

Scalable Multi-Source Pre-training for Graph Neural Networks

(Appendix)

A PRELIMINARIES FOR PRE-TRAINING

A.1 Conventional GNN Pretraining

In conventional GNN pre-training approaches, the key idea is to utilize easily accessible information to encode the intrinsic graph information through self-supervised tasks. Then, the pre-trained GNN can serve as a good initialization for various downstream applications. Formally, let $\mathcal{G}^{\text{pre}} = (X^{\text{pre}}, A^{\text{pre}})$ denote the graph for self-supervised pre-training, and Y^{pre} represents the labels corresponding to the self-supervised objective. The conventional pre-training process can be described as:

$$\theta^{\text{pre}}, \psi^{\text{pre}} = \arg \min_{\theta, \psi} f_{\psi}^{\text{pre}}(\text{GNN}(\mathcal{G}^{\text{pre}}; \theta), Y^{\text{pre}}). \quad (13)$$

In the equation, f_{ψ}^{pre} is the prediction function for pre-training and θ, ψ are the parameters to be optimized in the self-supervised task. The downstream graph used to fine-tune the pre-trained model is typically defined as a new task involving a graph \mathcal{G}^{fin} and labels Y^{fin} . The conventional fine-tuning process is thus described as:

$$\begin{aligned} \theta^{\text{fin}}, \psi^{\text{fin}} &= \arg \min_{\theta, \psi'} f_{\psi'}^{\text{fin}}(\text{GNN}(\mathcal{G}^{\text{fin}}; \theta^{\text{init}}), Y^{\text{fin}}), \\ \text{s.t. } \theta^{\text{init}} &= \theta^{\text{pre}}, \mathcal{G}^{\text{fin}} \simeq \mathcal{G}^{\text{pre}}, \end{aligned} \quad (14)$$

where the parameters θ^{pre} of the pre-trained GNN model act as the initialized parameters for the GNN model in downstream tasks. Typically, the downstream graph \mathcal{G}^{fin} and the pre-training graph \mathcal{G}^{pre} are from the same or closely related domains. Given the fact that the downstream tasks often differ from the pre-training tasks, a new prediction function $f_{\psi'}^{\text{fin}}$ with parameter ψ' needs to be re-trained during fine-tuning. Note that graph prompt-tuning methods [27, 32] are designed to bridge the gap between pre-training and fine-tuning by reshaping the training objective with $f_{\psi'} = f_{\psi}$.

A.2 Proposed Multi-Source Pre-training

Different from the conventional single-domain pre-training, our proposed multi-domain pre-training can extract diverse transferable knowledge and effectively transfer it to the downstream tasks in an unseen domain. Specifically in this work, we use $\{\mathcal{G}_k | k = 1, 2, \dots, K\}$ to represent K subgraphs sampled from multi-source domains, which are heterogeneous in feature and label spaces. Additionally, a target graph represented by \mathcal{G}_* for the downstream tasks is from another distinct unseen domain. We formulate the general multi-domain pre-training process as:

$$\theta^{\text{pre}}, \psi^{\text{pre}}, \Phi^{\text{pre}} = \arg \min_{\theta, \psi, \Phi} f_{\psi}^{\text{pre}}(\text{GNN}(\{\mathcal{G}_k^{\text{pre}}\}; \theta, \Phi), \{Y_k^{\text{pre}}\}). \quad (15)$$

The equation is similar to Eq. (13) but the GNN model is pre-trained on multi-source domains, and additional parameters Φ are introduced to assist the pre-training process (i.e., the parameters

introduced in the LAMP framework). Corresponding to this, the fine-tuning process in the downstream task can be described as:

$$\begin{aligned} \theta^{\text{fin}}, \psi^{\text{fin}} &= \arg \min_{\theta, \psi'} f_{\psi'}^{\text{fin}}(\text{GNN}(\mathcal{G}_*^{\text{fin}}; \theta^{\text{init}}), Y_*^{\text{fin}}), \\ \text{s.t. } \theta^{\text{init}} &= h(\theta^{\text{pre}}, \psi^{\text{pre}}, \Phi^{\text{pre}}, \mathcal{G}_*^{\text{fin}}), \end{aligned} \quad (16)$$

where the downstream task is unspecified and originated from a distinct domain rather than the multi-source domains regarding $\{\mathcal{G}_k\}$. Furthermore, the initialized parameter θ^{init} is designed to be determined by $h(\cdot)$, which takes into account all parameters in the pre-training process and the downstream graph.

Compared to the conventional pre-training process, the proposed multi-source pre-training has two major differences:

(1) **Uniform GNN model:** It can integrate diverse effective knowledge and patterns to form a uniform GNN model through co-training on various multi-source domain graphs.

(2) **Better generalization:** It can generalize the pre-trained knowledge to an unseen graph domain, which can be adapted to various graphs without accessible pre-training data.

A.3 Important Notations

The important notations introduced in the main content of the paper are summarized in Tab. 5.

Table 5: Notations

Notation	Description
$\mathcal{G} = (X, A)$	The undirected graph with N nodes
$\{Z^{(l)}\}$	The outputs of the GNN layers
$\theta = \{\theta^{(l)}\}$	The parameters to be learned for the GNN model
L	The number of GNN layers
ψ	The parameters for the prediction function f_{ψ}
$\{\mathcal{G}_k\}, \mathcal{G}_*$	Graphs from the multi-source and target domains
$\{\phi_{\mathcal{G}_k}\}, \phi_{\mathcal{G}_*}$	The parameters for the transform function $\{g_k\}, g_*$
$\{\mathcal{S}_t\}$	The set of synthetic graphs
$\{\phi_{\mathcal{S}_t}\}$	The parameters to identify synthetic graphs
K, T	The numbers of the source domains and synthetic graphs
d	The feature dimension of the homogeneous synthetic graphs
$P_{\mathcal{G}}^r$	The r -walk empirical distribution for graph \mathcal{G}
R	The total walks considered for distribution matching
ω	The parameter for Wasserstein Discriminator f_{ω}
ϕ_m	The parameters for modulation operation f_{ϕ_m}
$c(\mathcal{G})$	The graph context
$\hat{\theta} = \{\hat{\theta}^{(l)}\}$	The GNN parameters after modulation

B ALGORITHMS AND COMPLEXITY ANALYSIS

Multi-domain Pre-training. Given a set of multi-source domains, the multi-domain pre-training process transforms the graphs from these diverse domains through $\{g_k\}$ and distills them into a set of synthetic graphs $\{S_t\}$ so that meta-knowledge can be extracted to pre-train a GNN model (θ, ψ) . A modulation architecture ϕ_m is also co-training to adapt the GNN parameters. During pre-training, the synthetic graphs are used to pre-train an effective GNN model and the modulation architecture. The outputs are the parameters $\theta^{\text{pre}}, \psi^{\text{pre}}, \phi_m^{\text{pre}}$ and the synthetic graphs $\{S_t\}$.

The overall process of pre-training is shown in Alg. 1. Assume that n_k, n_t and m_k, m_t are the average scales for nodes and edges corresponding to the sampled graphs $\{G_k\}$ and synthetic graphs $\{S_t\}$. During each iteration for pre-training, in Line 3, LAMP firstly samples subgraphs $\{G_k\}$ from source domains and transforms them, the complexity of which is $O(Kn_k + Kn_k d^2)$. Then in Line 4, LAMP constructs synthetic graphs $\{S_t\}$ from $\{\phi_{S_t}\}$, the complexity of which is $O(Tn_t d^2)$ when using MLPs for synthetic graph generation. In Line 5, knowledge approximation loss $\mathcal{L}_{\text{appr}}$ is calculated, costing $O(Ld(Km_k + Tm_t) + Ld^2(Kn_k + Tn_t))$ when a L -layer GCN serving as the backbone. In Line 6, the complexity of computing distribution matching loss \mathcal{L}_{dis} is $O(Rd(Km_k + Tm_t) + Rd^2(Kn_k + Tn_t))$. Lines 7–10 divide synthetic graphs into support S_t^{spt} and query S_t^{qry} graphs to calculate pre-training loss \mathcal{L}_{pre} , which needs a time complexity of $O(Ld(Tm_t + Ld^2 Tn_t))$. To sum up, the overall complexity of the pre-training process is $O(d(R+L)(Km_k + Tm_t) + d^2(R+L)(Kn_k + Tn_t))$, where n_k, n_t and m_k, m_t are usually small.

Algorithm 1 Pre-training Process for LAMP

Input: Multi-source domain graphs $\{G_k\}$; Unified node feature dimensions d for T synthetic graphs; Loss hyperparameters α, β
Output: Pre-trained parameters $\theta^{\text{pre}}, \psi^{\text{pre}}$ for GNN model and ϕ_m^{pre} for modulation; The synthetic graphs $\{S_t\}$

- 1: Initialize parameters $\{\phi_{S_t}\}, \{\phi_{G_k}\}, \theta, \psi, \phi_m$
- 2: **while** not converged **do**
- 3: Sample subgraphs $\{G_k\}$ from source domains and transform them with $\{\phi_{G_k}\}$.
- 4: Construct synthetic graphs $\{S_t\}$ from $\{\phi_{S_t}\}$
- 5: Calculate knowledge approximation loss $\mathcal{L}_{\text{appr}}$ [Eq. (3)]
- 6: Calculate distribution matching loss \mathcal{L}_{dis} [Eq. (5)]
- 7: **for** $S_t \in \{S_t\}$ **do**
- 8: Divide S_t into support S_t^{spt} and query S_t^{qry} graphs
- 9: **end for**
- 10: Calculate pre-training loss \mathcal{L}_{pre} [Eq. (9)]
- 11: $\mathcal{L} = \mathcal{L}_{\text{appr}} + \alpha \mathcal{L}_{\text{dis}} + \beta \mathcal{L}_{\text{pre}}$
- 12: Backpropagate \mathcal{L} to update $\{\phi_{S_t}\}, \{\phi_{G_k}\}, \theta, \psi, \phi_m$
- 13: **end while**
- 14: **return** $\theta^{\text{pre}}, \psi^{\text{pre}}, \phi_m^{\text{pre}}$; Synthetic graphs $\{S_t\}$

Target Domain Fine-tuning. On this stage, we train the transformation function g_* to match the distribution of the unseen target graph G_* with the synthetic graphs $\{S_t\}$. Then we modulate the pre-trained GNN model θ^{pre} with the modulation architecture ϕ_m^{pre} . By

Algorithm 2 Fine-tuning Process for LAMP

Input: The unseen domain graph G_* ; The pre-trained parameters θ^{pre} for GNN; The prediction function for pretext ψ^{pre} ; The modulation parameters ϕ_m^{pre} ; Synthetic graphs $\{S_t\}$

Output: A well trained GNN model

- 1: Match the graph distribution to get ϕ_{G_*} by $\mathcal{L}'_{\text{dis}}$ [Eq. (10)]
 - 2: Modulate pre-trained GNN as $\hat{\theta}^{\text{pre}}$ [Eq. (11)]
 - 3: Reshape the downstream task to match ψ^{pre}
 - 4: Fine-tune GNN model with $\hat{\theta}^{\text{pre}}, \psi^{\text{pre}}, \phi_{G_*}$ on G_*
 - 5: **return** Fine-tuned GNN model
-

reshaping the downstream task ψ^{fin} in the form of the pre-trained task ψ^{pre} , the GNN model can be fine-tuned.

The fine-tuning process is shown in Alg. 2. It's worth noting that the complexity of the distribution alignment procedure is $O(Rd(m_* + Tm_t) + Rd^2(n_* + Tn_t))$ where n_* and m_* are the average scales for nodes and edges corresponding to the target graph G_* . Such alignment can be processed during preprocessing. What's more, the context and task alignments adjust the GNN model's representation capability without changing the model definition. Therefore, LAMP does not introduce extra visible overhead during the model fine-tuning stage.

C FURTHER DESCRIPTIONS OF DATASETS AND BASELINES

The experiments are implemented in PyTorch with Python 3.6.8, executed on a system featuring an Intel Xeon E5-2620 v2 2.10GHz CPU, GeForce RTX 2070 8G GPU, and 64GB memory on a 64-bit CentOS Linux 7.2. We provide further details of the datasets and baselines.

C.1 Datasets

- **Product [P]** [11]: Product is an Amazon product co-purchasing network from Open Graph Benchmark (OGB). Nodes represent products sold on Amazon, and edges between two products indicate that the products are purchased together.
- **Yelp [Y]** [53]: The Yelp dataset is a social network, extracted from the data of businesses, users and reviews provided on the open challenge website Yelp. The node means user and the edge is the friendship.
- **Reddit [R]** [9]: This is a comment network constructed from Reddit, a large online discussion forum. 50 large communities have been sampled to build a post-to-post graph, connecting posts if the same user comments on both.
- **Academic [A]** [11]: Academic is a citation network consisting of papers indexed by MAG [40]. Each node is a paper and each edge indicates that one paper cites another one.

These networks are from different graph domains. Academic and Product are from Open Graph Benchmark (OGB), and the node labels for Yelp are multi-label.

C.2 Baselines

- (1) *Graph supervised learning methods.*

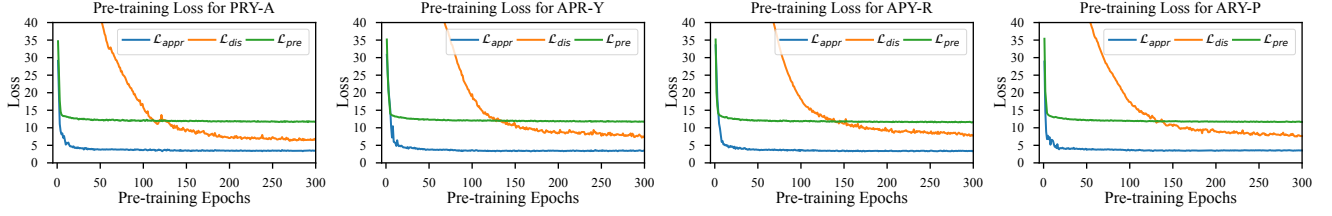


Figure 4: The results of continuous epochs for different loss functions during pre-training.

- **GCN** [19]: GCN resorted to weighted average-based neighborhood aggregation to receive messages from the neighboring nodes for node representation learning in an end-to-end manner.
- **GAT** [36]: GAT also used neighborhood aggregation for node representation learning in an end-to-end manner. At the same time, it assigned different weights to neighbors according to a self-attention operation.
- **GIN** [48]: GIN employed a sum-based aggregator, which is more powerful in expressing the graph structures.

(2) Graph self-supervised learning methods.

- **EdgePred** [12]: EdgePred randomly masked partial edges and then trained GNNs to reconstruct them.
- **AttrMask** [14]: AttrMask randomly masked partial node attributes and then trained GNNs to predict them.
- **DGI** [37]: DGI capitalized on a self-supervised method for pre-training, which is based on the concept of mutual information (MI). It maximized the MI between the local augmented instances and the global representation.
- **GCC** [30]: GCC leveraged contrastive learning to capture the universal network topological properties across multiple networks. Since it doesn't consider node properties, we assemble it with a GCN model during fine-tuning.
- **GPT-GNN** [11]: GPT-GNN utilized generative model pre-training and derived an autoregressive framework to perform reconstruction on given graphs.
- **L2P-GNN** [28]: L2P-GNN simulated the fine-tuning step during pre-training to enable the pre-trained models to learn to fine-tune for the downstream tasks.

(3) Graph prompt learning methods

- **GPPT** [32]: GPPT pre-trained a GNN model based on the link prediction task, and employed a learnable prompt to reformulate the downstream node classification task into the same format as link prediction.
- **GraphPrompt** [27]: GraphPrompt employed a learnable prompt to assist a downstream task in locating the most relevant knowledge from pretext.
- **ProG** [33]: ProG unified the format of graph prompts and used meta-learning to learn a better prompt initialization.

D PRE-TRAINING LOSS ANALYSIS

In this section, we investigate the loss functions associated with the pre-training stage of our proposed framework, LAMP. The trajectories over successive epochs for distinct loss functions, denoted as \mathcal{L}_{appr} , \mathcal{L}_{dis} and \mathcal{L}_{pre} , are illustrated in Fig. 4. The figures show

that each of these three loss functions experiences a gradual decrease, reaching a relatively low level finally. This suggests that the pre-training stage of LAMP can successfully converge, yielding an efficacious pre-trained GNN model.

Notably, both \mathcal{L}_{appr} and \mathcal{L}_{pre} exhibit rapid declines, reaching convergence within the initial 50 epochs of the pre-training process. Conversely, \mathcal{L}_{dis} initially manifests as notably high, indicative of prominent distribution divergences between source and synthetic graphs. However, as the pre-training advances, \mathcal{L}_{dis} gradually diminishes, eventually aligning with the same levels observed for \mathcal{L}_{appr} and \mathcal{L}_{pre} . In summary, the results demonstrate that LAMP can not only produce a set of homogeneous synthetic graphs to facilitate the effective distillation of generalized knowledge from multi-source graphs, but also pre-train an effective GNN model with the derived meta knowledge.

E GENERALIZATION ABILITY FOR PRE-TRAINED DOMAINS

Here, we investigate the effectiveness of our proposed LAMP framework when the target domain is drawn from the multi-source domains. It is essential to emphasize that LAMP is specifically designed for handling unseen target domains, which is a more challenging scenario than this. Therefore, we pre-train GNN models based on four datasets (i.e., APY, R, P, and Y) and fine-tune them in the tasks from one of the four domains (i.e., A, P, R, or Y). We also evaluate LAMP and the baselines from the perspectives of training efficiency and final results in node classification and link prediction tasks.

Training Efficiency. The results of training efficiency with 10% labels or edges known are shown in Fig. 5. Notably, the performance of LAMP, depicted by the purple line, consistently outperforms the baseline methods. These findings conform to the results observed in the context of unseen domains, as illustrated in Fig. 2. This consistency not only underscores the efficacy of LAMP in addressing tasks involving diverse domains but also highlights its remarkable generalization capability and robustness when subjected to pre-training on multi-source graph domains. A comparison with the outcomes presented in Fig. 2 reveals that most baselines, particularly those associated with self-supervised methods such as AttrMask and DGI, demonstrate improved performance. This improvement can be attributed to the fact that the domains they operate on have been encountered during pre-training. However, it is noteworthy that their performances fall short of that achieved by LAMP. This disparity arises due to the limitations of these methods in handling heterogeneous distributions and negative transfer, which the comprehensive approach of LAMP can effectively address.

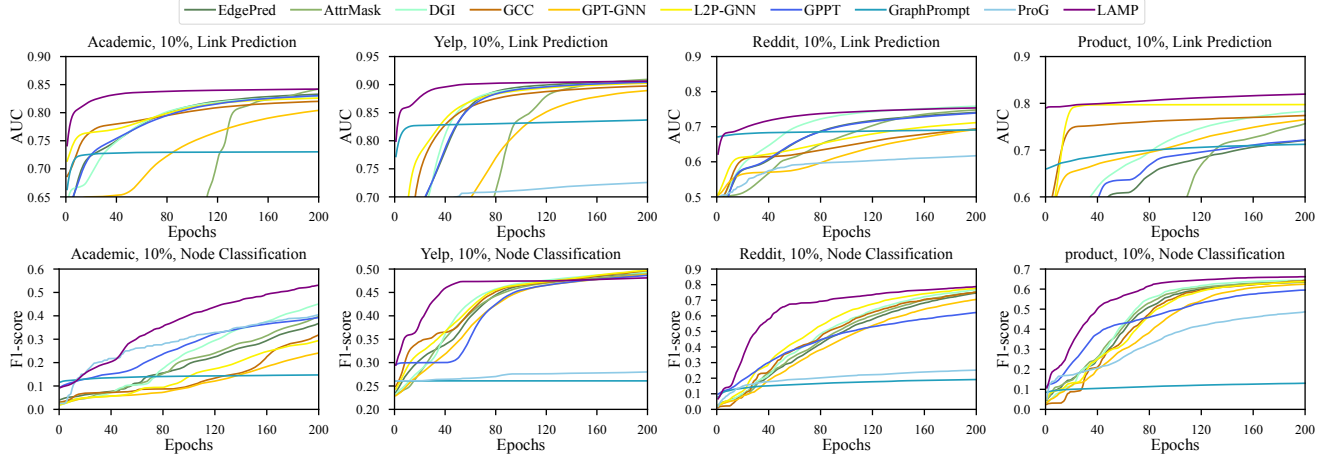


Figure 5: The results of continuous epochs for different methods in link prediction and node classification tasks.

Table 6: The final results for different methods with different fractions of known labels.

Methods	The final AUC (%) for link prediction								The final F1-score (%) for node classification							
	10% known edges				20% known edges				10% known labels				20% known labels			
	Acadmemic	Yelp	Reddit	Product	Acadmemic	Yelp	Reddit	Product	Acadmemic	Yelp	Reddit	Product	Acadmemic	Yelp	Reddit	Product
EdgePred	84.82	91.60	77.09	80.49	89.35	92.24	81.96	87.18	55.13	49.82	81.97	64.39	64.36	49.52	88.01	71.85
AttrMask	85.18	91.82	77.12	82.94	89.36	92.31	81.49	88.55	55.77	49.94	81.76	64.45	64.81	49.59	87.79	71.79
DGI	84.85	91.45	77.48	81.81	89.68	92.06	82.49	88.21	57.42	49.68	81.83	64.98	65.53	49.43	87.81	72.19
GCC	83.08	90.79	73.79	79.60	88.01	91.76	79.64	86.93	52.49	51.09	81.92	65.11	62.67	50.26	88.16	72.35
GPT-GNN	83.40	90.93	74.56	77.77	87.65	91.73	79.84	84.18	51.02	50.92	81.52	64.60	61.41	50.25	87.33	72.00
L2P-GNN	83.50	91.19	74.86	80.21	89.23	92.15	81.82	87.37	51.86	50.99	82.17	64.96	60.43	49.98	82.15	70.92
GPPT	84.56	91.52	77.09	80.72	88.37	91.92	80.31	87.21	48.99	50.44	70.24	62.57	61.93	50.25	87.88	72.23
GraphPrompt	73.05	84.93	69.69	71.90	75.53	85.76	73.18	77.52	15.44	26.09	25.19	14.73	16.61	24.00	27.93	14.91
ProG	63.33	74.23	63.47	56.70	59.91	74.52	60.62	56.49	48.27	31.70	45.26	52.20	56.43	31.58	52.68	63.97
LAMP	85.38	91.43	76.30	83.31	90.26	92.36	82.24	89.78	59.64	50.75	83.74	67.08	66.76	50.36	89.71	74.18

Final Results. The final results with 10% and 20% labels or edges known are shown in Tab. 6. Across most scenarios, LAMP consistently outperforms the baseline methods, showcasing its superior performances, except for Yelp and Reddit with 10% edges known in link prediction, Reddit with 20% edges known in link prediction, and Yelp with 10% labels known for node classification. Notably, the performance differences between LAMP and the second-best method frequently exceed 2%, underscoring the notable advantages of LAMP. Similar to the analysis of training efficiency, it is evident that the baselines can leverage pre-trained domain knowledge to some extent, leading to improved performances. However, the challenges associated with heterogeneous distributions and negative transfer remain limiting factors for these methods. In summary, whether dealing with pre-trained or previously unseen domains, LAMP consistently demonstrates its capacity to handle both scenarios, yielding superior performance outcomes.

F RESULTS FOR GRAPH CLASSIFICATION

This section presents the results when the downstream task is graph classification. For the task alignment of graph classification,

we utilize a virtualized graph node that establishes unidirectional links with all existing nodes within a graph to derive the graph representation. Similar to the node classification task, we reshape the graph classification task as the pretext task by predicting the links between graph nodes and labeled nodes.

In this way, we pre-train GCN models based on four datasets introduced in the main experiments (i.e., APRY). Subsequently, we fine-tune these models on graph classification tasks sourced from the domain of molecular informatics. Specifically, we select four datasets, DHFR, BZR, COX2, and PROTEIN-full, from TUDatasets [29], a well-established benchmark for graph classification. We allocate 10 to 20 labeled graphs for training during experimentation, reserving the remaining graphs for testing. The performances of LAMP and the baseline methods are visualized in Fig. 6 and summarized in Tab. 7. It is important to note that GPPT is excluded from the baseline methods as it is not applicable to graph-level tasks.

Fig. 6 illustrates the training efficiency of the methods in the graph classification task. A notable observation is the distinct curve patterns compared to those observed in node classification and link prediction tasks. Generally, with the exception of dataset DHFR,

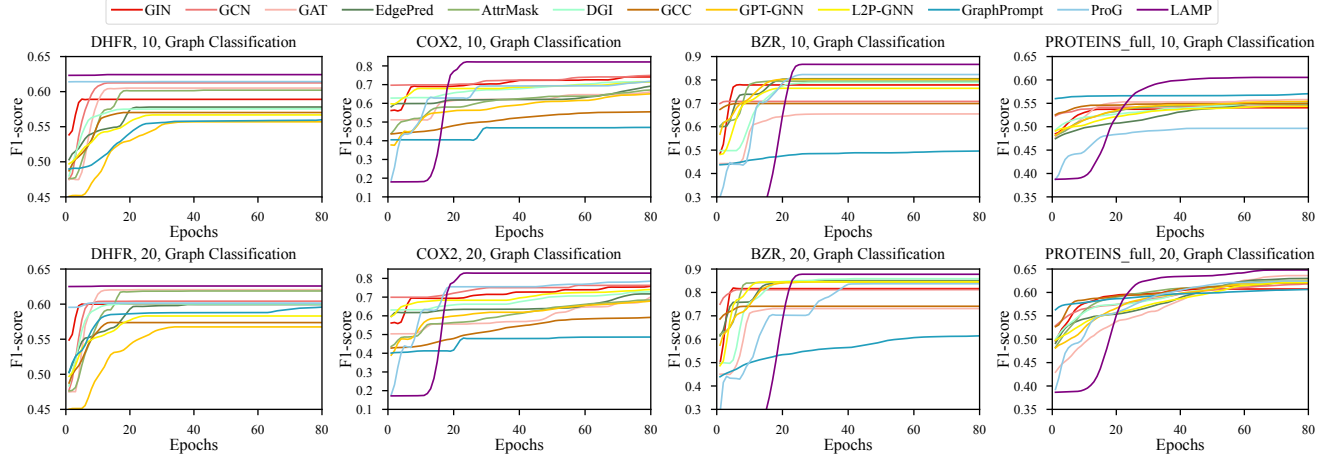


Figure 6: The results of continuous epochs for different methods in graph classification tasks.

the curves of LAMP in the other three datasets start at a lower level compared to the baselines. However, as the number of epochs increases to 20, the curves rise rapidly, resulting in significantly improved performances compared to the baselines. However, as the number of training graphs increases, the superiority of LAMP tends to weaken. In the case of DHFR, LAMP demonstrates promising performance with the pre-trained model from the beginning, with slight further improvement observed as training progresses. This suggests that LAMP effectively transfers positive knowledge to the target unseen domain, outperforming the baselines noticeably.

Tab. 7 presents the final results for the downstream graph classification tasks. The results indicate that LAMP achieves improvements of up to 1% in DHFR, 4.5% in BZR, 4.3% in COX2, and 2.4% in PROTEIN-full compared to the corresponding second-best baseline. Furthermore, when training graphs are scarce, LAMP notably outperforms other baselines, underscoring its ability to transfer effective knowledge across different domains and tasks with better capability of generalization.

Table 7: The final results for different methods with different numbers of known labels.

Methods	10 known labels				20 known labels			
	DHFR	BZR	COX2	PROT.	DHFR	BZR	COX2	PROT.
GCN	61.25	70.82	77.52	54.48	60.42	81.22	78.05	62.36
GIN	58.90	77.86	76.89	54.90	60.00	81.62	77.06	61.55
GAT	60.51	65.46	77.81	55.79	62.05	73.10	76.76	64.27
EdgePred	57.81	80.51	75.11	55.06	59.94	85.07	73.59	63.03
AttrMask	60.21	79.49	72.26	54.78	61.87	84.67	70.95	62.53
DGI	57.60	78.96	74.67	55.01	60.03	85.88	75.49	62.46
GCC	57.02	69.92	56.01	55.13	57.37	74.09	59.75	62.26
GPT-GNN	55.70	80.23	70.88	55.49	56.78	84.49	70.32	62.80
L2P-GNN	56.68	76.42	74.79	54.56	58.31	84.58	75.29	62.16
GraphPrompt	56.58	55.41	65.09	58.16	60.08	63.88	63.09	63.40
ProG	61.43	82.31	77.66	49.71	60.44	83.71	79.03	62.82
LAMP	62.43	86.76	82.19	60.54	62.58	89.45	82.92	64.83