Appendices

A Pseudocode of L2P-GNN

The pseudocode of the pre-training procedure for L2P-GNN is outlined in Algorithm 1. The training of L2P-GNN involves the initialization of parameters, task construction, as well as node-level and graph-level adaptations. At the beginning (Line 1), we randomly initialize all learnable parameters θ in our L2P-GNN, including the node-level aggregation parameters ψ and graph-level pooling parameters ω. Then, we construct child tasks and the parent task for the graph $\mathcal{G}$. Each child task consists of a support set and a query set, each of which contains edges randomly sampled from the edge distribution $p_{\mathcal{E}}$ of the graph (Line 2). In each training iteration, for each child task $\mathcal{T}_{\mathcal{G}}^{c}$, we perform node-level adaptation on the support set (Line 4–8). Furthermore, we conduct graph-level adaptation with the sub-structure and whole graph representations (Line 9–12). At last, we update all learnable parameters in L2P-GNN (Line 13). The process stops when the model converges.

Algorithm 1 Pre-training of L2P-GNN

Require: a pre-training graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{Z})$; node and graph-level update steps: s and t; node-level, graph-level and prior learning rates: α, β and γ; number of child-tasks: k; support set and query set size: m and n.

1: Randomly initialize GNN parameters $\theta = \{\psi, \omega\}$
2: Construct child tasks and the parent task for the graph $\mathcal{G}$ by Eq. (6): $\mathcal{T}_{\mathcal{G}} = (\mathcal{T}_{\mathcal{G}}^{1}, \mathcal{T}_{\mathcal{G}}^{2}, \cdots, \mathcal{T}_{\mathcal{G}}^{k})$,
each child task $\mathcal{T}_{\mathcal{G}}^{c}$ consisting of a support $\mathcal{S}_{\mathcal{G}}^{c}$ and a query set $\mathcal{Q}_{\mathcal{G}}^{c}$, and $\mathcal{S}_{\mathcal{G}}^{c} = \{(u, v) \sim p_{\mathcal{E}}\}$, $\mathcal{Q}_{\mathcal{G}}^{c} = \{(p, q) \sim p_{\mathcal{E}}\}$.
3: **while** not done **do**
4: **for all** child task $\mathcal{T}_{\mathcal{G}}^{c}$ w.r.t. graph $\mathcal{G}$ **do**
5: Compute node representation $\mathbf{h}_{v}^{l}$ by Eq. (1) for all nodes in support set $\mathcal{S}_{\mathcal{G}}^{c}$
6: Evaluate $\mathcal{L}^{node}(\psi, \mathcal{S}_{\mathcal{G}}^{c})$ by Eq. (7)
7: Node-level adaptation by Eq. (10) with s updates
8: **end for**
9: Compute sub-structure representation with $\mathbf{h}_{\mathcal{S}_{\mathcal{G}}^{c}} = \Omega\left(\omega; \left\{\mathbf{h}_{\mathbf{u}} | \forall u, \exists v : (u, v) \in \mathcal{S}_{\mathcal{G}}^{c}\right\}\right)$
10: Compute the whole graph representation $\mathbf{h}_{\mathcal{G}}$ by Eq. (2)
11: Evaluate $\mathcal{L}^{graph}(\omega, \mathcal{S}_{\mathcal{G}})$ by Eq. (8)
12: Graph-level adaptation by Eq. (11) with t updates
13: Update all learnable parameters θ in L2P-GNN by Eq. (9)
14: **end while**

B Details and Processing of Datasets

We conduct experiments on two large-scale datasets, including biology graphs (called Biology) and bibliographic graphs (called PreDBLP). Here we provide more details of the two datasets and any additional processing done.

Biology. Biology dataset comes from a public repository[3], covering 394,925 protein subgraphs. Following earlier work (Hu et al. 2020), we perform biological function prediction on the Biology data. Detailed information about the dataset can be found in Appendix D of the original paper (Hu et al. 2020).

PreDBLP. To enrich graph pre-training data from a different domain, we further present PreDBLP, a new compilation of bibliographic graphs. We derive the new PreDBLP data from AMiner[4] and DBLP[5]. Specifically, PreDBLP contains 1,054,309 paper subgraphs in 31 fields (e.g., artificial intelligence, data mining). Each subgraph is centered at a paper and contains the associated information of the paper.

The original Aminer/DBLP contains both the records of each paper and the implicit relations between papers, authors, venues and keywords. For each paper record in the Aminer/DBLP data, we generate a subgraph centered on the paper as follows: (1) according to the citation relationship, we perform a breadth-first search to select the subgraph nodes, with a search depth limit of 2 and a maximum number of 10 neighbors randomly expanded per node; (2) we include the selected paper nodes and all the edges between those paper nodes into the subgraph; (3) we convert the authors attached to each paper's record to nodes as well, and link them to the paper; (4) we utilize the same procedure as in (3) to incorporate the information of venues and keyword terms. As a result, each subgraph compiled contains four types of nodes (i.e., paper, author, venue and keywords) and edges (i.e., paper-paper, paper-author, paper-venue, paper-keywords).

We further utilize a set of node and edge features for the subgraph. For each subgraph, we set the node/edge features as their corresponding types. For instance, for nodes u and v connected via edge (u, v), the feature of u and v are their respective type and that of edge (u, v) is the type of (u, v).

During the pre-training process, we utilize 794,862 subgraphs that belong to 25 research fields to pre-train a GNN model. On average, each subgraph contains 262.43 nodes and 900.07 edges. In fine-tuning, we predict the research field of 299,447 labeled subgraphs from the remaining 6 research fields, including: *Artificial intelligence* (86,956 subgraphs), *Computational linguistics* (20,024 subgraphs), *Computer Vision* (95,729 subgraphs), *Data mining* (14,934 subgraphs), *Databases* (68,287 subgraphs) and *Fuzzy systems* (13,517 subgraphs).

C Implementation details of GNN Models

Here, we introduce the GNN architecture used in biological function prediction on Biology and research field prediction on PreDBLP. For both experiments, we utilize the GIN architectures (Xu et al. 2019b) as an example to explain how to incorporate the node features and edge features in the subgraphs.

Biological Function Prediction. Following previous work (Hu et al. 2020), the raw node features are uniform and the raw input edge features are binary vectors since the protein subgraphs only have edge features. We adopt the same GNN architecture as in (Hu et al. 2020) for protein function predic-

[3] http://snap.stanford.edu/gnn-pretrain

[4] https://www.aminer.cn/citation

[5] https://dblp.uni-trier.de

tion. Detailed implementation please refer to the Appendix A in (Hu et al. 2020).

Research Field Prediction. In research field prediction, the raw node features are 4-dimensional one-hot vectors, denoted as $\mathbf{x}_v \in \mathbb{R}^4$ for node v. The raw edge features are 1-dimensional type vector indicting the type of edge, denoted as $\mathbf{z}_{uv} \in \mathbb{R}^1$ (see Appendix B for details). As input features to GNNs, we first embed the feature vectors by

$$\mathbf{h}_v^0 = \mathbf{W}^{node}\mathbf{x}_u + \mathbf{b}^{node} \tag{A.1}$$

$$\mathbf{h}_{e_{uv}}^l = \mathbf{W}^{edge}\mathbf{z}_{uv} + \mathbf{b}^{edge} \quad \text{for } l = 0, 1, ..., L-1, \tag{A.2}$$

where $\mathbf{W}^{node}, b^{node}, \mathbf{W}^{edge}$ and b^{edge} are learnable parameters. At each layer, GNNs update node representations by

$$\mathbf{h}_v^l = \textsc{ReLU}(\textsc{Mlp}^l(\textsc{Concat}(\sum_{u\in\mathcal{N}_u\cup\{v\}} \mathbf{h}_u^{l-1}, \sum_{e_{uv}:u\in\mathcal{N}_u\cup\{v\}} \mathbf{h}_e^{l-1}))), \tag{A.3}$$

where $\textsc{Concat}(\cdot)$ takes two vectors as input and concatenates them, and $\mathcal{N}_u$ is a set of nodes adjacent to node v. Note that we remove the ReLU activation in the final layer so as to output negative values in $\mathbf{h}_v^l$.

With the aggregation and update of node/edge features, we generate node embeddings at final layer l to obtain the graph-level representation $\mathbf{h}_\mathcal{G}$:

$$\mathbf{h}_G = \textsc{Mlp}(\textsc{Mean}\left(\{\mathbf{h}_v^l | v \in \mathcal{G}\}\right)), \tag{A.4}$$

where Mean is the mean pooling operation and $\Omega(\cdot) = \textsc{Mlp}(\textsc{Mean}(\cdot))$ is the graph-level pooling calculation.

For other GNN architectures like GCN, GraphSAGE and GAT, we adopt the implementation in the Pytorch Geometric library[6]. More specifically, the number of GAT attention heads is set to 2 and the dimension of node/edge embeddings as well as the number of GNN layers are the same as GIN. Since these GNN models do not originally handle edge features, we incorporate edge features into them similarly to how we do it for the GIN; we add edge embeddings into node embeddings, and perform the GNN message-passing on the obtained node embeddings, as suggested in (Hu et al. 2020).

D Details of Experimental Settings

Implementation of Baselines To contextualize the empirical results of L2P-GNN on the pre-training benchmarks, we compare against four self-supervised or unsupervised baselines:

- EdgePred (Hamilton, Ying, and Leskovec 2017) is a self-supervised method to predict the connectivity of node pairs, which adapts the same objective function as node-level loss in L2P-GNN.
- DGI (Velickovic et al. 2019) learns node representations within graph-structured data in an unsupervised manner, which relies on maximizing mutual information between patch representations and corresponding high-level summaries of graphs—both derived using established graph convolutional network architectures.
- ContextPred (Hu et al. 2020) utilizes node-level self-supervised information to explore distribution of graph structure. We use the suggested parameters to sample subgraphs to predict their surrounding graph structures.
- AttrMasking (Hu et al. 2020) is also a node-level self-supervised pre-training strategy for GNNs, aiming to learn the regularities of the node and edge attributes distributed over graphs

All the above pre-training baselines and our L2P-GNN can be implemented for different GNN architectures. We experiment with four popular GNN architectures, namely, GCN (Kipf and Welling 2017), GraphSAGE (Hamilton, Ying, and Leskovec 2017), GAT (Velickovic et al. 2018) and GIN (Xu et al. 2019b). We implement these GNNs with PyTorch Geometric (PyG).

Parameter Settings We adopt Adaptive Moment Estimation (Adam) to optimize our L2P-GNN. We select the hyper-parameters that performed well across all downstream tasks in the validation sets. In pre-training procedure, for all datasets, we use a batch size of 64 and set the dimension of node representation to 300. We perform one step gradient descent update in both node-level and graph-level adaptations (i.e., $s = t = 1$). The prior learning rate, node-level and graph-level learning rates are all set to 0.001 (i.e., $\gamma = \alpha = \beta = 0.001$). We set the number of child tasks to 1 for all datasets, and set the sizes of support/query sets to 10/5 and 50/30 for Biology and PreDBLP dataset, respectively. The number of layers of GNNs is set to 5 for all datasets. The maximum number of epochs are set to 50 and 20 for pre-training GNNs on Biology and PreDBLP dataset, respectively. In fine-tuning procedure, all models are also trained with Adam optimizer with a learning rate of 0.001. For all downstream datasets, we use a batch size of 32 and train models for 50 epochs.

For baselines, we optimize their parameters empirically under the guidance of literature. Specifically, we also train the baselines with Adam optimizer with a learning rate of 0.001 and set the dimension of node representation to 300. As suggested in (Hu et al. 2020), we set the batch size to 256 for pre-training while 32 for fine-tuning Biology dataset. For PreDBLP dataset, we set the batch size and the number of epochs to be the same as in our L2P-GNN. Other baseline parameters either adopt the original optimal settings or are optimized by the validation set.

Experiment Environment All experiments are conducted on a Linux server with one GPU (GeForce RTX 2080) and CPU (Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz), and its operating system is Red Hat 4.8.5-16. We implement the proposed L2P-GNN with deep learning library PyTorch and PyTorch Geometric. The Python and PyTorch versions are 3.7.6 and 1.4.0, respectively.

[6]https://github.com/rusty1s/pytorch_geometric