Do Transformers Really Perform Bad for Graph Representation?

Anonymous Author(s)
Affiliation
Address
email

Abstract

The Transformer architecture has become a dominant choice in many domains, such as natural language processing and computer vision. Yet, it has not achieved competitive performance on popular leaderboards of graph-level prediction compared to mainstream GNN variants. Therefore, it remains a mystery how Transformers could perform well for graph representation learning. In this paper, we solve this mystery by presenting GraphFormer, which is built upon the standard Transformer architecture, and could attain excellent results on a broad range of graph representation learning tasks, especially on the recent OGB Large-Scale Challenge. Our key insight to utilizing Transformer in the graph is the necessity of effectively encoding the structural information of a graph into the model. To this end, we propose several simple yet effective structural encoding methods to help GraphFormer better model graph-structured data. Besides, we mathematically characterize the expressive power of GraphFormer and exhibit that with our ways of encoding the structural information of graphs, many popular GNN variants could be covered as the special cases of GraphFormer. The code and models of GraphFormer will be made publicly available at `https://github.com/anonymous/anonymous`.

1 Introduction

The Transformer [42] is well acknowledged as the most powerful neural network in modelling sequential data, such as natural language [11, 33, 6] and speech [17]. Model variants built upon Transformer have also been shown great performance in computer vision [12, 34]. However, to the best of our knowledge, Transformer has still not been the de-facto standard on public graph representation leaderboards [21, 14, 20]. There are many attempts of leveraging Transformer into the graph domain, but the only effective way is replacing some key modules (e.g., feature aggregation) in classic GNN variants by the softmax attention [43, 7, 22, 44, 50, 39, 13]. Therefore, it is still an open question whether Transformer architecture is suitable to model graphs and how to make it work in graph representation learning.

In this paper, we give an affirmative answer by developing GraphFormer, which is directly built upon the standard Transformer, and achieves state-of-the-art performance on a wide range of graph-level prediction tasks, including the very recent Open Graph Benchmark Large-Scale Challenge (OGB-LSC) [20], and several popular leaderboards (e.g., OGB [21], Benchmarking-GNN [14]). The Transformer is originally designed for sequence modeling. To utilize its power in graphs, we believe the key is to properly incorporate structural information of graphs into the model. Note that for each node i, the self-attention only calculates the semantic similarity between i and other nodes, without considering the structural information of a graph reflected on the nodes and the relation between node pairs. GraphFormer incorporates several simple yet effective encoding methods to leverage such information, which are described below.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

First, we propose a *Centrality Encoding* in GraphFormer to capture the node importance in the graph. In a graph, different nodes may have different importance, e.g., celebrities are considered to be more influential than the majority of web users in a social network. However, such information isn't reflected in the self-attention module as it calculates the similarities mainly using the node semantic features. To address the problem, we propose to encode the node centrality in GraphFormer. In particular, we leverage the simple *degree centrality* for the centrality encoding, where a learnable vector is assigned to each node according to its degree and added together with the node features in the input layer. Empirical studies show the simple centrality encoding is effective for Transformer in modeling the graph data.

Second, we propose a novel *Spatial Encoding* in GraphFormer to capture the structural relation between nodes. One notable geometrical property that distinguishes graph-structured data from other structured data, e.g., language, images, is that there does not exist a canonical grid to embed the graph. In fact, nodes can only lie in a non-Euclidean space and are linked by edges. To model such structural information, for each node pair, we assign a learnable embedding based on their spatial relation. Multiple measurements in the literature could be leveraged for modeling spatial relations. For a general purpose, we use the distance of the shortest path between any two nodes as a demonstration, which will be encoded as a bias term in the softmax attention and help the model accurately capture the spatial dependency in a graph. In addition, sometimes there is additional spatial information contained in edge features, such as the type of bond between two atoms in a molecular graph. We design a new edge encoding method to further take such signal into the Transformer layers. To be concrete, for each node pair, we compute an average of dot-products of the edge features and learnable embeddings along the shortest path, then use it in the attention module. Equipped with these encodings, GraphFormer could better model the relationship for node pairs and represent the graph.

By using the proposed encodings above, we further mathematically show that GraphFormer has strong expressiveness as many popular GNN variants are just its special cases. The great capacity of the model leads to state-of-the-art performance on a wide range of tasks in practice. On the large-scale quantum chemistry regression dataset[1] in the very recent Open Graph Benchmark Large-Scale Challenge (OGB-LSC) [20], GraphFormer outperforms most mainstream GNN variants by more than 10% points in terms of the relative error. On other popular leaderboards of graph representation learning (e.g., MolHIV, MolPCBA, ZINC) [21, 14], GraphFormer also surpasses the previous best results, demonstrating the potential and adaptability of the Transformer architecture.

2 Preliminary

In this section, we recap the preliminaries in Graph Neural Networks and Transformer.

Graph Neural Network (GNN). Let $G = (V, E)$ denote a graph where $V = \{v_1, v_2, \cdots, v_n\}$, $n = |V|$ is the number of nodes. Let the feature vector of node v_i be x_i. GNNs aim to learn representation of nodes and graphs. Typically, modern GNNs follow a learning schema that iteratively updates the representation of a node by aggregating representations of its first or higher-order neighbors. We denote $h_i^{(l)}$ as the representation of v_i at the l-th layer and define $h_i^{(0)} = x_i$. The l-th iteration of aggregation could be characterized by AGGREGATE-COMBINE step as

$$a_i^{(l)} = \text{AGGREGATE}^{(l)}\left(\left\{h_j^{(l-1)} : j \in \mathcal{N}(v_i)\right\}\right), \quad h_i^{(l)} = \text{COMBINE}^{(l)}\left(h_i^{(l-1)}, a_i^{(l)}\right), \quad (1)$$

where $\mathcal{N}(v_i)$ is the first or higher-order neighbor of v_i. The AGGREGATE function is used to gather the information from neighbors. Common aggregation functions include MEAN, MAX, SUM, which are used in different architectures of GNNs [25, 18, 43, 46]. The goal of COMBINE function is to fuse the information from neighbors into the node representation.

In addition, for graph representation tasks, a READOUT function is designed to aggregate node features $h_i^{(L)}$ of the final iteration into the representation h_G of the entire graph G:

$$h_G = \text{READOUT}\left(\left\{h_i^{(L)} \mid v_i \in G\right\}\right). \quad (2)$$

READOUT can be implemented by a simple permutation invariant function such as summation [46] or a more sophisticated graph-level pooling function [1].

[1] `https://ogb.stanford.edu/kddcup2021/pcqm4m/`

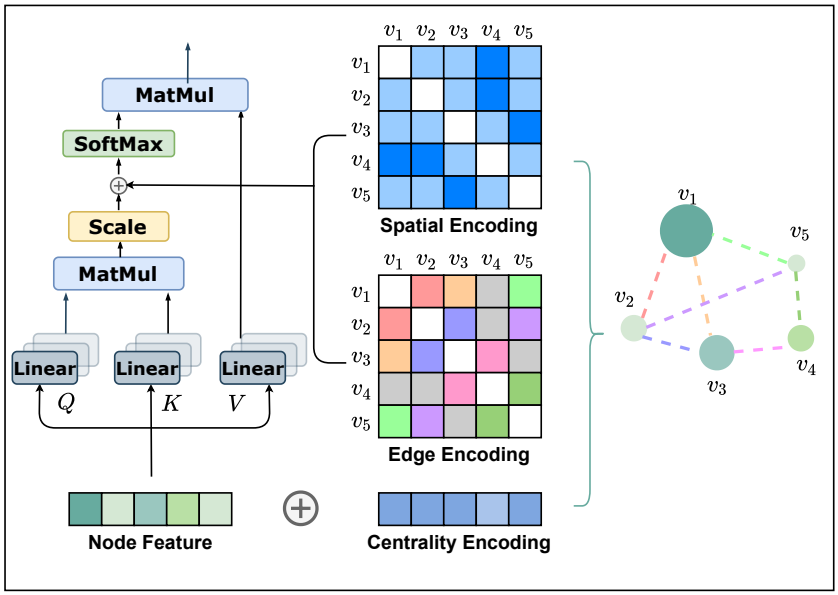

Figure 1: An illustration of our proposed centrality encoding, spatial encoding, and edge encoding in GraphFormer.

Transformer. The Transformer architecture consists of a composition of Transformer layers [42]. Each Transformer layer has two parts: a self-attention module and a position-wise feed-forward network (FFN). Let $H = \left[h_1^\top, \cdots, h_n^\top\right]^\top \in \mathbb{R}^{n \times d}$ denote the input of self-attention module where d is the hidden dimension and $h_i \in \mathbb{R}^{1 \times d}$ is the hidden representation at position i. The input H is projected by three matrices $W_Q \in \mathbb{R}^{d \times d_K}, W_K \in \mathbb{R}^{d \times d_K}$ and $W_V \in \mathbb{R}^{d \times d_V}$ to the corresponding representations Q, K, V. The self-attention is then calculated as:

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V, \tag{3}$$

$$A = \frac{QK^\top}{\sqrt{d_K}}, \quad \text{Attn}\left(H\right) = \text{softmax}\left(A\right)V, \tag{4}$$

where A is a matrix capturing the similarity between queries and keys. For simplicity of illustration, we consider the single-head self-attention and assume $d_K = d_V = d$. The extension to the multi-head attention is standard and straightforward, and we omit bias terms for simplicity.

3 GraphFormer

In this section, we present our GraphFormer for graph tasks. First, we elaborate on the several key designs in the GraphFormer, which serve as an inductive bias in the neural network to learn the graph representation. We further provide the detailed implementations of GraphFormer. Finally, we show that our proposed GraphFormer is more powerful as popular GNN models are its special cases.

3.1 Structural Encodings in GraphFormer

As discussed in the introduction, it is important to develop ways to leverage the structural information of graphs into the Transformer model. Towards this goal, we present three simple but effective designs of encoding in GraphFormer. See Figure 1 for an illustration.

3.1.1 Centrality Encoding

In Eq.4, the attention distribution is calculated based on the semantic correlation between nodes. However, node centrality, which measures how important a node is in the graph, is usually a strong signal for graph understanding. For example, celebrities who have a huge number of followers are

important factors in predicting the trend of a social network [36, 35]. Such information is neglected in the current attention calculation, and we believe it should be a valuable signal and used in the Transformer model.

In GraphFormer, we use the degree centrality, which is one of the standard centrality measures in literature, as an additional signal to the neural network. To be specific, we develop a *Centrality Encoding* which assigns each node two real-valued embedding vectors according to its indegree and outdegree. As the centrality encoding is applied to each node, we simply add it to the node features as the input.

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)}, \tag{5}$$

where $z^-, z^+ \in \mathbb{R}^d$ are learnable embedding vectors specified by the indegree $\deg^-(v_i)$ and outdegree $\deg^+(v_i)$ respectively. For undirected graphs, $\deg^-(v_i)$ and $\deg^+(v_i)$ could be unified to $\deg(v_i)$. By using the centrality encoding in the input, the softmax attention can catch the node importance signal in the queries and the keys. Therefore the model can capture both the semantic correlation and the node importance in the attention mechanism.

3.1.2 Spatial Encoding

An advantage of Transformer is its global receptive field. In each Transformer layer, each token can attend to the information at any position and then process its representation. But this operation has a byproduct problem that the model has to explicitly specify different positions or encode the positional dependency (such as locality) in the layers. For sequential data, one can either give each position an embedding (i.e., absolute positional encoding [42]) as the input or encode the relative distance of any two positions (i.e., relative positional encoding [38, 40]) in the Transformer layer.

However, for graphs, nodes are not arranged as a sequence. They can lie in a multi-dimensional spatial space and are linked by edges. To encode the structural information of a graph in the model, we propose a novel *Spatial Encoding*. Concretely, for any graph G, we consider a function $\phi\left(v_i, v_j\right) : V \times V \to \mathbb{R}$ which measures the spatial relation between v_i and v_j in graph G. The function ϕ can be defined by the connectivity between the nodes in the graph. In this paper, we choose $\phi(v_i, v_j)$ to be the distance of the shortest path between v_i and v_j if the two nodes are connected. If not, we set the output of ϕ to be a special value, i.e., -1. We assign each (feasible) output value a learnable scalar which will serve as a bias term in the self-attention module. Denote A_{ij} as the (i, j)-element of the Query-Key product matrix A, we have:

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i, v_j)}, \tag{6}$$

where $b_{\phi(v_i,v_j)}$ is a learnable scalar indexed by $\phi(v_i, v_j)$, and shared across all layers.

Here we discuss several benefits of our proposed method. First, we can see that in Eq. (6), each node can attend to all other nodes in the graph. Compared to conventional GNNs described in Section 2, where the receptive field is restricted to the neighbors, our network can easily obtain global information in a single layer and use that to process the representation of each node. Second, using $b_{\phi(v_i,v_j)}$ indeed can characterize the structural information of graphs. For example, if $b_{\phi(v_i,v_j)}$ is learned to be a decreasing function with respect to the distance $\phi(v_i, v_j)$, for each node, the model will likely pay more attention to the nodes near it and pay less attention to the nodes far away from it.

3.1.3 Edge Encoding in the Attention

In many graph tasks, edges also have structural features, e.g., in a molecular graph, atom pairs may have features describing the type of bond between them. Such features are important to the graph representation, and encoding them together with node features into the network is essential. There are mainly two edge encoding methods used in previous works. In the first method, the edge features are added to the associated nodes' features [21, 28]. In the second method, for each node, its associated edges' features will be used together with the node features in the aggregation [15, 46, 25]. However, such ways of using edge feature only propagate the edge information to its associated nodes, which may not be an effective way to leverage edge information in representation of the whole graph.

To better encode edge features into attention layers, we propose a new edge encoding method in GraphFormer. The attention mechanism needs to estimate correlations for each node pair (v_i, v_j),

and we believe the edges connecting them should be considered in the correlation as in [32, 44]. For each ordered node pair (v_i, v_j), we find (one of) the shortest path $\text{SP}_{ij} = (e_1, e_2, ..., e_N)$ from v_i to v_j, and compute an average of the dot-products of the edge feature and a learnable embedding along the path. The proposed edge encoding incorporates edge features via a bias term to the attention module. Concretely, we modify the (i, j)-element of A in Eq. (3) further with the edge encoding c_{ij} as:

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i, v_j)} + c_{ij}, \text{ where } c_{ij} = \frac{1}{N} \sum_{n=1}^{N} x_{e_n} (w_n^E)^T, \tag{7}$$

where x_{e_n} is the feature of the n-th edge e_n in SP_{ij}, $w_n^E \in \mathbb{R}^{d_E}$ is the n-th weight embedding, and d_E is the dimensionality of edge feature.

3.2 Implementation Details of GraphFormer

GraphFormer Layer. GraphFormer is built upon the classic Transformer encoder based on the original implementation described in [42]. In addition, we apply the layer normalization (LN) before the multi-head self-attention (MHA) and the feed-forward blocks (FFN) instead of after [45]. This modification has been unanimously adopted by all current Transformer implementations because it leads to more effective optimization [37]. Especially, for FFN sub-layer, we set the dimensionality of input, output, and the inner-layer to d. We formally characterize the GraphFormer layer as below:

$$h^{'(l)} = \text{MHA}(\text{LN}(h^{(l-1)})) + h^{(l-1)} \tag{8}$$

$$h^{(l)} = \text{FFN}(\text{LN}(h^{'(l)})) + h^{'(l)} \tag{9}$$

Special Node. As stated in the previous section, various graph pooling functions are proposed to represent the graph embedding. Inspired by [15], in GraphFormer, we add a special node called `[VNode]` to the graph, and make connection between `[VNode]` and each node individually. In the AGGREGATE-COMBINE step, the representation of `[VNode]` has been updated as normal nodes in graph, and the representation of the entire graph h_G would be the node feature of `[VNode]` in the final layer. In the BERT model [11, 33], there is a similar token, i.e., `[CLS]`, which is a special token attached at the beginning of each sequence, to represent the sequence-level feature on downstream tasks. While the `[VNode]` is connected to all other nodes in graph, which means the distance of shortest path is 1 for any $\phi(\texttt{[VNode]}, v_j)$ and $\phi(v_i, \texttt{[VNode]})$, the connection is not physical. To distinguish the connection of physical and virtual, inspired by [24], we reset all spatial encodings for $b_{\phi(\texttt{[VNode]}, v_j)}$ and $b_{\phi(v_i, \texttt{[VNode]})}$ to a distinct learnable scalar.

3.3 How Powerful is GraphFormer?

In the previous subsections, we introduce three structural encodings and the architecture of GraphFormer. Then a natural question is: *Do these modifications make GraphFormer more powerful than other GNN variants?* In this subsection, we first give an affirmative answer by showing that GraphFormer can represent the AGGREGATE and COMBINE steps in popular GNN models:

Fact 1. *By choosing proper weights and distance function ϕ, the GraphFormer layer can represent AGGREGATE and COMBINE steps of popular GNN models such as GIN, GCN, GraphSAGE.*

The proof sketch to derive this result are: 1) Spatial encoding enables self-attention module to distinguish neighbor set $\mathcal{N}(v_i)$ of node v_i so that the softmax function can calculate mean statistics over $\mathcal{N}(v_i)$; 2) Knowing the degree of a node, mean over neighbors can be translated to sum over neighbors; 3) With multiple heads and FFN, representations of v_i and $\mathcal{N}(v_i)$ can be processed separately and combined together later. We defer the proof of this fact to Appendix A.

Moreover, we show further that by using our spatial encoding, GraphFormer can go beyond classic message passing GNNs whose expressive power is no more than the 1-Weisfeiler-Lehman (WL) test. We give a concrete example in Appendix A to show how GraphFormer helps distinguish graphs that the 1-WL test fails to.

Connection between Self-attention and Virtual Node. Besides the superior expressiveness than popular GNNs, we also find an interesting connection between using self-attention and the virtual node heuristic [15, 29, 23, 21]. As shown in the leaderboard of OGB [21], the virtual node trick, which augments graphs with additional supernodes that are connected to all nodes in the original graphs, can significantly improve the performance of existing GNNs. Conceptually, the benefit of the virtual node is that it can aggregate the information of the *whole graph* (like the READOUT function) and then propagate it to *each node*. However, a naive addition of a supernode to a graph can potentially lead to inadvertent over-smoothing of information propagation [23]. We instead find that such a graph-level aggregation and propagation operation can be naturally fulfilled by vanilla self-attention without additional encodings. Concretely, we can prove the following fact:

Fact 2. *By choosing proper weights, every node representation of the output of a GraphFormer layer without additional encodings can represent MEAN READOUT functions.*

This fact takes the advantage of self-attention that each node can attend to all other nodes. Thus it can simulate graph-level READOUT operation to aggregate information from the whole graph. Besides the theoretical justification, we empirically find that GraphFormer does not encounter the problem of over-smoothing, which makes the improvement scalable. The fact also inspires us to introduce a special node for graph readout (see the previous subsection).

4 Experiments

We first conduct experiments on the recent OGB-LSC [20] quantum chemistry regression (i.e., PCQM4M-LSC) challenge, which is currently the biggest graph-level prediction dataset and contains more than 3.8M graphs in total. Then, we report the results on the other three popular tasks: ogbg-molhiv, ogbg-molpcba and ZINC, which come from the OGB [21] and benchmarking-GNN [14] leaderboards. Finally, we ablate the important design elements of GraphFormer. A detailed description of datasets and training strategies could be found in Appendix B.

4.1 OGB Large-Scale Challenge

Baselines. We benchmark the proposed GraphFormer with GCN [25] and GIN [46], and their variants with virtual node (-VN) [15]. They achieve the state-of-the-art valid and test mean absolute error (MAE) on the official leaderboard[2] [20]. In addition, we compare to GIN's multi-hop variant [5], and 12-layer deep graph network DeeperGCN [28], which also show promising performance on other leaderboards. We further compare our GraphFormer with the recent Transformer-based graph model GT [13].

Settings. We primarily report results on two model sizes: **GraphFormer** ($L = 12, d = 768$), and a smaller one **GraphFormer$_{\text{SMALL}}$** ($L = 6, d = 512$). Both the number of attention heads in the attention module and the dimensionality of edge features d_E are set to 32. We use AdamW as the optimizer, and set the hyperparameter ϵ to 1e-8 and $(\beta 1, \beta 2)$ to (0.99,0.999). The peak learning rate is set to 2e-4 (3e-4 for **GraphFormer$_{\text{SMALL}}$**) with a 60k-step warm-up stage followed by a linear decay learning rate scheduler. The total training steps are 1M. The batch size is set to 1024. All models are trained on 8 NVIDIA V100 GPUS for about 2 days.

Results. Table 1 summarizes performance comparisons on PCQM4M-LSC dataset. From the table, GIN-VN achieves the previous state-of-the-art validate MAE of 0.1395. The original implementation of GT [13] employs a hidden dimension of 64 to reduce the total number of parameters. For a fair comparison, we also report the result by enlarging the hidden dimension to 768, denoted by GT-Wide, which leads to a total parameter of 83.2M. While, both GT and GT-Wide do not outperform GIN-VN and DeeperGCN-VN. Especially, we do not observe a performance gain along with the growth of parameters of GT.

Compared to the previous state-of-the-art GNN architecture, GraphFormer noticeably surpasses GIN-VN by a large margin, e.g., 11.5% relative validate MAE decline for GraphFormer comparing to previous state-of-the-art. Noting that the test dataset is publicly unavailable, and the test MAE of

[2] https://github.com/snap-stanford/ogb/tree/master/examples/lsc/pcqm4m#performance

Table 1: Results on PCQM4M-LSC. * indicates the results are cited from the official leaderboard [20].

method	#param.	train MAE	validate MAE	test MAE
GCN [25]	2.0M	0.1318	0.1691 (0.1684*)	0.1791*
GIN [46]	3.8M	0.1203	0.1537 (0.1536*)	0.1543*
GCN-VN [25, 15]	4.9M	0.1225	0.1485 (0.1510*)	0.1603*
GIN-VN [46, 15]	6.7M	0.1150	0.1395 (0.1396*)	0.1419*
GINE-VN [5, 15]	13.2M	0.1248	0.1430	-
DeeperGCN-VN [28, 15]	25.5M	0.1059	0.1398	-
GT [13]	0.6M	0.0944	0.1400	-
GT-Wide [13]	83.2M	0.0955	0.1408	-
GraphFormer$_{\text{SMALL}}$	12.5M	0.0778	0.1264	-
GraphFormer	47.1M	**0.0582**	**0.1234**	**0.1328**

Table 2: Results on MolPCBA.

method	#param.	AP (%)
DeeperGCN-VN+FLAG [28]	5.6M	28.42±0.43
DGN [2]	6.7M	28.85±0.30
GINE-VN [5]	6.1M	29.17±0.15
PHC-GNN [27]	1.7M	29.47±0.26
GINE-APPNP [5]	6.1M	29.79±0.30
GIN-VN[46] (fine-tune)	3.4M	29.02±0.17
GraphFormer-FLAG	119.5M	**31.39**±0.32

Table 3: Results on MolHIV.

method	#param.	AUC (%)
GCN-GraphNorm [5, 8]	526K	78.83±1.00
PNA [10]	326K	79.05±1.32
PHC-GNN [27]	111K	79.34±1.16
DeeperGCN-FLAG [28]	532K	79.42±1.20
DGN [2]	114K	79.70±0.97
GIN-VN[46] (fine-tune)	3.3M	77.80±1.82
GraphFormer-FLAG	47.0M	**80.51**±0.53

GraphFormer is evaluated by the OGB team on 5% of test data through the first-stage test submission[3]. As stated in Section 3.3, we further find that the proposed GraphFormer does not encounter the problem of over-smoothing, i.e., the train and validate error keep going down along with the growth of depth and width of models.

4.2 Graph Classification

In this section, we further investigate the performance of GraphFormer on commonly used graph-level prediction tasks of popular leaderboards, i.e., OGB [21] (OGBG-MolPCBA, OGBG-MolHIV), and benchmarking-GNN [14] (ZINC). Since that pre-training is encouraged by OGB, we mainly explore the transferable capability of the pre-trained GraphFormer on OGB-LSC. Please note that the model configurations, hyper-parameters, and the pre-training performance of pre-trained GraphFormers used for MolPCBA and MolHIV are different from the models used in the previous subsection. Please refer to Appendix B for detailed descriptions. For benchmarking-GNN, which does not encourage large pre-trained model, we train an additional GraphFormer$_{\text{SLIM}}$ ($L = 12, d = 80$, total param.$= 489K$) from scratch on ZINC.

Baselines. We report performance of GNNs which achieve top-performance on the official leaderboards[4] *without additional domain-specific features*. Considering that the pre-trained GraphFormer leverages external data, for a fair comparison on OGB datasets, we additionally report performance for fine-tuning GIN-VN pre-trained on PCQM4M-LSC dataset, which achieves the previous state-of-the-art valid and test MAE on that dataset.

Settings. We report detailed training strategies in Appendix B. In addition, GraphFormer is more easily trapped in the over-fitting problem due to the large size of the model and the small size of the dataset. Therefore, we employ a widely used data augmentation for graph - FLAG [26], to mitigate the over-fitting problem on OGB datasets.

Results. Table 2, 3 and 4 summarize performance of GraphFormer comparing with other GNNs on MolHIV, MolPCBA and ZINC datasets. GraphFormer consistently and significantly outperforms

[3]The second-stage full evaluation will be opened after NeruIPS submission deadline.

[4]`https://ogb.stanford.edu/docs/leader_graphprop/`
`https://github.com/graphdeeplearning/benchmarking-gnns/blob/master/docs/07_leaderboards.md`

Table 4: Results on ZINC.

method	#param.	test MAE
GIN [46]	509,549	0.526±0.051
GraphSage [18]	505,341	0.398±0.002
GAT [43]	531,345	0.384±0.007
GCN [25]	505,079	0.367±0.011
GatedGCN-PE [4]	505,011	0.214±0.006
MPNN (sum) [15]	480,805	0.145±0.007
PNA [10]	387,155	0.142±0.010
GT [13]	588,929	0.226±0.014
$\text{GraphFormer}_{\text{SLIM}}$	489,321	**0.122**±0.006

Table 5: Ablation study results on PCQM4M-LSC dataset with different designs.

Node Relation Encoding		Centrality	Edge Encoding			valid MAE
Laplacian PE[13]	Spatial		via node	via Aggr	via attn bias(Eq.7)	
-	-	-	-	-	-	0.2276
✓	-	-	-	-	-	0.1483
-	✓	-	-	-	-	0.1427
-	✓	✓	-	-	-	0.1396
-	✓	✓	✓	-	-	0.1328
-	✓	✓	-	✓	-	0.1327
-	✓	✓	-	-	✓	0.1304

previous state-of-the-art GNNs on all three datasets by a large margin. Specially, except GraphFormer, the other pre-trained GNNs do not achieve competitive performance, which is in line with previous literature [19]. In addition, we conduct more comparisons to fine-tuning the pre-trained GNNs, please refer to Appendix C.

4.3 Ablation Studies

We perform a series of ablation studies on the importance of designs in our proposed GraphFormer, on PCQM4M-LSC dataset. The ablation results are included in Table 5. To save the computation resources, the Transformer models in table 5 have 12 layers, and are trained for 100K iterations.

Node Relation Encoding. We compare previously used positional encoding (PE) to our proposed spatial encoding, which both aim to encode the information of distinct node relation to Transformers. There are various PEs employed by previous Transformer-based GNNs, e.g., Weisfeiler-Lehman-PE (WL-PE) [50] and Laplacian PE [3, 14]. We report the performance for Laplacian PE since it performs well comparing to a series of PEs for Graph Transformer in previous literature [13]. Transformer architecture with the spatial encoding outperforms the counterpart built on the positional encoding, which demonstrates the effectiveness of using spatial encoding to capture the node spatial information.

Centrality Encoding. Transformer architecture with simple degree-based centrality encoding yields a large margin performance boost in relation to those without centrality information. This indicates that the centrality encoding is indispensable to Transformer architecture for modeling graph data.

Edge Encoding. We compare our proposed edge encoding (denoted as via attn bias) to two commonly used edge encodings described in Section 3.1.3 to incorporate edge features into GNN, denoted as via node and via Aggr in Table 5. From the table, the gap of performance is minor between the two conventional methods, but our proposed edge encoding performs significantly better, which indicates that edge encoding as attention bias is more effective for Transformer to capture spatial information on edges.

5 Related Work

In this section, we highlight the most recent works which attempt to develop standard Transformer architecture-based GNN or graph structural encoding, but spend less effort on elaborating the works by adapting attention mechanism to GNNs [31, 49, 7, 22, 1, 43, 44, 50, 41].

5.1 Graph Transformer

There are several works that study the performance of pure Transformer architectures (stacked by transformer layers) with modifications on graph representation tasks, which are more related to our GraphFormer. For example, several parts of the transformer layer are modified in [39], including an additional GNN employed in attention sub-layer to produce vectors of Q, K, and V, long-range residual connection, and two branches of FFN to produce node and edge representations separately. They pre-train their model on 10 million unlabelled molecules and achieve excellent results by fine-tuning on downstream tasks. Very recently, Dwivedi *et al.* [13] revisit a series of works for Transformer-based GNNs, and suggest that the attention mechanism in Transformers on graph data should only aggregate the information from neighborhood (i.e., using adjacent matrix as attention mask) to ensure graph sparsity, and propose to use Laplacian eigenvector as positional encoding. Their model GT surpasses baseline GNNs on graph representation task.

5.2 Structural Encodings in GNNs

Path and Distance in GNNs. Information of path and distance is commonly used in GNNs. For example, an attention-based aggregation is proposed in [9] where the node features, edge features, one-hot feature of the distance and ring flag feature are concatenated to calculate the attention probabilites; a distance-weighted aggregation scheme on graph is proposed in [48]; it has been proved in [30] that adopting distance encoding (i.e., one-hot feature of the distance as extra node attribute) could lead to a strictly more expressive power than the 1-WL test; similar to [9], path-based attention is leveraged in [47] to model the influence between the center node and its higher-order neighbors.

Positional Encoding in Transformer on Graph. Several works introduce positional encoding (PE) to Transformer-based GNNs to help the model capture the node position information. For example, Graph-BERT [50] introduces three types of PE to embed the node position information to model, i.e., an absolute WL-PE which represents different codes labeled by Weisfeiler-Lehman algorithm, an intimacy based PE and a hop based PE which are both variant to the sampled subgraphs. Absolute Laplacian PE is employed in [13] and empircal study shows that its performance surpasses the absolute WL-PE used in [50].

Edge Feature. Except the conventionally used methods to encode edge feature, which are described in previous section, there are several attempts that exploit how to better encode edge features: an attention-based GNN layer is developed in [16] to encode edge features. To be specific, the edge feature is weighted by the similarity of the features of its two nodes; edge feature has been encoded into the popular GIN [46] in [5]; in [13], the authors propose to project edge features to an embedding vector, then multiply it by attention coefficients, and send the result to an additional FFN sub-layer to produce edge representations.

6 Conclusion

We have explored the direct application of Transformers to graph representation. With three simple, yet effective graph structural encodings, the proposed GraphFormer works surprisingly well on a wide range of popular benchmark datasets. While these initial results are encouraging, many challenges remain. For example, the quadratic complexity of the self-attention module restricts GraphFormer's application on large graphs. Therefore, future development of efficient GraphFormer is necessary. Performance improvement could be expected by leveraging domain knowledge-powered encodings on particular graph datasets. Finally, an applicable graph sampling strategy is desired for node representation extraction with GraphFormer. We leave them for future works.

References

[1] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. *ICLR*, 2021.

[2] Dominique Beaini, Saro Passaro, Vincent Létourneau, William L Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. In *International Conference on Machine Learning*, 2021.

[3] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[4] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

[5] Rémy Brossard, Oriel Frigo, and David Dehaene. Graph convolutions that can finally model local structure. *arXiv preprint arXiv:2011.15069*, 2020.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[7] Deng Cai and Wai Lam. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7464–7471, 2020.

[8] Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-yan Liu, and Liwei Wang. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning*, 2021.

[9] Benson Chen, Regina Barzilay, and Tommi Jaakkola. Path-augmented graph transformer network. *arXiv preprint arXiv:1905.12712*, 2019.

[10] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33, 2020.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[13] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.

[14] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[15] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

[16] Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9211–9219, 2019.

[17] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

[18] William L Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[19] W Hu, B Liu, J Gomes, M Zitnik, P Liang, V Pande, and J Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[20] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

[21] 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.

[22] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020.

[23] Katsuhiko Ishiguro, Shin-ichi Maeda, and Masanori Koyama. Graph warp module: an auxiliary module for boosting the power of graph neural networks in molecular graph analysis. *arXiv preprint arXiv:1902.01020*, 2019.

[24] Guolin Ke, Di He, and Tie-Yan Liu. Rethinking the positional encoding in language pre-training. *ICLR*, 2020.

[25] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[26] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks. *arXiv preprint arXiv:2010.09891*, 2020.

[27] Tuan Le, Marco Bertolini, Frank Noé, and Djork-Arné Clevert. Parameterized hypercomplex graph neural networks for graph classification. *arXiv preprint arXiv:2103.16584*, 2021.

[28] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[29] Junying Li, Deng Cai, and Xiaofei He. Learning graph-level representation for drug discovery. *arXiv preprint arXiv:1709.03741*, 2017.

[30] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[31] Yuan Li, Xiaodan Liang, Zhiting Hu, Yinbo Chen, and Eric P. Xing. Graph transformer, 2019.

[32] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. *arXiv preprint arXiv:1808.10568*, 2018.

[33] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[34] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[35] P David Marshall. The promotion and presentation of the self: celebrity as marker of presentational media. *Celebrity studies*, 1(1):35–48, 2010.

[36] Alice Marwick and Danah Boyd. To see and be seen: Celebrity practice on twitter. *Convergence*, 17(2):139–158, 2011.

[37] Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, Thibault Fevry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, et al. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

[38] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[39] 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, 2020.

[40] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, 2018.

[41] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

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

[44] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Direct multi-hop attention based graph neural network. *arXiv preprint arXiv:2009.14332*, 2020.

[45] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.

[46] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[47] Yiding Yang, Xinchao Wang, Mingli Song, Junsong Yuan, and Dacheng Tao. Spagan: Shortest path graph attention network. *Advances in IJCAI*, 2019.

[48] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In *International Conference on Machine Learning*, pages 7134–7143. PMLR, 2019.

[49] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[50] 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.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]