PROMPT TUNING FOR GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

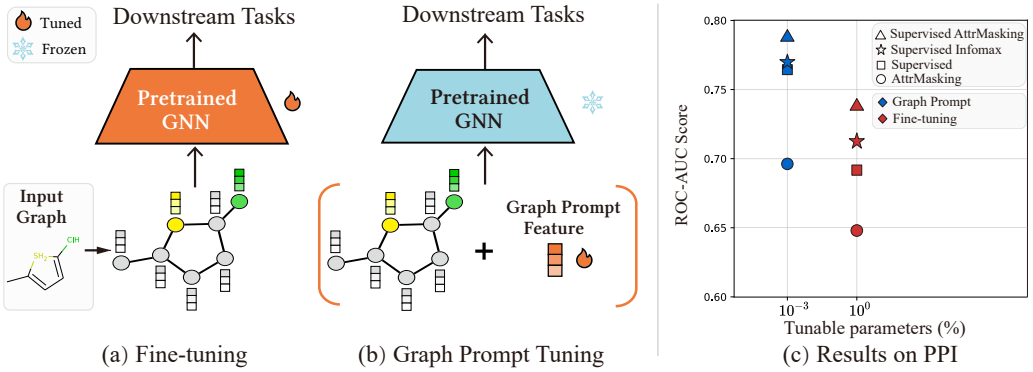
In recent years, prompt tuning has set off a research boom in the adaptation of pre-trained models. In this paper, we propose Graph Prompt as an efficient and effective alternative to full fine-tuning for adapting the pre-trianed GNN models to downstream tasks. To the best of our knowledge, we are the first to explore the effectiveness of prompt tuning on existing pre-trained GNN models. Specifically, without tuning the parameters of the pre-trained GNN model, we train a task-specific graph prompt that provides graph-level transformations on the downstream graphs during the adaptation stage. Then, we introduce a concrete implementation of the graph prompt, called GP-Feature (GPF), which adds learnable perturbations to the feature space of the downstream graph. GPF has a strong expressive ability that it can modify both the node features and the graph structure implicitly. Accordingly, we demonstrate that GPF can achieve the approximately equivalent effect of any graph-level transformations under most existing pre-trained GNN models. We validate the effectiveness of GPF on numerous pre-trained GNN models, and the experimental results show that with a small amount (about 0.1% of that for fine-tuning) of tunable parameters, GPF can achieve comparable performances as fine-tuning, and even obtain significant performance gains in some cases.

1 INTRODUCTION

Graph neural networks (GNNs) have achieved great success in graph representation learning (Kipf & Welling, 2017; Hamilton et al., 2017; Xu et al., 2019), which attract extensive attention from researchers. However, two fundamental challenges hinder the large-scale practical applications of GNNs. One is the scarcity of labeled data in the real world, which means the labeled data is only a tiny fraction of all collected data. In many practical situations, the acquisition of node labels is rather resource-consuming (Zitnik et al., 2018), and the lack of labeled data greatly degrades the performance of GNN models. The other challenge is low out-of-distribution generalization ability (Hu et al., 2020a; Knyazev et al., 2019; Yehudai et al., 2021; Morris et al., 2019). To be specific, if there exists a distribution shift between the training and testing data, the GNN model can no longer obtain satisfactory performance.

To overcome these challenges, recent efforts have been devoted to designing pre-trained GNN models (Xia et al., 2022b; Hu et al., 2020a;b; Lu et al., 2021). Similar to the pre-trained models in the language field, pre-trained GNN models are also trained on a large number of pre-training datasets, and then adapted to certain downstream tasks. Most existing pre-trained GNN models obey the "pre-train, fine-tune" learning strategy (Sun et al., 2022). Specifically, we train a GNN model with a massive corpus of pre-training graphs, then we apply the pre-trained GNN model as initialization and fine-tune the model parameters according to the downstream task.

The "pre-train, fine-tune" framework of pre-trained GNN models also arouses several crucial issues (Jin et al., 2020a). First, compared to ordinary GNN models, pre-trained GNN models usually contain several times the parameters of the normal one, which leads to high training overhead for adapting the pre-trained models on downstream tasks. Fine-tuning such models is quite difficult, time-consuming and space-consuming. Besides, it is difficult to guarantee that the model retains the generalization ability evaluated on the massive corpus of graphs. This problem is particularly acute when the downstream data is small in scale. In this case, the model is easy to over-fit on the downstream task, which makes the pre-training meaningless.



(a) Fine-tuning (b) Graph Prompt Tuning (c) Results on PPI

Figure 1: **Comparison between graph prompt tuning and fine-tuning.** (a) Current fine-tuning updates the parameters of the pre-trained GNN model. (b) Graph prompt tuning adds extra parameters in the input space while keeping the parameters of the pre-trained GNN model frozen. (c) Performance of different tuning methods on biology datasets (PPI) with the models pre-trained by different strategies in Hu et al. (2020a).

Inspired by the great success of natural language prompting (Li & Liang, 2021; Lester et al., 2021a; Liu et al., 2022a;b), we aim to investigate the effectiveness of prompt tuning on pre-trained GNN models. Prompt tuning brings us an alternative for the adaptation of pre-trained model on the downstream task: it freezes the parameters of the pre-trained model and modifies the input data. Unlike fine-tuning, prompt tuning does not change the pre-trained model but performs the data-space adaption by transforming the input of the downstream task. Such tuning strategy greatly reduces the overhead and difficulty of the adaptation on the downstream task.

However, applying prompt tuning in pre-trained GNN models is non-trivial and faces many challenges. First, compared to text data and image data, graph data is more irregular. Specifically, there does not exist a predetermined order for the nodes in graph, and the number of nodes in a graph and the number of the neighbors each node obtains are uncertain. These properties prevent existing prompt tuning methods from being directly transferred to graph data. Furthermore, graph data usually contains both structure information and node feature information, and they play different roles in different downstream tasks. For example, in social networks, the node features have a great impact on the classification results, while in protein networks, the graph structure plays a more essential role. Therefore, graph prompt tuning should consider both node feature information and structure information, and have the ability to transform any of them adaptively.

In this paper, we explore how to employ prompt tuning on pre-trained GNN models. We first propose a universal framework to express general prompt tuning on graph data, called *Graph Prompt (GP)*. Specifically, GP works on the input space and is defined as a learnable graph-level transformation. When adapting the pre-trained model on a certain downstream task, the input graph is first transformed by GP, then the prompted graph is processed through the pre-trained GNN model. During the adaptation stage, we keep the parameters of the pre-trained model frozen and only update the parameters of GP. After formally defining GP, we introduce a simple yet efficient implementation of GP, which we called *GP-Feature (GPF)*. GPF adds a shared learnable vector to all node features in the graph, which is easy to apply and can be used on most existing pre-trained GNN models. Figure 1 illustrates the difference of our proposed graph prompt tuning compared to traditional fine-tuning. Although GPF works directly on the feature space, we prove that it can also change the structure information of the graph implicitly. Specifically, we demonstrate that GPF can achieve approximately equivalent effect of any graph-level transformation theoretically under the architectures of most existing pre-trained GNN models. We validate the effectiveness of GPF on various existing pre-trained GNN models, pre-training and downstream datasets. The experimental results indicate that GPF can achieve satisfactory performances compared to fine-tuning. Overall, the contributions of our work can be summarized as follows:

- To the best of our knowledge, we are the first to investigate the universal prompt tuning on existing pre-trained GNN models, and we propose a framework called *Graph Prompt (GP)* to describe how to apply prompt tuning on pre-trained GNN models.

- We propose a concrete implementation of GP, called *GP-Feature (GPF)*, which works on the feature space of the input graph. By summarizing the architectures of existing pre-trained GNN models, we demonstrate that GPF can achieve the approximately equivalent effect of any graph-level transformation theoretically under the architectures of most existing pre-trained GNN models.
- We conduct extensive experiments on various pre-trained GNN models, pre-training and downstream datasets to validate the effectiveness of GPF. The experimental results indicate that with a small amount (about 0.1% of that for fine-tuning) of tunable parameters, GPF can achieve comparable performances of fine-tuning, and even obtain significant improvement in some cases.

2 Related works

2.1 Pre-training on Graphs

Pre-trained Language Models have revolutionized the landscape of natural language processing (NLP). Stimulated by that, tremendous efforts have been devoted to pre-trained graph models (PGMs). Supervised pre-training strategies are popular in the biochemical domain. Hu et al. (2020a) propose to pre-train GNNs for the prediction of molecules properties and protein functions existence. Both GROVER (Rong et al., 2020) and MGSSL (Zhang et al., 2021) propose to predict the presence of the motifs or generate them with the consideration that rich domain knowledge of molecules hides in the motifs. These works require domain-specific knowledge which limits their application areas. In light of this, unsupervised pre-training strategies proliferates recently. Graph Contrastive Learning (GCL) takes a large proportion of all the unsupervised strategies. DGI (Velickovic et al., 2019) and InfoGraph (Sun et al., 2019) are proposed to garner nodes or graphs representations via maximizing the mutual information between graph-level representations and substructure-level representations of different granularity. GraphCL (You et al., 2020) and its variant JOAO (You et al., 2021) propose various augmentations strategies for graph-level pre-training, creating different augmented views for contrastive learning. Apart from GCL, Masked Components Modeling (MCM) and Graph Context Prediction (GCP) are frequently adopted in unsupervised pre-training strategies (Xia et al., 2022b). MCM masks out some components of the graphs and trains the pre-training model to predict them. GCP is proposed to predict the context-aware properties of targets or their surrounding graph structures.

PGMs	Base Model	Optimal Pooling
Hu et al.	GIN	Mean/Sum
GPT-GNN	HGT	MLP
GCC	GIN	Sum
GraphCL	GIN	Sum
JOAO	GIN	Mean
AD-GCL	GIN	Mean/Sum
GROVER	GTransformer	Mean
DGI	GCN	Mean
LP-Info	GIN	Mean
SimGRACE	GIN	Sum

Table 1: Overview of pre-trained GNN models.

Most of model architectures of present PGMs are GCNs (Kipf & Welling, 2017) and GINs (Xu et al., 2019) according to Table 1. Besides, we can assert according to Table 1 that a majority of previous models take mean or sum as their graph pooling methods (or readout functions). Though more sophisticated pooling methods exist, Mesquita et al. (2020) propose that convolutions play a leading role in the learned representations, while graph pooling is not responsible for the success of GNNs on relevant and widely-used benchmarks. Linear graph pooling methods are effective and sufficient for learning representations.

2.2 Prompting Methods

Taking its rise from natural language processing (NLP), prompt-based learning is utilized to help pre-trained language models (LMs) "understand" the different downstream tasks (Liu et al., 2021a). It allows LM to be pre-trained on a massive amount of data. And by defining a new prompting function, the model can perform *few-shot* and even *zero-shot* learning, adapting to new scenarios with few data. Originally, Schick & Schütze (2020) propose a manually designed prompt pattern for NLP tasks, which is a language instruction prepended to the input text. Liu et al. (2021c) then propose a P-tuning

method to use the soft prompt instead of the previous manually designed prompt. P-tuning treats the prompts as task-specific continuous vectors and optimizes them via gradients during fine-tuning. Recently, prompting methods are further discussed on how to fix pre-trained models, while only updating the parameters of soft prompts to achieve excellent performance (Lester et al., 2021b; Liu et al., 2021b).

Benefiting from the success of prompts in NLP, the prompting method is then utilized in other areas. Jia et al. (2022) and Bahng et al. (2022) investigate the efficacy to adapt large-scale models in vision, such as CLIP (Radford et al., 2021) and other Transformer models like Vision Transformers (Dosovitskiy et al., 2021) and Swin Transformers (Liu et al., 2021d). Prompting methods have also brought great improvement on *few-shot* learning for Visual Language Models (VLMs). Frozen (Tsimpoukelli et al., 2021) and Flamingo (Alayrac et al., 2022) devote to stand on the shoulder of large pre-trained LMs, rapidly adapting to a variety of image and video understanding benchmarks. This pattern achieves a new state of the art for few-shot learning, simply by prompting the model with task-specific examples. However, the application of prompting methods in GNNs is still open. GPPT (Sun et al., 2022) is proposed to bridge the gap between pretext tasks and downstream tasks. It reformulates the downstream node classification looking the same as edge prediction, which is utilized as the pretext task during pre-training. To the best of our knowledge, we are the first to propose a general prompt tuning method for existing pre-trained GNN models.

3 METHODS

We propose the Graph Prompt (GP) for adapting pre-trained GNN models to downstream tasks. Specifically, during the downstream training stage, we keep the pre-trained model frozen and apply a learnable GP that transforms the input graph. Compared to fully fine-tuning the pre-trained models, GP only contains a small number of tunable parameters. We first define the notations in Section 3.1, then describe the definition of GP and GPF in Section 3.2. Finally, we provide some theoretical analysis of GPF in Section 3.3.

3.1 PRELIMINARIES

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}) \in \mathbb{G}$ represent a graph, where $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denote the node set and edge set respectively. The node features can be denoted as a matrix $\mathbf{X} = \{x_1, x_2, \ldots, x_N\} \in \mathbb{R}^{N \times F}$, where $x_i \in \mathbb{R}^F$ is the feature of the node v_i, and F is the dimensionality of node features. The adjacency matrix $\mathbf{A} \in \{0, 1\}^{N \times N}$ describes the structure relation between nodes, where $\mathbf{A}_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$. It is worth mentioning that there are many kinds of downstream tasks on graph, including node classification, link prediction, and graph classification. Our paper focuses on the graph classification task.

Pre-trained GNN Models. With the pre-training dataset $\mathcal{D}_{pt} = \{\mathcal{G}_{pt}\}$ and the specific pre-training task loss L_{pt} (Xia et al., 2022b), we obtain a pre-trained GNN encoder f by minimizing the pre-training loss on the pre-training dataset:

$$\min_f L_{pt} \tag{1}$$

Fine-Tuning Pre-trained Models. Given a pre-trained GNN encoder f and a downstream task dataset $\mathcal{D} = \{(\mathcal{G}_1, y_1), \ldots, (\mathcal{G}_m, y_m)\}$, usually, we fine-tune the parameters of the pre-trained model and maximize the likelihood of predicting the correct labels y:

$$\max_f P_f(y|\mathcal{G}) \tag{2}$$

3.2 GRAPH PROMPT

Graph Prompt Tuning. Drawing on the design of the prompt tuning in the NLP field (Liu et al., 2022a), we propose a GP that works on the input space. Given a frozen pre-trained GNN model f and a downstream task dataset $\mathcal{D} = \{(\mathcal{G}_1, y_1), \ldots, (\mathcal{G}_m, y_m)\}$, our target is to obtain a task-specific GP $g_\phi : \mathbb{G} \to \mathbb{G}$ parameterized by ϕ. The GP g_ϕ transforms the input graph $\mathcal{G}$ into the prompted graph $g_\phi(\mathcal{G})$. During the downstream task training, we select the optimal parameters of ϕ that maximizes

the likelihood of predicting the correct labels y without tuning the pre-trained model f, which can be formulated as:

$$\max_{\phi} P_{f,\phi}(y|g_\phi(\mathcal{G})) \tag{3}$$

During the evaluation stage, the input graph $\mathcal{G}$ is first transformed by GP g_ϕ, then the prompted graph $g_\phi(\mathcal{G})$ is processed through the frozen GNN model f.

Graph Prompt Design. Our goal is to explore GP as a practical adaptation method, thus, we discuss the concrete design of GP in this part. According to Formula 3, any learnable graph-level transformation can be designed as a GP. In this paper, we propose a simple yet efficient GP architecture, called *GP-Feature* (GPF), to adapt pre-trained models for specific downstream graph classification tasks. GPF concentrates on the transformations of the feature space for the input graph, and it is a learnable F-dimensional vector, which can be denoted as:

$$p \in \mathbb{R}^F \tag{4}$$

This GPF is added to the input graph features $\mathbf{X}$ and generate the prompted features $\mathbf{X}^*$:

$$\mathbf{X} = \{x_1, x_2, \ldots, x_N\} \tag{5}$$

$$\mathbf{X}^* = \{x_1 + p, x_2 + p, \ldots, x_N + p\} \tag{6}$$

Then, the prompted features $\mathbf{X}^*$ replace the initial features $\mathbf{X}$ and are processed by the pre-trained model.

3.3 Prompt Analysis

In this section, we further elaborate the properties of our proposed GPF, which guarantee its practical effectiveness.

Availability. GPF has high availability, because it works on the feature space of the input graph. As long as the pre-trained model supports the input node features, GPF can be easily adapted to this model. It is worth mentioning that some pre-trained GNN models focus on the graphs without initial node features (*e.g.*, GCC (Qiu et al., 2020)). In this case, the node features are generated by artificially synthesized representations, such as random features (Sato et al., 2021), and positional encoding (Wang et al., 2022; You et al., 2019). GPF can also be applied to these synthetic node features.

Transformation Diversity. According to Formula 3, GP is essentially a learnable graph-level transformation, which searches suitable adjustments of the downstream task graph for particular pre-trained model. In order to find the optimal transformation, GP should have a large enough transformation space. Now we will demonstrate that our proposed GPF is able to provide sufficiently complex transformations for most existing pre-trained GNN models.

From the Table 1, we can find that a large part of existing pre-trained GNN models apply *GCN* (Kipf & Welling, 2017) or *GIN* (Xu et al., 2019) as their backbone models. A single layer of these two models can be expressed as follows respectively:

$$\mathbf{H} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{\Theta} \qquad \text{(GCN)} \tag{7}$$

$$\mathbf{H} = h_{\mathbf{\Theta}}((\mathbf{A}+(1+\epsilon)\cdot\mathbf{I})\cdot\mathbf{X}) \qquad \text{(GIN)} \tag{8}$$

where $\widetilde{\mathbf{A}} = \mathbf{A}+\mathbf{I}$ represents the adjacency matrix with inserted self-loops, $\widetilde{\mathbf{D}}_{ii} = \sum_j \widetilde{\mathbf{A}}_{ij}$ represents the diagonal degree matrix, $\mathbf{\Theta}$ denotes a linear transformation layer and $h_{\mathbf{\Theta}}$ denotes a MLP. Except for the selection of the backbone model, the pooling strategies of the pre-training models are also similar. For GNN models, hierarchical pooling is not as effective as that in the image field. Previous work (Mesquita et al., 2020) indicates that GNN models do not need such complex pooling, and simple readout functions can achieve better performances. Consequentially, according to Table 1, most existing pre-trained GNN models use *mean* or *sum* readout function to calculate the graph representations, which can be expressed as follows:

$$h_{\mathcal{G}} = \sum_{v_i \in \mathcal{V}} h_i \quad \text{(sum)} \tag{9}$$

$$h_{\mathcal{G}} = \frac{1}{|\mathcal{V}|}\sum_{v_i \in \mathcal{V}} h_i \quad \text{(mean)} \tag{10}$$

Next, we prove that for a pre-trained GNN model with GCN/GIN as the backbone model and mean or sum as the readout function, our proposed GPF can *reach approximate results with arbitrary graph-level transformations*.

Proposition 1. *For a frozen pre-trained GNN model f^* with GCN/GIN as the backbone model and mean or sum as the readout function, an input graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ and an arbitrary graph-level transformation $g_{\phi^*} : \mathbb{G} \to \mathbb{G}$, there exists a GPF p^* that satisfies:*

$$f^*(\mathbf{A}, \mathbf{X} + p^*) \approx f^*(g_{\phi^*}(\mathbf{A}, \mathbf{X}))$$

For analytical simplification, we consider a single layer GIN with sum readout function, and replace the MLP with a linear transformation layer. Such pre-trained GNN model f can be expressed as:

$$\mathbf{H} = (\mathbf{A} + (1 + \epsilon) \cdot \mathbf{I}) \cdot \mathbf{X} \cdot \mathbf{\Theta} \tag{11}$$

$$h_{\mathcal{G}} = \sum_{v_i \in \mathcal{V}} h_i \tag{12}$$

where $\mathbf{\Theta}$ is a linear transformation layer. we assume this model is pre-trained as f^* in advance, which means the learnable parameters ϵ and $\mathbf{\Theta}$ are preset as ϵ^* and $\mathbf{\Theta}^*$ respectively by the pre-training tasks. When we apply graph prompt tuning to adapt the pre-trained model to specific downstream tasks, these learnable parameters are frozen.

Proposition 2. *With an input graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$, an arbitrary graph-level transformation g_ϕ can be decoupled to a series of following transformations:*

Feature transformations. *Changing the node features $g_{ft}(\mathbf{X}) = \mathbf{X}'$.*

Link transformations. *Adding or removing edges $g_{lt}(\mathbf{A}) = \mathbf{A}'$.*

Isolated component transformations. *Adding or removing isolated components (sub-graphs) $g_{ict}(\mathbf{A}, \mathbf{X}) = (\mathbf{A}', \mathbf{X}')$. The word "isolated" here means that the component (sub-graph) does not have any links with the rest of the graph.*

Proposition 2 indicates that an arbitrary graph-level transformation is a combination of above three transformations. For example, removing one node from the initial graph can be regarded as removing its connected edges first, and then removing the isolated node.

Proposition 3. *For a frozen pre-trained model f^*, an input graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ and an arbitrary feature transformation $g_{ft^*} : g_{ft^*}(\mathbf{X}) = \mathbf{X}'$, there exists a GPF $p*$ that satisfies:*

$$f^*(\mathbf{A}, \mathbf{X} + p^*) = f^*(\mathbf{A}, g_{ft^*}(\mathbf{X}))$$

The proof of Proposition 3 can be found in Appendix A.1. Proposition 3 indicates that our proposed GPF can cover all the feature transformations on the graph level. GPF adds the same feature perturbation $p \in \mathbb{R}^F$ to every node on the graph, but it can achieve the same effect as adding independent feature perturbations to each node respectively under the pre-trained GNN models described above. To achieve arbitrary feature transformations, a direct way is training a learnable matrix $\Delta\mathbf{X} \in \mathbb{R}^{N \times F}$, then calculating the transformed feature matrix $\mathbf{X}' = \mathbf{X} + \Delta\mathbf{X}$. Our proposed GPF can achieve a completely equivalent effect to this approach but reduces the number of learnable parameters from $N \times F$ to F, which indicates a huge gap in practical resource usage.

Proposition 4. *For a frozen pre-trained model f^*, an input graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ and an arbitrary link transformation $g_{lt^*} : g_{lt^*}(\mathbf{A}) = \mathbf{A}'$, there exists a GPF $p*$ that satisfies:*

$$f^*(\mathbf{A}, \mathbf{X} + p^*) = f^*(g_{ft^*}(\mathbf{A}), \mathbf{X})$$

More detailed proof can be found in Appendix A.2. Proposition 4 indicates that GPF can also cover all the link transformations. Intuitively, link transformations have a close connection with the change of the adjacency matrix, which is independent of the node features. However, our conclusion demonstrates that under the architecture of most existing GNN models, changes in the feature space and changes in the structure space can achieve equivalent effects. Learnable link transformations are also closely related to graph structure learning (Zhu et al., 2021; Gao et al., 2020; Jin et al., 2020b; Xu et al., 2021). These approaches treat the adjacency matrix as learnable variables, and add extra regularization to make it trainable. Our proposed GPF works on the feature space. It requires no training tricks, and reduces the number of parameters from $N \times N$ to F, but maintains the ability to achieve an equivalent effect with arbitrary link transformation.

Proposition 5. *For a frozen pre-trained model f^*, an input graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ and an arbitrary isolated component transformation $g_{ict^*} : g_{ict^*}(\mathbf{A}, \mathbf{X}) = (\mathbf{A}', \mathbf{X}')$, there exists a GPF $p*$ that satisfies:*

$$f^*(\mathbf{A}, \mathbf{X} + p^*) = f^*(g_{ict^*}(\mathbf{A}, \mathbf{X}))$$

The detailed proof is presented in Appendix A.3. Isolated component transformation has the ability to change the graph scale, and this kind of graph-level transformation is rarely considered by previous works. Proposition 5 expounds that GPF can also cover isolated component transformations.

Proposition 6. *For a frozen pre-trained model f^*, an input graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ and a series of transformations $\mathbf{g} = \{g_1, g_2, \cdots, g_k\}$ composed of g_{ft}, g_{lt} and g_{ist}, there exists a GPF $p*$ that satisfies:*

$$f^*(\mathbf{A}, \mathbf{X} + p^*) = f^*(\mathbf{g}(\mathbf{A}, \mathbf{X}))$$

The detailed proof can be found in Appendix A.4. From the above discussion, we demonstrate that GPF can achieve the same effect with any graph-level transformation for the pre-trained GNN model described as Formula 11 and Formula 12. GPF works directly on the feature space on the input graph, but it can provide the transformation space that covers all graph-level transformations. In practice, the pre-trained GNN models apply non-linear activation layers between GNN layers and may replace the linear transformations with MLPs, so Proposition 1 uses $\approx$ rather than $=$.

4 EXPERIMENTS

4.1 EXPERIMENT SETUP

Pre-trained GNN models. We employ four different pre-trained GNN models, and all of them use GIN (Xu et al., 2019) as the backbone model. These pre-trained models can be divided into following two collections according to whether they use node features during pre-training:

- *Pre-training with raw features.* This kind of pre-trained model uses raw node features during the pre-training process. We experiment with the pre-trained models proposed in Hu et al. (2020a), GraphCL (You et al., 2020), and SimGRACE (Xia et al., 2022a).
- *Pre-training without raw features.* This kind of pre-trained model does not use raw node features during the pre-training process. They generate synthesized node representations to replace raw node features, such as random features or Laplacian features. We experiment with GCC (Qiu et al., 2020) of this kind.

Pre-training datasets. For the pre-trained model proposed in Hu et al. (2020a), GraphCL and SimGRACE, we use the datasets that are published by Hu et al. (2020a) to pre-train the models, which include a dataset for the chemistry domain and a dataset for the biology domain. The dataset of the chemistry domain contains 2 million unlabeled molecules sampled from the ZINC15 database (Sterling & Irwin, 2015) an 256K labeled molecules from preprocessed ChEMBL dataset (Mayr et al., 2018; Gaulton et al., 2012). The dataset of the biology domain contains 395K unlabeled and 88K labeled protein ego-networks from PPI networks. To pre-train GCC, we follow the instructions in its paper and use the datasets of Academia (Ritchie et al., 2016), DBLP (SNAP) (Yang & Leskovec, 2012), DBLP (NetRep) (Ritchie et al., 2016), IMDB (Ritchie et al., 2016), Facebook (Ritchie et al., 2016) and LiveJournal (Backstrom et al., 2006) to pre-train the model.

Downstream datasets. We select the downstream datasets for the pre-trained model according to its pre-training datasets. For the models pre-trained on the chemistry dataset, we use 8 binary classification datasets published by Hu et al. (2020a) as downstream tasks, which is contained in MoleculeNet (Wu et al., 2017). For the models pre-trained on the biology dataset, we apply the pre-trained models to 40 binary classification tasks, and each task is to predict a fine-grained biological function. These downstream tasks are also provided by Hu et al. (2020a). As for GCC, we apply two graph classification datasets IMDB-BINARY and IMDB-MULTI from Yanardag and Vishwanathan (Yanardag & Vishwanathan, 2015) as downstream tasks.

More details about the experiments can be found in Appendix.

Dataset			BBBP	Tox21	ToxCast	SIDER	ClinTox	MUV	HIV	BACE
Pre-trained model	FT	GP	Out-of-distribution prediction (scaffold split)							
–	✓		65.87 ±2.28	74.69 ±0.78	62.59 ±1.10	58.74 ±1.74	59.36 ±4.13	72.40 ±3.16	74.08 ±1.78	69.44 ±6.23
Infomax	✓		**70.12 ±0.76**	74.93 ±0.37	62.25 ±0.55	57.43 ±0.85	68.82 ±2.94	75.87 ±1.54	76.21 ±0.81	77.79 ±0.56
		✓	60.85 ±0.74	67.75 ±0.60	58.75 ±0.17	58.05 ±0.31	61.62 ±1.51	75.35 ±1.33	71.90 ±0.28	65.98 ±0.43
EdgePred	✓		68.11 ±3.20	76.28 ±0.42	64.03 ±0.56	60.55 ±0.59	64.57 ±2.94	74.10 ±1.98	78.18 ±0.82	79.47 ±1.29
		✓	55.88 ±2.19	63.14 ±0.52	55.70 ±0.47	51.48 ±0.72	50.77 ±5.06	63.13 ±2.10	65.96 ±1.71	68.03 ±2.43
AttrMasking	✓		64.18 ±1.57	76.61 ±0.36	64.24 ±0.29	60.83 ±0.72	70.77 ±2.05	74.84 ±2.05	76.78 ±1.27	78.46 ±1.38
		✓	54.30 ±0.34	69.18 ±0.28	58.06 ±0.31	52.03 ±0.24	56.43 ±1.23	62.93 ±1.88	64.02 ±2.17	61.74 ±0.33
ContextPred	✓		65.27 ±2.53	75.43 ±0.76	64.11 ±0.41	60.68 ±0.59	65.01 ±3.55	75.73 ±0.72	77.28 ±1.04	79.17 ±1.29
		✓	58.57 ±0.60	67.79 ±0.93	58.79 ±0.47	59.36 ±0.21	39.76 ±2.75	71.83 ±0.80	68.02 ±1.37	58.67 ±4.59
Supervised	✓		67.58 ±1.08	77.01 ±0.41	64.55 ±0.23	61.59 ±0.81	56.98 ±3.59	77.13 ±2.77	75.63 ±0.56	77.01 ±0.70
		✓	63.81 ±0.17	75.58 ±0.09	63.32 ±0.11	59.83 ±0.06	50.79 ±0.64	79.01 ±0.59	70.93 ±0.60	75.60 ±0.45
Supervised-Infomax	✓		67.25 ±2.10	77.74 ±0.26	65.54 ±0.55	60.98 ±0.73	69.96 ±1.45	81.43 ±1.14	77.58 ±0.85	81.02 ±0.62
		✓	66.04 ±0.35	76.91 ±0.07	63.90 ±0.06	59.47 ±0.98	61.85 ±2.08	80.16 ±0.66	74.30 ±0.24	82.10 ±0.32
Supervised-EdgePred	✓		65.62 ±1.74	78.62 ±0.51	**66.19 ±0.31**	**63.69 ±0.66**	**73.08 ±1.57**	78.47 ±2.01	77.44 ±0.58	77.90 ±3.02
		✓	69.25 ±0.56	**79.66 ±0.12**	65.34 ±0.14	61.56 ±0.14	65.00 ±1.11	82.78 ±0.45	73.29 ±0.20	71.23 ±1.13
Supervised-AttrMasking	✓		66.09 ±1.06	77.61 ±0.46	65.31 ±0.22	63.12 ±0.73	75.55 ±5.44	82.38 ±1.42	77.26 ±0.90	80.46 ±0.99
		✓	65.18 ±0.23	78.56 ±0.10	64.73 ±0.14	62.49 ±0.23	69.19 ±0.66	79.10 ±0.44	77.34 ±0.13	79.54 ±0.34
Supervised-ContextPred	✓		69.31 ±1.40	78.37 ±0.39	65.53 ±0.22	62.92 ±0.44	72.78 ±2.25	82.89 ±1.24	**79.45 ±0.87**	**84.44 ±0.79**
		✓	68.35 ±0.37	77.93 ±0.06	65.50 ±0.13	61.71 ±0.15	69.59 ±0.22	**84.35 ±0.50**	73.25 ±0.34	78.96 ±3.42

Table 2: Test ROC-AUC (%) performance on molecular prediction benchmarks with different pre-trained GNN models and different tuning methods.

4.2 Main Results

Dataset			PPI
Pre-trained model	FT	GP	Species split
–	✓		65.86 ±1.89
Infomax	✓		63.28 ±0.38
		✓	62.89 ±0.96
EdgePred	✓		65.06 ±1.02
		✓	51.22 ±1.33
AttrMasking	✓		64.80 ±1.78
		✓	69.62 ±0.21
ContextPred	✓		66.01 ±1.38
		✓	67.06 ±0.61
Supervised	✓		69.17 ±2.36
		✓	76.44 ±0.13
Supervised-Infomax	✓		71.29 ±1.79
		✓	77.02 ±0.42
Supervised-EdgePred	✓		71.54 ±0.85
		✓	76.98 ±0.20
Supervised-AttrMasking	✓		73.93 ±1.17
		✓	**78.91 ±0.25**
Supervised-ContextPred	✓		72.10 ±1.94
		✓	77.42 ±0.07
GraphCL	✓		67.88 ±0.85
		✓	**68.22 ±0.11**
SimGRACE	✓		**70.67 ±0.31**
		✓	67.96 ±0.31

Table 3: Test ROC-AUC (%) performance on protein function prediction benchmarks with different pre-trained GNN models and different tuning methods.

Dataset			IMDB-B	IMDB-M
Pre-trained model	FT	GP		
–	✓		71.80 ±4.02	49.07 ±4.02
GCC (E2E)	✓		72.60 ±4.72	49.07 ±3.59
		✓	**73.40 ±3.80**	49.17 ±3.12
GCC (MoCo)	✓		71.70 ±4.98	48.07 ±2.91
		✓	72.50 ±3.20	**49.33 ±3.93**

Table 4: Test Acc (%) performance on graph classification benchmarks with GCC and different tuning methods.

Pre-trained model	FT	GP	Total params
Hu et al.	✓		$\sim$ 2M
		✓	$\sim$ 2K
GraphCL	✓		$\sim$ 2M
		✓	$\sim$ 2K
SimGRACE	✓		$\sim$ 2M
		✓	$\sim$ 2K
GCC	✓		$\sim$ 190K
		✓	$\sim$ 200

Table 5: The number of tunable parameters of pre-trained GNN models with different tuning methods.

We compare the performance of various pre-trained GNN models with ordinary fine-tuning (FT) and our proposed Graph Prompt (GP). Table 2,3 and 4 summarize the experimental results. GP here means using GPF to adapt the pre-trained GNN models to downstream tasks. For a pre-trained GNN model and all its variants, the best experimental results on a certain downstream task are **bolded** in the table. Meanwhile, the best results for the other tuning method are underlined (*e.g.*, if the overall best result of this downstream task comes from GP, then the best result of FT would be underlined for comparison). Due to the space limitation, more experimental results on molecular prediction benchmarks can be found in Appendix. Our systematic study suggests the following observations.

Observation (1): With a small amount of tunable parameters (about 0.1% of that for fine-tuning), GPF can achieve comparable performances with fine-tuning in most cases. According to Table 5, we can find that under the same pre-trained model, the amount of tunable parameters of GPF is just about 0.1% compared to that for fine-tuning. GPF achieves the optimal results on $2/3$ of the pre-trained GNN models in biology datasets (Table 3), and achieve the optimal results on all datasets with the pre-trained model GCC (Table 4). As for the chemistry datasets, GPF achieves the

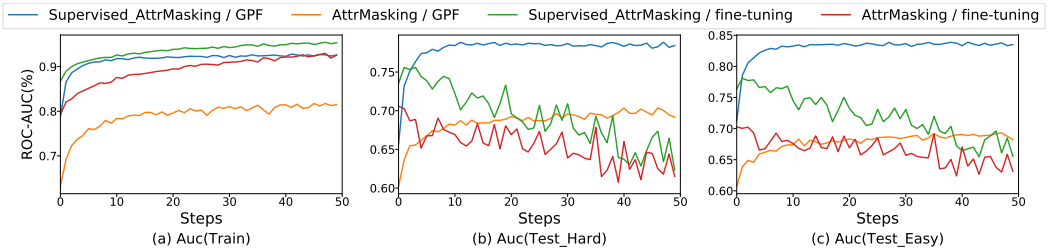


Figure 2: Training and test curves of different tuning methods.

optimal performance on $1/4$ of the downstream tasks. Under all the settings where GPF is inferior to fine-tuning, the optimal results on GPF are on average 2.08% lower than that of fine-tuning, which are comparable to the sub-optimal fine-tuning results.

Observation (2): GPF can achieve significant improvement compared to fine-tuning in some particular situations. In biology datasets, for the pre-trained models in Hu et al. (2020a) using supervised pre-training strategy, GPF outperforms fine-tuning about 5% on average. It is worth mentioning that this phenomenon rarely occurs in the NLP field, where prompt tuning matches, but cannot exceed fine-tuning. The overall results indicate that our proposed GPF outperforms fine-tuning in almost half of all experiments $(6/13)$.

Observation (3): Compared with fine-tuning, GPF is more stable when adapting the pre-trained GNN models on downstream tasks. For 75% $(65/87)$ of the pre-trained models and downstream tasks, GPF has smaller std values compared to fine-tuning. This result can be explained by the fact that GPF using a small number of task-specific parameters, which makes the adaptation more stable compared to full fine-tuning of the pre-trained model.

4.3 Additional Experiments

Training process analysis. We analyze the training process for different tuning methods on biology datasets (PPI) with two pre-trained GNN models pre-trained by two different strategies in Hu et al. (2020a): AttrMasking and Supervised AttrMasking. Figure 2 presents the training and test results during the adaptation stage. The test set is divided into Test Hard and Test Easy, which is based on whether the graph appears in the supervised pre-training process (more details can be found in Hu et al. (2020a)). From Figure 2 (a), we can find that the AUC scores of the training set are continually increasing during the adaptation stage for both GPF and fine-tuning. However, things are totally different on the test set. As for the fine-tuning method, the AUC scores on the test set fluctuate and decrease continuously after a brief improvement. When applying GPF to adapt pre-trained models, the AUC scores on the test set continue to grow and stay at a high value. These results indicate that fully fine-tuning a pre-trained GNN model on a downstream task may lose the generalization ability of the model, while employing GPF can alleviate this problem significantly.

Comparison with other tuning methods. Pre-trained GNN model is mostly tuned through fully updating the model with classification head parameters. We apply some other tuning methods, which are adopted extensively in other areas. The full experimental results can be found in Appendix A.7. The results indicate that GPF outperforms other tuning methods in most cases.

5 Conclusion

In this paper, we propose Graph Prompt, a prompt-based tuning method for adapting pre-trained GNN models on downstream tasks. GP introduces task-specific learnable graph-level transformations during the adaptation stage while keeping the pre-trained GNN models frozen. We also design a concrete implementation of GP, called GP-Feature, and demonstrate the effectiveness of GPF both theoretically and empirically. With only a small amount of tunable parameters, GPF achieves a satisfactory performance compared to fine-tuning. Our paper proves the feasibility of employing prompt tuning on pre-trained GNN models, and we hope our work will inspire future research on the design of graph prompts.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022.

Lars Backstrom, Daniel P. Huttenlocher, Jon M. Kleinberg, and Xiangyang Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD '06*, 2006.

Hyojin Bahng, Ali Jahanian, Swami Sankaranarayanan, and Phillip Isola. Exploring visual prompts for adapting large-scale models. 2022.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2021.

Xiang Gao, Wei Hu, and Zongming Guo. Exploring structure-adaptive graph learning for robust semi-supervised classification. *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, 2020.

Anna Gaulton, Louisa J. Bellis, A. Patrícia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, and John P. Overington. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic Acids Research*, 40:D1100 – D1107, 2012.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pp. 1024–1034, 2017.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Doll'ar, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *ArXiv*, abs/2111.06377, 2021.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *ICLR*, 2020a.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020b.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. *ECCV*, 2022.

Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhang Wang, Zitao Liu, and Jiliang Tang. Self-supervised learning on graphs: Deep insights and new direction. *ArXiv*, abs/2006.10141, 2020a.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020b.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Boris Knyazev, Graham W. Taylor, and Mohamed R. Amer. Understanding attention and generalization in graph neural networks. In *NeurIPS*, 2019.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *EMNLP*, 2021a.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021b.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *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)*, abs/2101.00190, 2021.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*, 2021a.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys (CSUR)*, 2022a.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021b.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *arXiv preprint arXiv:2103.10385*, 2021c.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *ACL*, 2022b.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9992–10002, 2021d.

Yuanfu Lu, Xunqiang Jiang, Yuan Fang, and Chuan Shi. Learning to pre-train graph neural networks. In *AAAI*, 2021.

Andreas Mayr, Günter Klambauer, Thomas Unterthiner, Marvin N. Steijaert, Jörg Kurt Wegner, Hugo Ceulemans, Djork-Arné Clevert, and Sepp Hochreiter. Large-scale comparison of machine learning methods for drug target prediction on chembl† †electronic supplementary information (esi) available: Overview, data collection and clustering, methods, results, appendix. see doi: 10.1039/c8sc00148k. *Chemical Science*, 9:5441 – 5451, 2018.

Diego Mesquita, Amauri H. de Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. *ArXiv*, abs/2010.11418, 2020.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *AAAI*, 2019.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

Scott C. Ritchie, Stephen C. Watts, Liam G. Fearnley, Kathryn E. Holt, Gad Abraham, and Michael Inouye. A scalable permutation approach reveals replication and preservation patterns of network modules in large datasets. *Cell systems*, 3 1:71–82, 2016.

Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. *Advances in Neural Information Processing Systems*, 33:12559–12571, 2020.

R. Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *SDM*, 2021.

Timo Schick and Hinrich Schütze. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*, 2020.

T. Sterling and John J. Irwin. Zinc 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55:2324 – 2337, 2015.

Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*, 2019.

Mingchen Sun, Kaixiong Zhou, Xingbo He, Ying Wang, and Xin Wang. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

Maria Tsimpoukelli, Jacob L Menick, Serkan Cabi, SM Eslami, Oriol Vinyals, and Felix Hill. Multimodal few-shot learning with frozen language models. *Advances in Neural Information Processing Systems*, 34:200–212, 2021.

Petar Velickovic, William Fedus, William L. Hamilton, Pietro Lio', Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. *ICLR*, 2019.

Hongya Wang, Haoteng Yin, Muhan Zhang, and Pan Li. Equivariant and stable positional encoding for more powerful graph neural networks. *ICLR*, 2022.

Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay S. Pande. Moleculenet: A benchmark for molecular machine learning. *arXiv: Learning*, 2017.

Jun Xia, Lirong Wu, Jintao Chen, Bozhen Hu, and Stan Z. Li. Simgrace: A simple framework for graph contrastive learning without data augmentation. *Proceedings of the ACM Web Conference 2022*, 2022a.

Jun Xia, Yanqiao Zhu, Yuanqi Du, and Stan Z Li. A survey of pretraining on graphs: Taxonomy, methods, and applications. *arXiv preprint arXiv:2202.07893*, 2022b.

Hui Xu, Liyao Xiang, Jiahao Yu, Anqi Cao, and Xinbing Wang. Speedup robust graph structure learning with low-rank information. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ICLR*, 2019.

Pinar Yanardag and S. V. N. Vishwanathan. Deep graph kernels. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42:181–213, 2012.

Gilad Yehudai, Ethan Fetaya, Eli A. Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *ICML*, 2021.

Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. *ICML*, 2019.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *NeurIPS*, 2020.

Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. In *International Conference on Machine Learning*, pp. 12121–12132. PMLR, 2021.

Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. In *ECCV*, 2016.

Z. Zhang, Q. Liu, H. Wang, C. Lu, and C. K. Lee. Motif-based graph self-supervised learning for molecular property prediction. 2021.

Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Yuanqi Du, Jieyu Zhang, Qiang Liu, Carl Yang, and Shu Wu. A survey on graph structure learning: Progress and opportunities. 2021.

Marinka Zitnik, Rok Sosi, and Jure Leskovec. Prioritizing network communities. *Nature Communications*, 9, 2018.

A APPENDIX

A.1 PROOF OF PROPOSITION 3

Proof. We set $\Delta\mathbf{X} = \mathbf{X}' - \mathbf{X}$, then we have:

$$\begin{aligned}\mathbf{H}' &= (\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\mathbf{X}'\cdot\mathbf{\Theta}^* \\ &= (\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot(\mathbf{X}+\Delta\mathbf{X})\cdot\mathbf{\Theta}^* \\ &= (\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\mathbf{X}\cdot\mathbf{\Theta}^* + (\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\Delta\mathbf{X}\cdot\mathbf{\Theta}^* \\ &= \mathbf{H} + (\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\Delta\mathbf{X}\cdot\mathbf{\Theta}^*\end{aligned}$$

As for GPF $p = [\alpha_1, \cdots, \alpha_F] \in \mathbb{R}^{1\times F}$, we can do the similar split:

$$\begin{aligned}\mathbf{H}_p &= (\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot(\mathbf{X} + [1]^N\cdot p)\cdot\mathbf{\Theta}^* \\ &= (\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\mathbf{X}\cdot\mathbf{\Theta}^* + (\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot[1]^N\cdot p\cdot\mathbf{\Theta}^* \\ &= \mathbf{H} + (\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot[1]^N\cdot p\cdot\mathbf{\Theta}^* \\ &= \mathbf{H} + [d_i + 1 + \epsilon^*]^N\cdot p\cdot\mathbf{\Theta}^*\end{aligned}$$

where $[1]^N \in \mathbb{R}^{N\times 1}$ denotes a column vector with N 1s, $[d_i + 1 + \epsilon^*]^N \in \mathbb{R}^{N\times 1}$ denotes a column vector with the value of i-th line is $d_i + 1 + \epsilon^*$ and d_i represents the degree number of v_i. To obtain the same graph representation $h_{\mathcal{G}}$, we have:

$$\begin{aligned}&h_{\mathcal{G},ft} = h_{\mathcal{G},p} \\ \rightarrow\ &\text{Sum}(\mathbf{H}') = \text{Sum}(\mathbf{H}_p)\end{aligned}$$

where $\text{Sum}(\mathbf{M}) = \sum_i m_i$ denotes the operation that calculates the sum vector for each row in the matrix. We can further simplify the above equation as:

$$\begin{aligned}&h_{\mathcal{G},ft} = h_{\mathcal{G},p} \\ \rightarrow\ &\text{Sum}(\mathbf{H}') = \text{Sum}(\mathbf{H}_p) \\ \rightarrow\ &\text{Sum}(\mathbf{H} + (\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\Delta\mathbf{X}\cdot\mathbf{\Theta}^*) = \text{Sum}(\mathbf{H} + [d_i + 1 + \epsilon^*]^N\cdot p\cdot\mathbf{\Theta}^*) \\ \rightarrow\ &\text{Sum}((\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\Delta\mathbf{X}\cdot\mathbf{\Theta}^*) = \text{Sum}([d_i + 1 + \epsilon^*]^N\cdot p\cdot\mathbf{\Theta}^*)\end{aligned}$$

where the results of $((\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\Delta\mathbf{X}) \in \mathbb{R}^{N\times F}$,and the frozen linear transformation $\mathbf{\Theta}^* \in \mathbb{R}^{F\times F'}$. We first calculate $\Delta h_{\mathcal{G},p} = \text{Sum}([d_i + 1 + \epsilon^*]^N\cdot p\cdot\mathbf{\Theta}^*) \in \mathbb{R}^{F'}$. We can obtain that:

$$\begin{aligned}\Delta h^i_{\mathcal{G},p} &= \sum_{j=1}^{F}\sum_{k=1}^{N}(d_k + 1 + \epsilon^*)\cdot\alpha_j\cdot\theta^*_{j,i} \\ &= \sum_{j=1}^{F}(D + N + N\cdot\epsilon^*)\cdot\alpha_j\cdot\theta^*_{j,i}\end{aligned}$$

where $h^i_{\mathcal{G},p}$ denotes the value of the i-th dimension in $h_{\mathcal{G},p}$, $D = \sum_{k=1}^{N} d_k$ denotes the total degree of all nodes in the whole graph, $\alpha_j, j \in [1, F]$ denotes the j-th learnable parameter in GPF p, and $\theta^*_{j,i}, j \in [1, F], i \in [1, F']$ denotes the frozen parameter in $\mathbf{\Theta}^*$. As for $\Delta h_{\mathcal{G},ft} = \text{Sum}((\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\Delta\mathbf{X}\cdot\mathbf{\Theta}^*)$, we assume $(\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\Delta\mathbf{X} = \mathbf{B} \in \mathbb{R}^{N\times F}$. Then we have:

$$\Delta h^i_{\mathcal{G},ft} = \sum_{j=1}^{F}(\sum_{k=1}^{N}\beta_{k,j})\cdot\theta^*_{j,i}$$

where $\beta_{k,j}, k \in [1, N], j \in [1, F]$ denotes the learnable parameter in $\mathbf{B}$. According to above analysis, to obtain a same graph representation $h_{\mathcal{G},p}$ with a certain $h_{\mathcal{G},ft^*}$, we have:

$$\begin{aligned}&h^i_{\mathcal{G},p} = h^i_{\mathcal{G},ft^*}\ ,\ \text{for every } i \in [1, F'] \\ \rightarrow\ &\Delta h^i_{\mathcal{G},p} = \Delta h^i_{\mathcal{G},ft^*} \\ \rightarrow\ &\alpha_j = \frac{\sum_{k=1}^{N}\beta^*_{k,j}}{D + N + N\cdot\epsilon^*}\ ,\ j \in [1, F]\end{aligned}$$

Therefore, for an arbitrary feature transformation g_{ft^*}, there exists a GPF p^* that satisfies above conditions and can obtain the same graph representation for pre-trained GNN model f^*. □

A.2 PROOF OF PROPOSITION 4

Proof. The proof of Proposition 4 is similar to that of Proposition 3. We set $\Delta\mathbf{A} = \mathbf{A}' - \mathbf{A}$. It is worth mentioning that $\mathbf{A}, \mathbf{A}' \in \{0,1\}^{N\times N}$ and $\Delta\mathbf{A} \in \{-1,0,1\}^{N\times N}$, which means they are of the same size, the values of $\mathbf{A}, \mathbf{A}'$ can only be 0 or 1, and the values of $\Delta\mathbf{A}$ can be -1, 0 or 1. We have:

$$\begin{aligned}\mathbf{H}' &= (\mathbf{A}' + (1+\epsilon^*)\cdot\mathbf{I})\cdot\mathbf{X}\cdot\mathbf{\Theta}^* \\ &= ((\mathbf{A}+\Delta\mathbf{A}) + (1+\epsilon^*)\cdot\mathbf{I})\cdot\mathbf{X}\cdot\mathbf{\Theta}^* \\ &= \mathbf{H} + \Delta\mathbf{A}\cdot\mathbf{X}\cdot\mathbf{\Theta}^*\end{aligned}$$

From the proof of Proposition 3, we obtain:

$$\mathbf{H}_p = \mathbf{H} + [d_i + 1 + \epsilon^*]^N \cdot p \cdot \mathbf{\Theta}^*$$

where $p = [\alpha_1, \cdots, \alpha_F] \in \mathbb{R}^{1\times F}$ denotes our learnable GPF, $[d_i + 1 + \epsilon^*]^N \in \mathbb{R}^{N\times 1}$ denotes a column vector with the value of i-th line is $d_i + 1 + \epsilon^*$ and d_i represents the degree number of v_i. With $\Delta h_{\mathcal{G},p} = \text{Sum}([d_i + 1 + \epsilon^*]^N \cdot p \cdot \mathbf{\Theta}^*) \in \mathbb{R}^{F'}$, we can obtain:

$$\begin{aligned}\Delta h^i_{\mathcal{G},p} &= \sum_{j=1}^{F}\sum_{k=1}^{N}(d_k + 1 + \epsilon^*)\cdot\alpha_j\cdot\theta^*_{j,i} \\ &= \sum_{j=1}^{F}(D + N + N\cdot\epsilon^*)\cdot\alpha_j\cdot\theta^*_{j,i}\end{aligned}$$

where $h^i_{\mathcal{G},p}$ denotes the value of the i-th dimension in $h_{\mathcal{G},p}$, $D = \sum_{k=1}^{N} d_k$ denotes the total degree of all nodes in the whole graph, $\alpha_j, j \in [1, F]$ denotes the j-th learnable parameter in GPF p, and $\theta^*_{j,i}, j \in [1, F], i \in [1, F']$ denotes the frozen parameter in $\mathbf{\Theta}^*$. As for $\Delta h_{\mathcal{G},lt} = \text{Sum}(\Delta\mathbf{A}\cdot\mathbf{X}\cdot\mathbf{\Theta}^*)$, we have:

$$\Delta h^i_{\mathcal{G},lt} = \sum_{j=1}^{F}\sum_{(k,l)\in N\times N}(\Delta a_{k,l}\cdot x_{l,j})\cdot\theta^*_{j,i}$$

where $\Delta a_{k,l}, k \in [1, N], l \in [1, N]$ denotes the element of $\Delta\mathbf{A}$, and $x_{l,j}, l \in [1, N], j \in [1, F]$ denotes the element of $\mathbf{X}$. To obtain a same graph representation $h_{\mathcal{G},p}$ with a certain $h_{\mathcal{G},lt^*}$, we have:

$$\begin{aligned}& h^i_{\mathcal{G},p} = h^i_{\mathcal{G},lt^*}\ , \text{ for every } i \in [1, F'] \\ \rightarrow\ & \Delta h^i_{\mathcal{G},p} = \Delta h^i_{\mathcal{G},lt^*} \\ \rightarrow\ & \alpha_j = \frac{\sum_{(k,l)\in N\times N}\Delta a^*_{k,l}\cdot x_{l,j}}{D + N + N\cdot\epsilon^*}\ ,\ j \in [1, F]\end{aligned}$$

Therefore, for an arbitrary link transformation g_{lt^*}, there exists a GPF p^* that satisfies above conditions and can obtain the same graph representation for pre-trained GNN model f^*. □

A.3 PROOF OF PROPOSITION 5

Proof. Unlike feature transformations and linear transformations, isolated component transformations will change the number of nodes in the graph, which means the scale of modified $\mathbf{A}'$ and $\mathbf{X}'$ is uncertain. We first express the isolated component transformation in more details. The adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$ can be divided into several isolated components, which can be expressed as:

$$\mathbf{A} = \begin{pmatrix}\mathbf{A}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{A}_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_m\end{pmatrix} \qquad \mathbf{X} = \begin{pmatrix}\mathbf{X}_1 \\ \mathbf{X}_2 \\ \vdots \\ \mathbf{X}_m\end{pmatrix}$$

Removing an isolated component $\mathcal{C}_k = \{\mathbf{A}_k, \mathbf{X}_k\}, k \in [1, m]$ means removing both $\mathbf{A}_k$ in the adjacency matrix and corresponding $\mathbf{X}_k$ in the feature matrix. Adding a new isolated component

$\mathcal{C}_{m+l} = \{\mathbf{A}_{m+l}, \mathbf{X}_{m+l}\}, l \geq 1$ means adding $\mathbf{A}_{m+l}$ to the adjacency matrix $\mathbf{A}$, and adding $\mathbf{X}_{m+l}$ to the corresponding position of $\mathbf{X}$. Then we have:

$$h_{\mathcal{G},ist} = \sum_k \text{Sum}((\mathbf{A}_k + (1+\epsilon^*)\cdot\mathbf{I})\cdot\mathbf{X}_k\cdot\mathbf{\Theta}^*)$$

To align with the proofs of Proposition 3 and Proposition 4, we set $\Delta h_{\mathcal{G},ist} = h_{\mathcal{G},ist} - \text{Sum}((\mathbf{A} + (1+\epsilon^*)\cdot\mathbf{I})\cdot\mathbf{X}\cdot\mathbf{\Theta}^*)$, and it can be expressed as:

$$\Delta h_{\mathcal{G},ist} = \sum_k I_k \cdot \text{Sum}((\mathbf{A}_k + (1+\epsilon^*)\cdot\mathbf{I})\cdot\mathbf{X}_k\cdot\mathbf{\Theta}^*)$$

where I_k is an indicator that satisfies:

$$I_k = \begin{cases} 0 & \text{if } \mathcal{C}_k \text{ has no change} \\ 1 & \text{if } \mathcal{C}_k \text{ is an additional component} \\ -1 & \text{if } \mathcal{C}_k \text{ is a removed component} \end{cases}$$

From the proof of Proposition 3, we have following conclusions:

$$\mathbf{H}_p = \mathbf{H} + [d_i + 1 + \epsilon^*]^N \cdot p \cdot \mathbf{\Theta}^*$$

where $p = [\alpha_1, \cdots, \alpha_F] \in \mathbb{R}^{1\times F}$ denotes our learnable GPF, $[d_i + 1 + \epsilon^*]^N \in \mathbb{R}^{N\times 1}$ denotes a column vector with the value of i-th line is $d_i + 1 + \epsilon^*$ and d_i represents the degree number of v_i. With $\Delta h_{\mathcal{G},p} = \text{Sum}([d_i + 1 + \epsilon^*]^N \cdot p \cdot \mathbf{\Theta}^*) \in \mathbb{R}^{F'}$, we can obtain:

$$\begin{aligned} \Delta h^i_{\mathcal{G},p} &= \sum_{j=1}^{F}\sum_{k=1}^{N}(d_k + 1 + \epsilon^*)\cdot\alpha_j\cdot\theta^*_{j,i} \\ &= \sum_{j=1}^{F}(D + N + N\cdot\epsilon^*)\cdot\alpha_j\cdot\theta^*_{j,i} \end{aligned}$$

where $h^i_{\mathcal{G},p}$ denotes the value of the i-th dimension in $h_{\mathcal{G},p}$, $D = \sum_{k=1}^{N} d_k$ denotes the total degree of all nodes in the whole graph, $\alpha_j, j \in [1, F]$ denotes the j-th learnable parameter in GPF p, and $\theta^*_{j,i}, j \in [1, F], i \in [1, F']$ denotes the frozen parameter in $\mathbf{\Theta}^*$. To obtain a same graph representation $h_{\mathcal{G},p}$ with a certain $h_{\mathcal{G},ist^*}$, we have:

$$\begin{aligned} & h^i_{\mathcal{G},p} = h^i_{\mathcal{G},ist^*} \text{ , for every } i \in [1, F'] \\ \rightarrow\ & \Delta h^i_{\mathcal{G},p} = \Delta h^i_{\mathcal{G},ist^*} \\ \rightarrow\ & \alpha_j = \frac{\sum_k I_k \cdot \text{Sum}((\mathbf{A}_k + (1+\epsilon^*)\cdot\mathbf{I})\cdot\mathbf{X}^j_k)}{D + N + N\cdot\epsilon^*} \text{ , } j \in [1, F] \end{aligned}$$

where $\mathbf{X}^j$ denotes the j-th column of the matrix $\mathbf{X}$. Therefore, for an arbitrary isolated component transformation g_{ict^*}, there exists a GPF p^* that satisfies above conditions and can obtain the same graph representation for pre-trained GNN model f^*. □

A.4 Proof of Proposition 6

Proof. Without loss of generality, we consider $\mathbf{g} = \{g_1, g_2\}$ with two transformations described in Proposition 2. Now we prove there exists a GPF $p*$ that satisfies:

$$f^*(\mathbf{A}, \mathbf{X} + p^*) = f^*(g_2(g_1(\mathbf{A}, \mathbf{X})))$$

We assume $g_1(\mathbf{A}, \mathbf{X}) = (\mathbf{A}', \mathbf{X}')$. According to Proposition 3, 4, 5, there exists a GPF p^*_1 that satisfies:

$$f^*(\mathbf{A}, \mathbf{X} + p^*_1) = f^*(g_1(\mathbf{A}, \mathbf{X}))$$

and a p^*_2 that satisfies:

$$f^*(\mathbf{A}, \mathbf{X} + p^*_2) = f^*(g_2(\mathbf{A}', \mathbf{X}'))$$

Therefore, there is a $p^* = p^*_1 + p^*_2$ that satisfies:

$$f^*(\mathbf{A}, \mathbf{X} + p^*) = f^*(g_2(g_1(\mathbf{A}, \mathbf{X})))$$

□

A.5 MORE DETAILS ABOUT THE EXPERIMENTS

Implementation details. We conduct graph classification tasks for various datasets and pre-trained models. To obtain the graph classification result, we first input the downstream graph to the pre-trained GNN model and get fixed-dimensional graph representations. Then, we use a linear transformation layer to transform the graph representations to classification results. When applying fine-tuning to adapt the pre-trained model to downstream tasks, we update all parameters of the pre-trained model and the final linear transformation layer. As for our proposed GPF, we freeze the parameters of the pre-trained model during adaption, but update the parameters of GPF and the final linear transformation layer. We strictly follow the training steps and the hyper-parameter settings presented in the papers of these pre-trained models (Hu et al., 2020a; You et al., 2020; Xia et al., 2022a; Qiu et al., 2020). For each pre-trained model and a specific downstream dataset, we conduct 5 rounds of experiments with different random seeds, and present the average result.

Hyper-parameter setting. This part presents the hyper-parameters involved during the adaptation stage of pre-trained GNN models on downstream tasks. Table 6 summarizes the hyper-parameter settings for fine-tuning, and Table 7 summarizes the hyper-parameter settings for GPF.

Downstream dataset	Model	Learning rate	Weight decay	Batch size	Training epoch
Biology	Hu et al.	0.001	0	32	50
	GraphCL	0.0001	0	32	50
	SimGrace	0.001	0	32	10
Chemistry	Hu et al.	0.001	0	32	100
	GraphCL	0.001	0	32	100
	SimGrace	0.001	0	32	100
IMDB-B	GCC	0.005	Linear	128	100
IMDB-M	GCC	0.005	Linear	128	100

Table 6: The hyper-parameter settings for fine-tuning.

Downstream dataset	Model	Prompt dimension	Learning rate	Weight decay	Batch size	Training epoch
Biology	Hu et al.	300	0.001	0	32	50
	GraphCL	300	0.0001	0	32	50
	SimGrace	300	0.0001	0	32	20
Chemistry	Hu et al.	300	0.001	0	32	100
	GraphCL	300	0.001	0	32	100
	SimGrace	300	0.001	0	32	100
IMDB-B	GCC	64	0.005	Linear	128	100
IMDB-M	GCC	64	0.005	Linear	128	100

Table 7: The hyper-parameter settings for GPF.

Details of pre-training datasets. The datasets published by Hu et al. (2020a) include two datasets named Biology and Chemistry for the biology domain and chemistry domain respectively. Biology contains 395K unlabeled protein ego-networks derived from PPI networks of 50 species for node-level self-supervised pre-training, and 88K labeled protein ego-networks to jointly predict 5000 *coarse-grained* biological functions for graph-level multi-task supervised pre-training. In Chemistry, 2 million unlabeled molecules are sampled from the ZINC15 database (Sterling & Irwin, 2015) for node-level self-supervised pre-training. A preprocessed ChEMBL dataset (Mayr et al., 2018; Gaulton et al., 2012) is utilized for graph-level multi-task supervised pre-training, which contains 456K molecules with 1310 kinds of biochemical assays. For datasets used to pre-train GCC, the self-supervised pre-training is performed on six graph datasets. Table 8 presents the detailed statistics of them.

Dataset	**Academia**	**DBLP(SNALP)**	**DBLP(NetRep)**	**IMDB**	**Facebook**	**LiveJournal**
$\|V\|$	137,969	317,080	540,486	896,305	3,097,165	4,843,953
$\|E\|$	739,984	2,099,732	30,491,158	7,564,894	47,334,788	85,691,368

Table 8: Statistics of datasets for pre-training

Details of downstream datasets. Detailed statistics of downstream datasets used for the models pre-trained on Biology and Chemistry are summarized in Table 9. For the downstream tasks of pre-trained GCC, we use IMDB-BINARY and IMDB-MULTI (Yanardag & Vishwanathan, 2015). Each dataset is a set of graphs where each graph is associated with a label. To evaluate GCC on this task, we use raw input graphs as the input of GCC.

Dataset	**PPI**	**BBBP**	**Tox21**	**ToxCast**	**SIDER**	**ClinTox**	**MUV**	**HIV**	**BACE**
# Proteins / Molecules	88K	2039	7831	8575	1427	1478	93087	41127	1513
# Binary prediction tasks	40	1	12	617	27	2	17	1	1

Table 9: Statistics of datasets for downstream tasks.

A.6 Additional results of main experiments.

Dataset			BBBP	Tox21	ToxCast	SIDER	ClinTox	MUV	HIV	BACE
Pre-trained model	FT	GP	Out-of-distribution prediction (scaffold split)							
GraphCL	✓		**69.49 ±0.35**	**73.35 ±0.70**	**62.54 ±0.26**	**60.63 ±1.26**	**75.17 ±2.14**	69.78 ±1.44	**78.26 ±0.73**	**75.51 ±2.01**
		✓	63.71 ±0.39	65.53 ±0.48	58.48 ±0.48	55.34 ±0.56	64.47 ±0.56	**70.10 ±0.68**	70.32 ±0.53	71.53 ±0.64

Table 10: Test ROC-AUC (%) performance on molecular prediction benchmarks with GraphCL and different tuning methods.

Table 10 presents the results of GraphCL on molecular prediction benchmarks with different tuning methods.

A.7 Results of comparison with other tuning methods.

We compare GPF with other tuning methods described as follows:

- PARTIAL-k: We finetune the last k layers of the model with the classfication head and freeze other parts, which is utilized in Zhang et al. (2016); He et al. (2021); Jia et al. (2022).
- MLP-k: We utilize a multilayer perceptron (MLP) with k layers as the classification head, instead of a single linear layer.

We conduct the experiments on the biology datasets (PPI) with pre-trained GNN models in Hu et al. (2020a), and Table 11 summaries the results. The experimental results indicate that GPF outperforms other tuning methods in most cases ($6/9$).

Pre-trained model	FT	MLP-3	Partial-1	Partial-3	GP (ours)
Infomax	63.28 ±0.38	67.51 ±0.73	**68.95 ±0.71**	64.69 ±0.46	62.89 ±0.96
EdgePred	65.06 ±1.02	65.71 ±0.41	**70.54 ±1.56**	66.53 ±0.28	51.22 ±1.33
AttrMasking	64.80 ±1.78	65.21 ±0.75	65.69 ±0.24	63.19 ±0.48	**69.62 ±0.21**
ContextPred	66.01 ±1.38	69.68 ±0.84	**67.27 ±0.54**	66.98 ±1.43	67.06 ±0.61
Supervised	69.17 ±2.36	74.02 ±0.37	71.93 ±0.99	71.11 ±2.22	**76.44 ±0.13**
Supervised-Infomax	71.29 ±1.79	74.68 ±0.56	74.36 ±0.92	73.28 ±0.18	**77.02 ±0.42**
Supervised-EdgePred	71.54 ±0.85	74.60 ±0.88	73.24 ±0.68	73.35 ±0.77	**76.98 ±0.20**
Supervised-AttrMasking	73.93 ±1.17	77.99 ±0.42	75.91 ±0.10	74.02 ±0.37	**78.91 ±0.25**
Supervised-ContextPred	72.10 ±1.94	76.01 ±0.68	76.62 ±0.92	74.86 ±0.79	**77.42 ±0.07**

Table 11: Test ROC-AUC (%) performance on protein function prediction benchmarks with different tuning methods.

A.8 Details of pre-trained GNN models.

In our experiments, we employ four different pre-trained GNN models, adopting GIN (Xu et al., 2019) as the backbone and average pooling for the READOUT function. These models are elaborated as follows:

- Hu et al. (2020a) Hu et al. (2020a) propose a new strategy and self-supervised methods for pre-training GNNs. It includes ContextPred (Context Prediction), Attribute Masking (AttrMasking) and Supervised Graph-Level Property Prediction (Supervised). They also use

Deep Graph Infomax (Infomax) (Velickovic et al., 2019) and Edge Prediction (EdgePred) (Hamilton et al., 2017) as pre-training baselines. Following their experiment settings, we also test different combinations of these methods.

- GraphCL You et al. (2020) propose a graph contrastive learning framework for learning unsupervised representations of graph data. They design four types of graph augmentations to incorporate various priors.
- SimGRACE Xia et al. (2022a) propose a simple framework for graph contrastive learning (SimGRACE), which doesn't require data augmentations. They take the original graph as input and GNN model with its perturbed version as two encoders to obtain two correlated views for contrast.
- GCC To capture the universal network topological properties across multiple networks, Qiu et al. (2020) propose a self-supervised graph neural network pre-training framework, which is named Graph Contrastive Coding (GCC).