Search to Fine-tune Pre-trained Graph Neural Networks for Graph-level Tasks

Zhili WANG[1], Shimin DI[1†], Lei CHEN[1,2], Xiaofang ZHOU[1]
[1]The Hong Kong University of Science and Technology, Hong Kong SAR, China
[2]The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China
zwangeo@connect.ust.hk, sdiaa@connect.ust.hk, leichen@hkust-gz.edu.cn, zxf@cse.ust.hk

2024 IEEE 40th International Conference on Data Engineering (ICDE) | 979-8-3503-1715-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDE60146.2024.00219

***Abstract*—Recently, graph neural networks (GNNs) have shown its unprecedented success in many graph-related tasks. However, GNNs face the label scarcity issue as other neural networks do. Thus, recent efforts try to pre-train GNNs on a large-scale unlabeled graph and adapt the knowledge from the unlabeled graph to the target downstream task. The adaptation is generally achieved by fine-tuning the pre-trained GNNs with a limited number of labeled data. However, current GNNs pre-training works focus more on how to better pre-train a GNN, but ignore the importance of fine-tuning to better leverage the transferred knowledge. Only a few works start to investigate a better fine-tuning strategy for pre-trained GNNs. But their designs either have strong assumptions or overlook the data-aware issue behind various downstream domains. To further boost pre-trained GNNs, we propose to search to fine-tune pre-trained GNNs for graph-level tasks (S2PGNN), which can adaptively design a suitable fine-tuning framework for the given pre-trained GNN and downstream data. Unfortunately, it is a non-trivial task to achieve this goal due to two technical challenges. First is the hardness of fine-tuning space design since there lack a systematic and unified exploration in existing literature. Second is the enormous computational overhead required for discovering suitable fine-tuning strategies from the discrete space. To tackle these challenges, S2PGNN first carefully summarizes a search space of fine-tuning strategies that is suitable for GNNs, which is expressive enough to enable powerful strategies to be searched. Then, S2PGNN integrates an efficient search algorithm to solve the computationally expensive search problem from a discrete and large space. The empirical studies show that S2PGNN can be implemented on the top of 10 famous pre-trained GNNs and consistently improve their performance by 9% to 17%. Our code is publicly available at https://github.com/zwangeo/icde2024.**

***Index Terms*—graph neural network, fine-tuning, pre-training graph neural networks**

I. Introduction

As one of the most ubiquitous data structures, graph is a powerful way to represent diverse and complex real-world systems, e.g., social networks [1], knowledge graphs [2]–[4], protein interactions [5], and molecules [6]–[10]. Graph representation learning [11] maps the original graph into the low-dimensional vector space to handle various graph scenarios. Recently, Graph Neural Networks (GNNs) [12]–[19], which follow the message-passing schema [20] to learn representations via iteratively neighboring message aggregation, have become the leading approaches towards powerful graph representation learning. GNNs have demonstrated state-of-the-art results in a variety of graph tasks. e.g., node classification [12]–[14], [21], [22], link prediction [17], [21], [23]–[26], and graph classification [6]–[8], [15], [20]–[22], [27], [28].

Despite the revolutionary success of GNNs on graph data, they are mainly trained in an end-to-end manner with task-specific supervision, which generally requires abundant labeled data. However, high-quality and task-specific labels can be scarce, which seriously impedes the application of GNNs on various graph data [29]. Especially, some scientific fields require extensive and laborious expert knowledge for adequate annotation, e.g., medicine, chemistry, and biology. Therefore, recent efforts [29]–[36] investigate pre-training [37] in GNNs so as to tackle this challenge and improve the generalization performances of GNNs. Pre-trained GNNs have demonstrated superiority and improved generalization performances on the downstream graph-level tasks, e.g., molecular property prediction [29]. They mainly follow the self-supervised way to pre-train GNNs on large-scale unlabeled graph data by exploiting various self-supervised learning (SSL) [38] strategies, such as Autoregressive Modeling (AM) [32], [39], Masked Component Modeling (MCM) [29], [40], and Contrastive Learning (CL) [30], [31], [34], [41]. Then, due to domain discrepancy [42], the *fine-tuning* strategy [43], [44] is proposed to transfer knowledge from pre-trained GNNs to the downstream domain by training the model with a limited number of labeled data.

Compared with the various pre-training mechanisms, the fine-tuning strategy on pre-trained GNNs has received little attention. The most common strategy is still the vanilla fine-tuning method [45], where the downstream GNN will be initialized by the parameters of a pre-trained GNN and trained on the labeled data of the targeted domain. However, the vanilla strategy may suffer from the issues of overfitting and poor generalization [46], [47]. Few recent efforts [42], [46], [48] dedicate to designing novel GNN fine-tuning strategies to mitigate potential issues of the vanilla solution. Unfortunately, their designs either have strong assumptions or overlook the data-aware issue behind the various downstream datasets. Firstly, the strategy [42] needs the pre-training data and task as prerequisites for the downstream domain, which unfortunately, are often inaccessible. Others [46], [48] assume the high relevance between the pre-trained domain with the downstream one, then enforce the similarity of model parameters or learned representations, which are incapable of handling out-of-distribution predictions. For example, molecular property prediction [29] often encompass novel substructures that have

†Corresponding Author

2375-026X/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDE60146.2024.00219

not been encountered during pre-training, a.k.a., *scaffold* [49]. Secondly, existing methods follow a fixed strategy to design fine-tuning methods, and their operators and structures remain unchanged when facing different data (i.e., not data-aware). However, complex and diverse downstream graph data may require data-specific strategy to better conduct generalization. For example, the GNN layers required for different graph data may be highly data-specific [18], [50] in terms of density and topology. Thus, how to selectively and comprehensively fuse the multi-scale information from different layers of pre-trained GNNs may be a crutial operator towards more data-aware and effective downstream generalization.

To fully unleash the potential of pre-trained GNNs on various downstream datasets, we propose a novel idea to search a suitable fine-tuning strategy for the given pre-trained GNN and downstream graph dataset. However, it is a non-trivial task to achieve this goal due to two technical challenges. Firstly, fine-tuning pre-trained GNNs is still a under-investigated problem and there lack a systematic and unified design space in existing literature. Thus, it is unclear how to design a powerful space for GNN fine-tuning. Secondly, the entire space equals to the Cartesian product of the fine-tuning strategy space and GNN model parameter space. The computational overhead of discovering suitable fine-tuning strategies from such a large space is enormous. To address these challenges, we present a novel framework in this paper, named *S*earch to fine-tune *P*re-trained *G*raph *N*eural *N*etworks for graph-level tasks (S2PGNN). More concretely, we investigate existing literature within and outside the area of GNNs, and systematically summarize a search space of fine-tuning frameworks that is suitable for pre-trained GNNs. The proposed space includes multiple influential design dimensions and enables powerful fine-tuning strategies to be searched. To reduce the search cost from the large and discrete space, we incorporate an efficient search algorithm, which suggests the parameter-sharing and continuous relaxation on the discrete space and solves the search problem by differentiable optimization. In summary, the main contributions of this work are listed as follows:

- In this paper, we systematically exploring GNN fine-tuning strategies, an important yet seriously under-investigated problem, to improve the utilization of pre-trained GNNs. We investigate fine-tuning within and outside GNN area and provide a new perspective for GNN fine-tuning.
- To further improve pre-trained GNNs, we propose S2PGNN that automatically search a suitable fine-tuning strategy for the given pre-trained GNN and downstream graph dataset, which broadens the perspective of GNN fine-tuning works. To the best of our knowledge, we are the first to develop automated fine-tuning search framework for pre-trained GNNs.
- We propose a novel search space of fine-tuning strategies in S2PGNN, which identifies key factors that affect GNN fine-tuning results and presents improved strategies. The S2PGNN framework is model-agnostic and can be plugged into existing pre-trained GNNs for better performance.
- The empirical studies demonstrate that S2PGNN can be implemented on the top of 10 famous pre-trained GNNs and consistently improve their performance. Besides, S2PGNN achieves better performance than existing fine-tuning strategies within and outside the GNN area.

TABLE I: A summary of common notations.

Notation	Definition
$\mathbb{R}^d$	The d-dimension real space.
$G=(V,E,\mathbf{A})$	An attributed graph with node-set V, edge-set E, and adjacency matrix $\mathbf{A}$.
V, E	$V=\{v_1,\ldots,v_n\}$, $E=\{(v_i,v_j)\|v_i,v_j\in V\}$.
$\mathbf{X}_v, \mathbf{X}_{uv}\in\mathbb{R}^d$	The node and edge attribute vector.
k, K	The current and maximum GNN layer index, $1\le k\le K$.
$\mathbf{H}_v, \mathbf{H}_G\in\mathbb{R}^d$	The node and graph representation vector.
$f_{\boldsymbol{\psi},\boldsymbol{\theta}}(\cdot)$	The GNN encoder with architectures $\boldsymbol{\psi}$ and parameters $\boldsymbol{\theta}$.
$p_{\boldsymbol{\alpha}}(\cdot)$	The GNN fine-tuning controller with parameters $\boldsymbol{\alpha}$.
$\boldsymbol{\Phi}_{ft}$	The GNN fine-tuning strategy.
$f_{dim}(\cdot)$	The GNN fine-tuning dimension for $dim\in\{conv,id,fuse,read\}$
$\mathcal{O}_{dim}$	The candidate-set of $f_{dim}(\cdot)$
$\mathcal{D}_{ssl}$, $\mathcal{D}_{ft}$	The pre-training and downstream dataset.
$\mathcal{L}_{ssl}(\cdot)$, $\mathcal{L}_{ft}(\cdot)$	The pre-training and downstream loss.

II. Related Work

A graph typically can be represented as $G=(V,E,\mathbf{A})$, where $V=\{v_1,\ldots,v_n\}$ is the node-set, $E=\{(v_i,v_j)|v_i,v_j\in V\}$ is the edge-set, and $\mathbf{A}\in\{0,1\}^{|V|\times|V|}$ is the adjacency matrix to define graph topology where $\mathbf{A}_{ij}=1$ iff $(v_i,v_j)\in E$. Each node v and edge (u,v) may be further equipped with attributes $\mathbf{X}_v\in\mathbb{R}^d$ and $\mathbf{X}_{uv}\in\mathbb{R}^d$.

In general, the learning on graph data first requires a graph encoder $f_{\boldsymbol{\psi},\boldsymbol{\theta}}(\cdot)$ with architectures $\boldsymbol{\psi}$ and parameters $\boldsymbol{\theta}$ to map the original graph G into the d-dimensional vector space: $f_{\boldsymbol{\psi},\boldsymbol{\theta}}: G\rightarrow\{\mathbf{H}\}$, where $\mathbf{H}\in\mathbb{R}^d$ can be the learned representation of single node/edge or the entire graph, depending on the prediction level of downstream tasks. Then, $\mathbf{H}$ can be fed into an additional prediction head (e.g., linear classifier) to predict the true label. Note that in the paper we discard the notation of prediction head for brievity. After that, the entire model can be trained in an end-to-end manner supervised by task-specific labels via the labeld training dataset $\mathcal{D}^{(tra)}$:

$$\boldsymbol{\theta}^*=\arg\min_{\boldsymbol{\theta}}\mathcal{L}_{sup}(f_{\boldsymbol{\psi},\boldsymbol{\theta}}(\cdot);\mathcal{D}^{(tra)}), \tag{1}$$

where ψ can be GCN,GIN and other architectures as introduced in Sec. II-A1 and Sec. II-A2. $\mathcal{L}_{sup}(\cdot)$ is the supervised loss function (e.g., cross entropy). The construction of $\mathcal{D}^{(tra)}$ also depends on the downstream task, e.g., $\mathcal{D}^{(tra)}=\{(G,y)\}$ for the graph classification.

A. Graph Neural Networks (GNNs)

Recent years have witnessed the unprecedented success of GNNs for modeling graph data and dealing with various graph tasks. As one of powerful graph encoders $f_{\boldsymbol{\theta}}(\cdot)$, GNNs rely on graph topology and node/edge features to achieve the representation learning. The majority of GNNs [12]–[15] follow

the message-passing paradigm [20] to learn representation $\mathbf{H}_v$ for given node v by iteratively aggregating messages from its neighbors $N(v)$. Formally, the intra-layer message passing for v can be formulated as:

$$\mathbf{M}_v \leftarrow AGG(\{(\mathbf{H}_u, \mathbf{H}_v, \mathbf{X}_{uv}) | u \in N(v)\}), \quad (2)$$

$$\mathbf{H}_v \leftarrow COMB(\mathbf{H}_v, \mathbf{M}_v), \quad (3)$$

where $AGG(\cdot)$ function aggregates neighboring messages to produce intermediate embedding $\mathbf{M}_v$, $COMB(\cdot)$ function combines information from neighbors and center node itself to update the representation $\mathbf{H}_v$. The aggregation process iterates for k times such that each node captures up to k-hop information in its learned representation $\mathbf{H}_v$. Furthermore, for graph-level tasks, a permutation-invariant readout function $READOUT(\cdot)$ is further required to obtain graph-level representation $\mathbf{H}_G$ of the entire graph G:

$$\mathbf{H}_G = READOUT(\{\mathbf{H}_v | v \in V\}). \quad (4)$$

1) Manual GNNs: The majority of GNNs are specific instantiations of Eq. (2), (3), and (4) and are designed manually. They mainly differ in several key functions, e.g., $N(v)$, $AGG(\cdot)$, $COMB(\cdot)$ and $READOUT(\cdot)$. We next present several classic GNNs which are adopted in later experiments.

- Graph Convolutional Network (GCN) [12] adopts mean aggregation function $MEAN(\cdot)$ and non-linear activation $\sigma(\cdot)$, e.g., $ReLU(\cdot)$. It proposes to transform intermediate embedding into representation by trainable matrix $\mathbf{W}$:

$$\mathbf{M}_v \leftarrow MEAN(\{\mathbf{H}_u | u \in N(v) \cup \{v\}\}),\ \mathbf{H}_v \leftarrow \sigma(\mathbf{W}\mathbf{M}_v).$$

- GraphSAGE (SAGE) [13] concatenates the intermediate embedding with the representation from previous iteration before the transformation:

$$\mathbf{M}_v \leftarrow MEAN(\{\mathbf{H}_u | u \in N(v)\}),\ \mathbf{H}_v \leftarrow \sigma(\mathbf{W}[\mathbf{H}_v || \mathbf{M}_v]).$$

- Graph Isomorphism Network (GIN) [15], as one of the most expressive GNN architectures, adopts sum aggregation function $SUM(\cdot)$ and multi-layer perceptron $MLP(\cdot)$ to transform the combined messages. The scalar ϵ is to balance the weights of messages from center node and its neighbors:

$$\mathbf{M}_v \leftarrow SUM(\{\mathbf{H}_u | u \in N(v)\}),\ \mathbf{H}_v \leftarrow MLP((1+\epsilon)\mathbf{H}_v + \mathbf{M}_v)$$

- Graph Attention Network (GAT) [14] introduces the attentive function $ATT(\cdot)$ [51] as its aggregation function:

$$\mathbf{M}_v \leftarrow ATT(\{\mathbf{H}_u | u \in N(v)\}),\ \mathbf{H}_v \leftarrow \sigma(\mathbf{W}\mathbf{M}_v).$$

For the graph-level $READOUT(\cdot)$ function, it can be simple non-parameterized function, e.g., sum pooling and mean pooling, or other more advanced methods [52]–[54].

2) Automated GNNs: To alleviate the extensive human labor in effective GNN architecture designs, recent efforts seek to automate this process and propose AutoGNNs [3], [21], [22], [55]–[57]. In general, AutoGNNs first design a unified GNN search space $\mathcal{O}$ to cover key design functions in Eq. (2), (3), and (4) (e.g., $AGG(\cdot)$, $COMB(\cdot)$) and promising candidates. They then adopt various search algorithms (e.g., differentiable methods [58]) to allow powerful GNNs to be discovered from the search space:

$$\boldsymbol{\psi}^*, \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\psi} \in \mathcal{O}, \boldsymbol{\theta}} \mathcal{L}_{sup}(f_{\boldsymbol{\psi}, \boldsymbol{\theta}}(\cdot); \mathcal{D}^{tra}). \quad (5)$$

Based on Eq. (5), novel GNNs identified by AutoGNNs have demonstrated superior results than their manually-designed counterparts on many graph scenarios [21], [55]–[57].

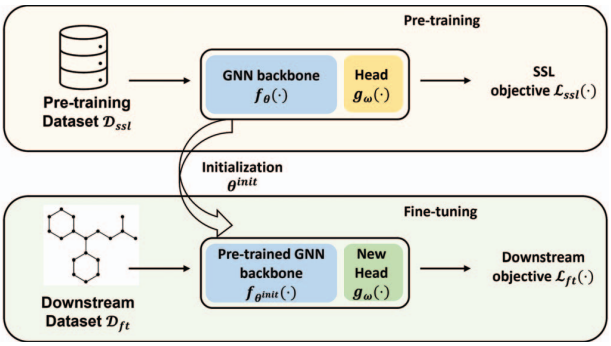


Fig. 1: The illustration of overall GNN pre-training and fine-tuning framework.

B. Fine-tuning Pre-trained GNNs

To relieve the reliance of GNNs on task-specific labels and improve their performances on scarcely-labeled graphs, recent works [29], [30], [32], [36] generalize the idea of self-supervised learning (SSL) [38] to graph data and propose to pre-train GNNs on large scale of unlabeled graph data $\mathcal{D}_{ssl}$:

$$\boldsymbol{\theta}^{init} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{ssl}(f_{\boldsymbol{\psi}, \boldsymbol{\theta}}(\cdot); \mathcal{D}_{ssl}), \quad (6)$$

where $\mathcal{L}_{ssl}(\cdot)$ is the loss function based on the adopted SSL pretext task. Depends on the instantiation of $\mathcal{L}_{ssl}(\cdot)$, existing pre-trained GNNs can be roughly categorized into: AutoEncoding [13], [35], Autoregressive Modeling [32], Maksed Component Modeling [29], [40], Context Prediction [29], and Contrastive Learning [30], [31], [34], [63]. More technical details can be found in Sec. IV-B.

Then, to allow the knowledge transfer from large-scale $\mathcal{D}_{ssl}$ to small-scale downstream $\mathcal{D}_{ft}$, they initialize the downstream GNN model with backbone architectures and pre-trained parameters $(\boldsymbol{\psi}, \boldsymbol{\theta}^{init})$. Then they perform model fine-tuning with task-specific labels in $\mathcal{D}_{ft}$ via the fine-tuning objective $\mathcal{L}_{ft}(\cdot)$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \boldsymbol{\Phi}_{ft}[\mathcal{L}_{ft}(f_{\boldsymbol{\psi}, \boldsymbol{\theta}}(\cdot); \mathcal{D}_{ft})], \quad (7)$$

where the optimization for initialization model $f_{\boldsymbol{\psi}, \boldsymbol{\theta}^{init}}(\cdot)$ is guided by the fine-tuning strategy $\boldsymbol{\Phi}_{ft}$. In the next, we elaborate on existing methods of fine-tuning techniques $\boldsymbol{\Phi}_{ft}$ within and outside pre-trained GNNs. We further summarize the comparisons between the proposed S2PGNN with existing ones in Tab. II.

- Vanilla Fine-Tuning (VFT): VFT is probably the most prevalent tuning strategy to adopt pre-trained GNNs among existing literatures [29], [30], [32], [34]. Under this schema,

TABLE II: Overview of common fine-tuning strategies in GNN and other domains. **Fine-tuning Scenarios** present the application scenario of the fine-tuning strategies, including whether is designed for GNN or other neural networks and which graph task the strategy can be applied to. **Fine-tuning Dimensions** presents what aspects they focus on during fine-tuning. **Automated** refers to whether they are capable of automatically designing the most suitable strategies to adaptively fine-tune the model for different downstream data.

Strategy Name		**Fine-tuning Scenarios**		**Fine-tuning Dimensions**					**Automated**
		GNN/Other	Graph Task	Model Architecture	Identity	Fusion	Readout	Model Weights	
Vanilla Fine-Tuning (VFT)		$\surd/\surd$	Node/Edge/Graph	$\times$	$\times$	$\times$	$\times$	$\surd$	$\times$
Regularized Fine-Tuning (RFT)	AUX-TS [42]	$\surd/\times$	Node/Edge	$\times$	$\times$	$\times$	$\times$	$\surd$	$\times$
	WordReg [46]	$\surd/\times$	Graph	$\times$	$\times$	$\times$	$\times$	$\surd$	$\times$
	GTOT-Tuning [48]	$\surd/\times$	Graph	$\times$	$\times$	$\times$	$\times$	$\surd$	$\times$
	L^2-SP [59]	$\times/\surd$	-	$\times$	$\times$	$\times$	$\times$	$\surd$	$\times$
	DELTA [60]	$\times/\surd$	-	$\times$	$\times$	$\times$	$\times$	$\surd$	$\times$
	BSS [61]	$\times/\surd$	-	$\times$	$\times$	$\times$	$\times$	$\surd$	$\times$
	StochNorm [62]	$\times/\surd$	-	$\surd$	$\times$	$\times$	$\times$	$\surd$	$\times$
Feature Extractor (FE)		$\times/\surd$	-	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
Last-k Tuning (LKT)		$\times/\surd$	-	$\times$	$\times$	$\times$	$\times$	$\surd$	$\times$
Adapter-Tuning (AT)		$\times/\surd$	-	$\surd$	$\times$	$\times$	$\times$	$\surd$	$\times$
S2PGNN		$\surd/\times$	Graph	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$

all parameters of the pre-trained GNN together with the new prediction head are fine-tuned with the task-specific supervised loss $\mathcal{L}_{sup}(\cdot)$ (e.g., cross-entropy loss for classification task) for given downstream scenarios:

$$\mathcal{L}_{ft}(\cdot) \equiv \mathcal{L}_{sup}(\cdot) \tag{8}$$

- Regularized Fine-Tuning (RFT): To prevent the overfitting and improve generalization, RFT methods [42], [46], [48] present improved fine-tuning strategies $\mathbf{\Phi}_{ft}$ by regularization. They conduct constrained downstream adaptation to enforce the similarity of model parameters or learned representations. This is generally achieved by incorporating a regularization item $\mathcal{L}_{reg}(\cdot)$ with the original $\mathcal{L}_{sup}(\cdot)$:

$$\mathcal{L}_{ft}(\cdot) = \mathcal{L}_{sup}(\cdot) + \mathcal{L}_{reg}(\cdot), \tag{9}$$

where different RFT methods differ in the way to instantiate $\mathcal{L}_{reg}(\cdot)$. Among literatures, AUX-TS [42] augments fine-tuning objectives with SSL objectives so as to reduce the gap between two stages and improve results. WordReg [46] develops smoothness-inducing regularizer built on dropout [64] to constrain representation distance induced by pre-trained and fine-tuned models. GTOT-Tuning [48] uses optimal transport to align graph topologies of pre-trained and fine-tuned models while preserving learned representations.

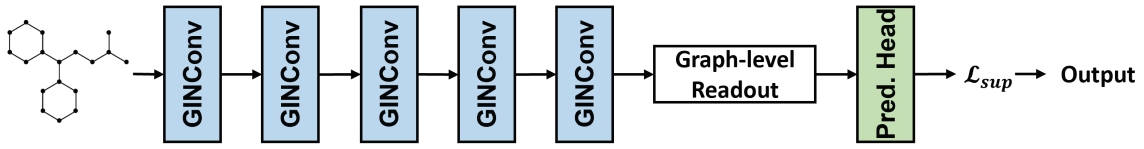
Remark 1 (Prompt Tuning): Prompt Tuning in GNNs [65]–[67] propose to unify the pre-training and downstream graph tasks with the shared task template and leverage the prompt technique [68] to prompt pre-trained knowledge for downstream learning. However, their methods may rely on the high relevance between the pre-training domain with the downstream one, which however may not hold in practical scenarios, especially when out-of-distribution predictions are demanded (e.g., molecular property prediction [29]). Besides, they mainly investigate how to unify tasks in two stages and task template designs, which is significantly different with our fine-tuning search problem (see Definition 1). Therefore, due to the significantly different research scope with this work, prompting methods are excluded for comparison in Tab. II.

Remark 2 (Fine-tuning Technique Outside GNNs): Additionally, we further discuss and test several RFT methods (including L^2-SP [59], DELTA [60], BSS [61], and StochNorm [62]) that are initially designed to fine-tune other types of deep models (e.g., CNNs) in the empirical study Sec. IV-C. Besides, for more comprehensive exploration, we further cover other classic fine-tuning techniques that are originally designed outside GNN area, including: Feature Extractor (FE) [69] that disables the fine-tuning to rely on parameter reuse, Last-k Tuning (LKT) [70] that freezes initial layers to fine-tune only last k layers, and Adapter-Tuning (AT) [44] that is representative of parameter-efficient method which fine-tunes only a small number of extra parameters in Adapter modules.

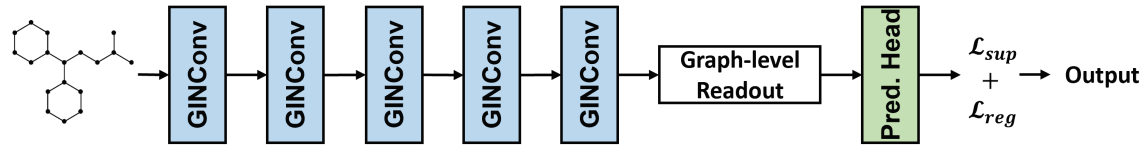
III. Methodology

As introduced in Sec. I, recently developed GNN pre-training techniques [29]–[36] have demonstrated promising to tackle the label-scarcity issue and improve the downstream learning. Despite the emergence of various pre-training strategies, how to fine-tune pre-trained GNNs for downstream scenarios, while important, is largely ignored in current literature. Only a few works start to investigate this direction, but existing methods either have strong assumptions or overlook the data-aware issue behind various downstream domains.

To fully unleash the potential of pre-trained GNNs on various downstream datasets, we aim to design a fine-tuning searching strategy to automatically design suitable fine-tuning framework for the given pre-trained GNN and downstream graph dataset. However, it is a non-trivial task to achieve this search idea due to the undefined search space and large computational overhead for optimizing the search problem. In this section, we first define a new problem of searching fine-tuning strategies for pre-trained GNNs. We next propose an expressive search space, which enables powerful models to be



(a) An example of vanilla fine-tuning strategy.



(b) An example of fine-tuning strategy with regularization.

Fig. 2: Illustration of GNN fine-tuning strategies (refer to Fig. 3 for the legend).

searched. Then, to reduce the search cost from the large and discrete space, we incorporate an efficient search algorithm and solve the search problem by differentiable optimization.

A. Problem Formulation

Definition 1 (The Fine-tuning Strategy Search Problem for Pre-trained GNNs): Given a pre-trained GNN model $f_{\psi,\theta}(\cdot)$ with architectures ψ and parameters θ, and downstream graph dataset with split $\mathcal{D}_{ft} = (\mathcal{D}_{ft}^{(tra)}, \mathcal{D}_{ft}^{(val)})$, the automatic and data-aware GNN fine-tuning search problem in S2PGNN can be formally defined as a bi-level optimization problem:

$$\mathbf{\Phi}_{ft}^{*} = \arg\min_{\mathbf{\Phi}_{ft},\boldsymbol{\theta}^{*}} \mathbf{\Phi}_{ft}[\mathcal{L}_{ft}(f_{\psi,\boldsymbol{\theta}^{*}}(\cdot); \mathcal{D}_{ft}^{(val)})], \tag{10}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^{*} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{ft}(f_{\psi,\boldsymbol{\theta}}(\cdot); \mathcal{D}_{ft}^{(tra)}), \tag{11}$$

where $\mathcal{L}_{ft}(\cdot)$ is the fine-tuning loss function, and $\mathbf{\Phi}_{ft}$ is the fine-tuning strategy that is to be searched.

By comparing Eq. (5) and (7) with above Eq. (10), it is intuitive that existing AutoGNNs focus on searching the GNN architecture ψ and optimizing $\boldsymbol{\theta}$. And existing pre-trained GNNs present a fixed fine-tuning strategy $\mathbf{\Phi}_{ft}$ and optimize the parameter $\boldsymbol{\theta}$. Both of them cannot be extended to solve the problem of searching fine-tuning strategies in Eq. (10). Furthermore, it is hard to solve Eq. (10) due to the unclear search space of $\mathbf{\Phi}_{ft}$ and large computational overhead of search algorithm. More specifically, fine-tuning pre-trained GNNs is still an under-investigated problem and there lack a systematic and unified design space in existing literature. Besides, it is very time-consuming to enumerate all possible choices of $\mathbf{\Phi}_{ft}$ because we need to train the model parameter $\boldsymbol{\theta}$ to convergence for each $\mathbf{\Phi}_{ft}$.

B. GNN Fine-tuning Search Space

To tackle the challenge of space design for GNN fine-tuning strategy, we provide a novel GNN fine-tuning search space from a brand new perspective, i.e., to incorporate the space of GNN structures with fine-tuning strategies. More specifically, to ensure the improvement brought by searching fine-tuning strategies, we identify that the backbone convolution $\boldsymbol{\phi}_{conv}(\cdot)$, identity augmentation $\boldsymbol{\phi}_{id}(\cdot)$, multi-scale fusion $\boldsymbol{\phi}_{fuse}(\cdot)$, and graph-level readout $\boldsymbol{\phi}_{read}(\cdot)$ for model structures are key functions to affect GNN fine-tuning results. Accordingly, the improved downstream message-passing with key functions in S2PGNN can be represented as:

$$\begin{cases} \mathbf{Z}_v \leftarrow \boldsymbol{\phi}_{conv}(\{\mathbf{H}_u, \mathbf{H}_v, \mathbf{X}_{uv} | u \in N(v)\}), \\ \mathbf{H}_v \leftarrow \boldsymbol{\phi}_{id}(\mathbf{H}_v, \mathbf{Z}_v), \quad 1 \le k \le K, \end{cases} \tag{12}$$

$$\mathbf{H}_v = \boldsymbol{\phi}_{fuse}(\{\mathbf{H}_v^{(k)} | 1 \le k \le K\}), \tag{13}$$

$$\mathbf{H}_G = \boldsymbol{\phi}_{read}(\{\mathbf{H}_v | v \in V\}), \tag{14}$$

where $\boldsymbol{\phi}_{conv}(\cdot)$ summarizes the intra-layer message passing Eq. (2) and (3), $\boldsymbol{\phi}_{id}(\cdot)$ augments the intra-layer message passing in pre-trained backbone $\boldsymbol{\phi}_{conv}(\cdot)$ with identity information from center node itself, $\boldsymbol{\phi}_{fuse}(\cdot)$ fuses the multi-scale information from different GNN layers k, $\boldsymbol{\phi}_{read}(\cdot)$ summarizes node representations to yield the graph representation. Based on Eq. (12), (13), and (14), the fine-tuning model structure is shown in Fig. 3, where we use the classic 5-layer GIN backbone [29] as an example for demonstration. Next, we elaborate on the illustrations of the proposed dimensions and their corresponding candidate-sets.

1) Backbone Convolution $\boldsymbol{\phi}_{conv}(\cdot)$: This dimension is the most basic building block in pre-trained GNNs. We directly transfer its structure and parameters to the downstream model.

2) Identity Augmentation $\boldsymbol{\phi}_{id}(\cdot)$: This dimension is important for data-aware fine-tuning due to several reasons. Firstly, in some downstream data, the identity information from node itself may be more essential than that aggregated from neighbors via $\boldsymbol{\phi}_{conv}(\cdot)$, especially when the neighboring messages can be missing, noisy, or unreliable. Secondly, the distinguishable information might be easily diluted and over-smoothed [71] in some backbone choices $\boldsymbol{\phi}_{conv}(\cdot)$, e.g., GCN. Thus, $\boldsymbol{\phi}_{id}(\cdot)$ may be beneficial to adjust the message flow in $\boldsymbol{\phi}_{conv}(\cdot)$. We include candidates from:

- **No augmentation.** We disable the identity augmentation and keep consistent as in pre-trained backbone with $zero\_aug$.

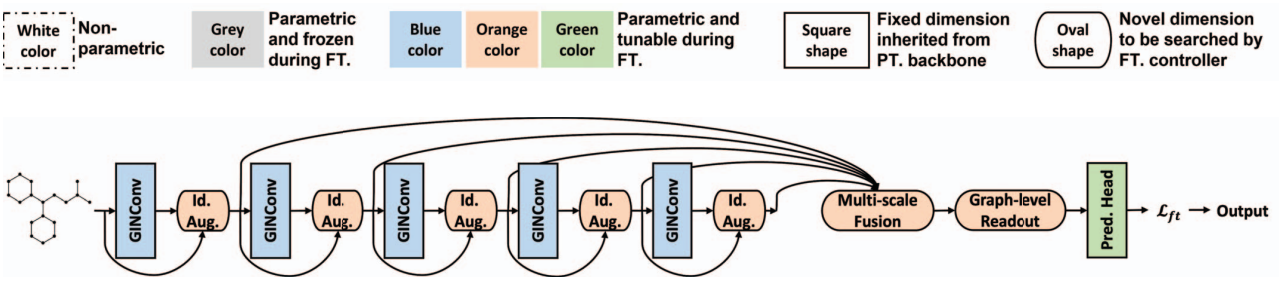


Fig. 3: Illustration to the framework of S2PGNN built on top of pre-trained 5-layer GIN. The orange part indicates the search dimensions in S2PGNN. PT., FT., and Id.Aug. are abbreviations for pre-training, fine-tuning, and identity augmentation.

- **Additive augmentation.** We allow direct skip-connection [72] with $identity\_aug$ as $\mathbf{H}_v \leftarrow \mathbf{H}_v + \mathbf{Z}_v$. We also allow transformed augmentation with $trans\_aug$ as $\mathbf{H}_v \leftarrow g(\mathbf{H}_v) + \mathbf{Z}_v$, where $g(\cdot)$ is a parameterized neural network with bottleneck architecture that maps $\mathbb{R}^d \rightarrow \mathbb{R}^m \rightarrow \mathbb{R}^d$, where we enforce $m \ll d$ for parameter-efficient transformation similar to Adapter Tuning [44].

3) Multi-scale Fusion $\phi_{fuse}(\cdot)$: Due to the diversity of downstream data structure (e.g., density, topology), the most suitable GNN layers required by different data may be highly data-specific [18], [50]. Besides, immediate representations in hidden layers of pre-trained GNNs often capture a spectrum of graph information with multiple scales, i.e., from local to global as layer increases [16]. Therefore, to make the fullest usage of pre-trained information, we propose to fuse the multi-scale information from different layers $\mathbf{H}_v = \sum_k w_v^{(k)} \mathbf{H}_v^{(k)}$ so as to allow the more adaptive and effective learning for downstream data. We cover candidates from:

- **Non-parametric fusion.** They use simple non-parametric approaches to determine weights $w_v^{(k)}$ for fusion. Among candidates listed in Tab. III, $last$, $concat$, max, $mean$, and ppr belong to this type. Specifically, $last$ disables the fusion to directly take the single-scale representations from last layer, $concat$ concatenates the multi-scale information for fusion, max takes the maximum value from each channel to induce the fused representations, $mean$ assigns equal importance weights for information in each layer, and ppr assigns decayed weights with Personalized PageRank [73].
- **Attentive fusion.** They allow the adaptive importance weights $w_v^{(k)}$ via attention mechanisms, where $w_v^{(k)} \in [0, 1]$ and $\sum_k w_v^{(k)} = 1$. We adopt the powerful $lstm$ fusion that is similar as in [16].
- **Gated fusion.** Gated fusion methods use gating functions to selectively filter information at different layers. We use gpr method for this category similar as in [18], which allows the adaptive scale as well as the sign of information, i.e., $w_v^{(k)} \in [-1, 1]$.

4) Graph-level Readout $\phi_{read}(\cdot)$: This is the compulsory function for the downstream graph-level predictions. Different readout methods focus on the capture of information from different aspects, e.g., node features or graph topology [74]. Thus, downstream data with different structures and properties may have their data-specific requirements towards the effective readout. Candidates for this dimension include:

- **Simple readout.** They are parameter-free and computationally fast, which may be suitable for graph data where the overall graph topology is less important than individual node features. We include $sum\_pooling$, $mean\_pooling$, and $max\_pooling$ readouts for this type.
- **Adaptive readout.** Adaptive methods aim to identify and capture most informative nodes or substructures into the graph representation via more sophisticated designs. We review existing literatures to cover powerful candidates $set2set$ [75], $sort\_pooling$ [76], $multiset\_pooling$ [54], and $neural\_pooling$ [77] for this category.

Remark 3 (Space Complexity of $\mathbf{\Phi}_{ft}$): As shown in Fig. 3 and summarized in Tab. III, the overall complexity of the proposed GNN fine-tuning strategy space $\mathbf{\Phi}_{ft}$ in S2PGNN equals to the Cartesian product of the size of all involved dimensions, i.e., $O(|\mathcal{O}_{conv}|^K \cdot |\mathcal{O}_{id}|^K \cdot |\mathcal{O}_{fuse}| \cdot |\mathcal{O}_{read}|)$. For the illustration in Fig. 3, the search space built on the 5-layer GIN will have 10,206 candidate fine-tuning strategies. Because each candidate will require to train the GNN parameter to convergence, we cannot simply use a brute-force algorithm to test all possible candidates and train thousands of GNNs, which can lead to enormous computational overhead.

C. GNN Fine-tuning Search Algorithm

Recall that as introduced in Sec. III-A, the GNN fine-tuning strategy search problem is formally defined as Eq. (10) and (11). To solve this problem, we need to compute the gradients of GNN fine-tuning strategy $\nabla_{\boldsymbol{\phi}} \mathcal{L}_{ft}$ and GNN model weights $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{ft}$. Computing $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{ft}$ is simple since $\boldsymbol{\theta}$ is continuous. Unfortunately, the GNN fine-tuning strategy $\boldsymbol{\phi}$ is categorical choices (see Tab. III) and thus discrete, thereby $\nabla_{\boldsymbol{\phi}} \mathcal{L}_{ft}$ does not exist. Thus, we propose to train a controller $p_{\boldsymbol{\alpha}}(\phi)$ that parameterized by $\boldsymbol{\alpha}$ and hope this controller can sample better fine-tuning strategy ϕ after achieving the performance of the sampled ϕ. Therefore, we reformulate the problem into:

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}, \boldsymbol{\theta}^*} \mathbb{E}_{\boldsymbol{\phi} \sim p_{\boldsymbol{\alpha}}(\boldsymbol{\phi})} [\mathcal{L}_{ft}(f_{\boldsymbol{\phi}, \boldsymbol{\theta}^*}(\cdot); \mathcal{D}_{ft}^{(val)})], \tag{15}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{ft}(f_{\boldsymbol{\phi}, \boldsymbol{\theta}}(\cdot); \mathcal{D}_{ft}^{(tra)}), \tag{16}$$

TABLE III: The GNN fine-tuning design space in S2PGNN: design dimensions and candidate-sets.

Design Dimension		Candidate-set
Backbone Convolution	$\phi_{conv}(\cdot)$	$\mathcal{O}_{conv} = \{pre\_trained\}$
Identity Augmentation	$\phi_{id}(\cdot)$	$\mathcal{O}_{id} = \{zero\_aug, identity\_aug, trans\_aug\}$
Multi-scale Fusion	$\phi_{fuse}(\cdot)$	$\mathcal{O}_{fuse} = \{last, concat, max, mean, ppr, lstm, gpr\}$
Graph-level Readout	$\phi_{read}(\cdot)$	$\mathcal{O}_{read} = \{sum\_pooling, mean\_pooling, max\_pooling, set2set, sort\_pooling, neural\_pooling\}$

where $\boldsymbol{\alpha}$ is the continuous controller parameter that controls the selection of GNN fine-tuning strategy $\boldsymbol{\phi}$, $\boldsymbol{\phi} \sim p_{\boldsymbol{\alpha}}(\boldsymbol{\phi})$ represents a GNN fine-tuning strategy $\boldsymbol{\phi}$ being sampled from the distribution $p_{\boldsymbol{\alpha}}(\boldsymbol{\phi})$ parameterized by $\boldsymbol{\alpha}$, and $\mathbb{E}[\cdot]$ is the expectation function. We aim to optimize $\boldsymbol{\alpha}$ such that it can yield the optimal fine-tuning strategy $\boldsymbol{\phi}^* \sim p_{\boldsymbol{\alpha}^*}(\boldsymbol{\phi})$. In the following, we demonstrate the concrete algorithm to solve Eq. (15) and (16) in Sec. III-C1 and Sec. III-C2.

1) Update Fine-tuning Strategy by Re-parameterization: We propose to employ dimension-specific controllers $\boldsymbol{\alpha} = \{\boldsymbol{\alpha}_{conv}, \boldsymbol{\alpha}_{id}, \boldsymbol{\alpha}_{fuse}, \boldsymbol{\alpha}_{raad}\}$ to guide the GNN fine-tuning strategy search from $\mathcal{O} = \{\mathcal{O}_{conv}, \mathcal{O}_{id}, \mathcal{O}_{fuse}, \mathcal{O}_{raad}\}$ for respective dimensions (see Sec. III-B and Tab. III), where $\boldsymbol{\alpha}_{dim} \in \mathbb{R}^{|\mathcal{O}_{dim}|}$ for $\forall dim \in \{conv, id, fuse, read\}$. Then, for the given fine-tuning dimension with candidate-set $\mathcal{O}$ (note that we discard the notation of dimension dim for brevity), let the one-hot vector $\boldsymbol{\phi} \sim p_{\boldsymbol{\alpha}}(\boldsymbol{\phi}) \in \mathbb{R}^{|\mathcal{O}|}$ represent the sampled strategy, let $\mathbf{Z}_{in}$ be the intermediate representation that to be fed into this dimension, then its output is calculated as $\mathbf{Z}_{out} = \sum_{i=1}^{|\mathcal{O}|} \boldsymbol{\phi}[i] \cdot \mathcal{O}[i](\mathbf{Z}_{in})$. However, the sampling process $\boldsymbol{\phi} \sim p_{\boldsymbol{\alpha}}(\boldsymbol{\phi})$ is discrete, which makes the derivative $\nabla_{\boldsymbol{\alpha}} \mathbf{Z}_{out}$ undefined and the gradient $\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{ft}$ non-existent.

To address this issue for optimizing strategy controller parameters $\boldsymbol{\alpha}$ in Eq. (15), we leverage the re-parameterization trick in [78], [79] to relax the discrete strategy sampling to be continuous and differentiable via $\boldsymbol{\phi} = g_{\boldsymbol{\alpha}}(\boldsymbol{U})$:

$$g_{\boldsymbol{\alpha}}(\boldsymbol{U}[i]) = \frac{\exp((\log \boldsymbol{\alpha}[i] - \log(-\log(\boldsymbol{U}[i])))/\tau)}{\sum_{j=1}^{|\mathcal{O}|} \exp((\log \boldsymbol{\alpha}[j] - \log(-\log(\boldsymbol{U}[i])))/\tau)}, \quad (17)$$

where $\boldsymbol{U}[i] \sim Uniform(0, 1)$ is sampled from the uniform distribution, and τ is the temperature that controls the discreteness of softmax output. In this way, the relaxed strategy $g_{\boldsymbol{\alpha}}(\boldsymbol{U})$ is differentiable w.r.t. $\boldsymbol{\alpha}$. Then, the gradient $\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{ft}$ for solving Eq. (15) can be computed as:

$$\begin{aligned} &\nabla_{\boldsymbol{\alpha}} \mathbb{E}_{\boldsymbol{\phi} \sim p_{\boldsymbol{\alpha}}(\boldsymbol{\phi})}[\mathcal{L}_{ft}(\boldsymbol{\phi}, \boldsymbol{\theta}^*; \mathcal{D}_{ft}^{(val)})] \\ &= \nabla_{\boldsymbol{\alpha}} \mathbb{E}_{\boldsymbol{U} \sim p(\boldsymbol{U})}[\mathcal{L}_{ft}(g_{\boldsymbol{\alpha}}(\boldsymbol{U}), \boldsymbol{\theta}^*; \mathcal{D}_{ft}^{(val)})] \\ &= \mathbb{E}_{\boldsymbol{U} \sim p(\boldsymbol{U})}[\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{ft}(g_{\boldsymbol{\alpha}}(\boldsymbol{U}), \boldsymbol{\theta}^*; \mathcal{D}_{ft}^{(val)})] \\ &= \mathbb{E}_{\boldsymbol{U} \sim p(\boldsymbol{U})}[\mathcal{L}_{ft}^{'}(g_{\boldsymbol{\alpha}}(\boldsymbol{U}), \boldsymbol{\theta}^*; \mathcal{D}_{ft}^{(val)}) \nabla_{\boldsymbol{\alpha}} g_{\boldsymbol{\alpha}}(\boldsymbol{U})], \end{aligned} \quad (18)$$

where $\nabla_{\boldsymbol{\alpha}} g_{\boldsymbol{\alpha}}(\boldsymbol{U})$ can be computed because $g_{\boldsymbol{\alpha}}(\boldsymbol{U})$ presented in Eq. (17) is differentiable and Eq. (18) can be easily approximated by Monte-Carlo (MC) sampling [80]. Note that $\tau \to 0$ makes the continuous output in Eq. (17) almost indistinguishable with the discrete one-hot vector, which thereby ensures the relaxation in Eq. (17) to be unbiased once converged.

2) Update GNN Weights by Weight-sharing: The gradient computation for high-level Eq. (15) requires frequent performance evaluation of sampled GNN fine-tuning strategy, which leads to heavy computational cost. To alleviate this issue and accelerating search, we suggest the weight-sharing [81] in low-level Eq. (16). Specifically, we evaluate every sampled strategy $\boldsymbol{\phi}$ (Eq. (17)) based on the shared GNN models weights $\boldsymbol{\theta}$, thereby we avoid repeatedly training the GNN model weights from scratch. The gradient w.r.t. shared model weights $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{ft}$ for solving Eq. (16) can be calculated as:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{ft}(\boldsymbol{\phi}, \boldsymbol{\theta}; \mathcal{D}_{ft}^{(tra)}) = \frac{1}{|\mathcal{D}_{ft}^{(tra)}|} \sum_{(x,y) \in \mathcal{D}_{ft}^{(tra)}} \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\phi}, \boldsymbol{\theta}; (x, y)),$$

where $\ell(\cdot)$ is the loss for each labeled data instance (x, y). Empirically, we leverage the cross-entropy loss for graph classification task and MSE loss for graph regression task.

IV. Experiments

As presented in Sec. II, the GNN pre-training approach generally contains a GNN backbone model (e.g., GCN [12], SAGE [13]), a GNN pre-training strategy (e.g., MCM [29], [40], CL [63]) to transfer knowledge from a large scale of unlabeled graph data $\mathcal{D}_{ssl}$, a GNN fine-tuning strategy (e.g., ST [29], RT [48]) to adapt the pre-trained GNNs to the domain-specific data with labels $\mathcal{D}_{ft}$. In this paper, we mainly investigate the improvement from the perspective of automatically designing a suitable fine-tuning strategy for the given data $\mathcal{D}_{ft}$. Thus, to validate the effectiveness of the proposed S2PGNN, we need to answer the following questions:

- **Q1**: Can S2PGNN be built on top of different GNN backbone and pre-training methods and consistently improve their performance? (see Sec. IV-B and Sec. IV-E)
- **Q2**: On the same configuration of GNN backbone model and pre-training strategy, can S2PGNN be more effective than other GNN fine-tuning strategies? (see Sec. IV-C)
- **Q3**: As discussed in Sec. II-B, there are several classic fine-tuning strategies in other domains (see Remark 2) that are not included in our search space. Will these models perform well on the GNN area? (see Sec. IV-C)
- **Q4**: What are the effects of each design dimension in S2PGNN? (see Sec. IV-D)
- **Q5**: Is there a risk that the fine-tuning search method will consume more computing resources to achieve improved performance? (see Sec. IV-F)

A. Experimental Settings

S2PGNN [1] is implemented based on PyTorch [82] and PyTorch Geometric [83] libraries. All experiments are conducted with one single NVIDIA Tesla V100 GPU.

[1]Code and data are available at https://github.com/zwangeo/icde2024.

TABLE IV: Summary of downstream GNN fine-tuning datasets $\mathcal{D}_{ft}$.

Dataset	#Molecules	#Tasks	Task Type	Metric	Domain
BBBP	2039	1	Classification	ROC-AUC (%) (↑)	Pharmacology
Tox21	7831	12	Classification	ROC-AUC (%) (↑)	Pharmacology
ToxCast	8575	617	Classification	ROC-AUC (%) (↑)	Pharmacology
SIDER	1427	27	Classification	ROC-AUC (%) (↑)	Pharmacology
ClinTox	1478	2	Classification	ROC-AUC (%) (↑)	Pharmacology
BACE	1513	1	Classification	ROC-AUC (%) (↑)	Biophysics
ESOL	1128	1	Regression	RMSE (↓)	Physical Chemistry
Lipophilicity (Lipo)	4200	1	Regression	RMSE (↓)	Physical Chemistry

1) GNN backbone models: For GNN backbone architectures, we mainly adopt classic and promising GNNs in recent years, including (5-layer) GCN [12], SAGE [13], Graph Isomorphism Network (GIN) [15], and GAT [14] (see more details in Sec. II-A). But due to the limited space, the experiments in sections Sec. IV-B, Sec. IV-C, Sec. IV-D, and Sec. IV-F are conducted on GIN. And we report the performance on other backbone models in Sec. IV-E.

2) GNN pre-training methods and datasets $\mathcal{D}_{ssl}$: To demonstrate the effectiveness of S2PGNN, we implement S2PGNN on top of 10 well-known and publicly available pre-training methods, including Infomax [63], EdgePred [13], ContextPred [29], AttrMasking [29], GraphCL [30], GraphLoG [31], MGSSL [32], SimGRACE [34], GraphMAE [35], and Mole-BERT [40]. As summarized in Tab. V, they cover a wide spectrum of various SSL strategies $\mathcal{L}_{ssl}(\cdot)$ (see more details in Sec. IV-B). Due to the space limit, the fine-tuning experiments in Sec. IV-B are conducted on top of all 10 pre-trained models, and experiments in Sec. IV-C, Sec. IV-D, Sec. IV-F, and Sec. IV-F are built on top of the pioneering pre-training work ContextPred [29].

In this paper, we follow the literature to adopt ZINC15 (250K) for MGSSL [32], which contains 250K unlabeled molecules collected from the ZINC15 database [84]. We use the larger version ZINC15 with 2 million molecules for methods other than MGSSL [32].

3) GNN fine-tuning tasks and datasets $\mathcal{D}_{ft}$: The proposed S2PGNN is built on top of existing pre-trained GNNs. Thus, we follow the vast majority of existing pre-trained GNNs to focus on the graph-level tasks (including graph classification and graph regression) for downstream evaluation. Specifically, we mainly conduct graph-level experiments on downstream molecular property prediction (MPP), which is an important task for a variety of domains (e.g., physics, chemistry, and materials science). In MPP, a molecule is represented as a graph, where nodes and edges denote atoms and bonds, and labels are related to molecular toxicity or enzyme binding properties. The aim of MPP is to predict the properties for unlabeled molecules. We employ ROC-AUC and RMSE for evaluating the classific and regressive MPP tasks, respectively. For datasets with multiple prediction tasks (see Tab. IV), we report average results over all their tasks.

As shown in Tab. VI, we follow the related literature [29], [85] to adopt 8 popular benchmark datasets: BBBP [86], Tox21 [87], ToxCast [88], SIDER [89], ClinTox [90], BACE [91], ESOL [92], and Lipophilicity (Lipo) [93] that are provided in MoleculeNet [94]. The selected datasets are from several domains, including pharmacology, biophysics, and physical chemistry. Among them, ESOL and Lipo are used for graph regression, while the rest are for graph classification. As for data split, we utilize *scaffold-split* [49] to split molecular datasets according to their substructures as suggested by [29], [85]. It provides a more challenging yet more realistic split to deal with real-world applications (where out-of-distribution predictions are often required) compared with random-split. Note that AutoGNNs are excluded for empirical comparisons because they focus on searching the GNN architecture (as in Eq. (5)), which thereby cannot be extended to solve the problem of searching fine-tuning strategies in ours (as in Definition 1 and Eq. (10)).

TABLE V: Summary of base GNN pre-training methods.

Method	SSL Strategy $\mathcal{L}_{ssl}(\cdot)$	SSL Data $\mathcal{D}_{ssl}$
Infomax	Contrastive Learning (CL)	ZINC15 (2M)
EdgePred	Autoencoding (AE)	ZINC15 (2M)
ContextPred	Context Prediction (CP)	ZINC15 (2M)
AttrMasking	Masked Component Modeling (MCM)	ZINC15 (2M)
GraphCL	Contrastive Learning (CL)	ZINC15 (2M)
GraphLoG	Contrastive Learning (CL)	ZINC15 (2M)
MGSSL	Autoregressive Modeling (AM)	ZINC15 (250K)
SimGRACE	Contrastive Learning (CL)	ZINC15 (2M)
GraphMAE	AutoEncoding (AE)	ZINC15 (2M)
Mole-BERT	Masked Component Modeling (MCM)	ZINC15 (2M)

4) Implementation details: For GNN pre-training methods, we employ their officially released pre-trained models to conduct the next stage fine-tuning. Readers may refer to the original papers for their detailed pre-training settings.

S2PGNN and other fine-tuning baselines are evaluated with the same protocol for rigorously fair comparisons. We follow pioneering literature [29] to setup fine-tuning configurations. Specifically, we leverage the simple linear classifier as downstream prediction head. To fine-tune the whole model, we use Adam optimizer with a learning rate of 1e-3. We set batch size as 32 and dropout rate as 50%. We perform fine-tuning for 100 epochs with early stopping based on validation set. The ratio to split train/validation/test set is 80%/10%/10%. We run all experiments for 10 times with different random seeds and report the mean results (standard deviations).

B. The implementation S2PGNN with Pre-trained GNNs and comparison with vanilla fine-tuning

As discussed in Sec. II-B, vanilla fine-tuning is still the most prevalent way to leverage pre-trained GNNs in existing works. Thus, we first implement and compare the gains of S2PGNN

TABLE VI: The performance comparison between proposed S2PGNN and vanilla fine-tuning (see Sec. II-B) with different pre-training objectives (see Sec. IV-B) and fixed GIN backbone. The rightmost column of each task type averages S2PGNN's gain/reduction over vanilla fine-tuning across all involved datasets given the specific pre-training objectives.

	Classification (ROC-AUC (%)) ↑						**Regression (RMSE) ↓**		**Avg.**
Dataset	BBBP	Tox21	ToxCast	SIDER	ClinTox	BACE	ESOL	Lipo	**Gain**
Infomax [63]	68.4 ± 1.7	75.6 ± 0.5	62.5 ± 0.8	58.3 ± 0.7	71.3 ± 2.6	75.5 ± 2.3	2.6 ± 0.1	1.0 ± 0.1	+17.7%
Infomax + S2PGNN	**69.9 ± 1.4**	**76.7 ± 0.5**	**65.8 ± 0.4**	**62.3 ± 1.3**	**74.8 ± 3.8**	**82.3 ± 1.3**	**1.5 ± 0.3**	**0.8 ± 0.0**	
EdgePred [13]	67.2 ± 2.9	75.8 ± 0.9	63.9 ± 0.4	60.5 ± 0.8	65.7 ± 4.1	79.4 ± 1.4	2.8 ± 0.0	1.0 ± 0.1	+14.4%
EdgePred + S2PGNN	**69.1 ± 0.8**	**77.1 ± 0.8**	**66.2 ± 0.3**	**62.3 ± 0.5**	**71.9 ± 1.1**	**82.2 ± 1.1**	**1.7 ± 0.2**	**0.9 ± 0.0**	
ContextPred [29]	69.0 ± 0.9	76.0 ± 0.4	63.5 ± 0.4	60.7 ± 0.6	69.7 ± 1.4	80.6 ± 0.8	2.8 ± 0.4	1.1 ± 0.0	+15.1%
ContextPred + S2PGNN	**70.9 ± 1.3**	**76.3 ± 0.4**	**67.0 ± 0.5**	**62.8 ± 0.3**	**75.9 ± 2.2**	**82.6 ± 0.7**	**1.7 ± 0.2**	**0.9 ± 0.0**	
AttrMasking [29]	65.1 ± 2.3	76.7 ± 0.6	64.4 ± 0.3	60.6 ± 0.9	72.0 ± 4.2	79.5 ± 0.7	2.8 ± 0.1	1.1 ± 0.0	+15.6%
AttrMasking + S2PGNN	**71.9 ± 1.1**	**77.3 ± 0.4**	**66.8 ± 0.5**	**62.9 ± 0.4**	**74.8 ± 3.1**	**82.7 ± 0.8**	**1.7 ± 0.1**	**0.9 ± 0.0**	
GraphCL [30]	68.3 ± 1.6	74.1 ± 0.8	62.6 ± 0.5	59.7 ± 1.1	71.4 ± 6.2	75.8 ± 2.7	2.5 ± 0.1	1.0 ± 0.0	+10.1%
GraphCL + S2PGNN	**70.8 ± 1.1**	**76.8 ± 0.5**	**66.6 ± 0.3**	**62.4 ± 1.2**	**75.2 ± 3.4**	**82.6 ± 2.3**	**1.9 ± 0.1**	**0.9 ± 0.0**	
GraphLoG [31]	66.5 ± 2.3	75.3 ± 0.3	63.3 ± 0.5	57.8 ± 1.2	69.6 ± 5.8	80.1 ± 2.4	2.5 ± 0.1	1.0 ± 0.0	+9.1%
GraphLoG + S2PGNN	**69.9 ± 1.5**	**76.8 ± 0.3**	**66.1 ± 0.2**	**62.0 ± 1.0**	**75.8 ± 2.1**	**85.1 ± 1.3**	**2.0 ± 0.1**	**0.9 ± 0.0**	
MGSSL [32]	67.4 ± 1.8	75.0 ± 0.6	63.1 ± 0.5	58.0 ± 1.2	68.1 ± 5.4	81.2 ± 3.3	2.4 ± 0.1	1.0 ± 0.0	+9.6%
MGSSL + S2PGNN	**69.4 ± 1.8**	**77.0 ± 0.6**	**66.3 ± 0.4**	**62.8 ± 1.2**	**76.7 ± 2.4**	**85.2 ± 1.2**	**1.9 ± 0.2**	**0.9 ± 0.0**	
SimGRACE [34]	67.9 ± 0.6	73.9 ± 0.5	61.9 ± 0.5	59.1 ± 0.7	61.1 ± 3.8	75.5 ± 1.4	2.6 ± 0.1	1.0 ± 0.0	+16.5%
SimGRACE + S2PGNN	**69.3 ± 0.9**	**75.9 ± 0.2**	**65.8 ± 0.3**	**62.3 ± 0.6**	**73.6 ± 3.2**	**83.9 ± 1.5**	**1.6 ± 0.3**	**0.9 ± 0.0**	
GraphMAE [35]	70.0 ± 1.0	75.1 ± 1.4	64.4 ± 2.0	60.7 ± 1.1	71.7 ± 7.2	79.8 ± 4.2	2.3 ± 0.4	1.0 ± 0.1	+10.3%
GraphMAE + S2PGNN	**70.3 ± 0.8**	**76.7 ± 0.8**	**66.4 ± 0.5**	**62.2 ± 0.6**	**77.0 ± 2.8**	**82.2 ± 1.2**	**1.6 ± 0.2**	**0.9 ± 0.0**	
Mole-BERT [40]	70.6 ± 1.4	77.4 ± 2.2	65.4 ± 1.9	61.9 ± 2.2	75.6 ± 3.1	77.4 ± 4.2	2.4 ± 0.4	1.0 ± 0.1	+14.5%
Mole-BERT + S2PGNN	**71.4 ± 0.4**	**79.5 ± 0.4**	**67.8 ± 0.2**	**63.8 ± 0.5**	**76.5 ± 0.5**	**84.2 ± 0.7**	**1.5 ± 0.2**	**0.8 ± 0.0**	

over vanilla strategy on top of 10 classic GNN pre-training methods (Tab. V) from various categories:

- AutoEncoding (AE): Given the partial access to graph, AE methods propose to reconstruct the input graph via autoencoder architecture [95]. Let $\tilde{G}$ be reconstructed graph, then their objective is: $\mathcal{L}_{ssl}(\cdot) = -\sum_{G\in\mathcal{D}_{ssl}} \log p(\tilde{G}|G)$.
- Autoregressive Modeling (AM): AM methods factorize input graph G as a sequence of components $\mathcal{C} = \{C_1, C_2, \dots\}$ (e.g., nodes, edges, and subgraphs) with some preset ordering and perform graph reconstruction in an autoregressive manner: $\mathcal{L}_{ssl}(\cdot) = -\sum_{G\in\mathcal{D}_{ssl}} \sum_{i=1}^{|\mathcal{C}|} \log p(C_i|C_{<i})$.
- Masked Component Modeling (MCM): MCM works masks out some components of input graphs (e.g., nodes, edges, and subgraphs), then aim to recover those masked ones $m(G)$ through the remaining ones $G\backslash m(G)$: $\mathcal{L}_{ssl}(\cdot) = -\sum_{G\in\mathcal{D}_{ssl}} \sum_{\hat{G}\in m(G)} \log p(\hat{G}|G\backslash m(G))$.
- Context Prediction (CP): CP explores graph structures and uses contextual information to design pre-training objectives. Let $t = 1$ if subgraph C_1 and surrounding context C_2 share the same center node, otherwise $t = 0$. CP leverages subgraphs to predict their surrounding context structures: $\mathcal{L}_{ssl}(\cdot) = -\sum_{G\in\mathcal{D}_{ssl}} \log p(t|C_1, C_2)$.
- Contrastive Learning (CL): CL conducts pre-training via maximizing the agreement between a pair of similar inputs, including Cross-Scale Contrastive Learning and Same-Scale Contrastive Learning. The former contrasts a pair of graph and its local substructure (G, C) against negative pairs (G, C^-): $\mathcal{L}_{ssl}(\cdot) = -\sum_{G\in\mathcal{D}_{ssl}}[\log s(G, C) - \sum_{C^-} \log s(G, C^-)]]$, where $s(\cdot,\cdot)$ is the similarity function. The latter maximizes the agreement between the augmented graph and its anchor graph (G, G^+) and meanwhile repel negative pairs (G, G^-): $\mathcal{L}_{ssl}(\cdot) = -\sum_{G\in\mathcal{D}_{ssl}}[\log s(G, G^+) - \sum_{C^-} \log s(G, G^-)]]$.

The main results on 8 downstream datasets are reported in Tab. VI. When equipped with S2PGNN, the pre-trained GNNs consistently demonstrate better fine-tuning performances on the graph classification and graph regression tasks than the vanilla fine-tuning. The gains are consistent on different datasets with diverse characteristics (see Tab. IV), and the average improvement across all datasets is significant ($9.1\% \sim 17.7\%$). Moreover, we also observe the superiority of S2PGNN is agnostic to GNN pre-training configurations, such as SSL strategy, SSL data, and attained pre-trained models (see Sec. IV-A2 and Tab. V). To summarize, observations from Tab. VI validate that S2PGNN fine-tuning provides a promisingly better solution than the vanilla strategy to achieve the better utilization of various pre-trained GNNs.

C. Comparison with other fine-tuning strategies

As discussed in Sec. II-B, we first investigate the vanilla fine-tuning and regularized fine-tuning strategies since they have already been adapted to the GNN area. Then, we explore more fine-tuning methods on other domains (e.g., computer vision) but have not been discussed in the GNN community.

1) GNN Fine-tuning Baselines: Apart from the vanilla strategy (see main results in Sec. IV-B), we also compare S2PGNN with another GNN fine-tuning work GTOT-Tuning [48], which belongs to regularized fine-tuning. We exclude AUX-TS [42] and WordReg [46] for baseline comparisons because: AUX-TS targets on different downstream tasks, and WordReg uses different data split with [29] but its code is not publicly available for reproducing results. Besides, we additionally report the results of several regularized fine-tuning baselines tailored from the computer vision domain (that are originally designed to fine-tune CNNs), including L^2-*SP* [59], DELTA [60], BSS [61], and StochNorm [62]. L^2-*SP* regularizes on model parameters to induce the fine-tuned

TABLE VII: The performance comparison between proposed S2PGNN fine-tuning and other fine-tuning strategies with fixed ContextPred pre-training objective and GIN backbone architecture.

	Classification (ROC-AUC (%)) ↑						Avg.
Dataset	BBBP	Tox21	ToxCast	SIDER	ClinTox	BACE	
Vanilla fine-tuning	68.0 ± 2.0	75.7 ± 0.7	63.9 ± 0.6	60.9 ± 0.6	65.9 ± 3.8	79.6 ± 1.2	69.0
L^2-*SP* [59]	68.2 ± 0.7	73.6 ± 0.8	62.4 ± 0.3	61.1 ± 0.7	68.1 ± 3.7	82.2 ± 2.4	69.3
DELTA [60]	67.8 ± 0.8	75.2 ± 0.5	63.3 ± 0.5	62.2 ± 0.4	73.4 ± 3.0	81.8 ± 1.1	70.6
BSS [61]	68.1 ± 1.4	75.9 ± 0.8	63.9 ± 0.4	60.9 ± 0.8	70.9 ± 5.1	82.4 ± 1.8	70.4
StochNorm [62]	69.3 ± 1.6	74.9 ± 0.6	63.4 ± 0.5	61.0 ± 1.1	65.5 ± 4.2	80.5 ± 2.7	69.1
GTOT-tuning [48]	70.0 ± 1.7	75.2 ± 0.9	63.0 ± 0.5	63.1 ± 0.6	71.8 ± 5.4	82.6 ± 2.0	71.0
S2PGNN (ContextPred)	70.9 ± 1.3	76.3 ± 0.4	67.0 ± 0.5	62.8 ± 0.3	75.9 ± 2.2	82.6 ± 0.7	72.6
S2PGNN (Mole-BERT)	**71.4 ± 0.4**	**79.5 ± 0.4**	**67.8 ± 0.2**	**63.8 ± 0.5**	**76.5 ± 0.5**	**84.2 ± 0.7**	**73.9**

TABLE VIII: The performance of more fine-tuning strategies that are excluded from S2PGNNX's design space with fixed ContextPred pre-training objective and GIN backbone architecture. Second best results are marked with underline.

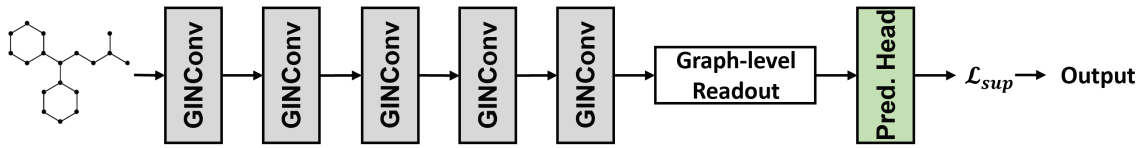
	Classification (ROC-AUC (%)) ↑						Avg.
Dataset	BBBP	Tox21	ToxCast	SIDER	ClinTox	BACE	
Vanilla tuning	68.0 ± 2.0	75.7 ± 0.7	63.9 ± 0.6	60.9 ± 0.6	65.9 ± 3.8	79.6 ± 1.2	69.0
Feature extractor	58.9 ± 0.4	68.7 ± 0.4	59.3 ± 0.2	59.9 ± 0.3	40.5 ± 2.7	61.6 ± 4.5	58.2
Last-k ($k = 3$)	68.1 ± 0.9	75.1 ± 0.4	64.0 ± 0.5	60.7 ± 0.5	63.9 ± 4.5	79.1 ± 1.2	68.5
Last-k ($k = 2$)	65.3 ± 1.0	74.5 ± 0.5	63.0 ± 0.7	61.6 ± 0.6	64.0 ± 3.5	80.6 ± 1.2	68.2
Last-k ($k = 1$)	64.6 ± 1.3	73.0 ± 0.5	61.4 ± 0.5	60.8 ± 0.5	66.1 ± 2.4	76.6 ± 0.8	67.1
Adapter ($m = 2$)	61.2 ± 0.4	71.4 ± 0.3	60.6 ± 0.1	59.6 ± 0.3	45.3 ± 3.1	71.1 ± 0.9	61.6
Adapter ($m = 4$)	62.5 ± 0.7	71.7 ± 0.3	60.6 ± 0.2	59.6 ± 0.3	47.4 ± 2.2	74.4 ± 1.8	62.7
Adapter ($m = 8$)	63.9 ± 0.7	71.8 ± 0.3	60.5 ± 0.4	59.9 ± 0.3	50.0 ± 1.0	76.8 ± 0.9	63.8
S2PGNN	**70.9 ± 1.3**	**76.3 ± 0.4**	**67.0 ± 0.5**	**62.8 ± 0.3**	**75.9 ± 2.2**	**82.6 ± 0.7**	**72.6**

weights to be close to pre-trained weights. DELTA imposes regularization on representations via the attention mechanism. BSS penalizes small eigenvalues of learned representations to suppress untransferable components. StochNorm regularizes on the encoder architecture in a dropout-like way. Please refer to Sec. II-B and Tab. II for more technical discussions.

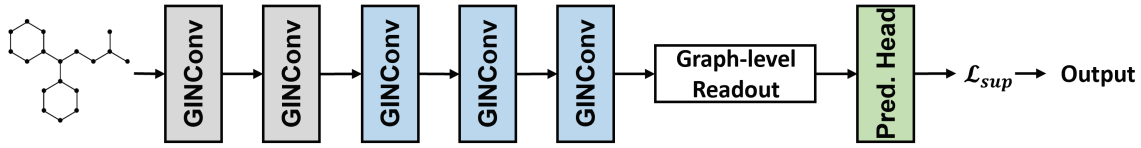
The comparisons on 6 classific MPP datasets have been summarized in Tab. VII. As shown in Tab. VII, we first observe the non-negligible improvement brought by S2PGNN compared with all baselines. Among baselines, regularized techniques (DELTA, BSS) from computer vision domain sometimes yield slightly better results than the vanilla fine-tuning strategy. However, in several cases (L^2-*SP*, StochNorm), the performance gains are not significant. This is probably due to their ignorance of characteristics in graph data (e.g., graph topology) and specific requirements in fine-tuning GNNs. By taking the graph topology into consideration, the method GTOT-Tuning, which is designed specifically to fine-tune GNNs, demonstrates higher performances than regularized variants extended from other domains. However, even the most competitive baseline GTOT-Tuning is still inferior to S2PGNN in 5 out of 6 datasets, indicating that the design dimensions proposed in S2PGNN (see Sec. III-B and Tab. II) may be indispensable and crucial designs towards more effective GNN fine-tuning. Furthermore, we note that various regularized strategies, such as GTOT-Tuning, is orthogonal to the proposed S2PGNN. Therefore, it may be promising to combine S2PGNN with other advanced regularized methods, i.e., combine an additional regularization term in S2PGNN's loss function Eq. (15) and (16) to further boost its performances, which we leave as future works.

2) Exploration of Other Fine-tuning Strategies: As mentioned in Sec. II-B and Remark 2, fine-tuning has been well explored in the domains beyond GNNs. Therefore, we try to investigate the performance of more fine-tuning strategies that are promising in other domains. However, we conclude that they may not be suitable for fine-tuning GNNs, thereby we discard them from our search space. Overall, we explore following fine-tuning strategies:

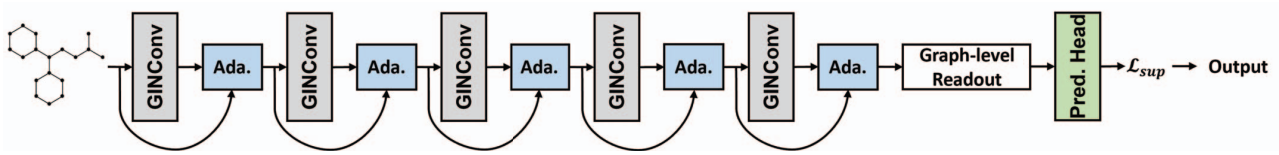
- Feature Extractor (FE): FE [69] proposes to reuse pre-trained model parameters to achieve better knowledge preservation. Once the pre-training is finished, all pre-trained layers are frozen and pre-trained model works as a pure feature extractor for downstream data. Only a small amount of additional parameters in specific prediction head are further tuned to perform downstream predictions.
- Last-k Tuning (LKT): With LKT [70], only parameters in last-k layers of the pre-trained models are further tuned, while the other initial layers are frozen and keep unchanged. LKT resides between ST and FE, and is popular in the computer version area [70]. In our experiments, we consider $k \in \{1, 2, 3\}$, which means we use around $20\% \sim 60\%$ tunable parameters of the original model (where $k = 5$)
- Adapter-Tuning (AT): AT [44] proposes to fine-tune only a small number of extra model parameters to attain the competitive performance, which is promising to achieve faster tuning and alleviate the over-fitting issue in natural language process and computer vision areas. More specifically, it adds small and task-specific neural network modules, called adapters, to pre-trained models. These adapters involve feature transformation $\mathbb{R}^d \to \mathbb{R}^m \to \mathbb{R}^d$ via the bottleneck architecture, where $m \ll d$ is to ensure parameter-efficient.



(a) An example of fine-tuning strategy with feature extractor.



(b) An example of fine-tuning strategy with last-k ($k = 3$) tunable layers.



(c) An example of fine-tuning strategy with adapter.

Fig. 4: Illustration to other fine-tuning strategies (refer to Fig. 3 for the legend).

Adapters are inserted between pre-trained layers and trained on the specific task, while the pre-trained layers remain fixed and unchanged to preserve the knowledge learned during pre-training. In our empirical explorations, we tailor the adapter design in [44] and consider adapter size $m \in \{2, 4, 8\}$ to use only around $1.3\% \sim 5.2\%$ tunable parameters of the original model (where $d = 300$).

The results on classific MPP tasks are summarized in Tab. VIII. Clearly, S2PGNN demonstrates consistent superior results than other approaches. Directly using pre-trained GNN as pure feature extractor (equivalent to $k = 0$) can reduce the total number of tunable parameters during fine-tuning, but it leads to the severe performance degradation on all 6 datasets. This indicates the fixed model may impede sufficient adaption when dealing with various downstream datasets. By gradually increasing tunable layers k ($1 \rightarrow 3$), the performance drop caused by insufficient adaption is mitigated. However, by tuning only partial model parameters, Last-k still yield inferior results than the vanilla strategy (equivalent to $k = 5$). Adapter method, although has demonstrated competitive fine-tuning capacity with the vanilla strategy when fine-tuning language models in natural language processing domain, fails to yield satisfactory results when fine-tuning GNNs for graph data.

To summarize, by investigating more fine-tuning strategies that are promising in other domains, we conclude that they may not be suitable for fine-tuning GNNs, which is probably due to the domain divergence. Instead, specific and innovative designs to fine-tune GNNs in graph domain is much more demanded. The performance comparison in Tab. VIII also indicates that the design dimensions in S2PGNN's search space may be more validated for searching the suitable fine-tuning strategy in pre-trained GNNs.

D. Ablation study on S2PGNN's design dimensions

To investigate S2PGNN's important design dimensions (see Sec. III-B) regarding GNN fine-tuning strategy: identity augmentation, multi-scale fusion, and adaptive graph-level readout, we further propose S2PGNN variants with degraded space and conduct ablation studies: S2PGNN-$\backslash id$ disables identity augmentation when aggregating messages from neighbors; S2PGNN-$\backslash fuse$ discards the multi-scale fusion and directly uses the last-layer output as learned node representations as most existing works does; S2PGNN-$\backslash read$ uses the simple and fixed mean pooling as [29] and follow-up works.

The man results of S2PGNN's variants are shown in Tab. IX. Significant performances drop are observed in each S2PGNN variant with degraded space, which provide empirical validation that the proposed design dimensions in S2PGNN are key factors that affect GNN fine-tuning results and should be incorporated during fine-tuning to achieve the optimal downstream results.

E. Effect of GNN backbone architectures

Recent works [96] have identified that GNN backbone is also crucial for GNN pre-training. Therefore, here we further provide additional results of S2PGNN when built on top of other classic GNN backbone architectures other than the GIN. Tab. X summarizes the performance of S2PGNN based on several pre-trained GNNs, including GCN, SAGE, and GAT models via the SSL strategy ContextPred [29]. We observe that pre-trained GNNs with all these backbone architectures can benefit from S2PGNN fine-tuning and achieve better performances than the vanilla strategy. This verifies that S2PGNN is agnostic to the base GNN architectures and is capable to achieve the consistent improvement.

TABLE IX: The performance comparison among S2PGNN's variants with degraded space.

	Classification (ROC-AUC (%)) ↑						Regression (RMSE) ↓		Avg.
Dataset	BBBP	Tox21	ToxCast	SIDER	ClinTox	BACE	ESOL	Lipo	**Drop**
S2PGNN-$\backslash id$	69.5 ± 2.0	75.9 ± 0.3	66.3 ± 0.4	61.3 ± 0.7	69.4 ± 6.3	79.5 ± 1.6	2.0 ± 0.3	0.9 ± 0.0	-5.2%
S2PGNN-$\backslash fuse$	69.0 ± 1.5	75.7 ± 0.6	65.7 ± 0.4	61.6 ± 0.7	61.6 ± 4.4	82.0 ± 1.0	2.5 ± 0.1	1.0 ± 0.0	-12.1%
S2PGNN-$\backslash read$	70.3 ± 1.6	75.2 ± 0.3	63.9 ± 0.3	62.2 ± 0.7	73.7 ± 4.4	80.3 ± 1.7	2.7 ± 0.0	1.0 ± 0.0	-12.3%
S2PGNN	**70.9 ± 1.3**	**76.3 ± 0.4**	**67.0 ± 0.5**	**62.8 ± 0.3**	**75.9 ± 2.2**	**82.6 ± 0.7**	**1.7 ± 0.2**	**0.9 ± 0.0**	-

TABLE X: The performance comparison between proposed S2PGNN fine-tuning and vanilla fine-tuning strategies with fixed ContextPred pre-training objective and other popular GNN backbone architectures.

	Classification (ROC-AUC (%)) ↑						Regression (RMSE) ↓		Avg.
Dataset	BBBP	Tox21	ToxCast	SIDER	ClinTox	BACE	ESOL	Lipo	**Gain**
ContextPred (GCN)	64.6 ± 2.2	73.0 ± 0.5	61.9 ± 1.1	56.8 ± 0.6	69.0 ± 1.3	79.9 ± 1.7	2.4 ± 0.1	**1.0 ± 0.0**	$+4.6\%$
ContextPred (GCN) + S2PGNN	**68.3 ± 1.0**	**75.7 ± 0.5**	**66.5 ± 0.3**	**62.3 ± 0.4**	**71.6 ± 1.3**	**81.5 ± 0.5**	**2.3 ± 0.1**	**1.0 ± 0.0**	
ContextPred (SAGE)	65.0 ± 3.0	74.7 ± 0.5	63.4 ± 0.2	**62.0 ± 0.6**	61.1 ± 3.1	78.8 ± 1.4	2.5 ± 0.1	**1.0 ± 0.0**	$+6.0\%$
ContextPred (SAGE) + S2PGNN	**69.0 ± 1.4**	**75.1 ± 0.4**	**66.4 ± 0.6**	61.6 ± 0.4	**67.1 ± 1.6**	**79.1 ± 0.7**	**2.0 ± 0.2**	**1.0 ± 0.0**	
ContextPred (GAT)	64.9 ± 1.2	69.6 ± 0.7	59.5 ± 0.8	52.5 ± 3.4	58.2 ± 6.9	60.5 ± 3.5	3.2 ± 0.1	1.1 ± 0.0	$+19.7\%$
ContextPred (GAT) + S2PGNN	**69.6 ± 0.9**	**75.0 ± 0.4**	**65.3 ± 0.3**	**61.8 ± 1.1**	**66.7 ± 3.2**	**80.4 ± 1.3**	**1.8 ± 0.1**	**1.0 ± 0.0**	

TABLE XI: The running time (seconds per epoch) of several fine-tuning strategies.

	Classification						Avg.
Dataset	BBBP	Tox21	ToxCast	SIDER	ClinTox	BACE	
Vanilla fine-tuning	5.2	14.0	11.3	3.3	2.7	6.8	7.2
L^2-*SP* [59]	5.3	23.8	27.3	5.3	6.8	4.3	12.1
DELTA [60]	5.7	11.4	11.1	5.8	3.2	5.0	7.0
BSS [61]	6.2	30.8	6.5	24.9	70.9	6.1	24.2
StochNorm [62].	4.5	17.8	31.1	3.3	3.3	3.8	10.6
GTOT-tuning [48]	5.7	22.7	34.1	4.6	3.0	11.3	13.6
S2PGNN	13.3	16.5	18.3	13.3	18.0	14.0	15.6

F. Efficiency

Apart from the effectiveness results provided in previous subsections, here we further report the concrete running time comparisons among several fine-tuning baselines. As shown in Tab. XI, we observe that the running time of S2PGNN is comparable with fine-tuning baselines, which eliminates the concern that the fine-tuning search method in S2PGNN may consume more computing resources to achieve improved perform ances. As demonstrated in Remark 3 and Tab. III, given that the space complexity in S2PGNN is $O(|\mathcal{O}_{conv}|^K \cdot |\mathcal{O}_{id}|^K \cdot |\mathcal{O}_{fuse}| \cdot |\mathcal{O}_{read}|)$ and the real space size built on the 5-layer GNN in our empirical study will have 10,206 candidate fine-tuning strategies, it further verifies that the proposed search algorithm in Sec. III-C is indeed powerful to tackle the effiency challenge (as mentioned in Sec. I) behind the fine-tuning search problem in this work.

V. Conclusion

In this paper, to fully unleash the potential of pre-trained GNNs on various downstream graph and bridge the missing gap between better fine-tuning strategies with pre-trained GNNs, we propose to search to fine-tune pre-trained GNNs for graph-level tasks, named S2PGNN. To achieve this, S2PGNN first investigates fine-tuning within and outside GNN area to identify key factors that affect GNN fine-tuning results and carefully present a novel search space that is suitable for GNN fine-tuning. To reduce the search cost from the large and discrete space, we incorporate an efficient search algorithm to suggest the parameter-sharing and continuous relaxation and solve the search problem by differentiable optimization. S2PGNN is model-agnostic and can be plugged into existing GNN backbone models and pre-trained GNNs. Empirical studies demonstrate that S2PGNN can consistently improve 10 classic pre-trained GNNs and achieve better performance than other fine-tuning works. Therefore, we expect S2PGNN to shed light on future directions towards better GNN fine-tuning innovation. For the future works, it may be worth trying to investigate how to derive a more robust and transferrable pre-trained GNNs that can be more easily adapted for various downstream graph scenarios.

Acknowledgment

Lei Chen's work is partially supported by National Science Foundation of China (NSFC) under Grant No. U22B2060, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Microsoft Research Asia Collaborative Research Grant and HKUST-Webank joint research lab grants. Xiaofang Zhou's work is supported by the JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust, HKUST-China Unicom Joint Lab on Smart Society, and HKUST-HKPC Joint Lab on Industrial AI and Robotics Research.

REFERENCES

[1] Z. Guo and H. Wang, "A deep graph neural network-based mechanism for social recommendations," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2776–2783, 2020.

[2] D. Shimin, Y. Quanming, Y. Zhang, and C. Lei, "Efficient relation-aware scoring function search for knowledge graph embedding," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 1104–1115, IEEE, 2021.

[3] S. Di and L. Chen, "Message function search for knowledge graph embedding," in *Proceedings of the ACM Web Conference 2023*, pp. 2633–2644, 2023.

[4] S. Di, Q. Yao, and L. Chen, "Searching to sparsify tensor decomposition for n-ary relational data," in *Proceedings of the Web Conference 2021*, pp. 4043–4054, 2021.

[5] M. Tsubaki, K. Tomii, and J. Sese, "Compound–protein interaction prediction with end-to-end learning of neural networks for graphs and sequences," *Bioinformatics*, vol. 35, no. 2, pp. 309–318, 2019.

[6] Z. Xiong, D. Wang, X. Liu, F. Zhong, X. Wan, X. Li, Z. Li, X. Luo, K. Chen, H. Jiang, *et al.*, "Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism," *Journal of medicinal chemistry*, vol. 63, no. 16, pp. 8749–8760, 2019.

[7] M. Sun, S. Zhao, C. Gilvary, O. Elemento, J. Zhou, and F. Wang, "Graph convolutional networks for computational drug development and discovery," *Briefings in bioinformatics*, vol. 21, no. 3, pp. 919–935, 2020.

[8] C. Lu, Q. Liu, C. Wang, Z. Huang, P. Lin, and L. He, "Molecular property prediction: A multilevel quantum interactions modeling perspective," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 1052–1060, 2019.

[9] H. Li, S. Di, Z. Li, L. Chen, and J. Cao, "Black-box adversarial attack and defense on graph neural networks," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pp. 1017–1030, IEEE, 2022.

[10] H. Li and L. Chen, "Early: Efficient and reliable graph neural network for dynamic graphs," *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1–28, 2023.

[11] F. Chen, Y.-C. Wang, B. Wang, and C.-C. J. Kuo, "Graph representation learning: a survey," *APSIPA Transactions on Signal and Information Processing*, vol. 9, p. e15, 2020.

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

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

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

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

[16] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International conference on machine learning*, pp. 5453–5462, PMLR, 2018.

[17] P. Li, Y. Wang, H. Wang, and J. Leskovec, "Distance encoding: Design provably more powerful neural networks for graph representation learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4465–4478, 2020.

[18] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized pagerank graph neural network," *arXiv preprint arXiv:2006.07988*, 2020.

[19] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," *Advances in neural information processing systems*, vol. 33, pp. 7793–7804, 2020.

[20] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*, pp. 1263–1272, PMLR, 2017.

[21] Z. Wang, S. Di, and L. Chen, "Autogel: An automated graph neural network with explicit link information," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24509–24522, 2021.

[22] Z. Wang, S. Di, and L. Chen, "A message passing neural network space for better capturing data-dependent receptive fields," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2489–2501, 2023.

[23] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[24] M. Zhang and Y. Chen, "Inductive matrix completion based on graph neural networks," *arXiv preprint arXiv:1904.12058*, 2019.

[25] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin, "Revisiting graph neural networks for link prediction," 2020.

[26] Y. Li, Y. Shen, L. Chen, and M. Yuan, "Zebra: When temporal graph neural networks meet temporal personalized pagerank," *Proceedings of the VLDB Endowment*, vol. 16, no. 6, pp. 1332–1345, 2023.

[27] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *Advances in neural information processing systems*, vol. 31, 2018.

[28] L. Wei, Z. He, H. Zhao, and Q. Yao, "Search to capture long-range dependency with stacking gnns for graph classification," *arXiv preprint arXiv:2302.08671*, 2023.

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

[30] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *Advances in neural information processing systems*, vol. 33, pp. 5812–5823, 2020.

[31] M. Xu, H. Wang, B. Ni, H. Guo, and J. Tang, "Self-supervised graph-level representation learning with local and global structure," in *International Conference on Machine Learning*, pp. 11548–11558, PMLR, 2021.

[32] Z. Zhang, Q. Liu, H. Wang, C. Lu, and C.-K. Lee, "Motif-based graph self-supervised learning for molecular property prediction," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15870–15882, 2021.

[33] S. Liu, H. Wang, W. Liu, J. Lasenby, H. Guo, and J. Tang, "Pre-training molecular graph representation with 3d geometry," *arXiv preprint arXiv:2110.07728*, 2021.

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

[35] Z. Hou, X. Liu, Y. Cen, Y. Dong, H. Yang, C. Wang, and J. Tang, "Graphmae: Self-supervised masked graph autoencoders," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 594–604, 2022.

[36] J. Xia, Y. Zhu, Y. Du, and S. Z. Li, "Pre-training graph neural networks for molecular representations: retrospect and prospect," in *ICML 2022 2nd AI for Science Workshop*, 2022.

[37] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 201–208, JMLR Workshop and Conference Proceedings, 2010.

[38] Y. Liu, M. Jin, S. Pan, C. Zhou, Y. Zheng, F. Xia, and S. Y. Philip, "Graph self-supervised learning: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 6, pp. 5879–5900, 2022.

[39] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun, "Gpt-gnn: Generative pre-training of graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1857–1867, 2020.

[40] J. Xia, C. Zhao, B. Hu, Z. Gao, C. Tan, Y. Liu, S. Li, and S. Z. Li, "Molebert: Rethinking pre-training graph neural networks for molecules," in *The Eleventh International Conference on Learning Representations*, 2022.

[41] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, "Gcc: Graph contrastive coding for graph neural network pre-training," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1150–1160, 2020.

[42] X. Han, Z. Huang, B. An, and J. Bai, "Adaptive transfer learning on graph neural networks," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 565–574, 2021.

[43] F. Radenović, G. Tolias, and O. Chum, "Fine-tuning cnn image retrieval with no human annotation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 7, pp. 1655–1668, 2018.

[44] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International Conference on Machine Learning*, pp. 2790–2799, PMLR, 2019.

[45] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[46] J. Xia, J. Zheng, C. Tan, G. Wang, and S. Z. Li, "Towards effective and generalizable fine-tuning for pre-trained molecular graph models," *bioRxiv*, pp. 2022–02, 2022.

[47] H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and T. Zhao, "Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization," *arXiv preprint arXiv:1911.03437*, 2019.

[48] J. Zhang, X. Xiao, L.-K. Huang, Y. Rong, and Y. Bian, "Fine-tuning graph neural networks via graph topology induced optimal transport," *arXiv preprint arXiv:2203.10453*, 2022.

[49] B. Ramsundar, P. Eastman, P. Walters, and V. Pande, *Deep learning for the life sciences: applying deep learning to genomics, microscopy, drug discovery, and more*. " O'Reilly Media, Inc.", 2019.

[50] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 338–348, 2020.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[52] C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò, "Towards sparse hierarchical graph classifiers," *arXiv preprint arXiv:1811.01287*, 2018.

[53] E. Ranjan, S. Sanyal, and P. Talukdar, "Asap: Adaptive structure aware pooling for learning hierarchical graph representations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5470–5477, 2020.

[54] J. Baek, M. Kang, and S. J. Hwang, "Accurate learning of graph representations with graph multiset pooling," *arXiv preprint arXiv:2102.11533*, 2021.

[55] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graph neural architecture search," in *International joint conference on artificial intelligence*, International Joint Conference on Artificial Intelligence, 2021.

[56] W. Zhang, Y. Shen, Z. Lin, Y. Li, X. Li, W. Ouyang, Y. Tao, Z. Yang, and B. Cui, "Pasca: A graph neural architecture search system under the scalable paradigm," in *Proceedings of the ACM Web Conference 2022*, pp. 1817–1828, 2022.

[57] B. M. Oloulade, J. Gao, J. Chen, T. Lyu, and R. Al-Sabri, "Graph neural architecture search: A survey," *Tsinghua Science and Technology*, vol. 27, no. 4, pp. 692–708, 2021.

[58] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[59] L. Xuhong, Y. Grandvalet, and F. Davoine, "Explicit inductive bias for transfer learning with convolutional networks," in *International Conference on Machine Learning*, pp. 2825–2834, PMLR, 2018.

[60] X. Li, H. Xiong, H. Wang, Y. Rao, L. Liu, Z. Chen, and J. Huan, "Delta: Deep learning transfer using feature map with attention for convolutional networks," *arXiv preprint arXiv:1901.09229*, 2019.

[61] X. Chen, S. Wang, B. Fu, M. Long, and J. Wang, "Catastrophic forgetting meets negative transfer: Batch spectral shrinkage for safe transfer learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[62] Z. Kou, K. You, M. Long, and J. Wang, "Stochastic normalization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16304–16314, 2020.

[63] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax.," *ICLR (Poster)*, vol. 2, no. 3, p. 4, 2019.

[64] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

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

[66] Z. Liu, X. Yu, Y. Fang, and X. Zhang, "Graphprompt: Unifying pre-training and downstream tasks for graph neural networks," in *Proceedings of the ACM Web Conference 2023*, pp. 417–428, 2023.

[67] X. Sun, H. Cheng, J. Li, B. Liu, and J. Guan, "All in one: Multi-task prompting for graph neural networks," 2023.

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

[69] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

[70] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *International conference on machine learning*, pp. 97–105, PMLR, 2015.

[71] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 3438–3445, 2020.

[72] G. Li, M. Muller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9267–9276, 2019.

[73] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.

[74] D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi, "Understanding pooling in graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[75] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *arXiv preprint arXiv:1511.06391*, 2015.

[76] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[77] D. Buterez, J. P. Janet, S. J. Kiddle, D. Oglic, and P. Liò, "Graph neural networks with adaptive readouts," *Advances in Neural Information Processing Systems*, vol. 35, pp. 19746–19758, 2022.

[78] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[79] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[80] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[81] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International conference on machine learning*, pp. 4095–4104, PMLR, 2018.

[82] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[83] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[84] T. Sterling and J. J. Irwin, "Zinc 15–ligand discovery for everyone," *Journal of chemical information and modeling*, vol. 55, no. 11, pp. 2324–2337, 2015.

[85] Y. Rong, Y. Bian, T. Xu, W. Xie, Y. Wei, W. Huang, and J. Huang, "Self-supervised graph transformer on large-scale molecular data," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12559–12571, 2020.

[86] I. F. Martins, A. L. Teixeira, L. Pinheiro, and A. O. Falcao, "A bayesian approach to in silico blood-brain barrier penetration modeling," *Journal of chemical information and modeling*, vol. 52, no. 6, pp. 1686–1697, 2012.

[87] "Tox21 data challenge 2014." https://tripod.nih.gov/tox21/challenge/.

[88] A. M. Richard, R. S. Judson, K. A. Houck, C. M. Grulke, P. Volarath, I. Thillainadarajah, C. Yang, J. Rathman, M. T. Martin, J. F. Wambaugh, *et al.*, "Toxcast chemical landscape: paving the road to 21st century toxicology," *Chemical research in toxicology*, vol. 29, no. 8, pp. 1225–1251, 2016.

[89] M. Kuhn, I. Letunic, L. J. Jensen, and P. Bork, "The sider database of drugs and side effects," *Nucleic acids research*, vol. 44, no. D1, pp. D1075–D1079, 2016.

[90] P. A. Novick, O. F. Ortiz, J. Poelman, A. Y. Abdulhay, and V. S. Pande, "Sweetlead: an in silico database of approved drugs, regulated chemicals, and herbal isolates for computer-aided drug discovery," *PloS one*, vol. 8, no. 11, p. e79568, 2013.

[91] G. Subramanian, B. Ramsundar, V. Pande, and R. A. Denny, "Computational modeling of β-secretase 1 (bace-1) inhibitors using ligand based

approaches," *Journal of chemical information and modeling*, vol. 56, no. 10, pp. 1936–1949, 2016.
[92] J. S. Delaney, "Esol: estimating aqueous solubility directly from molecular structure," *Journal of chemical information and computer sciences*, vol. 44, no. 3, pp. 1000–1005, 2004.
[93] A. Gaulton, L. J. Bellis, A. P. Bento, J. Chambers, M. Davies, A. Hersey, Y. Light, S. McGlinchey, D. Michalovich, B. Al-Lazikani, *et al.*, "Chembl: a large-scale bioactivity database for drug discovery," *Nucleic acids research*, vol. 40, no. D1, pp. D1100–D1107, 2012.
[94] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chemical science*, vol. 9, no. 2, pp. 513–530, 2018.
[95] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.
[96] R. Sun, H. Dai, and A. W. Yu, "Does gnn pretraining help molecular representation?," *Advances in Neural Information Processing Systems*, vol. 35, pp. 12096–12109, 2022.