Handling Feature Heterogeneity with Learnable Graph Patches

Yifei Sun
Zhejiang University
Hangzhou, China
yifeisun@zju.edu.cn

Yang Yang*
Zhejiang University
Hangzhou, China
yangya@zju.edu.cn

Xiao Feng
Zhejiang University
Hangzhou, China
functionendless@zju.edu.cn

Zijun Wang
Zhejiang University
Hangzhou, China
zwang745@ucsc.edu

Haoyang Zhong
Huazhong University of Science and Technology
Wuhan, China
haoyangzhong@hust.edu.cn

Chunping Wang
Finvolution Group
Shanghai, China
wangchunping02@xinye.com

Lei Chen
Finvolution Group
Shanghai, China
chenlei04@xinye.com

Abstract

In recent years, the rapid development of foundation models and graph pre-training technologies has spurred increasing interest in constructing a universal pre-trained graph model or Graph Foundation Model (GFM). However, a significant challenge is that existing models are unable to address feature heterogeneity in graph data without textual information, which hinders the transferability of graph models across different datasets. To bridge this gap, we propose the concept of *learnable graph patches*, which we regard as the smallest semantic units of any graph data. We decompose the graph into learnable graph patches by unfolding the node features and constructing corresponding patch structures separately. We then design PatchNet[1], a framework that mines transferable information from graph data across domains. Specifically, after extracting graph patches, we propose a patch encoder to extract knowledge from each unit and a patch aggregator to learn how the units is combined into a whole. Due to its domain-agnostic nature, the model can be applied to downstream data across different domains. Furthermore, we analyze the connection between PatchNet and existing graph models, as well as the transferability of the node embeddings it generates. Empirically, our approach not only achieves the capability to use multi-domain graphs for pre-training but also demonstrates continuous improvement in various downstream datasets and tasks. Moreover, we observe consistent improvement in downstream performance as the volume of pre-training data increases.

*Corresponding author.
[1]Code is available at https://github.com/zjunet/PatchNet.

KDD '25, August 3–7, 2025, Toronto, ON, Canada

ACM ISBN 979-8-4007-1245-6/25/08
https://doi.org/10.1145/3690624.3709242

CCS Concepts

• **Networks** → **Network algorithms**.

Keywords

graph neural networks, graph pre-training, feature heterogeneity

ACM Reference Format:
Yifei Sun, Yang Yang, Xiao Feng, Zijun Wang, Haoyang Zhong, Chunping Wang, and Lei Chen. 2025. Handling Feature Heterogeneity with Learnable Graph Patches. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3690624.3709242

1 Introduction

Graph, as a prevalent data structure, is ubiquitous in a wide range of applications, such as social network analysis [51], bio-informatics [53], finance [52], etc. These graphs form a vast knowledge base, encompassing rich and comprehensive information across various fields. With the advancement of graph neural networks (GNNs) and pre-training techniques, the development of universal graph pre-train models or graph foundation models (GFMs) to absorb knowledge across diverse graphs has gained significant attention [18, 22].

However, efforts to create transferable pre-trained models have faced numerous challenges. The reason is that graph data exhibits more complex heterogeneity compared to images in computer vision (CV) and sentences in natural language processing (NLP). One reason is that from a structural perspective, graph data exhibits vastly different patterns across various domains. For example, benzene rings are commonly found in molecular graphs, while triangles frequently appear in social networks. Fortunately, this issue has seen considerable breakthroughs in recent research [26, 62].

On the other hand, feature heterogeneity poses significant challenges and has been less explored due to its complex nature. Specifically, the sources of node information in graph data can be various, and the process of converting raw data into features also varies greatly. This leads to node features in different graph datasets potentially existing in entirely different semantic spaces. For example,

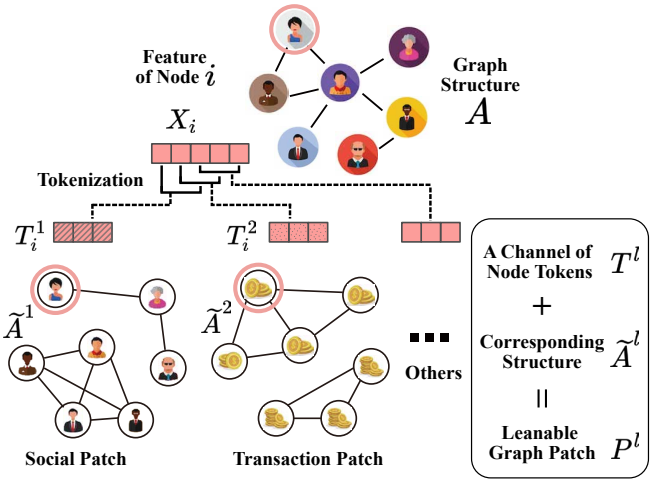

Figure 1: Decomposing a graph into graph patches. The upper side is an input graph (financial network as example). The lower side shows two of the decomposed graph patches. The first patch represents a social network, utilizing the social attributes of users for its node token, with edges indicating similarities in social aspects. The second patch is a transaction network, featuring nodes characterized by transactional information, where edges reflect transactional similarities.

node features in molecular graphs might be manually annotated by experts in laboratories without further processing, whereas in financial networks, features could be derived from tabular data using some feature selecting method.

One existing solution involves introducing text feature and using LLMs to map these into a common space [17]. However, most graph data do not include text as part of the node information, and in some cases, the original node information cannot be provided due to privacy concerns. Another approach is to use simple singular value decomposition (SVD) to convert feature to the same length [61], but this only standardizes feature length without explicitly aligning node features across different graphs. Thus, effectively addressing feature heterogeneity on graphs becomes crucial for building a unified graph pre-training model.

The core challenges in handling Feature Heterogeneity include, firstly, the difficulty in extracting meaningful and transferable information from the vastly diverse features across different graph data. Secondly, how to preserve such transferable information using a scalable model during the pre-training process also presents a significant challenge. In this paper, we propose that the association of feature structures could be the key to transferability. Hence, we aim to consider both node features and structural information when addressing feature heterogeneity. The key idea is to design a *learnable graph patching module* which is adaptable to various kinds of feature and train it to extract the transferable information to boost performance of downstream tasks.

First, we propose that each graph can be seen as a complex object consisting of graph patches. As the example shown in Fig. 1, the financial network on the upper side can be decomposed into a series of graph patches, including a social graph patch and a transaction graph patch, among others. Each graph patch contains a piece of relatively independent and transferable information from the original graph. Note that the node tokens T of each graph patch P are derived from the unfolding of the original node features, and its graph structure $\widetilde{A}$ is learned from both the original structure and the node tokens. We propose that such graph patches preserve the basic transferable information between graphs, as other graphs may also be decomposed into similar social and transactional information.

Second, we propose to build a scalable graph model, PatchNet, to encode the captured information in the learnable graph patches. Specifically, we first extract learnable graph patches from original graph, which are composed of node tokens and corresponding structure. Since the transferable information is hidden in the graph patches, we design a patch encoder to aggregate graph patch of each channel. Then we build a patch aggregation module to learn how to combine these transferable information across graph patches. Our method is the first one that can be pre-trained on data across domains without the need for text or other side information, and can be applied to downstream data across domains. Additionally, the high degree of modularity of our model allows for stacking to scale up the model's parameter. Finally, we evaluate our model on both existing settings and our new setting. The results indicate that this is a promising step toward building a GFM.

The contribution of this paper are summarized as follows:

- We propose the concept of learnable graph patches to handle feature heterogeneity, which enables the preserving of multi-domain knowledge across different graphs.
- We propose a graph model, PatchNet, composed of patch encoder and patch aggregator for generalized graph pre-training. We analyze the obtained embedding quality and the correlation between PatchNet and existing models.
- We empirically prove the effectiveness of PatchNet in various settings which demonstrate the superior performance across multiple graph datasets and tasks.

2 Related Work

Tackling feature heterogeneity. Developing transferable GNNs or GFM has always been a hot topic of research in the graph community. One of the great challenges is dealing with feature heterogeneity [18, 22]. Existing methods for graph pre-training or transfer learning suffer from feature heterogeneity due to varying feature origins and semantic spaces. OFA [17] manually converts heterogeneous features into textual descriptions using LLMs. However, most of the graphs mainly contain feature attributes without either side information like text or homogeneous origin. GCOPE [61] employs domain-specific virtual nodes that serve as inter-connectors linking nodes across various domains. Although these virtual nodes enable connections between domains, they only applied SVD to address feature heterogeneity without explicitly capturing the transferable information. However, we propose to use learnable graph patches to capturing transferrable information within feature heterogeneity.

Graph patching or tokenization. Due to the complex non-Euclidean nature of graph data, many existing studies have addressed the associated challenges by decomposing graphs into

smaller units like patches or tokens to enable more effective modeling and information extraction. These methods can be categorized into 4 types based on the granularity of the decomposition: node-level, path-level, subgraph-level, and combination-level. The most common approach is node-level decomposition, a method employed by various graph transformers [6, 24, 28, 54]. The advantage of this level is the natural way of segmentation, although it does not reduce the complexity of the graph structure. Next is the path level: PathNet [35] enhances the discriminability of graph models on heterophily graphs by capturing path patterns, while PathNNs [23] theoretically demonstrate that capturing full-range paths can enhance both expressiveness and model performance. The subgraph level includes NAGphormer [2], which efficiently improves node classification on large graphs by aggregating each layer of neighbors as separate patches, and GPatcher [60], which proposes aggregating p neighbors for each node based on topology and node features in heterophily graphs; Graph ViT [9] employs METIS for graph segmentation, validating that ViT-like approaches can also be effective on graphs. The final category is the combination level, to which our method belongs. We propose that learning such patch is transferable across graphs because these learnable graph patches can learn common unit information across graphs even with feature heterogeneity.

3 Method

Notations. We denote a graph as $\mathcal{G} = (V, A, X)$ and N is the number of nodes, $A \in \{0,1\}^{N\times N}$ is the adjacency matrix and $X_{\text{ori}} \in \mathbb{R}^{N\times D}$ is the node attribute matrix where D is the dimension of attributes that differs across domains. Due to feature heterogeneity, we normalize the node attributes as $X \in [0,1]^{N\times D}$.

Overview. The overall architecture of PatchNet is shown in Fig. 2 and the full algorithm can be found in Appendix. As discussed in Sec. 1, the goal is to handle feature heterogeneity between graphs in order to transfer the knowledge from various pre-training datasets into target downstream datasets. Moreover, we propose that the key is to build learnable patches that can capture the inherent feature and structure correlation. After obtaining graph patches, we first perform an encoding within each graph patch to obtain the inner-patch embedding, known as the Patch Encoder. Subsequently, we aggregate across all patches, referred to as Patch Aggregation, to produce the final node embedding F, which can be used for node classification or graph classification after pooling. This complete processing represents one layer of PatchNet, meaning that the entire process can be repeated multiple times to capture more extensive and richer patch information.

3.1 Building Structure-aware Graph Patches

The construction of graph patches consists of two parts: the formulation of node tokens and patch structure learning.

Definition 3.1 (Unfolding Node Tokens). Given a graph $\mathcal{G} = (V, A, X)$, the normalized attributes of each node $X_i \in [0,1]^D$ can be unfolded into node tokens $T_i \in [0,1]^{K\times M}$, where K is the number of channels of node tokens and M is the token size. This can be formalized as follows: $T_i^l = X[i, start_l : end_l], start_l = S\times l, end_l = S\times l + M, l = 0, ..., K$, M is the size of each token, step S refers to the stride of the unfolding window, which can be calculated by $S = \lfloor \frac{D-M}{K} \rfloor$. Here, each token $T_i^l \in [0,1]^M, l \in 1, ..., K$ is a smaller information unit of node. Each channel of tokens for all nodes is denoted as $T^l \in [0,1]^{N\times M}$. Hence, the node token matrix of this graph is $T \in [0,1]^{N\times K\times M}$.

The node token in Fig. 1 is represented by the tokenized pink vectors, which are extracted from the original node features. The core idea is to use the tensor unfold operation to losslessly expand each node's features into tokens. This unfold operation resembles the first half of a convolution operation, which is akin to spreading out potentially useful information. The single channel of tokens for all nodes in the graph contains a single perspective of all features.

Relation between node tokens. We explain the following two relations: The relation between node tokens on the same node T_i^1, T_i^2 and the relation between node tokens across different nodes T_1^l, T_2^l. Note that subscripts represent node indices, while superscripts denote token or patch indices. $T_i^l = X[i, start_l : end_l]$, the neighboring T_i^1, T_i^2 share the overlap feature. Since $start_1 < start_2 < end_1 < end_2$ and $end_1 - start_2 = (M - S)$, the shared part between T_i^1, T_i^2 is a subset of token T_i^1. Since the process of splitting tokens from node features does not involve interactions between nodes, the relationship between T_1^l, T_2^l is similar to that between X_1, X_2, meaning that they are relatively independent.

After we formulate tokens from each node with the original structure (dotted edges in Fig. 2), we then introduce a structure learning method to derive learnable graph patches based on the original graph structure and node tokens. This part is our key to handling feature heterogeneity across graphs.

Definition 3.2 (Learning Graph Patches). Given a channel of tokens T^l and the original graph structure A, a graph patch is denoted $P^l = \left(T^l, \widetilde{A}^l\right)$, where $l = 1, ..., K$. Here, $\widetilde{A}^l \in \{0,1\}^{N\times N}$ is learned for each token channel l with a shared graph learner Ψ in parallel. That is, $\widetilde{A}^l = \Psi(T^l, A)$.

Thus, a graph can be divided into K learnable graph patches for further modeling. We propose that the key to tackling feature heterogeneity is to find/learn the transferable information across different graphs. Here we propose to design a learnable module to extract the graph patches that contain feature-structure correlation information. In other words, the learned graph patches themselves are not transferable, but the learning mechanism within the parameters of graph learning module is transferable. Note that the graph learner module are learned simultaneously with patch encoder and aggregator which are introduced in following sections.

There are many ways to construct such a graph learner module. In this paper, we utilize an attention-based approach.

$$S_{i,j} = f_\phi(W \odot T_i^l, W \odot T_j^l), \tag{1}$$

$$\widetilde{A}_{i,j}(T) = \begin{cases} \sigma(S_{i,j}), & j \in \text{top-k}(S_{i,:}) \\ 0, & j \notin \text{top-k}(S_{i,:}) \end{cases}, \tag{2}$$

where T_i^l and T_j^l denote two node tokens, $\odot$ is the Hadamard operation, W is a learnable parameter vector, f_ϕ denotes the similarity metric such as cosine similarity, σ stands for non-linear activation function like relu. Moreover, we employ residual connections to

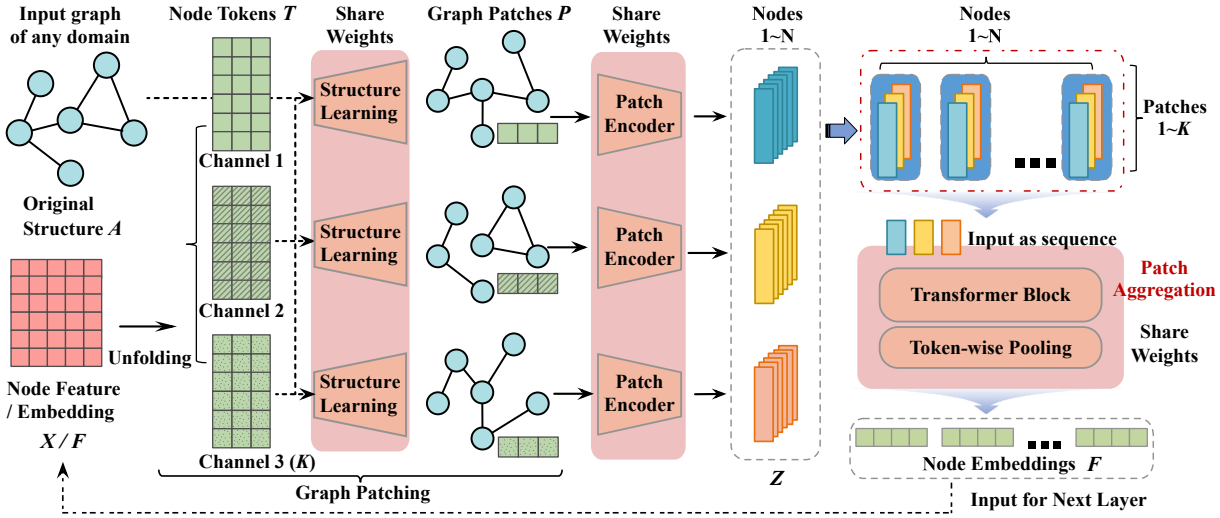

Figure 2: Overall architecture of PatchNet. (a) Building Graph Patches: The input node attributes X are unfolded into multiple node tokens, which are then paired with a graph learner to form patches. (b) Encoding Patches: Each patch is encoded using a shared-parameter encoder, resulting in patch embeddings Z without information passing between patches. (c) Aggregating Patches: Patch embeddings are aggregated within each node to yield the final node embeddings F.

update the learned $\widetilde{A}$ to accelerate and stabilize the training process:

$$\widetilde{A} = \alpha\widetilde{A} + (1-\alpha)A \tag{3}$$

where α stands for the trade-off parameter for how much trainable structure to adopt.

3.2 Theoretical Analysis of Graph Patches

Fundamental assumption. The success of transfer learning relies on the pre-training and downstream data having similar distributions as model inputs [63]. Compared to the original graph, patches of the same granularity are more likely to exhibit similar distributions, as the extracted tokens have the same dimensions, and the patch structures are learned by a transferable graph learner. This enables the model to transfer learned knowledge effectively when modeling and aggregating patches across different datasets.

Physical meaning. Any graph data is partitioned into learnable graph patches $P^l = (T^l, \widetilde{A}^l)$ with the same token feature space $T^l \in [0,1]^{N\times M}$ and the same space of learned structure $\widetilde{A}^l \in \{0,1\}^{N\times N}$, representing the division into smaller graph patches with the same Cartesian Product space of $T\times\widetilde{A}$. Ultimately, patching reduces the distributional differences between $\Pr(T(G_a), \widetilde{A}(G_a))$ and $\Pr(T(G_b), \widetilde{A}(G_b))$ of different graphs G_a, G_b.

Relation among patches. Note that when generating node tokens T^1, T^2 from the original node features, we use an unfolding operation with overlap ($T^1 \cap T^2 \neq \emptyset$). The $\widetilde{A}$ of each patch is learned based on the same A. Thus, the information between two adjacent patches (adjacent in the token splitting process) partially overlaps, which means these patches are not entirely independent. On the other hand, since the node tokens obtained through unfolding can be restored via the folding operation ($\bigcup_{l=1}^{K} T_i^l = X_i, i = 1, ..., N$), all the patches can be combined to reconstruct the original graph ($\bigcup_{l=1}^{K} P^l = G(X, A)$).

3.3 Encoding and Aggregating Graph Patches

In this section, we introduce scalable graph patch encoder and aggregator to extract transferable knowledge in graph patches.

Encoding Graph Patches. We propose a dual-branch attention mechanism to adaptively encode the patches. Specifically, given each token $T^l \in [0,1]^{N\times M}$, the original A and learned $\widetilde{A}$, the l-th patch are encoded in following two steps: we first use GNN with shared parameters to respectively encode both the learned patches and the combination of the original graph with node tokens.

$$H^l = \Phi(T^l, A), \quad \widetilde{H^l} = \Phi(T^l, \widetilde{A}(T^l)), \tag{4}$$

where Φ denotes a GNN. This results in embeddings $H^l, \widetilde{H^l}$, both in $\mathbb{R}^{N\times \text{hid}}$. Then we combine both H^l and $\widetilde{H^l}$ to get the representation of the l-th patch of the whole graph.

$$f_{\mathbf{MLP}}(\widetilde{H^l} || H^l) = E \in \mathbb{R}^{N\times 2}, \quad \delta(E) = \beta \in (0,1)^{N\times 2}, \tag{5}$$

where $f_{\mathbf{MLP}}$ is a two-layer MLP and δ stands for Softmax operation. The Softmax operation is applied along the second dimension to produce normalized importance scores β. Finally, we utilizes the scores β to combine $H^l, \widetilde{H^l}$.

$$Z^l = \sum \delta(E), \tag{6}$$

where the summation is applied along the second dimension. The output $Z^l \in \mathbb{R}^{N\times \text{hid}} (l = 1, ...K)$ is encoded patch embedding for N nodes.

Aggregating Graph Patches. For images or natural languages, after extracting the patches, it is vital to combine with positional embedding before aggregating the patches. However, the positional embedding of tokens, and even that of each dimension of node attributes, is meaningless for graph data. For example, in a social network where users are nodes, swapping the age and gender attributes does not affect the information of the nodes or the graph itself. Thus, we directly feed the patch embeddings into the patch aggregation process. Moreover, since we have node-wisely (along the 1st dimension) aggregated the tokens with graph structure, we propose to patch-wisely (along the 2nd dimension) aggregate the patches for every node.

One straight way is to aggregate all the patches through a MLP-based module. However, since PatchNet is designed to transfer across different domains which can not guarantees the same number of patches, we propose to employ a module with unlimited capacity of aggregation length, transformer block. It enables PatchNet to capture contextual information throughout the entire feature. Given $Z \in \mathbb{R}^{N\times K\times \text{hid}}$ as input, the

$$U = \text{LayerNorm}(\text{MHA}(Z)), \tag{7}$$

$$U' = \text{LayerNorm}(\text{FFN}(U)), \tag{8}$$

$$F = \text{Pooling}(W') \tag{9}$$

where MHA function computes the attention for each of the h heads and concatenates them together, FF function applies two linear layers with ReLU activation in between, LayerNorm is used to provide training stability and the Pooling refers to pooling along the direction of K patches. Thus, we get the final embeddings $F \in \mathbb{R}^{N\times f}$. Overall, the time complexity of patch aggregation is $O(NK^2)$.

It is noteworthy that, through the aggregation within patches and between patches, only one layer of the proposed framework is completed. Similar to the concept of the layer in classical GNNs, we can stack multiple layers of PatchNet. Specifically, when using PatchNet for the first layer, the input features are the original node features. In the second layer of using PatchNet, we perform a new layer of graph patch construction and modeling based on the node embeddings generated in the first layer.

3.4 Pipeline and Intuition

In this section, we aim to illustrate the process of using PatchNet during pre-training and fine-tuning, and to explain the intuition behind our model's ability to handle feature heterogeneity.

Detailed Implementation. Firstly, we shuffle multiple datasets and feed them into PatchNet. We segment all graphs into graph patches and encoding and aggregating graph patches to get node embedding. Then, we use existing self-supervised tasks to enable PatchNet to learn transferable information across graph datasets. During downstream fine-tuning, the downstream data must also be segmented according to the patching steps shown in Sec. 3.1. The major difference in fine-tuning is that the tasks are switched to downstream tasks. After fine-tuning on the training data, we use the tuned PatchNet to infer on the test set.

Here we give the analysis of time complexity of PatchNet. The pre-processing the tokens takes $O(NKM)$, where number of patches is K and token size is M. The encoding of structure-aware graph patches is divided into two parts: For the structure learning part, we account for complexities involving locality-sensitive kNN sparsification post-processing [8], where neighbors of each node are selected from a batch of nodes (batch size = b, hidden dimension $\approx M$). Thus, the complexity is $O(NM^2 + NMb)$ [21]. The GNN aggregation typically takes $O(N + E)$. Lastly, patch Aggregation takes $O(NK^2)$. Hence, the overall complexity is: $O(N(KM + M^2 + Mb + K^2))$.

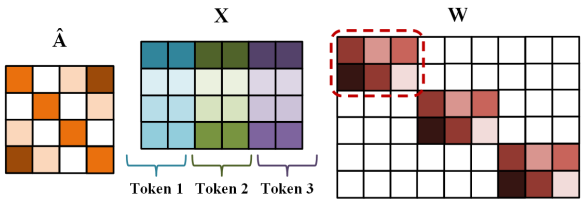

Figure 3: The decomposed operation of PatchNet from the perspective of MPNN.

Intuition. Here we intuitively explain why the proposed graph patch learning is transferable to tackle feature heterogeneity. The success of transfer learning relies on the pre-training and downstream data having similar distributions as model inputs [63]. Compared to the original graph, patches of the same granularity are more likely to exhibit similar distributions, as the extracted tokens have the same dimensions and the patch structures are learned by a transferable graph learner. This enables the model to transfer learned knowledge effectively when modeling and aggregating patches across different datasets. Note that Fig. 1 is merely illustrative, assuming that in a financial network, some features represent social information while some represent transaction information. In reality, the token features in the data may not always correspond to clearly defined or linguistically describable aspects. Thus, we use the the learnable parameters in structure learning module to automate learning the patch construction.

Since we have learned the transferable information across datasets during patch extraction, we further preserve these information by encoding and aggregating the patches. Then, we use encoder to acquire knowledge within graph patches with the same information granularity. Subsequent transformer-based patch aggregation learns how different graphs combine these graph patches. In summary, the information transferred by PatchNet is encapsulated within the extracted graph patches. The richer the variety of graph patches during pre-training, the more likely it is that similar graph patches will be encountered in downstream data, thus enhancing downstream performance.

3.5 Model Analysis

Connection to others. Compared to the existing graph pre-training backbone, where only transferable information could be learned from a complete graph, dividing the graph into patches allows for learning both fine-grained inner-patch patterns and correlations between coarse-grained inter-patches information. In this section, we analyze the correlation between PatchNet, existing MPNNs and other graph transformers.

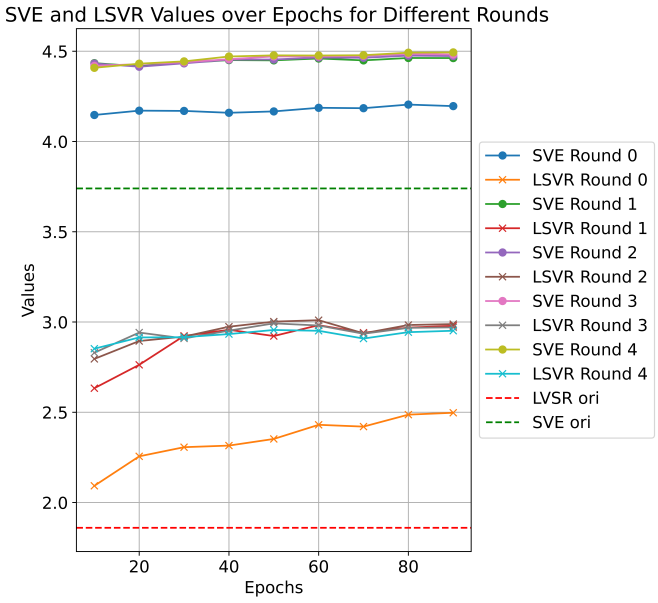

Figure 4: The SVE and LSVR of the embedding generated by PatchNet.

Theoretically, PatchNet can be seen as a decoupled version of existing MPNNs. Formally, each layer of an MPNN can be formulated as the combination of "propagation" and "transformation":

$$\mathbf{H} = \sigma(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}), \quad \hat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}, \tag{10}$$

where W is the weight matrix of the "transformation" process. As shown in Fig. 3, our patch encoder decouples the aggregation process between patches, which was originally achieved through aggregation with graph structure $\hat{A}$ in MPNNs, and implements it through the transformer block in Section 3.3. Thus, our patch encoder can be seen as setting the parameters of the white part in the W matrix to zeros. These blank parameters are replaced by the powerful transformer block, which not only retains the learning ability but also reduces the noise impact of the graph structure on the misaligned patches. In addition, since our structure-aware module shares weights for K patches, the parameters near the diagonal of our W matrix are duplicated and identical, which is shown to be parameter-efficient. From another perspective, W in Fig. 3 is essentially a block diagonal matrix. Such matrices are often used in scenarios aimed at enhancing efficiency while still maintaining the accuracy of the algorithm [5].

Generally, PatchNet reduces information exchange between patches compared to MPNN in the patch encoding stage and earlier steps, which we compensate for during patch aggregation. In other words, our method retains the flexibility of parameter transformations on input features similar to MPNN, while significantly surpassing MPNN in terms of generalizability.

Quality of embedding. To further analyze PatchNet, we propose to qualify the generated node embedding from the perspective of singular value. The singular value spectrum of the embedding space, which is widely considered to be related to the generalization performance [3, 25, 50]. More specifically, we perform singular value decomposition (SVD) on the node embedding $\mathbf{F} \in \mathbb{R}^{M\times D}$ by PatchNet: $\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^\top$.[1].

Definition 3.3 (Singular Value Entropy). Singular value entropy (SVE) is characterized as the entropy associated with the normalized singular values. It serves as a quantifier for the distribution's flatness among singular values.

$$\text{SVE} = -\sum_{i=1}^{D} \frac{\sigma_i}{\sum_{j=1}^{D}\sigma_j} \log \frac{\sigma_i}{\sum_{j=1}^{D}\sigma_j} \tag{11}$$

Higher SVE values suggest an enhanced capture of data structure within the feature space, attributed to either the learning of more distinct features or the memorization of noise, thereby expanding its dimensional span.

Definition 3.4 (Largest Singular Value Ratio). The largest singular value ratio (LSVR) is determined by taking the logarithm of the quotient obtained from dividing the largest singular value, denoted as σ_1, by the aggregate of all singular values:

$$\text{LSVR} = -\log \frac{\sigma_1}{\sum_{i=1}^{D}\sigma_i}. \tag{12}$$

LSVR quantifies the disparities in data encapsulated by the singular vector associated with the largest singular value, σ_1, indicative of the model's transferability [3].

We plot the SVE and LSVR for the embedding generated by PatchNet in Fig. 4. The input data here is PCQM4Mv2 [10]. Specifically, the dashed lines represent the average values of SVE and LSVR obtained from the original data's node features, while the different colored solid lines represent the average values of SVE and LSVR for the node representations obtained by our model after different numbers of rounds of training. The solid lines with dots represent the values of SVE, and the solid lines with crosses represent the values of LSVR. The results indicate that during the forward propagation process of PatchNet, both SVE and LSVR for the node representations are continuously increasing, demonstrating that our model is constantly improving the transferability and distinguishability of the representations.

4 Experiments

In this section, we answer the following four questions through experiments to validate the effectiveness of our method:

- **RQ1.** Can PatchNet handle feature heterogeneity on different downstream datasets through cross-domain pre-training?
- **RQ2.** Does the performance of PatchNet improve with the increase of the scale of pre-training datasets?
- **RQ3.** Can PatchNet outperform other pre-trained backbones in both graph and node classification tasks?
- **RQ4.** How sensitive is PatchNet to the size of node tokens?

Note that each set of experiments is repeated five times. Detailed hyper-parameters can be found in the Appendix.

[1] $\mathbf{U}$ and $\mathbf{V}$ denotes the left and right singular vector matrices, respectively, and Σ denoting the diagonal singular value matrix $\{\sigma_1, \ldots, \sigma_D\}$.

Table 1: Cross domain pre-training and fine-tuning performance in terms of mean and std. deviation of ROC-AUC (for Sider, HIV, Bace) and F1 (for Flickr and DBLP). Improvement (IMP) and P-value are used to measure the gap between using PatchNet with all pre-training data and without any pre-training.

Pre \ Down	Sider	HIV	Bace	Flickr	DBLP
RAW	52.14±0.56	56.58±2.57	55.84±3.16	47.25±3.54	74.86±2.26
PatchNet (RAW)	51.88±0.59	57.72±1.31	57.35±2.77	46.29±3.86	75.05±2.52
ZINC	53.06±0.47	58.74±1.72	59.74±0.54	45.30±2.98	76.44±2.32
Arxiv	52.77±0.58	57.01±1.80	56.00±2.66	50.64±2.69	79.01±2.84
ZINC+Arxiv	**53.71±0.48**	**60.23±1.15**	**59.75±1.02**	**50.81±1.94**	**79.12± 1.93**
IMP (%)	1.29	2.51	2.40	4.52	4.07
P-value	0.012	0.008	0.010	0.003	0.003

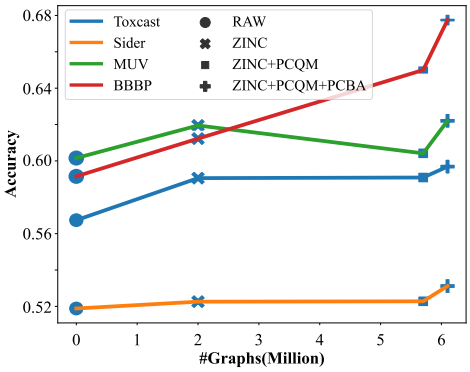

Figure 5: Performance on scaling of the size fo pre-training datasets.

4.1 Cross Domain Transfer Learning

To address **RQ1**, we set up experiments (Tab. 1) with cross-domain pre-training and fine-tuning. **Some graph data's features are derived from text and are not the target of our model, as they can either be uniformly regenerated using the same text encoder to obtain homogenous node features, or use models like OFA [17] with LLM.** To the best of our knowledge, no other models have yet been pre-trained across multiple domains and then applied to different tasks in various downstream domains, so we only use PatchNet to evaluate.

During pre-training, we selected molecular dataset ZINC [34] and paper citation dataset Ogbn-Arxiv [11]. These datasets not only exhibit feature heterogeneity but also have different forms: Arxiv is a single large graph, whereas ZINC consists of many smaller molecular graphs. Thus, we employ neighbor sampling for Arxiv and graph sampling for ZINC. We load these two datasets simultaneously to pre-train one PatchNet using existing two pre-text tasks, Attribute Masking (AM) and Context prediction (CP) [12]. We empirically find out that the using these multi-task trick [16] to combine AM and CP perform the best. Note that since ZINC has 2M samples (graphs) while Arxiv only has 0.17M samples (subgraph stems from nodes), we randomly choose 0.17M samples out of ZINC in each epoch. The reason is to keep balance between the pre-training information. In the fine-tuning phase, we employed two different types of tasks across datasets: graph and node classification; the graph classification involved Sider, HIV, and Bace from [43], while the node classification included datasets Flickr [58] and DBLP [1]. Notably, the downstream data also originated from different domains: Sider, HIV, and Bace are molecular datasets, while Flickr and DBLP are social network and citation graph, respectively. Not only is there feature heterogeneity between the pre-train and downstream datasets, but they also belong to different tasks. During fine-tuning, we conduct fine-tuning and testing according to the data splits specific to each dataset. In Tab. 1, each row represents a combination of pre-training data, and each column represents a different downstream dataset. The first two rows represent end-to-end learning without any transfer.

Results. From Tab. 1, it is evident that our model handles feature heterogeneity effectively, both between pre-train datasets and between pre-train and downstream datasets. Large IMP values demonstrate that our method has achieved performance improvements through pre-training and transfer learning, while small p-values ensure that these improvements are statistically significant. Moreover, although PatchNet may sometimes perform comparably to basic models like GIN [48] (used in RAW setting) in an end-to-end manner, it consistently shows improvement after pre-training. Furthermore, the underlined scores representing the second best generally indicate that pre-training with the same task data tends to yield relatively better results. Additionally, we find that even when the two pre-training datasets come from completely different backgrounds, combined pre-training still leads to improvements in downstream tasks. Even when there is significant variation among downstream datasets, this type of cross-domain transfer pre-training still achieves good results in downstream tasks. This is also the outcome we hope to see in future graph foundation models, making PatchNet a step towards achieving a GFM.

4.2 Data Scaling Perspective

To address **RQ2**, we chose a more intuitive scenario, which involves simply modifying the scale of the pre-training datasets. We tried different combinations of pre-training datasets to increase the volume of pre-training data. Fig. 5 features pre-training data that includes ZINC, PCQM [10], and PCBA [11]. By combining these for pre-training, we obtain three models, plus one model without pre-training (RAW), resulting in 4 horizontal axis data points. The downstream data includes Toxcast, Sider, MUV and BBBP from [43], corresponding to 4 line graphs.

Results. From Fig. 5, we find that PatchNet is capable of multi-dataset pre-training and efficiently handling larger amounts of pre-training data. Additionally, certain pre-train datasets contribute more significantly to improvements in specific downstream datasets compared to others. For example, PCBA has a greater impact on enhancing MUV than it does on Toxcast and Sider. This could be due to PCBA and MUV both belonging to the Biophysics category, whereas Sider is categorized under Physiology data [43]. Moreover, PatchNet performs better when pre-trained with more datasets under the same setting, which reveals that our model has great potential when trained with enormous data.

	Tox21	Toxcast	Sider	ClinTox	MUV	HIV	BBBP	Bace	Rank
RAW	67.90±1.48	58.39±0.96	52.14±0.56	56.43±4.23	58.53±2.52	56.58±2.57	58.57±7.72	55.84±3.16	12.5
ContextPred [12]	66.45±1.75	58.16±0.85	51.53±0.22	55.83±1.07	59.49±2.66	56.58±1.31	63.57±1.16	57.92±1.07	11.3
AttrMask [12]	67.17±1.31	59.33±1.37	52.21±1.12	56.69±1.78	58.58±2.18	57.34±0.98	63.65±2.18	57.27±1.94	9.3
AM+CP [16]	67.62±1.84	58.19±0.68	52.44±0.29	57.17±0.96	59.06±2.63	56.53±1.43	63.79±1.60	57.96±3.64	9.3
GPT-GNN [13]	67.98±1.75	58.39±1.51	52.97±0.91	57.07±1.73	58.56±1.54	56.68±1.03	65.06±3.05	56.25±2.05	8.5
GraphCL [57]	68.22±1.61	59.09±1.18	52.67±0.09	56.99±1.63	58.73±1.85	56.82±1.64	64.68±1.44	56.92±1.26	7.9
GraphMVP [19]	68.01±0.93	55.43±0.44	52.24±0.57	55.54±2.12	57.36±2.69	56.88±1.75	65.41±0.39	57.77±0.35	10.3
3D InfoMax [33]	67.05±1.26	58.22±0.58	52.58±0.35	54.56±2.55	59.85±2.36	56.65±1.67	67.64±1.33	58.66±1.40	9.0
Mole-BERT [45]	**70.07±0.69**	**59.72±0.15**	52.58±0.35	55.52±3.09	61.05±1.35	57.79±1.79	68.44±2.90	**60.07±1.71**	4.0
PatchNet (RAW)	67.77±1.80	56.74±0.59	51.88±0.59	55.78±1.15	60.16±1.17	57.72±1.31	59.15±2.85	57.35±2.77	10.4
PatchNet (AM)	64.84±1.41	56.95±0.74	52.29±0.39	57.26±2.81	60.65±2.50	54.27±1.39	60.32±4.27	57.20±1.94	11.1
PatchNet (CP)	66.22±1.83	59.56±0.27	53.06±0.52	58.60±1.35	61.79±1.47	54.11±1.46	55.01±6.11	57.55±1.34	7.9
PatchNet (AM+CP)	68.56±1.34	59.64±0.71	**53.06±0.47**	**59.50±1.79**	**62.19±1.46**	**58.74±1.72**	**68.55±3.75**	59.74±0.54	1.4

Table 2: The comparison on graph classification task (Full version see Appendix).

	Tox21	Toxcast	Sider	ClinTox	BBBP	Bace	Rank
GCN[41]	55.25 ± 1.68	52.14 ± 0.44	51.94 ± 0.26	50.96 ± 3.73	65.27 ± 1,76	53.62 ± 0.92	5.3
GIN [48]	60.17 ± 1.84	54.19 ± 0.68	51.44 ± 0.29	53.59 ± 0.96	68.10 ±1.60	50.02 ± 3.64	5.0
GAT [38]	61.62 ± 2.37	53.86 ± 0.46	51.83 ± 0.28	58.41 ± 2.08	68.14 ± 2.11	54.01 ± 1.73	3.8
Graphormer [55]	63.04 ± 1.47	57.14 ± 0.70	52.54± 0.19	55.61 ± 1.60	66.24 ± 1.62	57.56 ± 1.85	3.2
GraphGPS [28]	65.77 ± 1.78	56.13 ± 0.78	52.80± 0.21	58.75 ± 1.13	**68.73 ± 2.17**	52.11 ± 1.76	2.5
PatchNet	**68.56 ± 1.34**	**59.64 ± 0.71**	**53.06 ± 0.47**	**59.50 ± 1.79**	68.55 ± 3.75	**59.74 ± 0.54**	1.2

Table 3: The comparison of backbones on graph classification task.

	Coauthor-CS	Coauthor-physics
GCN	82.15±1.93	87.25±0.72
GAT	82.72±1.96	88.41±0.52
DGI	83.09±2.02	88.34±0.75
GRACE	83.43±2.96	88.69±0.87
GraphMAE	84.33±2.75	90.13±0.57
GraphMAE2	84.67±2.43	91.80±0.64
PatchNet	**87.69±1.28**	**92.87±0.27**

Table 4: Results of node classification evaluation.

4.3 Pre-training and Fine-tuning

To address **RQ3**, we conducted experiments on two types of downstream tasks: graph classification and node classification.

Graph classification evaluation. In this section, we conduct two parts of experiments: comparing with different pre-text tasks (Tab. 2) comparing with different backbones (Tab. 3). We choose ZINC as our pre-training datasets in both experiments. Note that the size of node feature is $D = 300$, which is different from the publicly available feature size $D = 2$. It's well known that most of the researches involving molecular graphs all expand two-dimensional original feature to 300 dimensions of learnable features [47]. Thus, we replace the learnable feature with non-trainable features by using group VQ-VAE [37] which is also used by Mole-bert [45] which are frozen throughout the training process.

For patch extraction, we use token size $M = 32$ and step size $K = 20$. For downstream evaluation, we adopt the widely-used 8 binary classification datasets from MoleculeNet [43]. Due to the difficulty of obtaining sufficient labels in practical applications, we adopt a 1:1:8 train-validation-test label split . Note that we employ scaffold splitting [29] to split molecules based on their structures, which emulates real-world scenarios.

On the one hand, we fix the backbone of baselines as their original GIN. Tab. 2 is divided into 4 main blocks by row. The first row is using GIN as backbone and without pre-training. The second block consists of current popular pre-training strategies. The third block contains those strategies designed for molecular graphs. In the last block we adopt 3 combination of pre-text tasks: attribute masking, context prediction, and both. From Tab. 2, we can see that: Compared to all baselines, PatchNet achieves competitive or better performance under the same experimental protocols. Since we only use simple pre-training strategies and its combination, it is apparent that our backbone plays the most significant role. Moreover, we find that single pre-training strategies may lead to negative transfer on both GIN and PatchNet. But after applying multi-task pre-training strategy, PatchNet opens up a significant gap with GIN, which means our model does better on combining pre-training strategies. When testing PatchNet's raw capability, we are surprised to find that PatchNet outperformed some GIN series models in certain datasets even without pre-training, which indicates that even under the condition of such extreme label ratio and more model parameters, PatchNet still achieves better performance, reflecting that PatchNet is adaptive and robust. Due to space limitation, we show the full version in Appendix.

On the other hand, we compare with different backbones using the same pre-training strategy as the combination of AM and CP. Tab. 3 is also divided into 3 blocks, universal backbones, molecular specialized backbones and our backbone from top to bottom.

	Tox21	Toxcast	Sider	ClinTox	MUV	HIV	BBBP	Bace
PatchNet	**68.56±1.34**	**59.64±0.71**	**53.06±0.47**	**59.50±1.79**	**62.19±1.46**	**58.74±1.72**	**68.55±3.75**	**59.74±0.54**
Inter-Mean	60.87±1.50	53.41±0.86	52.11±0.90	51.46±1.26	59.35±1.09	55.91±2.20	50.58±3.08	52.01±0.54
Inter-Sum	60.65±2.17	51.48±0.80	52.80±0.43	58.79±1.30	58.05±1.92	52.34±3.07	52.02±7.82	53.89±0.93
Inner-GCN	63.78±2.10	55.93±0.53	51.62±0.47	50.13±1.66	58.74±1.17	53.82±1.70	54.04±5.18	53.18±2.02
Inner-GAT	59.14±3.76	56.07±1.34	52.54±0.60	57.98±1.23	59.79±1.27	58.21±3.21	54.19±5.83	54.84±1.58
Non-overlap	60.62±2.36	54.68±0.92	52.89±0.41	58.31±1.54	61.47±1.86	56.60±2.39	53.71±4.57	55.54±1.59

Table 5: Ablation on our backbone

From Tab. 3, we can see that: As is known to all, the effect of transfer between pre-train and downstream datasets is an important measurement for backbones. PatchNet provides a well-performed transfer compared to traditional backbones such as GCN, GAT and GIN. This is because patch extraction process significantly increases the generalizability of our model, and the transformer module can avoid the old problems such as over-smoothing, allowing our model to effectively use a greater amount of parameters than normal GNNs. Moreover, our method performs better than molecule specialized backbones such as Graphormer. That's because we combine the merit from both GNN aggregation and transformer. Since our backbone can obtain more detailed semantic information in original feature by learning from tokens, it is much more efficient and effective than other backbones.

Node classification evaluation. Here we use two datasets, Coauthor-CS [32] and Coauthor-Physics [32] for both pre-training and downstream evaluation due to their rich node features. For patch extraction, we use token size $M = 1024$ and step size $K = 512$. Tab. 4 contains 3 blocks: classical end-to-end methods, well-known self-supervised methods and PatchNet. As for dataset split, we follow ParetoGNN [14] with 1:1:8 for training/validation/test. We report the average ROC-AUC with the corresponding standard deviation. Table 4 shows that our backbone is more powerful and capable of extracting more information from the limited downstream data. Since PatchNet is trained on both datasets, our method has achieved cross-dataset transferability, which enables us to pre-train a model using more large-scale datasets to get a more powerful model.

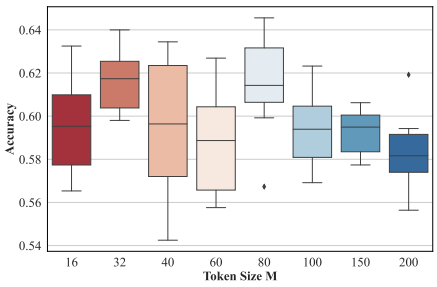

Figure 6: Performance of varying token size M.

4.4 Ablation and Sensitivity Study

To address **RQ4**, we conducted two experiments: an ablation study of PatchNet' s sub-modules and a sensitivity analysis of the most critical hyperparameter, the token size M.

As shown in Tab. 3, backbone with transformer performs significantly better than that with simple pooling, since the rich information among different patches can't be simply aggregated by pooling. And the inner-patch aggregation with GIN does a better work than others. This is due to the fact that GIN has high expressiveness, which is also the reason why GIN has become the preferred choice in many molecular graph-based research studies. What's more, we test the behavior of our model when there is no overlapping information between patches. Since splitting original feature of any dataset is quite challenging, we allow overlapping tokens to form graph patches. And as our expectation, backbone with token overlapping performs better. But to our surprise, some of the results are better than the setting without pre-training, which means our model could potentially learn to complete missing features. From Fig. 6, we find that having a large number of token size isn't a good idea. This is because if the number of token channels is relatively low, the learning process for aggregating information between tokens will be partially lost. In extreme scenarios where there's only one token, this process will be lost entirely, which means our backbone will involution to normal GNN model.Moreover, we can find out that even when token size is small, satisfactory results can still be achieved. It indicates that GNN's ability to learn structure is not abandoned even when the backbone focuses more on transformer's token aggregation.

5 Conclusion

In summary, we highlight the challenges in addressing the limitations of current graph pre-training models for the incapability of handling cross-domain transfer. We identify the main challenge as feature heterogeneity. Then, by introducing the learnable graph patches as a basic semantic unit for graph data, we propose PatchNet, a framework capable of mining transferable information across different graph domains. Our empirical analyses show PatchNet's capability to generate distinguishable and transferable node representations, advancing pre-training on multi-domain non-textual graphs and showing continuous improvement on various downstream datasets and tasks.

Limitation and future work. Our method has an inherent limitation in using a fixed size for unfolding node features into node tokens, determined by a hyperparameter based on empirical experience. Future work will explore automatic learning of optimal node token sizes to improve generalization.

6 Acknowledgment

This work is supported by National Natural Science Foundation of China (No. 62176233, No. 62322606, No. 62441605).

References

[1] Uchenna Akujuobi, Han Yufei, Qiannan Zhang, and Xiangliang Zhang. 2019. Collaborative Graph Walk for Semi-Supervised Multi-label Node Classification. In *2019 IEEE International Conference on Data Mining (ICDM)*. 1–10. https://doi.org/10.1109/ICDM.2019.00010

[2] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. 2023. NAGphormer: A Tokenized Graph Transformer for Node Classification in Large Graphs. In *Proceedings of the International Conference on Learning Representations*.

[3] Xinyang Chen, Sinan Wang, Mingsheng Long, and Jianmin Wang. 2019. Transferability vs. Discriminability: Batch Spectral Penalization for Adversarial Domain Adaptation. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 1081–1090.

[4] Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, et al. 2023. Exploring the potential of large language models (llms) in learning on graphs. *arXiv preprint arXiv:2307.03393* (2023).

[5] Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. 2022. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*. PMLR, 4690–4721.

[6] Vijay Prakash Dwivedi and Xavier Bresson. 2020. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699* (2020).

[7] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jian, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D Hwang, et al. 2023. Faith and Fate: Limits of Transformers on Compositionality. *arXiv preprint arXiv:2305.18654* (2023).

[8] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. 2021. Slaps: Self-supervision improves structure learning for graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 22667–22681.

[9] Xiaoxin He, Bryan Hooi, Thomas Laurent, Adam Perold, Yann LeCun, and Xavier Bresson. 2023. A generalization of vit/mlp-mixer to graphs. In *International conference on machine learning*. PMLR, 12724–12745.

[10] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs. arXiv:2103.09430 [cs.LG]

[11] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687* (2020).

[12] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265* (2019).

[13] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1857–1867.

[14] Mingxuan Ju, Tong Zhao, Qianlong Wen, Wenhao Yu, Neil Shah, Yanfang Ye, and Chuxu Zhang. 2023. Multi-task Self-supervised Graph Neural Networks Enable Stronger Task Generalization. (2023).

[15] Kevin Lin, Lijuan Wang, and Zicheng Liu. 2021. Mesh Graphormer. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), 12919–12928.

[16] Xi Lin, Hui-Ling Zhen, Zhenhua Li, Qing-Fu Zhang, and Sam Kwong. 2019. Pareto multi-task learning. *Advances in neural information processing systems* 32 (2019).

[17] Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2023. One for All: Towards Training One Graph Model for All Classification Tasks. *arXiv preprint arXiv:2310.00149* (2023).

[18] Jiawei Liu, Cheng Yang, Zhiyuan Lu, Junze Chen, Yibo Li, Mengmei Zhang, Ting Bai, Yuan Fang, Lichao Sun, Philip S Yu, et al. 2023. Towards graph foundation models: A survey and beyond. *arXiv preprint arXiv:2310.11829* (2023).

[19] Shengchao Liu, Hanchen Wang, Weiyang Liu, Joan Lasenby, Hongyu Guo, and Jian Tang. 2021. Pre-training molecular graph representation with 3d geometry. *arXiv preprint arXiv:2110.07728* (2021).

[20] Shengchao Liu, Hanchen Wang, Weiyang Liu, Joan Lasenby, Hongyu Guo, and Jian Tang. 2022. Pre-training Molecular Graph Representation with 3D Geometry. arXiv:2110.07728 [cs.LG]

[21] Yixin Liu, Yu Zheng, Daokun Zhang, Hongxu Chen, Hao Peng, and Shirui Pan. 2022. Towards unsupervised deep graph structure learning. In *Proceedings of the ACM Web Conference 2022*. 1392–1403.

[22] Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Michael Galkin, and Jiliang Tang. 2024. Graph foundation models. *arXiv preprint arXiv:2402.02216* (2024).

[23] Gaspard Michel, Giannis Nikolentzos, Johannes F Lutzeyer, and Michalis Vazirgiannis. 2023. Path neural networks: Expressive and accurate graph neural networks. In *International Conference on Machine Learning*. PMLR, 24737–24755.

[24] Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. 2022. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455* (2022).

[25] Samet Oymak, Zalan Fabian, Mingchen Li, and Mahdi Soltanolkotabi. 2019. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. *arXiv preprint arXiv:1906.05392* (2019).

[26] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. *KDD* (2020).

[27] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[28] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. 2022. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems* 35 (2022), 14501–14515.

[29] Bharath Ramsundar, Peter Eastman, Pat Walters, and Vijay Pande. 2019. *Deep learning for the life sciences: applying deep learning to genomics, microscopy, drug discovery, and more.* " O'Reilly Media, Inc.".

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

[31] Abulhair Saparov and He He. 2022. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240* (2022).

[32] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *arXiv preprint arXiv: 1811.05868* (2018).

[33] Hannes Stärk, Dominique Beaini, Gabriele Corso, Prudencio Tossou, Christian Dallago, Stephan Günnemann, and Pietro Liò. 2022. 3d infomax improves gnns for molecular property prediction. In *International Conference on Machine Learning*. PMLR, 20479–20502.

[34] Teague Sterling and John J Irwin. 2015. ZINC 15–ligand discovery for everyone. *Journal of chemical information and modeling* 55, 11 (2015), 2324–2337.

[35] Yifei Sun, Haoran Deng, Yang Yang, Chunping Wang, Jiarong Xu, Renhong Huang, Linfeng Cao, Yang Wang, and Lei Chen. 2022. Beyond Homophily: Structure-aware Path Aggregation Graph Neural Network. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, Lud De Raedt (Ed.). International Joint Conferences on Artificial Intelligence Organization, 2233–2240. https://doi.org/10.24963/ijcai.2022/310 Main Track.

[36] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. 2021. Adversarial graph augmentation to improve graph contrastive learning. *Advances in Neural Information Processing Systems* 34 (2021), 15920–15933.

[37] Aäron van den Oord, Oriol Vinyals, and K. Kavukcuoglu. 2017. Neural Discrete Representation Learning. *NIPS* (2017).

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[39] Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2023. Can Language Models Solve Graph Problems in Natural Language? *arXiv preprint arXiv:2305.10037* (2023).

[40] Yuyang Wang, Jianren Wang, Zhonglin Cao, and Amir Barati Farimani. 2022. Molecular Contrastive Learning of Representations via Graph Neural Networks. arXiv:2102.10056 [cs.LG]

[41] Max Welling and Thomas N Kipf. 2016. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*.

[42] Zhanghao Wu, Paras Jain, Matthew A. Wright, Azalia Mirhoseini, Joseph E. Gonzalez, and Ion Stoica. 2022. Representing Long-Range Context for Graph Neural Networks with Global Attention. arXiv:2201.08821 [cs.LG]

[43] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. 2018. MoleculeNet: a benchmark for molecular machine learning. *Chemical science* 9, 2 (2018), 513–530.

[44] Jun Xia, Lirong Wu, Jintao Chen, Bozhen Hu, and Stan Z Li. 2022. Simgrace: A simple framework for graph contrastive learning without data augmentation. In *Proceedings of the ACM Web Conference 2022*. 1070–1079.

[45] Jun Xia, Chengshuai Zhao, Bozhen Hu, Zhangyang Gao, Cheng Tan, Yue Liu, Siyuan Li, and Stan Z Li. 2023. Mole-bert: Rethinking pre-training graph neural networks for molecules. (2023).

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

[47] Jun Xia, Yanqiao Zhu, Yuanqi Du, and Stan Z. Li. 2022. A Systematic Survey of Chemical Pre-trained Models. *arXiv preprint arXiv: Arxiv-2210.16484* (2022).

[48] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.

[49] Minghao Xu, Hang Wang, Bingbing Ni, Hongyu Guo, and Jian Tang. 2021. Self-supervised graph-level representation learning with local and global structure. In *International Conference on Machine Learning*. PMLR, 11548–11558.

[50] Yihao Xue, Kyle Whitecross, and Baharan Mirzasoleiman. 2022. Investigating why contrastive learning benefits robustness against label noise. In *International*

	Type	Name	N	E
Pre-training	Graph level	ZINC [34]	53,254,058	115,472,818
		PCQM4Mv2 [10]	52,970,652	109,093,626
		Ogbg-molpcba [11]	11,373,137	24,618,372
	Node level	Ogbn-arxiv [11]	169,343	1,166,243
		Flickr [58]	105,938	2,316,948
		DBLP [1]	28,702	68,335
Downstream	Graph level	Tox21 [43]	145,459	302,190
		Toxcast [43]	161,088	330,356
		Sider [43]	48,006	100,912
		ClinTox [43]	38,637	82,372
		MUV [43]	2,255,846	4,892,252
		HIV [43]	1,049,163	2,259,376
		BBBP [43]	49,068	105842
		Bace [43]	51,577	111,536
	Node level	Flickr [58]	105,938	2,316,948
		DBLP [1]	28,702	68,335
		Coauthor-CS [32]	18,333	81,894
		Coauthor-Physics [32]	34,493	247,962

Table 6: The statistics of all datasets.

Hyperparameter	Range
K	{15 → 25}
M	64
number of GIN layers	3
number of Attention heads	3
Learning Rate	{1e-3 → 3e-3}
Weight decay	{0 → 1e-6}
GIN dropout rate	0.2
Attention dropout rate	{0.3 → 0.7}
Batch size	{256, 512, 1024}
Optimizer	Adam
Epoch	100
GPU	GeForce RTX 4090

Table 7: Hyper-parameter of graph classification task.

Conference on Machine Learning. PMLR, 24851–24871.
[51] Yang Yang, Yuhong Xu, Yizhou Sun, Yuxiao Dong, Fei Wu, and Yueting Zhuang. 2019. Mining fraudsters and fraudulent strategies in large-scale mobile social networks. *IEEE Transactions on Knowledge and Data Engineering* 33, 1 (2019), 169–179.
[52] Yang Yang, Yuhong Xu, Chunping Wang, Yizhou Sun, Fei Wu, Yueting Zhuang, and Ming Gu. 2019. Understanding default behavior in online lending. In *CIKM*. 2043–2052.
[53] Jiacai Yi, Chengkun Wu, Xiaochen Zhang, Xinyi Xiao, Yanlong Qiu, Wentao Zhao, Tingjun Hou, and Dongsheng Cao. 2022. MICER: a pre-trained encoder–decoder architecture for molecular image captioning. *Bioinformatics* 38, 19 (2022), 4562–4572.
[54] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems* 34 (2021), 28877–28888.
[55] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do Transformers Really Perform Badly for Graph Representation?. In *Thirty-Fifth Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=OeWooOxFwDa
[56] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. 2021. Graph contrastive learning automated. In *International Conference on Machine Learning*. PMLR, 12121–12132.
[57] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *Advances in neural information processing systems* 33 (2020), 5812–5823.
[58] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).
[59] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. 2020. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140* (2020).
[60] Shuaicheng Zhang, Haohui Wang, Si Zhang, and Dawei Zhou. 2023. GPatcher: A Simple and Adaptive MLP Model for Alleviating Graph Heterophily. *arXiv preprint arXiv:2306.14340* (2023).
[61] Haihong Zhao, Aochuan Chen, Xiangguo Sun, Hong Cheng, and Jia Li. 2024. All in one and one for all: A simple yet effective method towards cross-domain graph pretraining. *arXiv preprint arXiv:2402.09834* (2024).
[62] Qi Zhu, Carl Yang, Yidan Xu, Haonan Wang, Chao Zhang, and Jiawei Han. 2021. Transfer learning of graph neural networks with ego-graph information maximization. *Advances in Neural Information Processing Systems* 34 (2021), 1766–1779.
[63] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning. *Proc. IEEE* 109, 1 (2020), 43–76.

A Details of Experiments

A.1 Datasets

The datasets used are shown in Tab. 7. The scale of the graphs used in pre-training and downstream tasks also highlights the high scalability of PatchNet.

A.2 Hyperparemeter Strategy

Overall, our proposed framework is implemented via PyTorch. As for software versions, we use Python 3.7.0, PyTorch 1.13.1, OGB 1.3.6, and CUDA 11.3. Moreover, the range of hyperparameters are listed in Table 7.

A.3 Additional Experiments

Here we present the complete graph evaluation results in Tab. 8, comparing our model against a broader range of self-supervised tasks. It is evident that our model indeed outperforms existing self-supervised methods. Please refer to the main text for the simplified Tab. 2. As is shown in the table, our model's performance, combined with simple pre-training tasks, surpasses other existing approaches on datasets within a single domain. This result demonstrates that our model not only exhibits strong generalization capabilities but also effectively learns domain-specific knowledge.

B Algorithm

The algorithm of Fig. 2 is shown in Algo. 1. The computation of PatchNet includes three parts: node tokenization, patch construction, encoding and aggregation of graph patches. Hence, the overall complexity is: $O(N(KM + M^2 + Mb + K^2))$.

C More Related Work

We provide additional information here on related work concerning pre-training and fine-tuning GNNs. The concept of graph foundation model (GFM) is comprehensively established [18, 22] and envisioned to be adept across various tasks and datasets. Yet, there currently does not exist a GFM that fully meets the criteria. However, building a cross-domain and cross-task graph model has always been a hot topic. One of the pathways to build graph foundation models is to design graph pre-training framework.

	Tox21	Toxcast	Sider	ClinTox	MUV	HIV	BBBP	Bace	Rank
w/o pre	67.90±1.48	58.39±0.96	52.14±0.56	56.43±4.23	58.53±2.52	56.58±2.57	58.57±7.72	55.84±3.16	12.5
AD-GCL [36]	64.81±1.22	55.55±0.79	51.13±0.17	53.46±2.20	59.41±2.42	56.01±1.02	59.26±1.86	52.27±2.08	15.3
ContextPred [12]	66.45±1.75	58.16±0.85	51.53±0.22	55.83±1.07	59.49±2.66	56.58±1.31	63.57±1.16	57.92±1.07	11.3
AttrMask [12]	67.17±1.31	59.33±1.37	52.21±1.12	56.69±1.78	58.58±2.18	57.34±0.98	63.65±2.18	57.27±1.94	9.3
AM+CP [16]	67.62±1.84	58.19±0.68	52.44±0.29	57.17±0.96	59.06±2.63	56.53±1.43	63.79±1.60	57.96±3.64	9.3
SimGRACE [44]	67.49±1.37	58.79±0.34	52.73±0.08	56.54±2.34	59.98±2.45	57.65±1.96	63.27±2.30	56.03±1.84	8.8
GraphLoG [49]	64.09±1.47	58.88±0.59	52.74±0.55	57.38±1.76	60.39±1.99	57.04±1.07	62.49±2.80	56.14±1.88	8.6
GPT-GNN [13]	67.98±1.75	58.39±1.51	52.97±0.91	57.07±1.73	58.56±1.54	56.68±1.03	65.06±3.05	56.25±2.05	8.5
GraphCL [57]	68.22±1.61	59.09±1.18	52.67±0.09	56.99±1.63	58.73±1.85	56.82±1.64	64.68±1.44	56.92±1.26	7.9
JOAO [56]	68.41±1.59	58.92±0.42	52.45±0.14	57.78±1.36	60.73±2.06	56.88±1.45	62.99±2.29	57.35±1.46	7.0
GraphMVP [19]	68.01±0.93	55.43±0.44	52.24±0.57	55.54±2.12	57.36±2.69	56.88±1.75	65.41±0.39	57.77±0.35	10.3
3D InfoMax [33]	67.05±1.26	58.22±0.58	52.58±0.35	54.56±2.55	59.85±2.36	56.65±1.67	67.64±1.33	58.66±1.40	9.0
Mole-BERT [45]	**70.07±0.69**	**59.72±0.15**	52.58±0.35	55.52±3.09	61.05±1.35	57.79±1.79	68.44±2.90	**60.07±1.71**	4.0
Ours(AM)	64.84±1.41	56.95±0.74	52.29±0.39	57.26±2.81	60.65±2.50	54.27±1.39	60.32±4.27	57.20±1.94	11.1
Ours w/o pre	67.77±1.80	56.74±0.59	51.88±0.59	55.78±1.15	60.16±1.17	57.72±1.31	59.15±2.85	57.35±2.77	10.4
Ours(CP)	66.22±1.83	59.56±0.27	53.06±0.52	58.60±1.35	61.79±1.47	54.11±1.46	55.01±6.11	57.55±1.34	7.9
Ours(AM+CP)	68.56±1.34	59.64±0.71	**53.06±0.47**	**59.50±1.79**	**62.19±1.46**	**58.74±1.72**	**68.55±3.75**	59.74±0.54	1.4

Table 8: The comparison of overall performance on graph classification task.

Algorithm 1 Pseudo code for the forward process of the model

The Patch Encoder Φ, the multi-head self-attention encoder W and a feed forward network using ReLU F. An Attention Mechanism Att.
The graph structure and node feature after patch extraction is g, x. If we regarded each channel separately, we will get $g, x_i, i = 0..K$ for different tokens i.
/* Model training starts */
Obtain their node embeddings z_0 by [Some Method].
for each token j **do**
  Learn the new graph structure g'.
  for each node u, node v **do**
    Calculate similarity $S^j_{uv} = SIM(W \cdot x_{uj}, W \cdot x_{vj})$.
  Apply non-linear transformation to S^j.
  Normalize S^j.
  Update graph structure $g' \leftarrow Topk(g, S^j)$.
  Combine node embeddings using attention mechanism: $z_{0,j} \leftarrow Att(\Phi(g, x_j), \Phi(g', x_j))$.
end for
Aggregate through patches by W and pass through F: $z_1 \leftarrow W(z_0)$.
Combine embeddings: $z_2 \leftarrow F(z_0 + z_1)$.
Merge all channels by mean-pooling: $z_3 \leftarrow P(z_0 + z_1 + z_2)$.

Pre-training graph models has achieved significant success, with various self-supervised pre-training methods proposed for both node-level and graph-level tasks. For node classification task, following the generative language model GPT [27], GPT-GNN [13] factorizes graph generation into Attribute Generation and Edge Generation. While for graph classification task, GraphCL [57] maximizes agreement between two representations of the same node by injecting random perturbations, and [12] use subgraphs to develop several self-supervised learning strategies combining node-level and graph-level pre-training information. Graph Contrastive Coding (GCC) [26] captures the universal network topological properties through subgraph instance discrimination as pre-training task. In particular, these predictive and contrastive methods are effective for graphs with rich annotated information, such as molecular graphs, protein interaction graphs and social networks. However, a majority of these works still utilize a plain GNN, such as the 5-layer Graph Isomorphism Network(GIN) [46, 48], and therefore cannot be reused when encountering different downstream tasks without corresponding data. Graph-BERT [59] trains GNN and transformer in parallel with Attribute Reconstruction and Structure Recovery tasks. GROVER [30] first uses GNNs to capture local structural information, which then serves as queries, keys and values for a Transformer encoder. GraphTrans [42] proposes the first hybrid architecture, using a stack of MPNN layers before fully-connecting the graph. Mesh Graphormer [15] proposes a hybrid architecture stacking Graph Residual Block (GRB) on a multi-head Transformer block. Besides, some other works utilize the uniqueness of input data in designing architectures. For example, MolCLR [40] uses molecule SMILES to implement graph augmentation for contrastive learning. GraphMVP [20] pretrains by leveraging the consistency between 3D geometry and 2D topology. Since these models utilize more semantic information, they are even more domain-specific. Some of above methods only transfer structural information, neglecting the node attributes that contain valuable information.

Recently, attempts have been made to adapt Large Language Models (LLMs) to tasks associated with graph analysis. Despite their proficiency in natural language processing, directly converting graph data for LLM processing has not been entirely effective, leading to less than ideal outcomes, as demonstrated in research on both textual [4] and non-textual graphs [39]. Nevertheless, LLMs still encounter challenges when processing graph data [7, 31].

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009