transformers for Representation Learning on Textual-Edge Networks. In *The Eleventh International Conference on Learning Representations*.

[26] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-Efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 565–576. https://doi.org/10.18653/v1/2021.acl-long.47

[27] Ling Min Serena Khoo, Hai Leong Chieu, Zhong Qian, and Jing Jiang. 2020. Interpretable Rumor Detection in Microblogs by Attending to User Interactions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 8783–8790. `https://doi.org/10.1609/aaai.v34i05.6405`

[28] Jinwoo Kim, Dat Tien Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. 2022. Pure Transformers Are Powerful Graph Learners. In *Advances in Neural Information Processing Systems*.

[29] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

[30] Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. 2022. Rethinking Graph Transformers with Spectral Attention. In *Advances in Neural Information Processing Systems*.

[31] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 3045–3059. `https://doi.org/10.18653/v1/2021.emnlp-main.243`

[32] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 7871–7880. `https://doi.org/10.18653/v1/2020.acl-main.703`

[33] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 4582–4597. `https://doi.org/10.18653/v1/2021.acl-long.353`

[34] Dongze Lian, Zhou Daquan, Jiashi Feng, and Xinchao Wang. 2022. Scaling & Shifting Your Features: A New Baseline for Efficient Model Tuning. In *Advances in Neural Information Processing Systems*.

[35] Kevin Lin, Lijuan Wang, and Zicheng Liu. 2021. Mesh Graphormer. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 12919–12928. `https://doi.org/10.1109/ICCV48922.2021.01270`

[36] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-Tuning: Prompt Tuning Can Be Comparable to Fine-tuning Across Scales and Tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 61–68. `https://doi.org/10.18653/v1/2022.acl-short.8`

[37] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692*.

[38] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient Low-Rank Hypercomplex Adapter Layers. In *Advances in Neural Information Processing Systems*.

[39] Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. 2022. UniPELT: A Unified Framework for Parameter-Efficient Language Model Tuning. *arXiv preprint arXiv:2110.07577*.

[40] Łukasz Maziarka, Tomasz Danel, Sławomir Mucha, Krzysztof Rataj, Jacek Tabor, and Stanisław Jastrzębski. 2020. Molecule Attention Transformer. *arXiv preprint arXiv:2002.08264*.

[41] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. 2021. GraphiT: Encoding Graph Structure in Transformers. *arXiv preprint arXiv:2106.05667*.

[42] Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. 2022. Transformer for Graphs: An Overview from Architecture Perspective. *arXiv preprint arXiv:2202.08455*.

[43] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-Destructive Task Composition for Transfer Learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. 487–503. `https://doi.org/10.18653/v1/2021.eacl-main.39`

[44] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. AdapterHub: A Framework for Adapting Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 46–54. `https://doi.org/10.18653/v1/2020.emnlp-demos.7`

[45] Sabina Podlewska and Rafał Kafel. 2018. MetStabOn—Online Platform for Metabolic Stability Predictions. *International Journal of Molecular Sciences* 19, 4 (2018), 1040. `https://doi.org/10.3390/ijms19041040`

[46] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. *Improving Language Understanding by Generative Pre-Training*. Technical Report. OpenAI.

[47] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. *Language Models Are Unsupervised Multitask Learners*. Technical Report. OpenAI.

[48] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67.

[49] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying WEI, Wenbing Huang, and Junzhou Huang. 2020. Self-Supervised Graph Transformer on Large-Scale Molecular Data. In *Advances in Neural Information Processing Systems*, Vol. 33. 12559–12571.

[50] Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. AdapterDrop: On the Efficiency of Adapters in Transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 7930–7946. `https://doi.org/10.18653/v1/2021.emnlp-main.626`

[51] Teague Sterling and John J. Irwin. 2015. ZINC 15 – Ligand Discovery for Everyone. *Journal of Chemical Information and Modeling* 55, 11 (2015), 2324–2337. `https://doi.org/10.1021/acs.jcim.5b00559`

[52] Asa Cooper Stickland and Iain Murray. 2019. BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning. In *Proceedings of the 36th International Conference on Machine Learning*. 5986–5995.

[53] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, Vol. 30.

[55] Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou', and Daniel Cer. 2022. SPoT: Better Frozen Model Adaptation through Soft Prompt Transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 5039–5059. `https://doi.org/10.18653/v1/2022.acl-long.346`

[56] Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Jianshu Ji, Guihong Cao, Daxin Jiang, and Ming Zhou. 2021. K-Adapter: Infusing Knowledge into Pre-Trained Models with Adapters. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 1405–1418. `https://doi.org/10.18653/v1/2021.findings-acl.121`

[57] Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. AdaMix: Mixture-of-Adapter for Parameter-efficient Tuning of Large Language Models. *arXiv preprint arXiv:2205.12410*.

[58] Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. 2023. Multitask Prompt Tuning Enables Parameter-Efficient Transfer Learning. In *The Eleventh International Conference on Learning Representations*.

[59] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. 2018. MoleculeNet: A Benchmark for Molecular Machine Learning. *Chemical Science* 9, 2 (2018), 513–530. `https://doi.org/10.1039/C7SC02664A`

[60] Jun Xia, Yanqiao Zhu, Yuanqi Du, and Stan Z. Li. 2022. A Survey of Pretraining on Graphs: Taxonomy, Methods, and Applications. *arXiv preprint arXiv:2202.07893*.

[61] Keyulu Xu*, Weihua Hu*, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful Are Graph Neural Networks?. In *International Conference on Learning Representations*.

[62] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do Transformers Really Perform Badly for Graph Representation?. In *Advances in Neural Information Processing Systems*.

[63] Bruce X. B. Yu, Jianlong Chang, Lingbo Liu, Qi Tian, and Chang Wen Chen. 2022. Towards a Unified View on Visual Parameter-Efficient Transfer Learning. *arXiv preprint arXiv:2210.00788*.

[64] Weihao Yuan, Xiaodong Gu, Heng Li, Zilong Dong, and Siyu Zhu. 2023. Monocular Scene Reconstruction with 3D SDF Transformers. In *The Eleventh International Conference on Learning Representations*.

[65] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. 2020. Graph-Bert: Only Attention Is Needed for Learning Graph Representations. *arXiv preprint arXiv:2001.05140*.

[66] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning. In *The Eleventh International Conference on Learning Representations*.

[67] Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Chee-Kong Lee. 2021. Motif-Based Graph Self-Supervised Learning for Molecular Property Prediction. In *Advances in Neural Information Processing Systems*, Vol. 34. 15870–15882.

[68] Jianan Zhao, Chaozhuo Li, Qianlong Wen, Yiqi Wang, Yuming Liu, Hao Sun, Xing Xie, and Yanfang Ye. 2021. Gophormer: Ego-Graph Transformer for Node Classification. *arXiv preprint arXiv:2110.13094*.

[69] Han Zhou, Xingchen Wan, Ivan Vulić, and Anna Korhonen. 2023. AutoPEFT: Automatic Configuration Search for Parameter-Efficient Fine-Tuning. *arXiv preprint arXiv:2301.12132*.

A   Appendix

A.1   More Detailed Preliminaries

Transformer   Transformer [54], as one of the most popular network architectures so far, has been widely employed in diverse domains, such as NLP, computer vision and graph. In a standard encoder of Transformer, Multi-Head self-Attention (MHA) and Feed-Forward Networks (FFN) are two core components. Given the input $X \in \mathbb{R}^{n \times d}$, where n is the length of input sequence and d refers to the hidden size of representation, the query Q and key-value pairs K, V are first obtained by: $Q = XW_q + b_q, K = XW_k + b_k, V = XW_v + b_v$, where three projection matrices $W_{q/k/v} \in \mathbb{R}^{d \times d}$ and the bias $b_{q/k/v} \in \mathbb{R}^d$. Then, they are split into N_h heads ($Q_i, K_i, V_i \in \mathbb{R}^{d \times d_k}, d_k = d/N_h$) to pass the self-attention operation:

$$\text{Attn}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}}\right) V_i \tag{5}$$

After that, all head outputs are concatenated by a linear projection transformation ($W_o \in \mathbb{R}^{d \times d}, b_o \in \mathbb{R}^d$), then we can attain the final output of MHA:

$$\text{MAH}(X) = \text{Concat}(\text{head}_1, \text{head}_2, \cdots, \text{head}_h)W_o + b_o \tag{6}$$

where $\text{head}_i = \text{Attn}(Q_i, K_i, V_i)$. Another important module is FFN, which consists of two linear layers with a ReLU nonlinear activation function (where we still take X as the input for simplicity):

$$\text{FFN(X)} = \text{ReLU}(XW_{f1} + b_{f1})W_{f2} + b_{f2} \tag{7}$$

where $W_{f1} \in \mathbb{R}^{d \times d_{ff}}, W_{f2} \in \mathbb{R}^{d_{ff} \times d}, b_{f1} \in \mathbb{R}^{d_{ff}}, b_{f2} \in \mathbb{R}^d$. Note that $d_{ff} = d$ in some GTNs, such as Graphormer [62] and MAT [40]. In the following demonstration, for simplicity, we take $X, X' \in \mathbb{R}^{d \times d}$ as the input, output of a certain module, respectively.

Adapter   Houlsby et al. [15] insert two compact modules (i.e., Adapter blocks in Fig. 3) into the encoders of Transformer. Specifically, an Adapter block is composed of the down-projection transformation W_{down}, the up-projection transformation W_{up}, the nonlinear activation function $\sigma(\cdot)$, and the skip connection:

$$X' = X + \sigma(XW_{down})W_{up} \tag{8}$$

where $W_{down} \in \mathbb{R}^{d \times r}, W_{up} \in \mathbb{R}^{r \times d}$ and r is the bottleneck size of Adapter, which satisfies the condition $r \ll d$ for reducing the number of learnable parameters. To introduce the graph structure information ($S \in \mathbb{R}^{n \times n}$) in Sec. 5.3, we modify Eq. (8) as follows:

$$X' = X + \sigma(SXW_{down})W_{up} \tag{9}$$

LoRA   Based on the *low intrinsic rank* hypothesis [1], LoRA [16] reparametrizes the updating of pre-trained weight W by the low-rank decomposition, i.e., $W + \Delta W = W + W_{down}W_{up}$. For the practice in Transformer, LoRA injects two low-rank modules into the query and value weights (W_q, W_v) in a parallel connection manner. For the weight $W_* \in \{W_q, W_v\}$, we can obtain:

$$X' = X(W_* + s \cdot W_{down}W_{up}) \tag{10}$$

where $s \geq 1$ is a scalar hyper-parameter, and X' can be regarded as the new query or value. For an example of query $Q = XW_q$, the updated query $Q' = Q + s \cdot XW_{down}W_{up}$, which has a similar updating formulation with Adapter. To introduce the graph structure information ($S \in \mathbb{R}^{n \times n}$) in Sec. 5.3, we modify Eq. (10) as follows:

$$X' = XW_* + s \cdot SXW_{down}W_{up} \tag{11}$$

BitFit   Ben-Zaken et al. [3] employ a straightforward strategy to expose knowledge of the pre-trained models for downstream tasks via tuning the bias terms (b) of the model. To be specific, for the linear operation, the updated output is equal to:

$$X' = XW + b \tag{12}$$

where W refers to the pre-trained weights of the linear layer, and $b \in \{b_q, b_k, b_v, b_o, b_{f1}, b_{f2}\}$ (including the parameters of LayerNorm layers). Note that only b is updated during fine-tuning.

Hyperformer This method can be regarded as a variant of Adapter via using shared hypernetworks in multi-task scenarios. Specifically, Hyperformer [26] leverages the task conditioned hypernetworks to obtain the parameters of Adapter modules, i.e.:

$$X' = X + \text{LN}\big(\sigma(XW_{down})W_{up}\big) \tag{13}$$

where $W_{down} = W^D I_\tau, W_{up} = W^U I_\tau$, $W^D \in \mathbb{R}^{(d \times r) \times t}, W^U \in \mathbb{R}^{(r \times d) \times t}$, and $I_\tau \in \mathbb{R}^t$ is task embedding for each individual task (τ). Similar operations are also conducted on the parameters of LayerNorm layer $\text{LN}(\cdot)$.

Compacter Mahabadi et al. [38] introduce the Kronecker product and weights sharing tricks to reduce the ratio of trainable parameters in Adapter. Specifically, for the learnable weight $W \in \mathbb{R}^{d \times r}$, we can decompose W into multiple "small" matrices via Kronecker product ($\otimes$):

$$W = \sum_{i=1}^{n} A_i \otimes B_i \tag{14}$$

where $A_i \in \mathbb{R}^{n \times n}, B_i \in \mathbb{R}^{\frac{d}{n} \times \frac{r}{n}}$. This decomposition method can be applied to the trainable weights W_{down} and W_{up} in Adapter. Besides, Compacter also shares the A_i across all Adapter blocks.

MAM He et al. [14] investigate the traditional PEFTs from three perspectives: updated functional form, insertion form, and modified representation, and then offer a unified view to understand existing PEFTs: $X' = X + \Delta X$, where ΔX is learned by PEFT modules. Furthermore, based on their findings (e.g., FFN can better utilize modification than MHA at larger capacities), they propose a new PEFT method (MAM) by combining the most optimal choices.

A.2 More Descriptions about Datasets

We evaluate our proposed method and other baselines on nine benchmark datasets: MolHIV, MolPCBA, FreeSolv, ESOL, BBBP, Estrogen-α, Estrogen-β, MetStab$_{low}$, and MetStab$_{high}$ [13, 19, 45, 59]. Following the previous settings [40, 62], we split the dataset into the training set, the validation set, and the test set in the ratio of $8 : 1 : 1$, and the statistical information is shown in Tab. 6. Specifically, for each of dataset, MolHIV and MolPCBA are two molecular property prediction datasets, which are derived from the popular graph benchmark OGB. The target of this task is to predict the binary labels for each molecule, which indicates whether it has a particular property or not. FreeSolv and ESOL are regression tasks for predicting water solubility in terms of hydration free energy and log solubility. BBBP is a binary classification task for predicting the ability of a molecule to penetrate the blood-brain barrier. The aim of Estrogen-α and Estrogen-β is to predict whether a compound is active towards a given target based on experimental data from the ChEMBL database. Last, MetStab$_{low}$ and MetStab$_{high}$ are also binary classification tasks for predicting whether a compound has high or low metabolic stability.

Table 6: Statistics for different datasets. "# Train", "# Valid", and "# Test" indicate the number of training, validation, and test sets, respectively. Here, "Clf." and "Reg." refer to the classification and regression tasks, respectively.

Datasets	MolHIV	MolPCBA	FreeSolv	ESOL	BBBP	Estrogen-α	Estrogen-β	MetStab$_{low}$	MetStab$_{high}$
# Train	32,901	350,343	513	902	1,631	1,918	1,568	1,701	1,701
# Valid	4,113	43,793	64	113	204	240	196	213	213
# Test	4,113	43,793	65	113	204	240	197	213	213
Task Type	Clf.	Clf.	Reg.	Reg.	Clf.	Clf.	Clf.	Clf.	Clf.
Metric	AUC	AP	RMSE	RMSE	AUC	AUC	AUC	AUC	AUC

A.3 Detailed Experimental Configurations and Implementations

For a variety of methods, we perform a relatively fair hyper-parameter search in terms of learning rate and batch size from $\{1e-3, 2e-3, 1e-4, 1e-4, 1e-5, 2e-5\}$ and $\{32, 64, 128\}$, respectively. For the PEFTs with bottleneck architecture, we select the bottleneck size (r) from $\{4, 8, 16, 32, 48, 64, 128\}$ on various datasets. The detailed configurations are reported in Tab. 7. In addition, we also provide some general pseudo-code to illustrate how to integrate our proposed method into existing GTNs in Alg. 1.

Table 7: The detailed experimental configurations (batch size, learning rate, and bottleneck size) of various methods on a range of datasets, where Full-FT denotes full fine-tuning.

Method	MolHIV	MolPCBA	FreeSolv	ESOL	BBBP	Estrogen-α	Estrogen-β	MetStab$_{low}$	MetStab$_{high}$
				Batch Size / Learning Rate					
Full-FT	128 / 2e-5	128 / 2e-5	32 / 1e-5	32 / 1e-5	32 / 1e-5	32 / 1e-5	32 / 1e-5	32 / 1e-5	32 / 1e-5
Adapter	128 / 2e-3	128 / 2e-3	64 / 1e-3	64 / 2e-3	64 / 2e-3	32 / 1e-3	64 / 1e-3	32 / 2e-3	64 / 2e-3
Hyperformer	128 / 2e-3	128 / 1e-3	64 / 2e-3	32 / 1e-3	32 / 2e-3	32 / 1e-3	32 / 2e-3	64 / 1e-3	32 / 1e-3
Compacter	128 / 1e-3	128 / 2e-3	32 / 1e-3	32 / 1e-3	32 / 2e-3	64 / 2e-3	32 / 2e-3	64 / 2e-3	32 / 2e-3
MAM	128 / 2e-3	128 / 1e-3	32 / 1e-3	64 / 1e-3	32 / 1e-3	64 / 2e-3	32 / 1e-3	32 / 2e-3	64 / 2e-3
LoRA	128 / 1e-3	128 / 2e-3	32 / 2e-3	32 / 1e-3	64 / 2e-3	64 / 2e-3	32 / 1e-3	32 / 1e-3	32 / 1e-3
BitFit	128 / 1e-3	128 / 1e-3	32 / 1e-3	32 / 1e-3	32 / 1e-3	32 / 1e-3	32 / 1e-3	32 / 1e-3	32 / 1e-3
G-Adapter	128 / 2e-3	128 / 1e-3	32 / 1e-3	64 / 2e-3	64 / 1e-3	32 / 2e-3	32 / 2e-3	32 / 2e-3	32 / 2e-3
				Bottleneck Size (r)					
Adapter	16	64	16	8	16	4	32	128	64
Hyperformer	8	48	16	4	16	8	16	64	48
Compacter	16	48	32	8	16	4	16	48	32
MAM	8	32	8	16	32	4	8	32	32
LoRA	4	32	16	8	32	8	4	4	16
G-Adapter	4	48	4	16	4	4	4	16	64

A.4 More Experimental Results

Here, we conduct more experiments to demonstrate the feature distribution shift issue on different datasets (MolHIV, Estrogen-β, and MetStab$_{low}$) with two pre-trained GTNs (Graphormer and MAT). The experimental results are shown in Fig. 8, 9, and 10. In addition, we depict the relationship between Jensen-Shannon divergence (which measures the degree of discrepancy in feature distribution between PEFT and full fine-tuning) and performance across various datasets and methods in Fig. 6. We also supplement more comparisons in terms of the training efficiency, the impact of different designs, and the effect of graph structure information for traditional PEFTs on more datasets. The results are shown in Fig. 7, Tab. 8, 9, where we could draw the consistent conclusion as before.
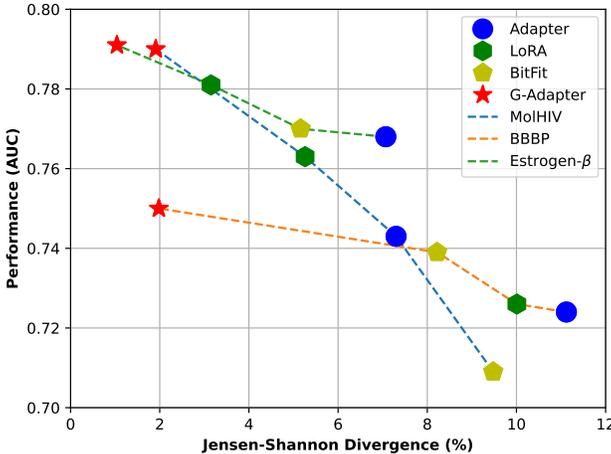

Figure 6: The relationship between Jensen-Shannon divergence and performance.

Table 8: The impact of graph structure information on traditional PEFTs.

Method	FreeSolv	ESOL	Estrogen-α	Estrogen-β	MetStab$_{high}$
Adapter	0.327	0.320	0.978	0.768	0.859
Adapter + S	0.311	0.314	0.980	0.773	0.866
LoRA	0.309	0.284	0.979	0.781	0.878
LoRA + S	0.297	0.280	0.978	0.789	0.880

Table 9: The impact of insertion position and components on performance.

Method	FreeSolv	ESOL	BBBP	Estrogen-α	Estrogen-β	MetStab$_{low}$	MetStab$_{high}$
G-Adapter	0.280	0.279	0.750	0.976	0.791	0.865	0.881
G-Adapter (pre_mha)	0.317	0.312	0.739	0.966	0.772	0.832	0.865
G-Adapter (post_mha)	0.304	0.298	0.733	0.963	0.787	0.847	0.863
G-Adapter (pre_ffn)	0.291	0.281	0.741	0.973	0.789	0.873	0.875
G-Adapter (post_ffn)	0.289	0.284	0.739	0.974	0.788	0.869	0.872
G-Adapter (mha + ffn)	0.294	0.296	0.735	0.972	0.777	0.864	0.869
G-Adapter (w/o. S)	0.351	0.335	0.706	0.950	0.737	0.823	0.832
G-Adapter (w/o. pre_ln)	0.321	0.319	0.711	0.962	0.745	0.852	0.841
G-Adapter (w/o. post_ln)	0.314	0.323	0.713	0.955	0.751	0.846	0.845
G-Adapter (w/o. ln)	0.343	0.328	0.705	0.956	0.726	0.833	0.839
G-Adapter (w/o. act_fn)	0.331	0.301	0.713	0.962	0.755	0.844	0.855
G-Adapter (w/o. breg)	0.346	0.325	0.704	0.951	0.743	0.849	0.834

(a) FreeSolv   (b) ESOL   (c) BBBP

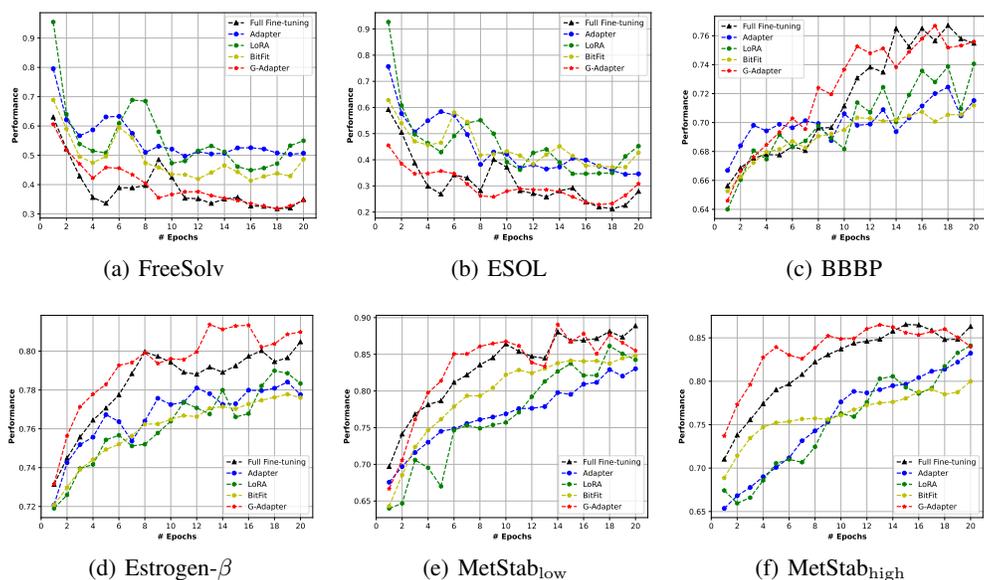(d) Estrogen-β (e) MetStab$_{low}$ (f) MetStab$_{high}$

Figure 7: More comparisons of training efficiency between PEFTs and full fine-tuning on more datasets. Note that the evaluation protocol for FreeSolv and ESOL is RMSE (the lower, the better), while the others are AUC (the higher, the better).

Adapter: 7.30%   LoRA: 5.26%   BitFit: 9.48%   G-Adapter: 1.91%
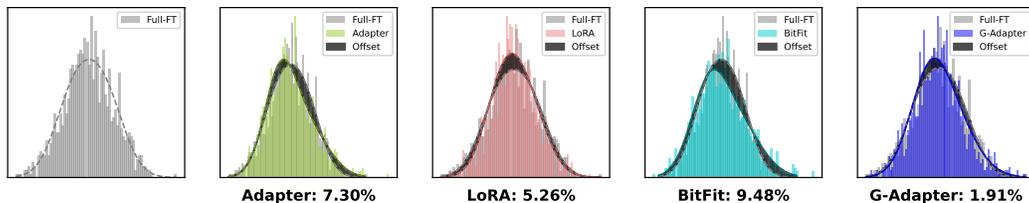
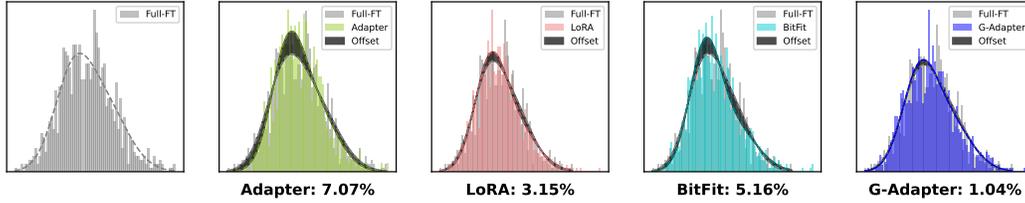Figure 8: Illustration of feature distribution shift on MolHIV with pre-trained Graphormer.

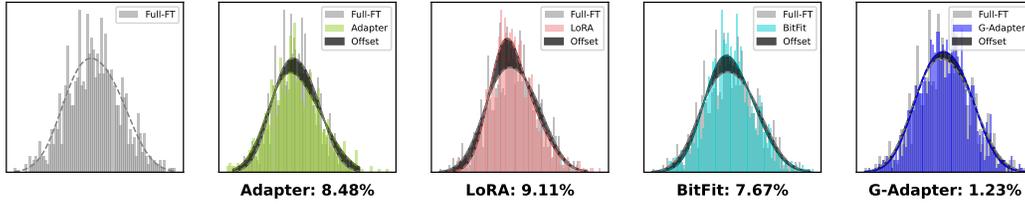Figure 9: Illustration of feature distribution shift on Estrogen-β with pre-trained MAT.

Figure 10: Illustration of feature distribution shift on MetStab$_{\text{low}}$ with pre-trained MAT.

Algorithm 1 Pseudo-code of G-Adapter in a PyTorch-like style.

```python
# Define the G-Adapter block
class GAdapter(nn.Module):
    def __init__(self, hidden_size, bottleneck_size)
        super(GAdapter, self).__init__()
        self.down = nn.Linear(hidden_size, bottleneck_size)
        self.up   = nn.Linear(bottleneck_size, hidden_size)
        self.pre_ln  = nn.LayerNorm(hidden_size)
        self.post_ln = nn.LayerNorm(hidden_size)
        self.act_fn  = nn.ReLU()

    def forward(self, x, s):
        # x: batch_size * sequence_length * hidden_size
        # s: batch_size * sequence_length * sequence_length
        x = self.pre_ln(x)
        x = self.act_fn(self.up(self.down(torch.matmul(s, x)))) + x
        x = self.post_ln(x)
        return x

# Apply the G-Adapter block into the built-in module
class Encoder(nn.Module):
    def __init__(self, hidden_size, bottleneck_size, *args, **kwargs)
        super(Encoder, self).__init__()
        ...
        # +++ #
        self.gadapter = GAdapter(hidden_size, bottleneck_size)
        # +++ #
        ...

    def forward(self, x, s):
        ...
        # +++ #
        x = self.gadapter(x, s)
        # +++ #
        ...
        return x
```

19