Generalized Graph Prompt: Toward a Unification of Pre-Training and Downstream Tasks on Graphs

Xingtong Yu, Zhenghao Liu, Yuan Fang, *Senior Member, IEEE,* Zemin Liu, Sihong Chen, and Xinming Zhang, *Senior Member, IEEE,*

***Abstract*—Graphs can model complex relationships between objects, enabling a myriad of Web applications such as online page/article classification and social recommendation. While graph neural networks (GNNs) have emerged as a powerful tool for graph representation learning, in an end-to-end supervised setting, their performance heavily relies on a large amount of task-specific supervision. To reduce labeling requirement, the "pre-train, fine-tune" and "pre-train, prompt" paradigms have become increasingly common. In particular, prompting is a popular alternative to fine-tuning in natural language processing, which is designed to narrow the gap between pre-training and downstream objectives in a task-specific manner. However, existing study of prompting on graphs is still limited, lacking a universal treatment to appeal to different downstream tasks. In this paper, we propose GRAPHPROMPT, a novel pre-training and prompting framework on graphs. GRAPHPROMPT not only unifies pre-training and downstream tasks into a common task template, but also employs a learnable prompt to assist a downstream task in locating the most relevant knowledge from the pre-trained model in a task-specific manner. In particular, GRAPHPROMPT adopts simple yet effective designs in both pre-training and prompt tuning: During pre-training, a link prediction-based task is used to materialize the task template; during prompt tuning, a learnable prompt vector is applied to the READOUT layer of the graph encoder. To further enhance GRAPHPROMPT in these two stages, we extend it into GRAPHPROMPT+ with two major enhancements. First, we generalize a few popular graph pre-training tasks beyond simple link prediction to broaden the compatibility with our task template. Second, we propose a more generalized prompt design that incorporates a series of prompt vectors within every layer of the pre-trained graph encoder, in order to capitalize on the hierarchical information across different layers beyond just the readout layer. Finally, we conduct extensive experiments on five public datasets to evaluate and analyze GRAPHPROMPT and GRAPHPROMPT+.**

***Index Terms*—Graph mining, few-shot learning, meta-learning, pre-training, fine-tuning, prompting.**

Xingtong Yu is with the University of Science and Technology of China, Hefei, Anhui 230052, China, and also with Singapore Management University, Singapore 188065 (email: yxt95@mail.ustc.edu.cn). Work was done as a visiting student at Singapore Management University.

Zhenghao Liu is with the University of Science and Technology of China, Hefei, Anhui 230052, China (email: salzh@mail.ustc.edu.cn).

Yuan Fang is with Singapore Management University, Singapore 188065 (email: yfang@smu.edu.sg).

Zemin Liu is with the Zhejiang University, Hangzhou, Zhejiang 310058 (e-mail: liu.zemin@zju.edu.cn).

Sihong Chen is with the Tecent AI, China, Shenzhen, Guangdong 518063 (email: cshwhale@sina.com).

Xinming Zhang is with the University of Science and Technology of China, Hefei, Anhui 230052, China (email: xinming@ustc.edu.cn).

Corresponding author: Yuan Fang; Xinming Zhang.

Manuscript received April 19, 2021; revised August 16, 2021.

I. INTRODUCTION

The ubiquitous Web is becoming the ultimate data repository, capable of linking a broad spectrum of objects to form gigantic and complex graphs. The prevalence of graph data enables a series of downstream tasks for Web applications, ranging from online page/article classification to friend recommendation in social networks. Modern approaches for graph analysis generally resort to graph representation learning including graph embedding and graph neural networks (GNNs). Earlier graph embedding approaches [1], [2], [3] usually embed nodes on the graph into a low-dimensional space, in which the structural information such as the proximity between nodes can be captured [4]. More recently, GNNs [5], [6], [7], [8], [9] have emerged as the state of the art for graph representation learning. Their key idea boils down to a message-passing framework, in which each node derives its representation by receiving and aggregating messages from its neighboring nodes recursively [10].

Graph pre-training. Typically, GNNs work in an end-to-end manner, and their performance depends heavily on the availability of large-scale, task-specific labeled data as supervision. This supervised paradigm presents two problems. First, task-specific supervision is often difficult or costly to obtain. Second, to deal with a new task, the weights of GNN models need to be retrained from scratch, even if the task is on the same graph. To address these issues, pre-training GNNs [11], [12], [13], [14], [15], [16] has become increasingly popular, inspired by pre-training techniques in language and vision applications [17], [18], [19]. The pre-training of GNNs leverages self-supervised learning on more readily available label-free graphs (*i.e.*, graphs without task-specific labels), and learns intrinsic graph properties that intend to be general across tasks and graphs in a domain. In other words, the pre-training extracts a task-agnostic prior, and can be used to initialize model weights for a new task. Subsequently, the initial weights can be quickly updated through a lightweight fine-tuning step on a smaller number of task-specific labels.

However, the "pre-train, fine-tune" paradigm suffers from the problem of inconsistent objectives between pre-training and downstream tasks, resulting in suboptimal performance [20]. On one hand, the pre-training step aims to preserve various intrinsic graph properties such as node/edge features [12], [11], node connectivity/links [6], [11], [14], and local/global patterns [13], [12], [14]. On the other hand, the fine-tuning

step aims to reduce the task loss, *i.e.*, to fit the ground truth of the downstream task. To narrow the gap between pre-training and downstream tasks, prompting [21] has first been proposed for language models, which is a natural language instruction designed for a specific downstream task to elicit the semantic relevance between the task and the language model [22].

Research problem and challenges. To address the divergence between graph pre-training and various downstream tasks, in this paper we investigate the design of pre-training and prompting for graph neural networks. We aim for a unified design that can suit different downstream tasks flexibly. This problem is non-trivial due to the following two challenges.

Firstly, to enable effective knowledge transfer from the pre-training to a downstream task, it is desirable that the pre-training step preserves graph properties that are compatible with the given task. However, since different downstream tasks often have different objectives, *how do we unify pre-training with various downstream tasks on graphs*, so that a single pre-trained model can universally support different tasks? That is, we try to convert the pre-training task and downstream tasks to follow the same "template". Using pre-trained language models as an analogy, both their pre-training and downstream tasks can be formulated as masked language modeling.

Secondly, under the unification framework, it is still important to identify the distinction between different downstream tasks, in order to attain task-specific optima. For pre-trained language models, prompts in the form of natural language tokens or learnable word vectors have been designed to give different hints to different tasks, but it is less apparent what form prompts on graphs should take. Hence, *how do we design prompts on graphs*, so that they can guide different downstream tasks to effectively make use of the pre-trained model?

Present work: GRAPHPROMPT. To address these challenges, we propose a novel graph pre-training and prompting framework, called GRAPHPROMPT, aiming to unify the pre-training and downstream tasks for GNNs. Drawing inspiration from the prompting strategy for pre-trained language models, GRAPHPROMPT capitalizes on a unified template to define the objectives for both pre-training and downstream tasks, thus bridging their gap. We further equip GRAPHPROMPT with task-specific learnable prompts, which guides the downstream task to exploit relevant knowledge from the pre-trained GNN model. The unified approach endows GRAPHPROMPT with the ability of working on limited supervision such as few-shot learning tasks.

More specifically, to address the first challenge of unification, we focus on graph topology, which is a key enabler of graph models. In particular, subgraph is a universal structure that can be leveraged for both node- and graph-level tasks. At the node level, the information of a node can be enriched and represented by its contextual subgraph, *i.e.*, a subgraph where the node resides in [23], [24]; at the graph level, the information of a graph is naturally represented by the maximum subgraph (*i.e.*, the graph itself). Consequently, we unify both the node- and graph-level tasks, whether in pre-training or downstream, into the same template: the similarity calculation of (sub)graph. In this work, we adopt link prediction as the self-supervised pre-training task, given that links are readily available in any graph without additional annotation cost. Meanwhile, we focus on the popular node classification and graph classification as downstream tasks, which are node- and graph-level tasks, respectively. All these tasks can be cast as instances of learning subgraph similarity. On one hand, the link prediction task in pre-training boils down to the similarity between the contextual subgraphs of two nodes. On the other hand, the downstream node or graph classification task boils down to the similarity between the target instance (a node's contextual subgraph or the whole graph, resp.) and the class prototypical subgraphs constructed from labeled data. The unified template bridges the gap between the pre-training and different downstream tasks.

Toward the second challenge, we distinguish different downstream tasks by way of the READOUT operation on subgraphs. The READOUT operation is essentially an aggregation function to fuse node representations in the subgraph into a single subgraph representation. For instance, sum pooling, which sums the representations of all nodes in the subgraph, is a practical and popular scheme for READOUT. However, different downstream tasks can benefit from different aggregation schemes for their READOUT. In particular, node classification tends to focus on features that can contribute to the representation of the target node, while graph classification tends to focus on features associated with the graph class. Motivated by such differences, we propose a novel task-specific learnable prompt to guide the READOUT operation of each downstream task with an appropriate aggregation scheme. The learnable prompt serves as the parameters of the READOUT operation of downstream tasks, and thus enables different aggregation functions on the subgraphs of different tasks. Hence, GRAPHPROMPT not only unifies the pre-training and downstream tasks into the same template based on subgraph similarity, but also recognizes the differences between various downstream tasks to guide task-specific objectives.

Extension to GRAPHPROMPT+. Although GRAPHPROMPT bridges the gap between pre-training and downstream tasks, we further propose a more generalized extension called GRAPHPROMPT+ to enhance the versatility of the unification framework. Recall that in GRAPHPROMPT, during the pre-training stage, a simple link prediction-based task is employed that naturally fits the subgraph similarity-based task template. Concurrently, in the prompt-tuning stage, the model integrates a learnable prompt vector only in the READOUT layer of the pre-trained graph model. While these simple designs are effective, we take the opportunity to further raise two significant research questions.

First, toward a universal "pre-train, prompt" paradigm, it is essential to extend beyond a basic link prediction task in pre-training. While our proposed template can unify link prediction with typical downstream tasks on graph, researchers have proposed many other more advanced pre-training tasks on graphs, such as DGI [25] and GraphCL [26], which are able to capture more complex patterns from the pre-training graphs. Thus, to improve the compatibility of our framework with alternative

pre-training tasks, a natural question arises: *How can we unify a broader array of pre-training tasks within our framework?* In addressing this research question, we show that how a standard contrastive learning-based pre-training task on graphs can be generalized to fit our proposed task template, using a *generalized pre-training loss*. The generalization anchors on the core idea of subgraph similarity within our task template, while preserves the established sampling strategies of the original pre-training task. Hence, the uniqueness of each pre-training task is still retained while ensuring compatibility with our task template. We further give generalized variants of popular graph pre-training tasks including DGI [25], InfoGraph [27], GraphCL [26], GCC [13].

Second, graph neural networks [5], [6], [7], [28] typically employ a hierarchical architecture in which each layer learns distinct knowledge in a certain aspect or at some resolution. For instance, the initial layers primarily processes raw node features, thereby focusing on the intrinsic properties of individual nodes. However, as the number of layers in the graph encoder increases, the receptive field has been progressively enlarged to process more extensive neighborhood data, thereby shifting the focus toward subgraph or graph-level knowledge. Not surprisingly, different downstream tasks may prioritize the knowledge encoded at different layers of the graph encoder. Consequently, it is important to generalize our prompt design to leverage the layer-wise hierarchical knowledge from the pre-trained encoder, beyond just the READOUT layer in GRAPHPROMPT. Thus, *how do we design prompts that can adapt diverse downstream tasks to the hierarchical knowledge within multiple layers of the pre-trained graph encoder?* To solve this question, we extend GRAPHPROMPT with a generalized prompt design. More specifically, instead of a single prompt vector applied to the READOUT layer, we propose *layer-wise prompts*, a series of learnable prompts that are integrated into each layer of the graph encoder in parallel. The series of prompts modifies all layers of the graph encoder (including the input and hidden layers), so that each prompt vector is able to locate the layer-specific knowledge most relevant to a downstream task, such as node-level characteristics, and local or global structural patterns.

Contributions. To summarize, our contributions are four-fold. (1) We recognize the gap between graph pre-training and downstream tasks, and propose a unification framework GRAPHPROMPT and its extension GRAPHPROMPT+based on subgraph similarity for both pre-training and downstream tasks, including both node and graph classification tasks. (2) We propose a novel prompting strategy for GRAPHPROMPT, hinging on a learnable prompt to actively guide downstream tasks using task-specific aggregation in the READOUT layer, in order to drive the downstream tasks to exploit the pre-trained model in a task-specific manner. (3) We extend GRAPHPROMPT to GRAPHPROMPT+, which further unifies existing popular graph pre-training tasks for compatibility with our task template, and generalizes the prompt design to capture the hierarchical knowledge within each layer of the pre-trained graph encoder. (4) We conduct extensive experiments on five public datasets, and the results demonstrate the superior performance of GRAPHPROMPT and GRAPHPROMPT+in comparison to the state-of-the-art approaches.

A preliminary version of this manuscript has been published as a conference paper in The ACM Web Conference 2023 [29]. We highlight the major changes as follows. (1) Introduction: We reorganized Section I to highlight the motivation, challenges, and insights for the extension from GRAPHPROMPT to GRAPHPROMPT+. (2) Methodology: We proposed the extension GRAPHPROMPT+ to allow more general pre-training tasks and prompt tuning in Section V. The proposed GRAPHPROMPT+ not only broadens the scope of our task template to accommodate any standard contrastive pre-training task on graphs, but also enhances the extraction of hierarchical knowledge from the pre-trained graph encoder with layer-wise prompts. (3) Experiments: We conducted additional experiments to evaluate the extended framework GRAPHPROMPT+ in Section VII, which demonstrate significant improvements over GRAPHPROMPT.

II. RELATED WORK

Graph pre-training. Inspired by the application of pre-training models in language [17], [30] and vision [31], [18] domains, graph pre-training [32] emerges as a powerful paradigm that leverages self-supervision on label-free graphs to learn intrinsic graph properties. While the pre-training learns a task-agnostic prior, a relatively light-weight fine-tuning step is further employed to update the pre-trained weights to fit a given downstream task. Different pre-training approaches design different self-supervised tasks based on various graph properties such as node features [12], [11], links [33], [6], [11], [14], local or global patterns [13], [12], [14], local-global consistency [25], [27], and their combinations [26], [34], [35].

However, the above approaches do not consider the gap between pre-training and downstream objectives, which limits their generalization ability to handle different tasks. Some recent studies recognize the importance of narrowing this gap. L2P-GNN [14] capitalizes on meta-learning [36] to simulate the fine-tuning step during pre-training. However, since the downstream tasks can still differ from the simulation task, the problem is not fundamentally addressed.

Prompt-based learning. In other fields, as an alternative to fine-tuning, researchers turn to prompting [21], in which a task-specific prompt is used to cue the downstream tasks. Prompts can be either handcrafted [21] or learnable [37], [38]. On graph data, the study of prompting is still limited [39]. One recent work called GPPT [40] capitalizes on a sophisticated design of learnable prompts on graphs, but it only works with node classification, lacking an unification effort to accommodate other downstream tasks like graph classification. VNT [41] utilizes a pre-trained graph transformer as graph encoder and introduces virtual nodes as soft prompts within the embedding space. However, same as GPPT, its application is only focused on the node classification task. On the other hand, ProG [42] and SGL-PT [43] both utilize a specific pre-training method for prompt-based learning on graphs across multiple downstream tasks. Additionally, HGPrompt [44] extends GraphPrompt to address heterogeneous graph

learning, and MultiGPrompt [45] proposes a multi-task pre-training and prompting framework. Despite this, they neither explore the unification of different pre-training tasks, nor exploit the hierarchical knowledge inherent in graph encoder. Besides, there is a model also named as GraphPrompt [46], but it considers an NLP task (biomedical entity normalization) on text data, where graph is only auxiliary. It employs the standard text prompt unified by masked language modeling, assisted by a relational graph to generate text templates, which is distinct from our work.

Comparison to other settings. Our few-shot setting is different from other paradigms that also deal with label scarcity, including semi-supervised learning [5] and meta-learning [36]. In particular, semi-supervised learning cannot cope with novel classes not seen in training, while meta-learning requires a large volume of labeled data in their base classes for a meta-training phase, before they can handle few-shot tasks in testing.

III. Preliminaries

In this section, we give the problem definition and introduce the background of GNNs.

A. Problem Definition

Graph. A graph can be defined as $G = (V, E)$, where V is the set of nodes and E is the set of edges. Equivalently, the graph can be represented by an adjacency matrix $\mathbf{A}$, such as $\mathbf{A}_{ij} = 1$ iff $(v_i, v_j) \in E$, for any $v_i, v_j \in V$. We also assume an input feature matrix of the nodes, $\mathbf{X} \in \mathbb{R}^{|V| \times d}$, is available. Let $\mathbf{x}_i \in \mathbb{R}^d$ denote the feature vector of node $v_i \in V$. In addition, we denote a set of graphs as $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$.

Problem. In this paper, we investigate the problem of graph pre-training and prompting. For the downstream tasks, we consider the popular node classification and graph classification tasks. For node classification on a graph $G = (V, E)$, let C be the set of node classes with $\ell_i \in C$ denoting the class label of node $v_i \in V$. For graph classification on a set of graphs $\mathcal{G}$, let $\mathcal{C}$ be the set of graph labels with $L_i \in \mathcal{C}$ denoting the class label of graph $G_i \in \mathcal{G}$.

In particular, the downstream tasks are given limited supervision in a few-shot setting: for each class in the two tasks, only k labeled samples (*i.e.*, nodes or graphs) are provided, known as k-shot classification.

B. Graph Neural Networks

The success of GNNs boils down to the message-passing mechanism [10], in which each node receives and aggregates messages (*i.e.*, features or embeddings) from its neighboring nodes to generate its own representation. This operation of neighborhood aggregation can be stacked in multiple layers to enable recursive message passing. In the l-th GNN layer, the embedding of node v, denoted by $\mathbf{h}_v^l$, is calculated based on the embeddings in the previous layer, as follows:

$$\mathbf{h}_v^l = \textsc{Aggr}(\mathbf{h}_v^{l-1}, \{\mathbf{h}_u^{l-1} : u \in \mathcal{N}_v\}; \theta^l), \tag{1}$$

where $\mathcal{N}_v$ is the set of neighboring nodes of v, θ^l is the learnable GNN parameters in layer l. $\textsc{Aggr}(\cdot)$ is the neighborhood aggregation function and can take various forms [6], [7], [28] Note that in the first layer, the input node embedding $\mathbf{h}_v^0$ can be initialized as the node features in $\mathbf{X}$. The total learnable GNN parameters can be denoted as $\Theta = \{\theta^1, \theta^2, \ldots\}$. For brevity, we simply denote the output node representations of the last layer as $\mathbf{h}_v$.

For brevity of notation, GNNs can also be described in a alternative, matrix-based format. Consider the embedding matrix at the l-th layer, denoted as $\mathbf{H}^l$, in which each row, $\mathbf{h}_i^l$, denotes the embedding vector of node v_i. The embedding matrix at the l-th layer is calculated based on the embedding matrix from the previous, *i.e.*, $(l-1)$-th, layer:

$$\mathbf{H}^l = \textsc{Aggr}(\mathbf{H}^{l-1}, \mathbf{A}; \theta^l). \tag{2}$$

The initial embedding matrix $\mathbf{H}^0$ is set to be the same as the input feature matrix, *i.e.*, $\mathbf{H}^0 = \mathbf{X}$. After encoding through all the GNN layers, we simply denote the output embedding matrix as $\mathbf{H}$. For easy of reference, we further abstract the multi-layer encoding process as follows.

$$\mathbf{H} = \textsc{GraphEncoder}(\mathbf{X}, \mathbf{A}; \Theta). \tag{3}$$

IV. Proposed Approach: GraphPrompt

In this section, we present GraphPrompt, starting with a unification framework for common graph tasks. Then, we introduce the pre-training and downstream phases.

A. Unification Framework

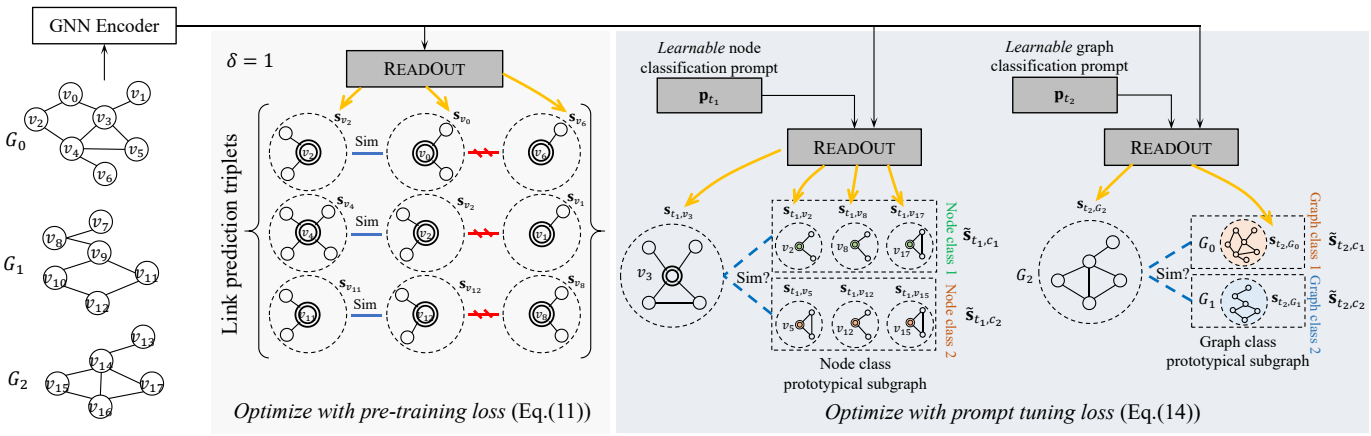
We first introduce the overall framework of GraphPrompt in Fig. 1. Our framework is deployed on a set of label-free graphs shown in Fig. 1(a), for pre-training in Fig. 1(b). The pre-training adopts a link prediction task, which is self-supervised without requiring extra annotation. Afterward, in Fig. 1(c), we capitalize on a learnable prompt to guide each downstream task, namely, node classification or graph classification, for task-specific exploitation of the pre-trained model. We explain how the framework supports a unified view of pre-training and downstream tasks below.

Instances as subgraphs. The key to the unification of pre-training and downstream tasks lies in finding a common template for the tasks. The task-specific prompt can then be further fused with the template of each downstream task, to distinguish the varying characteristics of different tasks.

In comparison to other fields such as visual and language processing, graph learning is uniquely characterized by the exploitation of graph topology. In particular, subgraph is a universal structure capable of expressing both node- and graph-level instances. On one hand, at the node level, every node resides in a local neighborhood, which in turn contextualizes the node [47], [48], [49]. The local neighborhood of a node v on a graph $G = (V, E)$ is usually defined by a contextual subgraph $S_v = (V(S_v), E(S_v))$, where its set of nodes and edges are respectively given by

$$V(S_v) = \{d(u, v) \leq \delta \mid u \in V\}, \text{ and} \tag{4}$$

$$E(S_v) = \{(u, u') \in E \mid u \in V(S_v), u' \in V(S_v)\}, \tag{5}$$



(a) Toy graphs (b) Pre-training (c) Prompting for node classification (left) or graph classification (right)

Fig. 1: Overall framework of GRAPHPROMPT.

where $d(u, v)$ gives the shortest distance between nodes u and v on the graph G, and δ is a predetermined threshold. That is, S_v consists of nodes within δ hops from the node v, and the edges between those nodes. Thus, the contextual subgraph S_v embodies not only the self-information of the node v, but also rich contextual information to complement the self-information [23], [24]. On the other hand, at the graph level, the maximum subgraph of a graph G, denoted S_G, is the graph itself, *i.e.*, $S_G = G$. The maximum subgraph S_G spontaneously embodies all information of G. In summary, subgraphs can be used to represent both node- and graph-level instances: Given an instance x which can either be a node or a graph (*e.g.*, $x = v$ or $x = G$), the subgraph S_x offers a unified access to the information associated with x.

Unified task template. Based on the above subgraph definitions for both node- and graph-level instances, we are ready to unify different tasks to follow a common template. Specifically, the link prediction task in pre-training and the downstream node and graph classification tasks can all be redefined as *subgraph similarity learning*. Let $\mathbf{s}_x$ be the vector representation of the subgraph S_x, and $\text{sim}(\cdot, \cdot)$ be the cosine similarity function. As illustrated in Figs. 1(b) and (c), the three tasks can be mapped to the computation of subgraph similarity, which is formalized below.

- ***Link prediction***: This is a node-level task. Given a graph $G = (V, E)$ and a triplet of nodes (v, a, b) such that $(v, a) \in E$ and $(v, b) \notin E$, we shall have

$$\text{sim}(\mathbf{s}_v, \mathbf{s}_a) > \text{sim}(\mathbf{s}_v, \mathbf{s}_b). \tag{6}$$

  Intuitively, the contextual subgraph of v shall be more similar to that of a node linked to v than that of another unlinked node.
- ***Node classification***: This is also a node-level task. Consider a graph $G = (V, E)$ with a set of node classes C, and a set of labeled nodes $D = \{(v_1, \ell_1), (v_2, \ell_2), \ldots\}$ where $v_i \in V$ and ℓ_i is the corresponding label of v_i. As we adopt a k-shot setting, there are exactly k pairs of $(v_i, \ell_i = c) \in D$ for every class $c \in C$. For each class $c \in C$, further define a *node class prototypical subgraph* represented by a vector $\tilde{\mathbf{s}}_c$, given by

$$\tilde{\mathbf{s}}_c = \frac{1}{k} \sum_{(v_i, \ell_i) \in D, \ell_i = c} \mathbf{s}_{v_i}. \tag{7}$$

  Note that the class prototypical subgraph is a "virtual" subgraph with a latent representation in the same embedding space as the contextual subgraphs. It is simply constructed as the mean representation of the contextual subgraphs of labeled nodes in a given class. Then, given a node v_j not in the labeled set D, its class label ℓ_j shall be

$$\ell_j = \arg\max_{c \in C} \text{sim}(\mathbf{s}_{v_j}, \tilde{\mathbf{s}}_c). \tag{8}$$

  Intuitively, a node shall belong to the class whose prototypical subgraph is the most similar to its contextual subgraph.
- ***Graph classification***: This is a graph-level task. Consider a set of graphs $\mathcal{G}$ with a set of graph classes $\mathcal{C}$, and a set of labeled graphs $\mathcal{D} = \{(G_1, L_1), (G_2, L_2), \ldots\}$ where $G_i \in \mathcal{G}$ and L_i is the corresponding label of G_i. In the k-shot setting, there are exactly k pairs of $(G_i, L_i = c) \in \mathcal{D}$ for every class $c \in \mathcal{C}$. Similar to node classification, for each class $c \in \mathcal{C}$, we define a *graph class prototypical subgraph*, also represented by the mean embedding vector of the (sub)graphs in c:

$$\tilde{\mathbf{s}}_c = \frac{1}{k} \sum_{(G_i, L_i) \in \mathcal{D}, L_i = c} \mathbf{s}_{G_i}. \tag{9}$$

  Then, given a graph G_j not in the labeled set $\mathcal{D}$, its class label L_j shall be

$$L_j = \arg\max_{c \in \mathcal{C}} \text{sim}(\mathbf{s}_{G_j}, \tilde{\mathbf{s}}_c). \tag{10}$$

  Intuitively, a graph shall belong to the class whose prototypical subgraph is the most similar to itself. ■

It is worth noting that node and graph classification can be further condensed into a single set of notations. Let (x, y) be an annotated instance of graph data, *i.e.*, x is either a node or a graph, and $y \in Y$ is the class label of x among the set of classes Y. Then,

$$y = \arg\max_{c \in Y} \text{sim}(\mathbf{s}_x, \tilde{\mathbf{s}}_c). \tag{11}$$

Finally, to materialize the common task template, we discuss how to learn the subgraph embedding vector $\mathbf{s}_x$ for the subgraph S_x. Given node representations $\mathbf{h}_v$ generated by a GNN (see Sect. III-B), a standard approach of computing $\mathbf{s}_x$ is to employ a READOUT operation that aggregates the representations of nodes in the subgraph S_x. That is,

$$\mathbf{s}_x = \text{READOUT}(\{\mathbf{h}_v : v \in V(S_x)\}). \tag{12}$$

The choice of the aggregation scheme for READOUT is flexible, including sum pooling and more advanced techniques [50], [28]. In our work, we simply use sum pooling.

In summary, the unification framework is enabled by the common task template of subgraph similarity learning, which lays the foundation of our pre-training and prompting strategies as we will introduce in the following parts.

B. Pre-Training Phase

As discussed earlier, our pre-training phase employs the link prediction task. Using link prediction/generation is a popular and natural way [6], [11], [51], [14], as a vast number of links are readily available on large-scale graph data without extra annotation. In other words, the link prediction objective can be optimized on label-free graphs, such as those shown in Fig. 1(a), in a self-supervised manner.

Based on the common template defined in Sect. IV-A, the link prediction task is anchored on the similarity of the contextual subgraphs of two candidate nodes. Generally, the subgraphs of two positive (*i.e.*, linked) candidates shall be more similar than those of negative (*i.e.*, non-linked) candidates, as illustrated in Fig. 1(b). Subsequently, the pre-trained prior on subgraph similarity can be naturally transferred to node classification downstream, which shares a similar intuition: The subgraphs of nodes in the same class shall be more similar than those of nodes from different classes. On the other hand, the prior can also support graph classification downstream, as graph similarity is consistent with subgraph similarity not only in letter (as a graph is technically always a subgraph of itself), but also in spirit. The "spirit" here refers to the tendency that graphs sharing similar subgraphs are likely to be similar themselves, which means graph similarity can be translated into the similarity of the containing subgraphs [52], [53], [54].

Formally, given a node v on graph G, we randomly sample one positive node a from v's neighbors, and a negative node b from the graph that does not link to v, forming a triplet (v, a, b). Our objective is to increase the similarity between the contextual subgraphs S_v and S_a, while decreasing that between S_v and S_b. More generally, on a set of label-free graphs $\mathcal{G}$, we sample a number of triplets from each graph to construct an overall training set $\mathcal{T}_{\text{pre}}$. Then, we define the following pre-training loss.

$$\mathcal{L}_{\text{pre}}(\Theta) = -\sum_{(v,a,b)\in\mathcal{T}_{\text{pre}}} \ln \frac{\exp(\text{sim}(\mathbf{s}_v, \mathbf{s}_a)/\tau)}{\sum_{u\in\{a,b\}} \exp(\text{sim}(\mathbf{s}_v, \mathbf{s}_u)/\tau)}, \tag{13}$$

where τ is a temperature hyperparameter to control the shape of the output distribution. Note that the loss is parameterized by Θ, which represents the GNN model weights.

The output of the pre-training phase is the optimal model parameters $\Theta_0 = \arg\min_\Theta \mathcal{L}_{\text{pre}}(\Theta)$. Θ_0 can be used to initialize the GNN weights for downstream tasks, thus enabling the transfer of prior knowledge downstream.

C. Prompting for Downstream Tasks

The unification of pre-training and downstream tasks enables more effective knowledge transfer as the tasks in the two phases are made more compatible by following a common template. However, it is still important to distinguish different downstream tasks, in order to capture task individuality and achieve task-specific optimum.

To cope with this challenge, we propose a novel task-specific learnable prompt on graphs, inspired by prompting in natural language processing [21]. In language contexts, a prompt is initially a handcrafted instruction to guide the downstream task, which provides task-specific cues to extract relevant prior knowledge through a unified task template (typically, pre-training and downstream tasks are all mapped to masked language modeling). More recently, learnable prompts [37], [38] have been proposed as an alternative to handcrafted prompts, to alleviate the high engineering cost of the latter.

Prompt design. Nevertheless, our proposal is distinctive from language-based prompting for two reasons. Firstly, we have a different task template from masked language modeling. Secondly, since our prompts are designed for graph structures, they are more abstract and cannot take the form of language-based instructions. Thus, they are virtually impossible to be handcrafted. Instead, they should be topology related to align with the core of graph learning. In particular, under the same task template of subgraph similarity learning, the READOUT operation (used to generate the subgraph representation) can be "prompted" differently for different downstream tasks. Intuitively, different tasks can benefit from different aggregation schemes for their READOUT. For instance, node classification pays more attention to features that are topically more relevant to the target node. In contrast, graph classification tends to focus on features that are correlated to the graph class. Moreover, the important features may also vary given different sets of instances or classes in a task.

Let $\mathbf{p}_t$ denote a learnable *prompt vector* for a downstream task t, as shown in Fig. 1(c). The prompt-assisted READOUT operation on a subgraph S_x for task t is

$$\mathbf{s}_{t,x} = \text{READOUT}(\{\mathbf{p}_t \odot \mathbf{h}_v : v \in V(S_x)\}), \tag{14}$$

where $\mathbf{s}_{t,x}$ is the task t-specific subgraph representation, and $\odot$ denotes the element-wise multiplication. That is, we perform a *feature weighted* summation of the node representations from the subgraph, where the prompt vector $\mathbf{p}_t$ is a dimension-wise reweighting in order to extract the most relevant prior knowledge for the task t.

Prompt tuning. To optimize the learnable prompt, also known as *prompt tuning*, we formulate the loss based on the common template of subgraph similarity, using the prompt-assisted task-specific subgraph representations.

Formally, consider a task t with a labeled training set $\mathcal{T}_t = \{(x_1, y_1), (x_2, y_2), \ldots\}$, where x_i is an instance (*i.e.*, a node

Algorithm 1 PROMPT TUNING FOR GRAPHPROMPT

Input: Graph encoder with pre-trained parameters Θ_0, a set of n downstream tasks $\{t_1, \ldots, t_n\}$.
Output: Optimized prompt vectors $\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$.

```
1: for i ← 1 to n do
2:     /* Encoding graphs with pre-trained weights */
3:     for each graph G = (X, A) in task t_i do
4:         H ← GRAPHENCODER(X, A; Θ_0)
5:         Let h_v ← H[v], where v is a node in G
6:     p_i ← initialization
7:     while not converged do
8:         /* Prompt-assisted readout by Eq. (14) */
9:         for each instance x in task t_i do
10:            s_{t_i,x} ← READOUT({p_i ⊙ h_v : v ∈ V(S_x)})
11:        /* Update prototypical subgraphs */
12:        for each class c in task t_i do
13:            s̃_{t_i,c} ← MEAN(s_{t_i,x} : x belongs to class c)
14:        /* Optimizing the prompt vector */
15:        Calculate L_prompt by Eq. (15)
16:        Update p_i by backpropagating L_prompt
17: return {p_1, ..., p_n}.
```

or a graph), and $y_i \in Y$ is the class label of x_i among the set of classes Y. The loss for prompt tuning is defined as

$$\mathcal{L}_{\text{prompt}}(\mathbf{p}_t) = -\sum_{(x_i,y_i)\in\mathcal{T}_t} \ln \frac{\exp(\text{sim}(\mathbf{s}_{t,x_i}, \tilde{\mathbf{s}}_{t,y_i})/\tau)}{\sum_{c\in Y}\exp(\text{sim}(\mathbf{s}_{t,x_i}, \tilde{\mathbf{s}}_{t,c})/\tau)}, \quad (15)$$

where the class prototypical subgraph for class c is represented by $\tilde{\mathbf{s}}_{t,c}$, which is also generated by the prompt-assisted, task-specific READOUT.

Note that, the prompt tuning loss is only parameterized by the learnable prompt vector $\mathbf{p}_t$, without the GNN weights. Instead, the pre-trained GNN weights Θ_0 are frozen for downstream tasks, as no fine-tuning is necessary. This significantly decreases the number of parameters to be updated downstream, thus not only improving the computational efficiency of task learning and inference, but also reducing the reliance on labeled data.

We outline the main steps for prompt tuning in Algorithm 1. In short, we iterate through each downstream task to learn their prompt vectors one by one. In lines 3–5, we obtain the embedding of each node using the pre-trained graph encoder, where the pre-trained weights Θ_0 will be frozen throughout the prompt-tuning process. In lines 8–16, we optimize the prompt vectors. More specifically, we perform prompt-assisted readout (lines 10–11), and update the embeddings of the prototypical subgraphs based on the few-shot labeled data given in the task (lines 12–13). Note that the steps of updating prototypical subgraphs are only required for classification tasks.

V. EXTENSION TO GRAPHPROMPT+

Next, we generalize the pre-training tasks and prompts in GRAPHPROMPT, extending it to GRAPHPROMPT+.

A. Generalizing Pre-Training Tasks

Although our proposed task template in GRAPHPROMPT easily accommodate the link prediction task, which is a simple, effective and popular pre-training task on graphs, it is not immediately apparent how the task template can fit other more advanced pre-training tasks on graphs. This presents a significant limitation in GRAPHPROMPT: Our "pre-train, prompt" framework is confined to only one specific pre-training task, thereby precluding other pre-training tasks that can potentially learn more comprehensive knowledge. To unify a broader range of pre-training tasks in GRAPHPROMPT+, we first show that any standard contrastive graph pre-training task can be generalized to leverage subgraph similarity, the pivotal element in our task template. Meanwhile, each contrastive task can still preserve its unique sampling approach for the positive and negative pairs. Next, we illustrate the generalization process using several mainstream contrastive pre-training tasks.

Unification of graph contrastive tasks. We formulate a general loss term for any standard contrastive pre-training task on graphs. The generalized loss is compatible with our proposed task template, *i.e.*, it is based on the core idea of subgraph similarity. The rationale for this generalization centers around two reasons.

First, the key idea of contrastive learning boils down to bringing positively related instances closer, while pushing away negatively related ones in their latent space [55]. To encapsulate this objective in a generalized loss, the definition of instances needs to be unified for diverse contrastive tasks on graphs, so that the distance or proximity between the instances can also be standardized. Not surprisingly, *subgraphs* can serve as a unified definition for various types of instances on graphs, including nodes [25], [26], subgraphs [14], [56] or the whole graphs [27], [56], which are common forms of instance in contrastive learning on graphs. By treating these instances as subgraphs, the contrastive objective can be accomplished by calculating a similarity score between the subgraphs, so as to maximize the similarity between a positive pair of subgraphs whilst minimizing that between a negative pair.

Second, the general loss term should flexibly preserve the unique characteristics of different contrastive tasks, thereby enabling the capture of diverse forms of knowledge through pre-training. Specifically, various contrastive approaches diverge in the sampling or generation of positive and negative pairs of instances. Consequently, in our generalized loss, the set of positive or negative pairs can be materialized differently to accommodate the requirements of each contrastive objective.

Given the above considerations, we first define a standard contrastive graph pre-training task. Consider a set of pre-training data $\mathcal{T}_{\text{pre}}$, which consists of the target instances (or elements that can derive the target instances). For each target instance $o \in \mathcal{T}_{\text{pre}}$, a contrastive task samples or constructs a set of positive instances Pos_o, as well as a set of negative instances Neg_o, both w.r.t. o. Hence, $\{(o, a) : a \in Pos_o\}$ is the set of positive pairs involving the target instance o, such that the objective is to maximize the similarity between each pair of o and a. On the other hand, $\{(o, b) : b \in Neg_o\}$ is the set of negative pairs, such that the objective is to minimize the similarity between each pair of o and b.

Then, we propose the following generalized loss that can be used for any standard contrastive task.

$$\mathcal{L}(\Theta) = -\sum_{o\in\mathcal{T}_{\text{pre}}} \ln \frac{\sum_{a\in Pos_o}\exp(\text{sim}(\mathbf{s}_a, \mathbf{s}_o)/\tau)}{\sum_{b\in Neg_o}\exp(\text{sim}(\mathbf{s}_b, \mathbf{s}_o)/\tau)}, \quad (16)$$

TABLE I: Materializing the target, positive and negative instances for mainstream contrastive graph pre-training approaches for the generalized loss in Eq. (16). We also include link prediction (LP), which is used by GRAPHPROMPT.

	Target instance o	Positive instance a	Negative instance b	Loss
LP [29]	a node v	a node linked to v	a node not linked to v	Eq. (13)
DGI [25]	a graph G	a node in G	a node in G', a corrupted graph of G	Eq. (17)
InfoGraph [27]	a graph G	a node in G	a node in $G' \neq G$	Eq. (18)
GraphCL [26]	an augmented graph G_i from a graph G by strategy i	an augmented graph G_j from a graph G by strategy j	an augmented graph G'_j from a graph $G' \neq G$ by strategy j	Eq. (19)
GCC [13]	a random walk induced subgraph G_v^r from a node v's r-egonet	a random walk induced subgraph $\tilde{G}_v^r \neq G_v^r$ from v's r-egonet	a random walk induced subgraph $G_v^{r'}$ from v's r'-egonet, $r' \neq r$	Eq. (20)

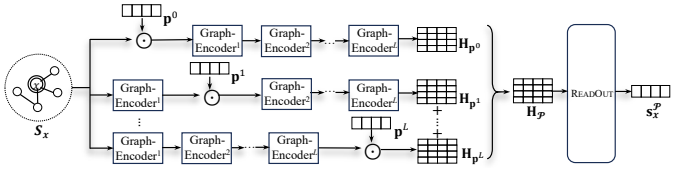


Fig. 2: Layer-wise prompt tuning for a pre-trained graph encoder with L layers. GRAPHENCODER^l represents the l-th layer of the graph encoder, and $\mathbf{p}^l$ represents the prompt vector that modifies the l-th layer.

where $\mathbf{s}_o$ denotes the subgraph embedding of the target instance, $\mathbf{s}_a, \mathbf{s}_b$ are the subgraph embedding of positive and negative instances, respectively, and τ is a temperature hyper-parameter.

Using this generalized loss, GRAPHPROMPT+ broadens the scope of our proposed task template that is based on subgraph similarity, which can serve as a more general framework than GRAPHPROMPT to unify standard contrastive pre-training tasks on graphs beyond link prediction.

Template for mainstream contrastive tasks. We further illustrate how the generalized loss term can be materialized for several mainstream contrastive graph pre-training approaches, in order to fit the task template. In Table I, we summarize how various definitions of instances in these mainstream approaches are materialized, as also detailed below. Note that the strategies for sampling positive and negative instances are preserved as originally established.

DGI [25] operates on the principle of mutual information maximization, where the primary objective is to maximize the consistency between the local node representations and the global graph representation. Given a target instance $G \in \mathcal{T}_{\text{pre}}$, a positive instance is any node a in G, while a negative instance is any node b in G', a corrupted version of G. Then, we can reformulate the pre-training loss of DGI as

$$\mathcal{L}_{\text{DGI}}(\Theta) = -\sum_{G \in \mathcal{T}_{\text{pre}}} \ln \frac{\sum_{a \in V(G)} \exp(\text{sim}(\mathbf{s}_a, \mathbf{s}_G)/\tau)}{\sum_{b \in V(G')} \exp(\text{sim}(\mathbf{s}_b, \mathbf{s}_G)/\tau)}, \quad (17)$$

where $V(G)$ denotes the set of nodes in G, and G' is obtained by corrupting G.

InfoGraph [27] extends the concept of DGI, which also attempts to maximize the mutual information between the global graph representation and local node representations, but differs from DGI in the sampling of negative instances. Similar to DGI, given a target instance $G \in \mathcal{T}_{\text{pre}}$, a positive instance is any node a in G. However, a negative instance is a node sampled from a different graph $G' \neq G$ from the pre-training data, instead of a corrupted view of G in DGI. Another difference from DGI is, the local representation of a node v is derived by fusing the embeddings of v from all layers of the graph encoder, an architecture that would not be affected by our reformulated loss as

$$\mathcal{L}_{\text{IG}}(\Theta) = -\sum_{G \in \mathcal{T}_{\text{pre}}} \ln \frac{\sum_{a \in V(G)} \exp(\text{sim}(\mathbf{s}_a, \mathbf{s}_G)/\tau)}{\sum_{G' \in \mathcal{T}_{\text{pre}}, G' \neq G} \sum_{b \in V(G')} \exp(\text{sim}(\mathbf{s}_b, \mathbf{s}_G)/\tau)}. \quad (18)$$

GraphCL [26] aims to maximize the mutual information between distinct augmentations of the same graph, while reducing that between augmentations of different graphs. Given a graph $G \in \mathcal{T}_{pre}$, we can obtain distinct augmentations of G, say G_i by augmentation strategy i and G_j by strategy j. One of them, say G_i, can serve as a target instance, while the other, say G_j, can serve as a positive instance w.r.t. G_i. Meanwhile, we can obtain an augmentation of a different graph $G' \neq G$, say G'_j, which can serve as a negative instance w.r.t. G_i. Thus, we can reformulate the pre-training loss as

$$\mathcal{L}_{\text{GCL}}(\Theta) = -\sum_{G \in \mathcal{T}_{\text{pre}}} \ln \frac{\exp(\text{sim}(\mathbf{s}_{G_j}, \mathbf{s}_{G_i})/\tau)}{\sum_{G' \in \mathcal{T}_{\text{pre}}, G' \neq G} \exp(\text{sim}(\mathbf{s}_{G'_j}, \mathbf{s}_{G_i})/\tau)}. \quad (19)$$

GCC [13] employs a framework where the model learns to distinguish between subgraphs that are contextually similar or originated from the same root graph, and subgraphs that are derived from different root graphs. Given a node $v \in \mathcal{T}_{\text{pre}}$, a target instance G_v^r is a subgraph induced by performing random walk in the r-egonet of node v[1]. To sample a positive instance, another random walk is performed in the same r-egonet, resulting in another induced subgraph $\tilde{G}_v^r$ that is different from G_v^r. On the other hand, a negative instance can be sampled by random walk from the r'-egonet of v such that $r' \neq r$. Letting $\mathcal{G}_v^{r'}$ denote a set of random walk induced subgraphs from the r'-egonet of v for some $r' \neq r$, we can reformulate the pre-training loss of GCC as

$$\mathcal{L}_{\text{GCC}}(\Theta) = -\sum_{v \in \mathcal{T}_{\text{pre}}} \ln \frac{\exp(\text{sim}(\mathbf{s}_{\tilde{G}_v^r}, \mathbf{s}_{G_v^r})/\tau)}{\sum_{G_v^{r'} \in \mathcal{G}_v^{r'}} \exp(\text{sim}(\mathbf{s}_{G_v^{r'}}, \mathbf{s}_{G_v^r})/\tau)}. \quad (20)$$

B. Generalizing Prompts

To adapt different downstream tasks more effectively, we propose a more generalized layer-wise prompt design to strategically utilize hierarchical knowledge across multiple layers of the pre-trained graph encoders.

[1]The r-egonet of node v is defined as a subgraph consisting of all nodes within r hops of v and all edges between these nodes

Layer-wise prompt design. Consider a pre-trained graph encoder comprising L layers. In GRAPHPROMPT+, let $\mathcal{P}$ be a set of $L+1$ prompt vectors, with one prompt vector allocated for each layer, including the input layer (*i.e.*, $l = 0$):

$$\mathcal{P} = \{\mathbf{p}^0, \mathbf{p}^1, \ldots, \mathbf{p}^L\}. \tag{21}$$

That is, $\mathbf{p}^l$ is a learnable vector representing the prompt vector that modifies the l-th layer of the pre-trained encoder for the downstream task, for $0 \leq l \leq L$. Note that in GRAPHPROMPT, there is only one prompt vector applied to the last layer $l = L$, which is then used by the READOUT layer to obtain subgraph-level representations. In contrast, in GRAPHPROMPT+, these $L+1$ prompt vectors are applied to different layers in parallel in order to focus on different layers, as illustrated in Fig. 2.

Specifically, let $\mathbf{H_p}$ denote the output from the pre-trained graph encoder after applying a single prompt vector $\mathbf{p}$ to a specific layer associated with $\mathbf{p}$, as follows.

$$\mathbf{H_p} = \textsc{GraphEncoder}_{\mathbf{p}}(\mathbf{X}, \mathbf{A}; \Theta), \tag{22}$$

where $\textsc{GraphEncoder}_{\mathbf{p}}(\cdot)$ indicates that a specific layer of the encoder has been modified by $\mathbf{p}$. Taking the prompt vector $\mathbf{p}^l$ as an example, it modifies the embedding matrix generated by the l-th layer of the graph encoder via element-wise multiplication, which gives $\mathbf{p}^l \odot \mathbf{H}^l$. Here $\mathbf{p}^l$ is multiplied element-wise with each row of $\mathbf{H}^l$ in a manner analogous to that in GRAPHPROMPT. Hence, the embedding matrix of the next layer is generated as follows:

$$\mathbf{H}^{l+1} = \textsc{Aggr}(\mathbf{p}^l \odot \mathbf{H}^l, \mathbf{A}; \theta^{l+1}), \tag{23}$$

while the calculation of other layers remains unchanged, resulting in the output embedding matrix $\mathbf{H}_{\mathbf{p}^l}$.

Finally, we apply each prompt $\mathbf{p}^l \in \mathcal{P}$ to the pre-trained graph encoder in parallel, and obtain a series of embedding matrices $\{\mathbf{H}_{\mathbf{p}^0}, \mathbf{H}_{\mathbf{p}^1}, \ldots, \mathbf{H}_{\mathbf{p}^L}\}$. We further fuse these "post-prompt" embedding matrices into an final output embedding matrix $\mathbf{H}_{\mathcal{P}}$, which will be employed to calculate the downstream task loss. In particular, we adopt a learnable coefficient w^l to weigh $\mathbf{H}_{\mathbf{p}^l}$ in the fused output, *i.e.*,

$$\mathbf{H}_{\mathcal{P}} = \textstyle\sum_{l=0}^{L} w^l \mathbf{H}_{\mathbf{p}^l}, \tag{24}$$

as different downstream tasks may depend on information from diverse layers with varying degrees of importance.

Note that compared to GRAPHPROMPT, we suffer a L-fold increase in the number of learnable parameters during prompt tuning in GRAPHPROMPT+. However, a typical graph encoder adopts a shallow message-passing architecture with a small number of layers, *e.g.*, $L = 3$, which does not present a significant overhead.

Prompt tuning. Given a specific downstream task t, we have a set of task t-specific prompts $\mathcal{P}_t = \{\mathbf{p}_t^0, \mathbf{p}_t^1, \ldots, \mathbf{p}_t^L\}$, corresponding to the $L+1$ layers in the graph encoder. After applying $\mathcal{P}_t$ to the pre-trained encoder, we obtain the embedding for the subgraph S_x as

$$\mathbf{s}_{t,x} = \textsc{ReadOut}(\{\mathbf{h}_{\mathcal{P}_t, v} : v \in V(S_x)\}), \tag{25}$$

where $\mathbf{s}_{t,x}$ is the task t-specific subgraph representation after applying the series of prompts $\mathcal{P}_t$, and $\mathbf{h}_{\mathcal{P}_t, v}$ is a row of $\mathbf{H}_{\mathcal{P}_t}$ that corresponds to node v.

To optimize the generalized prompt vectors, we adopt the same loss function as utilized in GRAPHPROMPT, as given by Eq. (15). That is, we also leverage the task-specific subgraph representations after applying the generalized prompts, $\mathbf{s}_{t,x}$, in the prompt tuning loss.

Complexity analysis. We compare the complexity of GRAPHPROMPT and GRAPHPROMPT+. On a downstream graph G, the computation consists of two main parts: Calculating the node embeddings with a pre-trained GNN, and prompt tuning. First, we calculate node embeddings with a pre-trained GNN, which is common to both GRAPHPROMPT and GRAPHPROMPT+, as well as any other method that utilizes a pre-trained GNN. This part depends on the complexity of the GNN. In a typical GNN, in each layer each node would access its $\bar{d}$ neighbors for aggregation, assuming at most $\bar{d}$ neighbors are involved in the aggregation. Hence, given L layers, the overall complexity of calculating the node embeddings is $O(\bar{d}^L \cdot |V|)$, where $|V|$ is the number of nodes. Second, we perform prompt tuning. On one hand, for GRAPHPROMPT, a prompt vector modifies each node in G, leading to a complexity of $O(|V|)$. On the other hand, for GRAPHPROMPT+, $L+1$ prompt vectors modify each node in G, leading to a complexity of $O(L \cdot |V|)$.

Thus, for both GRAPHPROMPT and GRAPHPROMPT+, the first part is dominant in the complexity, since $O(\bar{d}^L \cdot |V|)$ is much larger than $O(|V|)$ and $O(L \cdot |V|)$. Hence, the prompt tuning step for both of our methods adds a negligible overhead. Furthermore, comparing GRAPHPROMPT+ to GRAPHPROMPT, the overhead is also manageable, since L is often a small value, generally ranging from 1 to 3.

VI. EXPERIMENTS

In this section, we conduct extensive experiments including node classification and graph classification as downstream tasks on five benchmark datasets to evaluate GRAPHPROMPT.

A. Experimental Setup

Datasets. We employ five benchmark datasets for evaluation. (1) *Flickr* [57] is an image sharing network which is collected by SNAP[2]. (2) *PROTEINS* [58] is a collection of protein graphs which include the amino acid sequence, conformation, structure, and features such as active sites of the proteins. (3) *COX2* [59] is a dataset of molecular structures including 467 cyclooxygenase-2 inhibitors. (4) *ENZYMES* [60] is a dataset of 600 enzymes collected from BRENDA enzyme database. (5) *BZR* [59] is a collection of 405 ligands for benzodiazepine receptor.

We summarize these datasets in Table II. Note that the "Task" column indicates the type of downstream task performed on each dataset: "N" for node classification and "G" for graph classification.

Baselines. We evaluate GRAPHPROMPT against the state-of-the-art approaches from three main categories. (1) *End-to-end graph neural networks*: GCN [5], GraphSAGE [6], GAT [7]

[2]https://snap.stanford.edu/data/

TABLE II: Summary of datasets.

	Graphs	Graph classes	Avg. nodes	Avg. edges	Node features	Node classes	Task (N/G)
Flickr	1	-	89,250	899,756	500	7	N
PROTEINS	1,113	2	39.06	72.82	1	3	N, G
COX2	467	2	41.22	43.45	3	-	G
ENZYMES	600	6	32.63	62.14	18	3	N, G
BZR	405	2	35.75	38.36	3	-	G

and GIN [28]. They capitalize on the key operation of neighborhood aggregation to recursively aggregate messages from the neighbors, and work in an end-to-end manner. (2) *Graph pre-training models*: DGI [25], InfoGraph [27], and GraphCL [26]. They work in the "pre-train, fine-tune" paradigm. In particular, they pre-train the GNN models to preserve the intrinsic graph properties, and fine-tune the pre-trained weights on downstream tasks to fit task labels. (3) *Graph prompt models*: GPPT [40]. GPPT utilizes a link prediction task for pre-training, and resorts to a learnable prompt for the node classification task, which is mapped to a link prediction task.

Other few-shot learning baselines on graphs, such as Meta-GNN [61] and RALE [62], often adopt a meta-learning paradigm [36]. They cannot be used in our setting, as they require a large volume of labeled data in their base classes for the meta-training phase. In our approach, only label-free graphs are utilized for pre-training.

Settings and parameters. To evaluate the goal of our GRAPHPROMPT in realizing a unified design that can suit different downstream tasks flexibly, we consider two typical types of downstream tasks, *i.e.*, node classification and graph classification. In particular, for the datasets which are suitable for both of these two tasks, *i.e.*, *PROTEINS* and *ENZYMES*, we only pre-train the GNN model once on each dataset, and utilize the same pre-trained model for the two downstream tasks with their task-specific prompting.

The downstream tasks follow a k-shot classification setting. For each type of downstream task, we construct a series of k-shot classification tasks. The details of task construction will be elaborated later when reporting the results in Sect. VI-B. For task evaluation, as the k-shot tasks are balanced classification, we employ accuracy as the evaluation metric following earlier work [63], [62].

For all the baselines, based on the authors' code and default settings, we further tune their hyper-parameters to optimize their performance. More implementation details of the baselines and our approach can be found in Appendix D of GraphPrompt [29].

B. Performance Evaluation

As discussed, we perform two types of downstream task, namely, node classification and graph classification in few-shot settings. We first evaluate on a fixed-shot setting, and then vary the shot numbers to see the performance trend.

Few-shot node classification. We conduct this node-level task on three datasets, *i.e.*, *Flickr*, *PROTEINS*, and *ENZYMES*. Following a typical k-shot setup [61], [63], [62], we generate a series of few-shot tasks for model training and validation. In particular, for *PROTEINS* and *ENZYMES*, on each graph we randomly generate ten 1-shot node classification tasks (*i.e.*, in each task, we randomly sample 1 node per class) for training and validation, respectively. Each training task is paired with a validation task, and the remaining nodes not sampled by the pair of training and validation tasks will be used for testing. For *Flickr*, as it contains a large number of very sparse node features, selecting very few shots for training may result in inferior performance for all the methods. Therefore, we randomly generate ten 50-shot node classification tasks, for training and validation, respectively. On Flickr, 50 shots are still considered few, accounting for less than 0.06% of all nodes on the graph.

TABLE III: Accuracy evaluation on node classification.
Results are in percent, with best **bolded** and runner-up underlined.

Methods	Flickr 50-shot	PROTEINS 1-shot	ENZYMES 1-shot
GCN	9.22 ± 9.49	59.60 ± 12.44	61.49 ± 12.87
GRAPHSAGE	13.52 ± 11.28	59.12 ± 12.14	61.81 ± 13.19
GAT	16.02 ± 12.72	58.14 ± 12.05	60.77 ± 13.21
GIN	10.18 ± 5.41	60.53 ± 12.19	63.81 ± 11.28
DGI	17.71 ± 1.09	54.92 ± 18.46	63.33 ± 18.13
GRAPHCL	18.37 ± 1.72	52.00 ± 15.83	58.73 ± 16.47
GPPT	18.95 ± 1.92	50.83 ± 16.56	53.79 ± 17.46
GRAPHPROMPT	**20.21** ± 11.52	**63.03** ± 12.14	**67.04** ± 11.48

Table III illustrates the results of few-shot node classification. We have the following observations. First, our proposed GRAPHPROMPT outperforms all the baselines across the three datasets, demonstrating the effectiveness of GRAPHPROMPT in transferring knowledge from the pre-training to downstream tasks for superior performance. In particular, by virtue of the unification framework and prompt-based task-specific aggregation in READOUT function, GRAPHPROMPT is able to close the gap between pre-training and downstream tasks, and derive the downstream tasks to exploit the pre-trained model in task-specific manner. Second, compared to graph pre-training models, GNN models usually achieve comparable or even slightly better performance. This implies that the discrepancy between the pre-training and downstream tasks in these pre-training models, obstructs the knowledge transfer from the former to the latter. Even with sophisticated pre-training, they cannot effectively promote the performance of downstream tasks. Third, the graph prompt model GPPT is only comparable to or even worse than the other baselines, despite also using prompts. A potential reason is that GPPT requires much more learnable parameters in their prompts than ours, which may not work well with very few shots.

Few-shot graph classification. We further conduct few-shot graph classification on four datasets, *i.e.*, *PROTEINS*, *COX2*, *ENZYMES*, and *BZR*. For each dataset, we randomly generate 100 5-shot classification tasks for training and validation, following a similar process for node classification tasks.

We illustrate the results of few-shot graph classification in Table IV, and have the following observations. First, our proposed GRAPHPROMPT significantly outperforms the baselines on these four datasets. This again demonstrates the necessity of unification for pre-training and downstream tasks, and the effectiveness of prompt-assisted task-specific aggregation for

TABLE IV: Accuracy evaluation on graph classification.

Results are in percent, with best **bolded** and runner-up underlined.

Methods	PROTEINS 5-shot	COX2 5-shot	ENZYMES 5-shot	BZR 5-shot
GCN	54.87 ± 11.20	51.37 ± 11.06	20.37 ± 5.24	56.16 ± 11.07
GRAPHSAGE	52.99 ± 10.57	52.87 ± 11.46	18.31 ± 6.22	57.23 ± 10.95
GAT	48.78 ± 18.46	51.20 ± 27.93	15.90 ± 4.13	53.19 ± 20.61
GIN	58.17 ± 8.58	51.89 ± 8.71	20.34 ± 5.01	57.45 ± 10.54
INFOGRAPH	54.12 ± 8.20	54.04 ± 9.45	20.90 ± 3.32	57.57 ± 9.93
GRAPHCL	56.38 ± 7.24	55.40 ± 12.04	28.11 ± 4.00	59.22 ± 7.42
GRAPHPROMPT	**64.42** ± 4.37	**59.21** ± 6.82	**31.45** ± 4.32	**61.63** ± 7.68

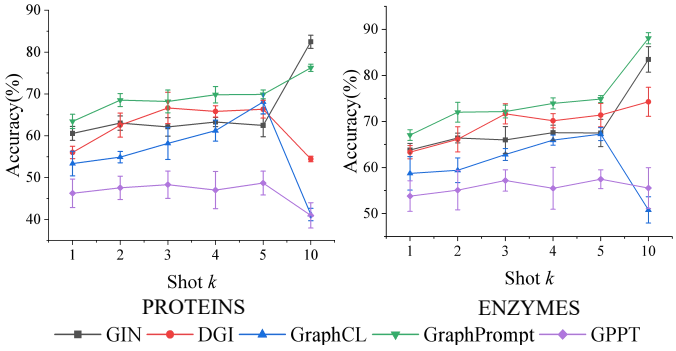


Fig. 3: Impact of shots on few-shot node classification.

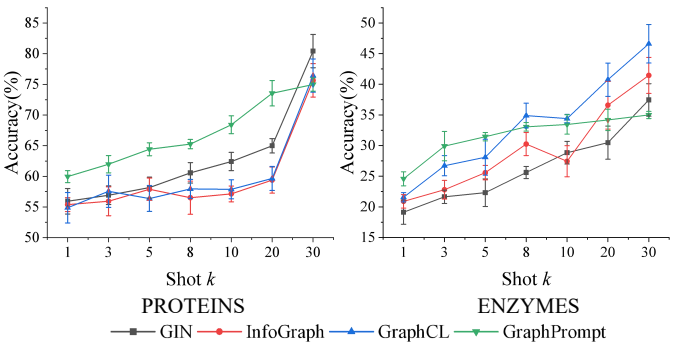


Fig. 4: Impact of shots on few-shot graph classification.

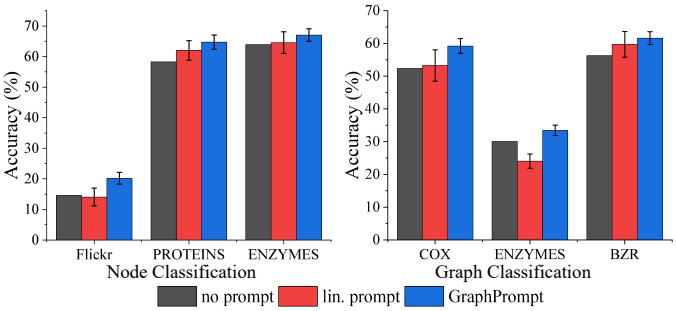


Fig. 5: Ablation study.

READOUT. Second, as both node and graph classification tasks share the same pre-trained model on *PROTEINS* and *ENZYMES*, the superior performance of GRAPHPROMPT on both types of tasks further demonstrates that, the gap between different tasks is well addressed by virtue of our unification framework. Third, the graph pre-training models generally achieve better performance than the end-to-end GNN models. This is because both InfoGraph and GraphCL capitalize on graph-level tasks for pre-training, which are naturally not far away from the downstream graph classification.

Performance with different shots. We further tune the number of shots for the two few-shot classification tasks to evaluate its influence on the performance. In particular, for few-shot node classification, we tune the number of shots in {1, 2, 3, 4, 5, 10}, and employ the most competitive baselines (*i.e.*, GIN, DGI, GraphCL, and GPPT) for comparison. Similarly, for few-shot graph classification, we tune the number of shots in {1, 3, 5, 8, 10, 20}, and also employ the most competitive baselines (*e.g.*, GIN, InfoGraph, and GraphCL) for comparison. The number of tasks is identical to the above settings of few-shot node classification and graph classification. We conduct experiments on two datasets, *i.e.*, *PROTEINS* and *ENZYMES* for the two tasks.

We illustrate the comparison in Figs. 3 and 4 for node and graph classification[3], respectively, and have the following observations. First, for few-shot node classification, in general our proposed GRAPHPROMPT consistently outperforms the baselines across different shots. The only exception occurs on *PROTEINS* with 10-shot, which is possibly because 10-shot labeled data might be sufficient for GIN to work in an end-to-end manner. Second, for few-shot graph classification, our proposed GRAPHPROMPT outperforms the baselines when very limited labeled data is given (*e.g.*, when number of shots is 1, 3, or 5), and might be surpassed by some competitive baselines when fed with adequate labeled data (*e.g.*, GraphCL with number of shot 8, 10, and 20).

[3]The error bars in the figures represent standard deviation.

C. Ablation Study

To evaluate the contribution of each component, we conduct an ablation study by comparing GRAPHPROMPT with different prompting strategies: (1) *no prompt*: for downstream tasks, we remove the prompt vector, and conduct classification by employing a classifier on the subgraph representations obtained by a direct sum-based READOUT. (2) *lin. prompt*: we replace the prompt vector with a linear transformation matrix as an alternative design.

We report the comparison in Fig. 5, and note the following[3]. (1) Without the prompt vector, *no prompt* usually performs the worst in the four models, showing the necessity of prompting the READOUT operation differently for different downstream tasks. (2) Converting the prompt vector into a linear transformation matrix also hurts the performance, as the matrix involves more parameters thus increasing the reliance on labeled data.

VII. EXPERIMENTS ON GRAPHPROMPT+

In this section, we further conduct experiments to evaluate our extended approach GRAPHPROMPT+ in comparison to the vanilla GRAPHPROMPT.

We follow the same experiment setup of GRAPHPROMPT, as detailed in Sect. VI-A. In particular, we evaluate the same

few-shot node classification or graph classification tasks on the same five datasets. Consistent with GRAPHPROMPT, in the implementation of GRAPHPROMPT+, we utilize a three-layer GIN as the backbone, and set the hidden dimensions as 32. We also set $\delta = 1$ to construct 1-hop subgraphs for the nodes same as GRAPHPROMPT.

A. Effect of Generalized Layer-wise Prompts

We first investigate the impact of layer-wise prompts to evaluate their ability of extracting hierarchical pre-trained knowledge. To isolate the effect of prompts and be comparable to GRAPHPROMPT, we fix the pre-training task in GRAPHPROMPT+ to link prediction; further experiments on alternative pre-training tasks will be presented in Sect. VII-B. Furthermore, we compare to three variants of GRAPHPROMPT+, namely, GRAPHPROMPT+/0, GRAPHPROMPT+/1 and GRAPHPROMPT+/2. Here, GRAPHPROMPT+/l denotes the variant where only one prompt vector $\mathbf{p}^l$ is applied to modify the l-th layer of the pre-trained encoder. As we have a total of $L = 3$ layers, GRAPHPROMPT is equivalent to GRAPHPROMPT+/3.

In the following, we perform few-shot node and graph classification as the downstream tasks, and analyze the results of GRAPHPROMPT, GRAPHPROMPT+ and the variants.

Few-shot node classification. The results of node classification are reported in Table V. We first observe that GRAPHPROMPT+ outperforms all variants and GRAPHPROMPT. Except GRAPHPROMPT+, only a single prompt vector is added to a specific layer of the graph encoder. This indicates the benefit of leveraging multiple prompt vectors in a layer-wise manner to extract hierarchical knowledge within the pre-trained graph encoder. Second, the application of prompts at different layers leads to varying performance. This observation confirms the nuanced distinctions across the layers of the graph encoder, suggesting the existence of hierarchical structures within pre-trained knowledge. Third, the significance of prompt vectors at different layers varies across datasets. Specifically, for the *Flickr* dataset, applying the prompt to a deeper layer generally yields progressively better performance. In contrast, for the *PROTEINS* and *ENZYMES* datasets, implementing the prompt at the last layer (*i.e.*, GRAPHPROMPT) is generally not as good compared to its application at the shallow layers. This discrepancy is attributed to the notable difference in the nature and size of the graphs. Note that the *Flickr* graph is a relatively large image sharing network, while the graphs in the *PROTEINS* and *ENZYMES* datasets describe small chemical structures, as detailed in Table II. In smaller graphs, knowledge from shallower layers could be adequate, given the relative size of the receptive field to the graph. Nevertheless, GRAPHPROMPT+ automatically selects the most relevant layers and achieves the best performance.

Few-shot graph classification. The results for graph classification are presented in Table VI. We first observe that GRAPHPROMPT+ continues to outperform all variants and GRAPHPROMPT on graph classification, similar to the results on node classification. This demonstrates the robustness of our layer-wise prompts on different types of graph tasks.

TABLE V: Effect of layer-wise prompts (node classification). Results are in percent, with best **bolded** and runner-up underlined.

Methods	Flickr 50-shot	PROTEINS 1-shot	ENZYMES 1-shot
GRAPHPROMPT+/0	17.18 ± 11.49	62.64 ± 13.31	68.32 ± 10.54
GRAPHPROMPT+/1	18.14 ± 11.82	63.02 ± 12.04	68.52 ± 10.77
GRAPHPROMPT+/2	20.11 ± 12.58	63.61 ± 11.89	69.09 ± 10.19
GRAPHPROMPT	20.21 ± 11.52	63.03 ± 12.14	67.04 ± 11.48
GRAPHPROMPT+	**20.55** ± 11.97	**63.64** ± 11.69	**69.28** ± 10.74
(↑ vs. GRAPHPROMPT)	(+1.68%)	(+0.97%)	(+3.34%)

TABLE VI: Effect of layer-wise prompts (graph classification). Results are in percent, with best **bolded** and runner-up underlined.

Methods	PROTEINS 5-shot	COX2 5-shot	ENZYMES 5-shot	BZR 5-shot
GRAPHPROMPT+/0	60.80 ± 2.96	51.14 ± 12.93	44.93 ± 3.91	53.93 ± 5.37
GRAPHPROMPT+/1	56.49 ± 8.79	53.53 ± 6.73	35.91 ± 4.64	48.50 ± 1.80
GRAPHPROMPT+/2	61.63 ± 6.86	58.16 ± 6.97	34.36 ± 5.16	48.50 ± 2.43
GRAPHPROMPT	64.42 ± 4.37	59.21 ± 6.82	31.45 ± 4.32	61.63 ± 7.68
GRAPHPROMPT+	**67.71** ± 7.09	**65.23** ± 5.59	**45.35** ± 4.15	**68.61** ± 3.99
(↑ vs. GRAPHPROMPT)	(+5.11%)	(+10.17%)	(+44.20%)	(+11.33%)

Moreover, compared to node classification, GRAPHPROMPT+ shows a more pronounced improvement over the variants. This difference stems from the nature of graph classification as a graph-level task, which emphasizes the need for more global knowledge from all layers of the graph encoder. Thus, effective integration of hierarchical knowledge across these layers is more important to graph classification, which can enhance performance more significantly.

B. Compatibility with Generalized Pre-training Tasks

Finally, we conduct experiments using alternative contrastive learning approaches for pre-training, beyond the simple link prediction task. Specifically, we select the two most popular contrastive pre-training task on graphs, *i.e.*, DGI [25] and GraphCL [26], and implement the generalized pre-training loss for each of them as discussed in Sect. V-A. For each pre-training task, we compare among GRAPHPROMPT+, GRAPHPROMPT, and the original architecture in their paper without prompt tuning (denoted as "Original").

The results of node and graph classification tasks are reported in Tables VII and VIII, respectively. It is evident that both GRAPHPROMPT+ and GRAPHPROMPT exhibit superior performance compared to the original versions of DGI and GraphCL[4]. The results imply that our proposed prompt-based framework can flexibly incorporate well-known contrastive pre-training models like DGI and GraphCL, overcoming the limitation of a singular pre-training approach based on link prediction. Furthermore, empirical results also reveal a consistent trend where GRAPHPROMPT+ demonstrates superior performance over GRAPHPROMPT. This trend further corroborates the effectiveness of layer-wise prompts under alternative pre-training tasks, extending the analysis in Sect. VII-A.

[4]One exception occurs when using GraphCL as the pre-training method for the 5-shot graph classification task on *PROTEINS*. The reason could be that 5-shot labeled data already provide adequate supervision in this case, but our methods are still superior in more label-scarce scenarios. Specifically, in the one-shot scenario, the accuracy of original GraphCL, GRAPHPROMPT and GRAPHPROMPT+ is 53.26, 53.77 and 54.07, respectively.

TABLE VII: Compatibility with popular contrastive pre-training on graphs, for downstream node classification.
Results are in percent, with best **bolded** and runner-up underlined.

Pre-training	Methods	Flickr 50-shot	PROTEINS 1-shot	ENZYMES 1-shot
DGI	Original	17.71 ± 1.09	54.92 ± 18.46	63.33 ± 18.13
	GRAPHPROMPT	17.78 ± 4.98	60.79 ± 12.00	66.46 ± 11.39
	GRAPHPROMPT+	**20.98** ± 11.60	**65.24** ± 12.51	**68.92** ± 10.77
GraphCL	Original	18.37 ± 1.72	52.00 ± 15.83	58.73 ± 16.47
	GRAPHPROMPT	19.33 ± 4.11	60.15 ± 13.31	63.14 ± 11.02
	GRAPHPROMPT+	**19.95** ± 12.48	**62.55** ± 12.63	**68.17** ± 11.30

TABLE VIII: Compatibility with popular contrastive pre-training on graphs, for downstream graph classification.
Results are in percent, with best **bolded** and runner-up underlined.

Pre-training	Methods	PROTEINS 5-shot	COX2 5-shot	ENZYMES 5-shot	BZR 5-shot
DGI	Original	54.12 ± 8.20	54.04 ± 9.45	20.90± 3.32	57.57± 9.93
	GRAPHPROMPT	54.32 ± 0.61	**54.60** ± 5.96	27.69 ± 2.23	58.53 ± 7.41
	GRAPHPROMPT+	**56.16** ± 0.67	54.30 ± 6.58	**41.40** ± 3.77	**62.83** ± 8.97
GraphCL	Original	**56.38** ± 7.24	55.40 ± 12.04	28.11 ± 4.00	59.22± 7.42
	GRAPHPROMPT	55.30 ± 1.05	57.87 ± 1.52	29.82 ± 2.87	61.35 ± 7.54
	GRAPHPROMPT+	55.59 ± 2.63	**59.32** ± 17.00	**41.42** ± 3.73	**61.75** ± 7.56

VIII. CONCLUSIONS

In this paper, we studied the research problem of prompting on graphs and proposed GRAPHPROMPT, in order to overcome the limitations of graph neural networks in the supervised or "pre-train, fine-tune" paradigms. In particular, to narrow the gap between pre-training and downstream objectives on graphs, we introduced a unification framework by mapping different tasks to a common task template. Moreover, to distinguish task individuality and achieve task-specific optima, we proposed a learnable task-specific prompt vector that guides each downstream task to make full of the pre-trained model. We further extended GRAPHPROMPT into GRAPHPROMPT+by enhancing both the pre-training and prompt tuning stages. Finally, we conduct extensive experiments on five public datasets, and show that GRAPHPROMPT and GRAPHPROMPT+significantly outperforms various state-of-the-art baselines.

ACKNOWLEDGMENTS

This research / project is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Proposal ID: T2EP20122-0041). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore. This work is also supported in part by the National Key Research and Development Program of China under Grant 2020YFB2103803. The first author extends his heartfelt gratitude to Ms. Mengzhuo Fang for her invaluable support and assistance during challenging periods.

REFERENCES

[1] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *SIGKDD*, 2014, pp. 701–710.
[2] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.
[3] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD*, 2016, pp. 855–864.
[4] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *TKDE*, vol. 30, no. 9, pp. 1616–1637, 2018.
[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
[6] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *NeurIPS*, vol. 30, pp. 1025–1035, 2017.
[7] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.
[8] X. Yu, Z. Liu, Y. Fang, and X. Zhang, "Learning to count isomorphisms with graph neural networks," in *AAAI*, 2023.
[9] D. Zhang, W. Feng, Y. Wang, Z. Qi, Y. Shan, and J. Tang, "Dropconn: Dropout connection based random gnns for molecular property prediction," *IEEE TKDE*, 2023.
[10] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *TNNLS*, vol. 32, no. 1, pp. 4–24, 2020.
[11] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun, "GPT-GNN: Generative pre-training of graph neural networks," in *SIGKDD*, 2020, pp. 1857–1867.
[12] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," in *ICLR*, 2020.
[13] 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 *SIGKDD*, 2020, pp. 1150–1160.
[14] Y. Lu, X. Jiang, Y. Fang, and C. Shi, "Learning to pre-train graph neural networks," in *AAAI*, 2021, pp. 4276–4284.
[15] X. Xu, F. Zhou, K. Zhang, and S. Liu, "CCGL: Contrastive cascade graph learning," *IEEE TKDE*, vol. 35, no. 5, pp. 4539–4554, 2022.
[16] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," *IEEE TKDE*, vol. 35, no. 1, pp. 857–876, 2021.
[17] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H.-W. Hon, "Unified language model pre-training for natural language understanding and generation," *NeurIPS*, vol. 32, 2019.
[18] H. Bao, L. Dong, S. Piao, and F. Wei, "BEit: BERT pre-training of image transformers," in *ICLR*, 2022.
[19] L. Hu, Z. Liu, Z. Zhao, L. Hou, L. Nie, and J. Li, "A survey of knowledge enhanced pre-trained language models," *IEEE TKDE*, 2023.
[20] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *arXiv preprint arXiv:2107.13586*, 2021.
[21] 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," *NeurIPS*, vol. 33, pp. 1877–1901, 2020.
[22] Y. Zhu, X. Zhou, J. Qiang, Y. Li, Y. Yuan, and X. Wu, "Prompt-learning for short text classification," *arXiv preprint arXiv:2202.11345*, 2022.
[23] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *NeurIPS*, vol. 31, 2018.
[24] K. Huang and M. Zitnik, "Graph meta learning via local subgraphs," *NeurIPS*, vol. 33, pp. 5862–5874, 2020.
[25] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2019.
[26] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *NeurIPS*, vol. 33, pp. 5812–5823, 2020.
[27] F.-Y. Sun, J. Hoffman, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," in *ICLR*, 2020.
[28] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.
[29] Z. Liu, X. Yu, Y. Fang, and X. Zhang, "GraphPrompt: Unifying pre-training and downstream tasks for graph neural networks," in *WWW*, 2023, pp. 417–428.
[30] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pretrained language model for scientific text," in *NAACL*, 2019, pp. 3615–3620.
[31] J. Lu, D. Batra, D. Parikh, and S. Lee, "ViLBERT: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks," *NeurIPS*, vol. 32, 2019.
[32] J. Xia, Y. Zhu, Y. Du, and S. Z. Li, "A survey of pretraining on graphs: Taxonomy, methods, and applications," *arXiv preprint arXiv:2202.07893*, 2022.
[33] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.
[34] Y. You, T. Chen, Y. Shen, and Z. Wang, "Graph contrastive learning automated," in *ICML*, 2021, pp. 12 121–12 132.

[35] S. Suresh, P. Li, C. Hao, and J. Neville, "Adversarial graph augmentation to improve graph contrastive learning," *NeurIPS*, vol. 34, pp. 15 920–15 933, 2021.
[36] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017, pp. 1126–1135.
[37] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang, "GPT understands, too," *arXiv preprint arXiv:2103.10385*, 2021.
[38] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," in *EMNLP*, 2021, pp. 3045–3059.
[39] X. Yu, Y. Fang, Z. Liu, Y. Wu, Z. Wen, J. Bo, X. Zhang, and S. C. Hoi, "Few-shot learning on graphs: from meta-learning to pre-training and prompting," *arXiv preprint arXiv:2402.01440*, 2024.
[40] M. Sun, K. Zhou, X. He, Y. Wang, and X. Wang, "Gppt: Graph pre-training and prompt tuning to generalize graph neural networks," in *SIGKDD*, 2022, pp. 1717–1727.
[41] Z. Tan, R. Guo, K. Ding, and H. Liu, "Virtual node tuning for few-shot node classification," *arXiv preprint arXiv:2306.06063*, 2023.
[42] X. Sun, H. Cheng, J. Li, B. Liu, and J. Guan, "All in one: Multi-task prompting for graph neural networks," in *SIGKDD*, 2023, pp. 2120–2131.
[43] Y. Zhu, J. Guo, and S. Tang, "Sgl-pt: A strong graph learner with graph prompt tuning," *arXiv preprint arXiv:2302.12449*, 2023.
[44] X. Yu, Z. Liu, Y. Fang, and X. Zhang, "Hgprompt: Bridging homogeneous and heterogeneous graphs for few-shot prompt learning," in *AAAI*, 2024, pp. 16 578–16 586.
[45] X. Yu, C. Zhou, Y. Fang, and X. Zhang, "Multigprompt for multi-task pre-training and prompting on graphs," in *WWW*, 2024, pp. 515–526.
[46] J. Zhang, Z. Wang, S. Zhang, M. M. Bhalerao, Y. Liu, D. Zhu, and S. Wang, "Graphprompt: Biomedical entity normalization using graph-based prompt templates," *arXiv preprint arXiv:2112.03002*, 2021.
[47] Z. Liu, W. Zhang, Y. Fang, X. Zhang, and S. C. H. Hoi, "Towards locality-aware meta-learning of tail node embeddings on networks," in *CIKM*, 2020, pp. 975–984.
[48] Z. Liu, Y. Fang, C. Liu, and S. C. Hoi, "Node-wise localization of graph neural networks," in *IJCAI*, 2021, pp. 1520–1526.
[49] Z. Liu, T.-K. Nguyen, and Y. Fang, "Tail-gnn: Tail-node graph neural networks," in *SIGKDD*, 2021, pp. 1109–1119.
[50] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *NeurIPS*, vol. 31, pp. 4805–4815, 2018.
[51] D. Hwang, J. Park, S. Kwon, K. Kim, J.-W. Ha, and H. J. Kim, "Self-supervised auxiliary learning with meta-paths for heterogeneous graphs," *NeurIPS*, vol. 33, pp. 10 294–10 305, 2020.
[52] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *JMLR*, pp. 2539–2561, 2011.
[53] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *AAAI*, 2018, pp. 4438–4445.
[54] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt, "Wasserstein weisfeiler-lehman graph kernels," *NeurIPS*, vol. 32, 2019.
[55] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon, "A survey on contrastive self-supervised learning," *Technologies*, vol. 9, no. 1, p. 2, 2020.
[56] M. Xu, H. Wang, B. Ni, H. Guo, and J. Tang, "Self-supervised graph-level representation learning with local and global structure," in *ICML*. PMLR, 2021, pp. 11 548–11 558.
[57] Z. Wen, Y. Fang, and Z. Liu, "Meta-inductive node classification across graphs," in *SIGIR*, 2021, pp. 1219–1228.
[58] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.
[59] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015, pp. 4292–4293.
[60] S. Wang, Y. Dong, X. Huang, C. Chen, and J. Li, "FAITH: Few-shot graph classification with hierarchical task graphs," in *IJCAI*, 2022.
[61] F. Zhou, C. Cao, K. Zhang, G. Trajcevski, T. Zhong, and J. Geng, "Meta-GNN: On few-shot node classification in graph meta-learning," in *CIKM*, 2019, pp. 2357–2360.
[62] Z. Liu, Y. Fang, C. Liu, and S. C. Hoi, "Relative and absolute location embedding for few-shot node classification on graph," in *AAAI*, 2021, pp. 4267–4275.
[63] N. Wang, M. Luo, K. Ding, L. Zhang, J. Li, and Q. Zheng, "Graph few-shot learning with attribute matching," in *CIKM*, 2020, pp. 1545–1554.

Xingtong Yu received his PhD degree in computer science from the University of Science and Technology of China in 2024, and the bachelor's degree from School of the Gifted Young, University of Science and Technology of China in 2019. His current research focuses on graph-based machine learning, graph mining, and prompting on graphs.

Zhenghao Liu is currently a MSc student with the School of Computer Science and Technology, University of Science and Technology of China. He received his B.S. Degree in computer science from Shanghai University of Finance and Economics Shanghai, China in 2020. His current research focuses on graph-based machine learning, graph mining, and prompting on graphs.

Yuan Fang (Senior Member, IEEE) received his bachelor's degree in computer science from the National University of Singapore in 2009 and his Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 2014. He is currently an assistant professor with the School of Computing and Information Systems, Singapore Management University. His current research focuses on graph learning and their applications.

Zemin Liu is currently an assistant professor with the College of Computer Science and Technology, Zhejiang University. He received his Ph.D. degree in Computer Science from Zhejiang University, Hangzhou, China in 2018, and B.S. Degree in Software Engineering from Shandong University, Jinan, China in 2012. His research interests lie in graph embedding, graph neural networks, and learning on heterogeneous information networks.

Sihong Chen is currently a senior researcher at Tencent AI. Her current research focuses on computer vision and multi-modal machine learning.

Xinming Zhang (Senior Member, IEEE) received the BE and ME degrees in electrical engineering from the China University of Mining and Technology, Xuzhou, China in 1985 and 1988, respectively, and the PhD degree in computer science and technology from the University of Science and Technology of China, Hefei, China in 2001. Since 2002, he has been with the faculty of the University of Science and Technology of China, where he is currently a professor with the School of Computer Science and Technology. From 2005 to 2006, he was a visiting professor with the Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea. His research interest includes wireless networks, Big Data and deep learning. He has published more than 100 papers. He won the second prize of Science and Technology Award of Anhui Province of China in Natural Sciences in 2017. He won the awards of Top Reviewers (1%) in Computer Science & Cross Field by Publons in 2019.