GRAPHGENT: Foundation Model for Graph Pre-training

Anonymous Author(s)
Affiliation
Address
email

Abstract

The success of "pre-train and fine-tune" paradigm has drawn considerable research interests. The pre-training frameworks of GPT-3 and SAM have nearly standardized NLP and CV. However, existing graph pre-training methods are limited in their generalizability since they can only transfer knowledge within the same domain or even the same graph and discard valuable node attributes. We attribute this limitation to the failure of defining the transferable token and the foundation model for graph. Inspired by ViT and PatchTST, we propose a new concept called *semantic graph patches*, which consist of divided node semantic tokens and corresponding topology structures from the original graph data. Accordingly, we design the **GRAPHGENT**[1](Graph Generalized pre-Training), a foundation model for generalized graph pre-training that leverages patch encoder and patch aggregator to learn transferable knowledge from different graphs. Moreover, we provide theoretical analysis and empirical results demonstrating the effectiveness of their proposed method. **GRAPHGENT** shows an average of 0.59% and 2.46% improvement against other SSL tasks and SOTA backbones respectively.

1 Introduction

Graph, as a prevalent data structure, is ubiquitous in a wide range of applications, such as social network analysis [45], bio-informatics [48], finance [46], etc. Due to difficulties in obtaining sufficient labels and the increasing demand to learn general knowledge from massive data, recent years have witnessed the flourishing of pre-training methods like GPT-3 [2] and SAM [14] which have revolutionized NLP and CV.

However, existing graph pre-training methods are not that generalized. Ideally, pre-training requires data from as many domains as possible to learn fundamental knowledge [13]. Besides, the downstream application should be agnostic to the domain of the pre-training data. Nevertheless, most existing graph pre-training models can only transfer knowledge within the same domain or even the same graph. Additionally, others assume that only the topology structure is transferable and, therefore, discard the node attributes of both pre-training and downstream graphs. On the one hand, if the downstream graphs come from domains that are difficult to acquire data, such as the SEEG brain network [3], it is almost impossible to obtain a large amount of pre-training data within the same domain. Moreover, for every new downstream graph dataset, the backbone must be pre-trained from scratch. On the other hand, in the compound data structure of a graph, both node attributes and graph topology are critical for achieving satisfactory performance on graph application tasks [20]. Thus, discarding the valuable node attributes leads to sub-optimal results since they are the primary prerequisite for GNNs to ensure the quality of node embedding [20].

[1]Code is available at `https://anonymous.4open.science/r/Fondation4Graph-33EB`.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

In this paper, we attribute the low generalizability of graph pre-training models to the failure of defining the transferable token and the "foundation model" [1]. Most existing graph pre-training models vaguely assume the correlation between the whole node attributes and the topology structure as a transferable token by leveraging a plain GNN such as GIN [43], or as the backbone model [41]. However, the graphs from different domains or distributions do not share the correlation due to their inconsistency on node attributes. Specifically, the distribution and even the length of node features vary across different domains or graphs. For instance, if pre-trained on a graph with node attributes of d dimensions, the pre-training model cannot be applied to downstream graphs with node attributes of different dimensions.

Inspired by ViT [4] and PatchTST [19], we propose that each graph can be seen as a complex consisting of **semantic graph patches**. Specifically, semantic graph patches is a combination of the divided node semantic tokens and the corresponding topology structures from the original graph structure. For example, in a financial network, each node contains complex semantic information such as personal information like age, gender, transaction information like borrowing and lending records and the amount of deposits. Therefore, such a financial network can be divided into a social network composed of personal information and a transaction network composed of transaction information, and so on. For graphs without obvious semantic division in node attributes, each node attributes can be understood as describing a complex color, which can be divided into multiple semantic graph patches with each of them being composed of single colors.

Given the transferable semantic graph patches, it is still challenging to design a foundation model to learn the fundamental knowledge. The ideal foundation model behind pre-training strategy is supposed to leverage large data from various domains for pre-training so as to learn fundamental knowledge [13], and generalize to datasets beyond those seen during pre-training. Therefore, we propose **GRAPHGENT** (graph generalized pre-training) model, which is to learn to aggregate first within each semantic graph patch for the structure correlations and then across all the semantic graph patches for the semantic correlations. Specifically, we first split the original node attributes into node semantic tokens. Then, we design a structure learning GNN to aggregate each semantic graph patch with its corresponding learned structure. Last, we exploit a transformer-based block to aggregate the semantic graph patches from each graph. We evaluate our foundation model on existing settings and our new setting. The results show its promising effect on generalized graph pre-training on both graph classification and node classification.

The contribution of this paper are summarized as follows:

- We identify the transferable token for graph pre-training as semantic graph patch, which consists of node semantic tokens and corresponding structure.
- We propose a foundation model **GRAPHGENT** composed of patch encoder and patch aggregator for generalized graph pre-training. We also theoretically analyze the correlation between **GRAPHGENT** and MPNNs.
- We empirically prove the effectiveness of our method against other pre-training methods on both node classification and graph classification tasks. In graph classification, **GRAPHGENT** demonstrates an average of 0.59% and 2.46% against other SSL tasks and other backbones.

2 Related Work

Pre-training Graph Neural Networks(GNNs) has achieved significant success, with various self-supervised pre-training methods proposed for both node-level and graph-level tasks. In node classification, following the generative language model GPT [22], GPT-GNN [11] factorizes graph generation into Attribute Generation and Edge Generation. While in graph classification, GraphCL [51] maximizes agreement between two representations of the same node by injecting random perturbations, and hu *et al.* [10] use subgraphs to develop several self-supervised learning strategies combining both node-level and graph-level pre-training information. Graph Contrastive Coding (GCC) [21] 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-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) [43, 41], and therefore cannot be reused when encountering different downstream tasks without corresponding data.

Several other works focus on designing pre-training graph models(PGMs), with most leveraging the self-attention mechanism of transformers. Graph-BERT[Zhang *et al.*] [53] trains GNN and transformer in parallel with Attribute Reconstruction and Structure Recovery tasks. GROVER [25] first uses GNNs to capture local structural information, which then serves as queries, keys and values for a Transformer encoder. GraphTrans [37] 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. However, most of these models are still limited to relatively small graphs due to the at least $\mathcal{O}(N^2)$ time complexity and similar semantic distribution, making them less effective in cross-domain datasets.

Besides, some other works utilize the uniqueness of input data in designing architectures. For example, MolCLR [34] uses molecule SMILES to implement graph augmentation for contrastive learning. GraphMVP [18] pretrains by leveraging the consistency between 3D geometry and 2D topology. While these models utilize more semantic information, they are even more domain-specific.

Moreover, some methods only transfer structural information, neglecting the node attributes that contain valuable information for downstream tasks. To tackle the aforementioned challenges, we propose **GRAPHGENT**, which views each graph as a combination of semantic graph patches, effectively utilizing both semantic and structural information. Based on the transferable design, our model exhibits stronger generalizability and bridges the gap between different graph types from pre-training to downstream.

3 Method

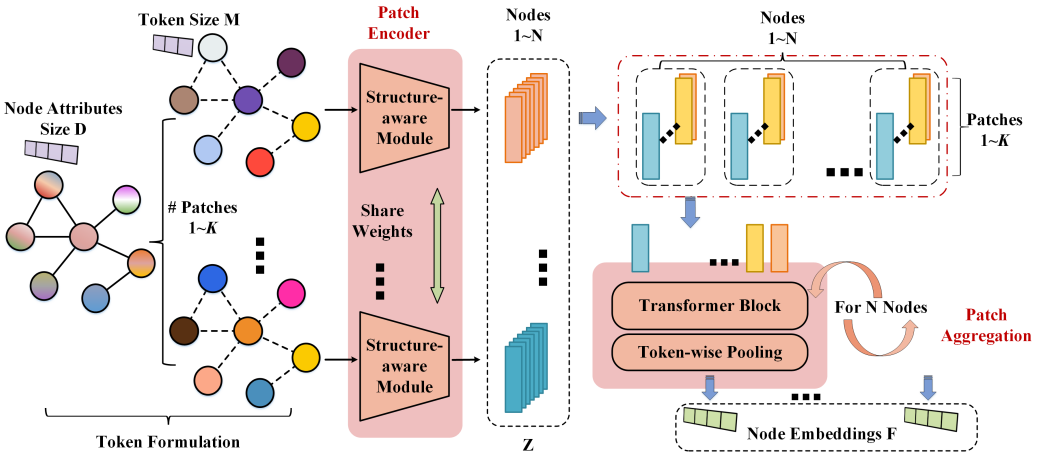

Figure 1: Overall Workflow of **GRAPHGENT**.

3.1 General Description

We begin with some necessary definitions.

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 attributes matrix where D is the dimension of attributes. Due to the varying scales of node features in different graphs, we use the normalized node attributes $X \in [0,1]^{N\times D}$.

The overall architecture of **GRAPHGENT** is shown in Figure 1 and the full algorithm can be found in Algorithm **??** from Appendix §C. We begin by extraction of the transferable semantic graph patches. For images and natural language, patching is straightforward: ViT [4] divides the image into grid-like patches/tokens; for natural language, each word is a patch/token. However, there are two main challenges patching the irregular graph data: firstly, the graph data contains both node features and

graph structure information. Secondly, the graph structure has complex geometric properties such as permutation invariance and typically different from graphs to graphs. Thus, a natural question is how to define transferable graph patches and extract them efficiently. Here we give formal definitions of node tokens and semantic graph patches.

Definition 3.1 (Node Tokens) *Given a graph $\mathcal{G} = (V, A, X)$, the attributes of each node $X_i \in \{0,1\}^D$ can be divided into node tokens $T_i \in \{0,1\}^{K\times M}$, where K is the number of token channels and M is the token size. Different token extraction methods can generate different number of token channels. Each token channel for node i is $T_i^j \in \{0,1\}^M$. Thus, the node token matrix is $T \in \{0,1\}^{N\times K\times M}$.*

Definition 3.2 (Semantic Graph Patches) *Given tokens T of N nodes and the original graph structure A, we generate corresponding rewired graph structure $\widetilde{A}(T_:^j) \in \{0,1\}^{N\times N}$ for each token channel j. Thus, each semantic graph patch is $P_j = \left(T_:^j, \widetilde{A}(T_:^j)\right)$, where $j = 1, ..., K$.*

The extraction of semantic graph patches consists of two parts: the formulation of node tokens and exclusive patch structure learning. Follow the Definition 3.1, we formulate K tokens of size M from each node before training and thus derive K graphs with the original structure (dotted line). Next, we design a structure-aware module for exclusive patch structure learning and patch encoding. Note that this module works parallel on the derived K graphs and share weights.

After encoding the semantic graph patches as $Z \in \mathbb{R}^{N\times K\times \text{hid}}$, we regard patch embeddings Z as N sequences, K as the effective input sequence length for the Transformer block and hid as the hidden dimension of the embedded tokens. Then, the transformer block send the entangled patch embeddings with the same size as Z directly to the token-wise pooling block. The patch embeddings Z are aggregated as the final node embeddings $F \in \mathbb{R}^{N\times f}$, where f is the output dimension. From a macro perspective, the workflow shown in Figure 1 is about mapping the original D-dimensional node attributes to f-dimensions. This process can also be repeated multiple times, similar to stacked GNN layers.

3.2 Pre-process of Tokens

The core idea of formulating node tokens is introduced in Section 1. We propose to uniformly extract the node tokens for every graph before training process, which is the foundation of the transferable semantic graph patch generation.

Although the semantic information of nodes may vary across different graphs, we propose a unified patch extraction approach to delineate the semantic boundaries of these tokens. In order to cover as many tokens with relatively independent semantics as accurately as possible, we employ a sliding window mechanism with hyperparameters token size M and step S to partition the normalized attributes $X_i \in [0,1]^D$ of each node i into the corresponding tokens $T_i \in [0,1]^{K\times M}$, where $K = \lfloor \frac{D-M}{S} \rfloor$.

Although the semantic boundaries of tokens are difficult to confirm, setting the step $S < M$ can provide more overlapping token perspectives and increase the likelihood of enabling appropriate semantic boundaries to some extent. Since $N > K \times M$ holds in most cases, the time consumption for token extraction is $\mathcal{O}(N)$, which can be done as a pre-processing step before training.

Having obtained the node tokens, it is not yet in the complete form of the semantic graph patches. As described in Definition 3.2, we generate a structure of graph semantic patches $\widetilde{A}$ corresponding to each token channel by using the original graph structure A and node tokens T. The most straightforward approach is to set $\widetilde{A}$ as A. However, this approach not only cannot completely extract independent semantic patches from the original graph, but may also lead to under-reaching and over-squashing. Thus, we design a structure-aware patch encoder to first learn $\widetilde{A}$ through each channel of node tokens, and then encode the complete patch.

3.3 Structure-aware Patch Encoder

Inspired by GAT [33], we design an attention-based graph learner to learn the structure $\widetilde{A}$ corresponding to each token. We assume that each feature plays a distinct role in the presence of an edge, while

there are no notable interrelationships among the features. Here we show the details about *graph learner module* in Figure 2 for a given token $T^l \in [0,1]^{N\times M}$:

$$S_{i,j} = f_\phi(W \odot T_i^l, W \odot T_j^l), \quad \widetilde{A}_{i,j}(T) = \begin{cases} \sigma(S_{i,j}), & j \in \text{top-k}(S_{i,:}) \\ 0, & j \notin \text{top-k}(S_{i,:}) \end{cases}, \tag{1}$$

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 like cosine similarity, σ stands for non-linear activation function like relu. Moreover, we employ residual connections to update the learned $\widetilde{A}$ to accelerate and stabilize the training process:

$$\widetilde{A} = \alpha\widetilde{A} + (1-\alpha)A \tag{2}$$

where α stands for the trade-off parameter for how much trainable structure to adopt. Then, we propose a dual-branch attention mechanism to adaptively encode the patch. Specifically, given each original token $T^l \in [0,1]^{N\times M}$ and the origin structure A and learned $\widetilde{A}$, the l-th patch are encoded as

$$H^l = \Phi(T^l, A), \quad \widetilde{H^l} = \Phi(T^l, \widetilde{A}(T^l)), \tag{3}$$

$$\beta = \delta(f_{\mathbf{MLP}}(\widetilde{H^l}||H^l)), \quad Z^l = \sum \beta(\widetilde{H^l}||H^l), \tag{4}$$

where Φ denotes a GNN, $f_{\mathbf{MLP}}$ is a two-layer MLP and δ stands for Softmax operation. The output $Z^l \in \mathbb{R}^{N\times \text{hid}}(l = 1, ...K)$ is encoded patch embedding for N nodes.

3.4 Patch Aggregation

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 graph 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 already 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 nodes.

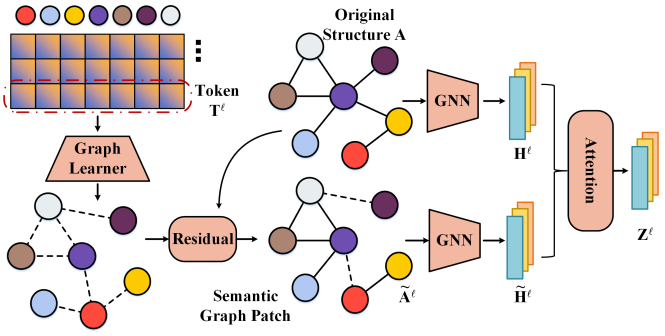

Figure 2: Details about Structure-aware Module in Figure 1.

One straight way is to aggregate all the patches through a MLP-based module. However, since **GRAPHGENT** is designed to transfer across different graphs 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 **GRAPHGENT** to capture contextual information throughout the entire feature. Given $Z \in \mathbb{R}^{N\times K\times \text{hid}}$ as input, the

$$W = \text{LayerNorm}(\text{MHA}(Z)), \tag{5}$$

$$W' = \text{LayerNorm}(\text{FFN}(W)), \tag{6}$$

$$F = \text{Pooling}(W') \tag{7}$$

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 node embeddings $F \in \mathbb{R}^{N\times f}$. Overall, the patch aggregation requires $\mathcal{O}(NK^2)$ time complexity.

3.5 Theoretical Analysis

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 analyse the correlation between **GRAPHGENT**, existing MPNNs and other graph transformers.

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

$$\mathbf{H} = \sigma\big(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}\big), \quad \hat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}, \tag{8}$$

where W is the weight matrix of the "transformation" process. As shown in Figure 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 3.4. 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.

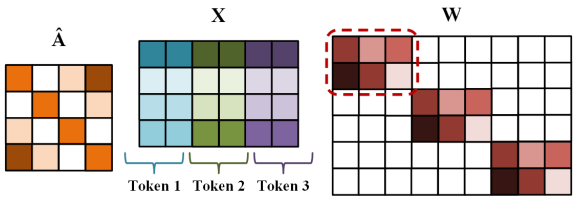

Figure 3: The decomposed operation of **GRAPHGENT** from the perspective of MPNN.

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. Last but not least, according to above sections, the time complexity of **GRAPHGENT** is $\mathcal{O}(N)$ when M, K, hid, D are constants. Thus, **GRAPHGENT** is relatively more efficient than existing graph transformers.

4 Experiments

4.1 Pre-training

Datasets. For graph classification task, We choose three datasets as our pre-training datasets, which are ZINC-agent [28], ogbg-molpcba [9] and PCQM4Mv2 [8]. 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 [42]. In this case, due to the simultaneous training of learnable features and backbone, it will affect our evaluation on the effectiveness of the backbone. Thus, we replace the learnable feature with non-trainable features by using group VQ-VAE [31] which is also used by Mole-bert [40]. We leverage group VQ-VAE as a pre-processed converter for each dataset respectively, which enhances the original node feature in molecule datasets from 2 dimensions to 300 dimensions, after that, the feature are frozen throughout the training process. We use the same way to acquire stationary node feature in all of our experiments about molecule graph classification for fair comparison.

For node classification task, our self-supervised pre-training is performed on ogbn-Papers100M [9]. We first run local clustering to produce local clusters for each node as the pre-processing step. Their detailed information of all datasets is shown in from Appendix § A.1.

Pre-training settings. For graph classification task, we apply patch extraction with $M = 32$ and $K = 20$. Our backbone architecture consists of a transformer encoder and a 3-layer Graph Isomorphism Network (GIN) whose hidden dimension is 128. Finally we employ mean pooling as the readout function.

For node classification task, we apply a patch extraction with token size set to 32 and step set to 10 to extract node features as the input of our backbone, whose architecture consists of a transformer encoder and a 4-layer Graph Isomorphism Network (GIN) with hidden dimension is 128. Finally

we employ mean pooling as the readout function. Detailed hyper-parameters can be found in the Appendix § A.2.

4.2 Graph Classification Evaluation

Datasets. We adopt the widely-used 8 binary classification datasets contained in MoleculeNet [38] as our downstream datasets. Since other baselines can only support one dataset during pre-training, we choose only ZINC-agent dataset as the pre-train dataset in Table 1 and Table 2.

Settings. Due to the difficulty of obtaining sufficient labels in practical biological and chemical applications, which is also one of the original intentions of pre-training technology, we adopt a 1:1:8 train-validation-test label split for molecular graph classification in this section. Note that we employ scaffold splitting [24] to split molecules based on their structures, which emulates real-world scenarios. During the fine-tuning phase, we train for 30 epochs with a batch size of 256. Due to the scaffold split, there is some data distribution shift between the validation set and test set [5]. Therefore, we report the experimental results (including baseline and our own) based on the better of the test performance corresponding to the best validation performance and the performance at the last epoch. Moreover, since previous studies have used different evaluation protocols, we meticulously reproduce all results using the same protocol as the pioneering work [10] to ensure fairness. As a result, there may be some inconsistency between our findings and their original papers.

Graph classification overall results. Table 1 is divided into 3 main blocks by row. The first block consists of current universal graphic pre-training strategies supported by GIN, the second block contains those strategies designed for molecular graphs, which are also supported by GIN, and in the last block we adopt ParetoGNN multi-task training framework [16] to combine AttrMask and ContextPred as one pre-training strategy. That's because our model is complicated, single task won't show the effectiveness of our backbone. Choosing AttrMask and ContextPred is a natural decision, since AttrMask is the easiest task for learning node features, while ContextPred is the simplest task for learning graphic structures. For fairness, only 2D inputs for molecules are used, including the molecule specialized tasks in the second block.

	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 [29]	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 [10]	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 [10]	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 [39]	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 [44]	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 [11]	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 [51]	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 [50]	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 [17]	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 [27]	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 [40]	**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 1: The comparison of overall performance on graph classification task.

We document the main results of graph classification on molecular datasets in Table 1, and it suggests the following trends: Compared to all self-supervised baselines in Table 1, **GRAPHGENT** 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.

Table 1 also suggests that single pre-training strategies may lead to negative transfer on both GIN and **GRAPHGENT**. But after applying multi-task pre-training strategy, **GRAPHGENT** opens up a significant gap with GIN, which means our model does better on synthesizing pre-training strategies.

When testing **GRAPHGENT**'s raw capability, we are surprised to find that **GRAPHGENT** 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, **GRAPHGENT** still achieves better performance, reflecting that **GRAPHGENT** is adaptive and robust.

Backbone comparison on graph classification task Table 2 is also divided into 3 blocks by row, and they stands for universal backbones, molecular specialized backbones and our backbone from top to bottom. All of them use AM+CP as pre-training strategy.

	Tox21	Toxcast	Sider	ClinTox	BBBP	Bace	Rank
GCN[35]	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 [43]	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 [32]	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 [49]	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 [23]	65.77 ± 1.78	56.13 ± 0.78	52.80± 0.21	58.75 ± 1.13	**68.73 ± 2.17**	52.11 ± 1.76	<u>2.5</u>
Ours	**68.56 ± 1.34**	**59.64 ± 0.71**	**53.06 ± 0.47**	**59.50 ± 1.79**	<u>68.55 ± 3.75</u>	**59.74 ± 0.54**	1.2

Table 2: The comparison of backbones on graph classification task.

In particular, we fix the pre-training strategy and list the behavior of different backbones in Table 2, and it suggests the following trends: As is known to all, the effect of transfer between upstream and downstream datasets is an important measurement for backbones. And as demonstrated in Table 2, **GRAPHGENT** provides a well-performed transfer when 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.

When treated with insufficient amount of labels, which is very common in reality, our backbone performs better than molecule specialized backbones such as Graphormer. That's because these backbones tile up too many parameters, so that it is hard for them to transfer their backbone architectures from upstream to downstream when label is insufficient. While our backbone can obtain more detailed semantic information in original feature by learning from tokens, which is much more efficient and effective than other backbones.

4.3 Node Classification Evaluation

Datasets. We conducted experiments on the widely used dataset, ogbn-arxiv [9], which contains academic papers from the arXiv repository, covering various research disciplines.

Settings. We compare our proposed method, **GRAPHGENT**, with several state-of-the-art self-supervised graph learning methods and generative methods. To demonstrate the generality of our proposed method, we followed the public split of ogbn-arxiv.

Results. Table 3 shows the results of **GRAPHGENT** on node classification task.

Note that all the baselines in the table (except for the randomly initialized, untrained model) require to pre-train on the same dataset as the downstream task. In comparison, our model's cross-domain ability allows us to pre-train on datasets with different distribution. Compared to most SSL baselines, **GRAPHGENT** achieves better results. Our competetive performance can be attributed to two factors. Firstly, our backbone is more powerful and capable of extracting more information from the limited downstream data. This allows our model to learn more effectively. Secondly, our method has achieved cross-dataset transferability, which enables us to pretrain a robust model using large-scale datasets such as ogbn-Papers100M. Existing methods are unable to achieve this unless they discard node features, as their backbones are constrained by the requirement that upstream and downstream tasks must use the same dataset. We can observe that even without a pre-trained model ('Ours w/o pre' in Table 1), **GRAPHGENT** outperforms most methods in the table requiring pre-training.

4.4 Model Analysis

Multiple pre-training datasets.

Our models are capable of multi-dataset pre-training to efficiently handle larger amounts of data. As is shown in Figure 4, our model performs better when pre-trained with more datasets under the same setting. And it reveals that our model has great potential when trained with enormous data.

	Arxiv
Ours w/o pre	70.08±0.48
MLP	55.50±0.23
SGC [36]	66.92±0.08
CCA-SSG [52]	68.57±0.02
GRACE [54]	69.34±0.01
BGRL [30]	70.51±0.03
GraphMAE [7]	71.03±0.02
GraphMAE2 [6]	**71.89±0.03**
Ours(GraphMAE2)	71.42±0.43

Table 3: Accuracy(%) on node classification task

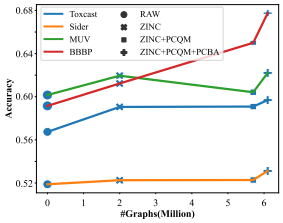

Figure 4: Performance varying # Pre-training datasets.

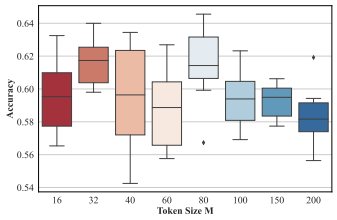

Figure 5: Performance varying token size M.

Ablation on backbone. We substitute or remove some components of the proposed approaches to study their effectiveness. As can be observed in Table 4, 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 backbone with GIN does a better work than those with GCN and GAT. This is due to the fact that GIN is the most powerful graph neural network(GNN) model among the three, 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 we don't know how to divide original feature to maximize its semantic information in these tokens, we allow overlapping inter-patches, so that our model will be able to learn a reasonable division. 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 can still learn something out even if the division is not reasonable.

	Tox21	Toxcast	Sider	ClinTox	MUV	HIV	BBBP	Bace
Ours	**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 4: Ablation on our backbone

Based on the right part of Figure 5, 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 kept to a small number, satisfactory results can still be achieved. It indicates that GNN's ability to learn structure is not be abandoned even when the backbone focuses more on transformer's token aggregation, which comprises a larger number of parameters. Due to the space limitation, we show other additional experiments in Appendix § B.

5 Conclusion

In conclusion, we highlight the challenges faced by existing graph pre-training models in terms of their limited generalizability. We propose a novel concept called semantic graph patches, which combines the divided node semantic tokens and corresponding topology structures from the original graph structure. Corresponding to semantic graph patches, we propose the **GRAPHGENT** as foundation model for generalized graph pre-training, which aggregates within each patch for structure correlations and across all patches for semantic correlations. The authors demonstrate the effectiveness of their proposed method through theoretical analysis and empirical results, showcasing the promising impact on both graph and node classification tasks. Overall, this paper offers a new perspective and approach towards improving the generalizability of graph pre-training models.

A Details of Experiments

A.1 Datasets

	Type	Name	N	E
Pre-training	**Graph level**	ZINC-agent [28]	53,254,058	115,472,818
		PCQM4Mv2 [8]	52,970,652	109,093,626
		Ogbg-molpcba [9]	11,373,137	24,618,372
	Node level	Paper100M [9]	3,097,165	47,334,788
Downstream	**Graph level**	Tox21 [38]	145,459	302,190
		Toxcast [38]	161,088	330,356
		Sider [38]	48,006	100,912
		ClinTox [38]	38,637	82,372
		MUV [38]	2,255,846	4,892,252
		HIV [38]	1,049,163	2,259,376
		BBBP [38]	49,068	105842
		Bace [38]	51,577	111,536
	Node level	Ogbn-arxiv [9]	169,343	1,166,243
		Pubmed [47]	19,717	44,324
		Citeseer [47]	2,110	3,668
		Cora [47]	2,485	5,069
		Coauthor-CS [26]	18,333	81,894
		Coauthor-Physics [26]	34,493	247,962

Table 3: The statistics of all datasets.

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.

For graph classification, we use Cross-entropy as downstream loss and Adam optimizer to train **GRAPHGENT**. Moreover, the range of hyperparameters are listed in Table 4, where K is the number of token channels, M is the token size, and the step S can be calculated by the original dimension of node attributes D and K, M.

Hyperparameter	Range
K	$\{15 \rightarrow 25\}$
M	64
number of GIN layers	3
number of Attention heads	3
Learning Rate	{1e-3 $\rightarrow$ 3e-3}
Weight decay	{0 $\rightarrow$ 1e-6}
GIN dropout rate	0.2
Attention dropout rate	$\{0.3 \rightarrow 0.7\}$
Batch size	{256, 512, 1024}
Optimizer	Adam
Epoch	100
GPU	GeForce RTX 3090

Table 4: Hyper-parameter of graph classification task.

For node classification, we use Cross-entropy as downstream loss and AdamW optimizer to train **GRAPHGENT**. Moreover, the range of hyperparameters are listed in Table 5. The number of token channels K can be calculated by the original dimension of node attributes D and S, M.

	Arxiv	Coauthor-CS	Coauthor-Physics
token size M	60	1024	1024
step S	30	512	1000
number of GNN layers	4	4	4
hidden size	64	300	300
ffn size	64	150	150
dropout rate	0.2	0.5	0.5
attention dropout rate	0.2	0.5	0.5
number of heads	3	4	4
pooling method	mean	mean	mean
Optimizer	AdamW	AdamW	AdamW
Epoch	1000	1000	1000
Learning Rate	0.001	0.001	0.001
Weight decay	1e-5	1e-07	1e-08
Batch size	128	128	128

Table 5: Hyper-parameter of node classification task.

B Additional Experiments

Here shows more experiments on node classification task.

Pre-training Datasets Our self-supervised pre-training is performed on two datasets, Coauthor-CS [26] and Coauthor-Physics [26]. Their detailed information of datasets is shown in from Section A.1.

Pre-training settings. We apply a patch extraction with token size set to 1024 and step set to 512 to extract node features as the input of our backbone, whose architecture consists of a transformer encoder and a 4-layer Graph Isomorphism Network (GIN) with hidden dimension is 300. Finally we employ mean pooling as the readout function. Detailed hyper-parameters can be found in the Section A.2.

	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
Ours	**89.69±1.28**	**93.87±0.27**

Table 6: Results of fine-tuning the pre-trained model (Coauthor-CS+Coauthor-physics) with split 1:1:8(train:validation:test)

Evaluation Settings. We compare our proposed method, **GRAPHGENT**, with several state-of-the-art self-supervised graph learning methods and generative methods. As for dataset split, following ParetoGNN [12], we randomly split the nodes into training, validation, and test sets with ratios of 1:1:8 for training/validation/test. For each dataset, we run all the algorithms 10 times and report the average ROC-AUC with the corresponding standard deviation.

Results. Table 6 shows the results of **GRAPHGENT** in node classification task on Coauthor-CS and Coauthor-Physics. Based on the results, our paper consistently outperforms current backbone models and self-supervised methods. Therefore, our conclusion is that our method achieves better performance even on moderately sized graph data, thanks to its flexible use of pre-training data and general feature decoupling techniques.

C Algorithm

Algorithm 1 Pseudo code for the forward process of **GRAPHGENT**

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.

/* **GRAPHGENT** training starts */

for all token j **do**
  for all node u, node v **do**
    $S^j_{uv} = SIM(W \cdot x_{uj}, W \cdot x_{vj})$
  end for
  $S^j \leftarrow NonLinear(S^j)$
  $S^j \leftarrow Normalize(S^j)$
  $g' \leftarrow Topk(g, S^j)$
  $z_{0,j} \leftarrow Att(\Phi(g, x_j), \Phi(g', x_j))$
end for
$z_1 \leftarrow W(z_0)$
$z_2 \leftarrow F(z_0 + z_1)$
$z_3 \leftarrow P(z_0 + z_1 + z_2)$

References

[1] Rishi Bommasani, Drew A. Hudson, and E. Adeli et al. On the opportunities and risks of foundation models. *ARXIV.ORG*, 2021.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] Junru Chen, Yang Yang, Tao Yu, Yingying Fan, Xiaolong Mo, and Carl Yang. Brainnet: Epileptic wave detection from seeg with hierarchical graph diffusion learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2741–2751, 2022.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[5] Shurui Gui, Xiner Li, Limei Wang, and Shuiwang Ji. GOOD: A graph out-of-distribution benchmark. In *NeurIPS*, 2022.

[6] Zhenyu Hou, Yufei He, Yukuo Cen, Xiao Liu, Yuxiao Dong, Evgeny Kharlamov, and Jie Tang. Graphmae2: A decoding-enhanced masked self-supervised graph learner, 2023.

[7] Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. *arXiv e-prints*, pages arXiv–2205, 2022.

[8] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs, 2021.

[9] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

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

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

[12] Mingxuan Ju, Tong Zhao, Qianlong Wen, Wenhao Yu, Neil Shah, Yanfang Ye, and Chuxu Zhang. Multi-task self-supervised graph neural networks enable stronger task generalization. 2023.

[13] Zixuan Ke, Yijia Shao, Haowei Lin, Tatsuya Konishi, Gyuhak Kim, and Bing Liu. Continual pre-training of language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[14] A. Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, A. Berg, Wan-Yen Lo, Piotr Dollár, and Ross B. Girshick. Segment anything. *ARXIV.ORG*, 2023.

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

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

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

[18] Shengchao Liu, Hanchen Wang, Weiyang Liu, Joan Lasenby, Hongyu Guo, and Jian Tang. Pre-training molecular graph representation with 3d geometry, 2022.

[19] Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

[20] Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *ARXIV.ORG*, 2019.

[21] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. *KDD*, 2020.

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

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

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

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

[26] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv: 1811.05868*, 2018.

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

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

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

[30] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L. Dyer, Rémi Munos, Petar Veličković, and Michal Valko. Large-scale representation learning on graphs via bootstrapping, 2023.

[31] Aäron van den Oord, Oriol Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. *NIPS*, 2017.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[33] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[34] Yuyang Wang, Jianren Wang, Zhonglin Cao, and Amir Barati Farimani. Molecular contrastive learning of representations via graph neural networks, 2022.

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

[36] Felix Wu, Tianyi Zhang, A. Souza, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. *International Conference On Machine Learning*, 2019.

[37] Zhanghao Wu, Paras Jain, Matthew A. Wright, Azalia Mirhoseini, Joseph E. Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention, 2022.

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

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

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

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

[42] Jun Xia, Yanqiao Zhu, Yuanqi Du, and Stan Z. Li. A systematic survey of chemical pre-trained models. *arXiv preprint arXiv: Arxiv-2210.16484*, 2022.

[43] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

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

[45] Yang Yang, Yuhong Xu, Yizhou Sun, Yuxiao Dong, Fei Wu, and Yueting Zhuang. Mining fraudsters and fraudulent strategies in large-scale mobile social networks. *IEEE Transactions on Knowledge and Data Engineering*, 33(1):169–179, 2019.

[46] Yang Yang, Yuhong Xu, Chunping Wang, Yizhou Sun, Fei Wu, Yueting Zhuang, and Ming Gu. Understanding default behavior in online lending. In *CIKM*, pages 2043–2052, 2019.

[47] Zhilin Yang, William W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *International Conference On Machine Learning*, 2016.

[48] Jiacai Yi, Chengkun Wu, Xiaochen Zhang, Xinyi Xiao, Yanlong Qiu, Wentao Zhao, Tingjun Hou, and Dongsheng Cao. Micer: a pre-trained encoder–decoder architecture for molecular image captioning. *Bioinformatics*, 38(19):4562–4572, 2022.

[49] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[50] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. In *International Conference on Machine Learning*, pages 12121–12132. PMLR, 2021.

[51] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.

[52] Hengrui Zhang, Qitian Wu, Junchi Yan, David Wipf, and Philip S. Yu. From canonical correlation analysis to self-supervised graph neural networks, 2021.

[53] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

[54] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.