LGI-GT: Graph Transformers with Local and Global Operators Interleaving

Shuo Yin , Guoqiang Zhong*

College of Computer Science and Technology, Ocean University of China
yinshuo@stu.ouc.edu.cn, gqzhong@ouc.edu.cn

Abstract

Since Transformers can alleviate some critical and fundamental problems of graph neural networks (GNNs), such as over-smoothing, over-squashing and limited expressiveness, they have been successfully applied to graph representation learning and achieved impressive results. However, although there are many works dedicated to make graph Transformers (GTs) aware of the structure and edge information by specifically tailored attention forms or graph-related positional and structural encodings, few works address the problem of how to construct high-performing GTs with modules of GNNs and Transformers. In this paper, we propose a novel graph Transformer with local and global operators interleaving (LGI-GT), in which we further design a new method propagating embeddings of the [CLS] token for global information representation. Additionally, we propose an effective message passing module called edge enhanced local attention (EELA), which makes LGI-GT a full-attention GT. Extensive experiments demonstrate that LGI-GT performs consistently better than previous state-of-the-art GNNs and GTs, while ablation studies show the effectiveness of the proposed LGI scheme and EELA. The source code of LGI-GT is available at https://github.com/shuoyinn/LGI-GT.

1 Introduction

In recent years, deep learning approaches for graph-structured data have been increasingly popular due to the great successes of graph neural networks (GNNs). While the convolutional neural networks (CNNs) [LeCun *et al.*, 1989] and recurrent neural networks (RNNs) [Elman, 1990] are good at handling grid-like and sequential data respectively, the inherent ability to deal with irregular data (i.e., graphs) and deeply learn their representations make GNNs to be widely applied in many areas, such as data mining and information retrieval. Broadly speaking, most GNNs are instances of a general framework called message passing neural network (MPNN) [Gilmer *et al.*, 2017], and many effective models have been proposed, such as GCN [Kipf and Welling, 2017], GraphSAGE [Hamilton *et al.*, 2017], GIN [Xu *et al.*, 2019], APPNP [Klicpera *et al.*, 2019], and PNA [Corso *et al.*, 2020].

However, the performance of aforementioned GNNs has been inevitably limited by some critical and fundamental problems, including over-smoothing [Li *et al.*, 2018; Chen *et al.*, 2020; Oono and Suzuki, 2020], over-squashing [Alon and Yahav, 2021] and limited expressiveness [Morris *et al.*, 2019]. Concretely, over-smoothing means that with the model becoming sufficiently deep, all node representations will converge to the same and cannot be differentiated any more, while over-squashing refers to that information from a large receptive field is "squashed" into some fixed-length vectors. Due to over-smoothing and over-squashing, GNNs cannot be too deep and meanwhile a node cannot effectively interact with another far away. In addition, message passing GNNs actually encode the rooted subtree around each node, thus their expressive ability is bounded by the 1-Weisfeiler-Lehman (1-WL) graph isomorphism test [Xu *et al.*, 2019], i.e., expressiveness of GNNs is limited.

Recently, Transformer [Vaswani *et al.*, 2017] has shown its dominance in the field of deep learning and Transformer-based models have performed the best on many computer vision and natural language processing tasks. Considering the great success of Transformer and its potential to address the critical issues of GNNs, graph Transformers (GT) [Dwivedi and Bresson, 2020; Kreuzer *et al.*, 2021] are proposed and have been attracting more and more attention.

Particularly, there are two main categories of GTs. The first category (**Type I**) combines Transformer encoder with connectivity information and edge features via specially tailored attention forms or graph-related positional and structural encodings (PE/SE), i.e., the Transformer encoder is still taken as the network backbone, but adapted for graph structures. Examples are [Dwivedi and Bresson, 2020], Graphormer [Ying *et al.*, 2021], SAN [Kreuzer *et al.*, 2021], UniMP (TransformerConv) [Shi *et al.*, 2021] and GRPE [Park *et al.*, 2022]. This class of GTs do not employ the relatively independent GNN modules. The second one (**Type II**), however, explicitly utilizes both the message passing GNN modules and Transformer encoder layers, such as GraphTrans [Wu *et al.*, 2021], SAT [Chen *et al.*, 2022] and GPS [Rampasek *et al.*, 2022]. Generally, in Type I methods, the local and global operations

*Corresponding author

are tightly coupled (i.e., node interactions within a neighborhood and over the entire graph are simultaneously considered) , while in Type II models they are loosely coupled. Although model structures of Type I are usually more complex, state-of-the-art (SOTA) results on many benchmarks [Dwivedi *et al.*, 2020; Hu *et al.*, 2020a] are obtained by GTs from Type II.

Specifically, there are several manners to construct GTs in Type II, and among them, GNN+Transformer (i.e., stacking a multi-layer Transformer on a multi-layer GNN, like Graph-Trans [Wu *et al.*, 2021]) and parallelization (i.e., feeding a same input into a GNN layer and a global self-attention layer independently in each block then followed by a fusing operation, like GPS [Rampasek *et al.*, 2022]) are two effective schemes. We argue that both of these schemes have shortcomings in model structures, i.e., global and local information cannot be sufficiently utilized. To overcome this problem, we propose a new way to construct GTs with GNN layers and Transformer layers.

In this paper, we introduce a novel architecture of GTs, in which **l**ocal and **g**lobal operators **i**nterleave (**LGI**). **Local operator** refers to a family of message passing GNN modules since they aggregate information from local neighborhoods, while **global operator** refers to any global attention–based methods like Transformer. Hence, we alternately place few GNN layers and few Transformer layers in sequence and repeat this several times. Via such an LGI scheme, GTs can sufficiently fuse information from neighboring nodes locally and distant nodes globally. For this reason, we call our proposed model **LGI-GT**, which can to some extent address the critical and fundamental problems of traditional GNNs and perform better than previous GTs. In particular, to propagate global information in LGI-GT, we design a new method to forward propagate embeddings of the `[CLS]` token [Devlin *et al.*, 2018] in a skip manner. Additionally, we propose a novel message passing method called **e**dge **e**nhanced **l**ocal **a**ttention (**EELA**), which can be leveraged as a local operator in LGI-GT and makes LGI-GT a full-attention GT model (both local and global operators are based on attention mechanisms).

Our main contributions can be summarized as follows:

- We propose LGI-GT, which is a GT model arranging message passing GNN modules and Transformers (as local operators and global operators, respectively) in an interleaving way.
- We propose a novel attention-based message passing module called EELA, which enables LGI-GT to be a full-attention GT.
- We empirically demonstrate that LGI-GT outperforms the state-of-the-art GNNs and GTs, by applying it to multiple graph-level and node-level tasks. In addition, we validate the effectiveness of EELA via an ablation study, and show the superiority of the LGI scheme via a comparison experiment, a depth study and visualization.

2 Related Work

In this section, we review the two categories of existing GTs.

Type I GTs. Dwivedi and Bresson [2020] propose an early example using self-attention to construct Transformer for processing graphs with Laplacian eigenvectors as positional encodings. In their model, attention is performed only on the neighbors instead of all the nodes of the graph. Hence, it can be seen as an example of the message passing network and very like to GAT [Velickovic *et al.*, 2018] with different attention implementation details. Likewise, Transformer-Conv designed in [Shi *et al.*, 2021] also uses local attention only over the 1-hop neighborhood. They adapt the original multi-head attention computing formula for graphs with edge attributes, and combine a gated mechanism to prevent from over-smoothing. Since their model is proposed for semi-supervised learning on graphs with nodes partially labeled, they additionally employ a new input for the Transformer by combining node features and node labels. SAN [Kreuzer *et al.*, 2021] firstly uses an auxiliary Transformer mapping eigenvectors and eigenvalues of the graph's Laplacian matrix to a learned positional encoding; then a main Transformer works using full-graph attention. Attention computing formulas of the main Transformer are tailored, by replacing the dot product operation (between key vectors and query vectors) with another one involving edge encoding vectors. Therefore, besides global dependencies, it also considers local connectivity and edge features. Graphormer [Ying *et al.*, 2021] leverages node degree and shortest path to get centrality encoding, spatial encoding and edge encoding, while the attention computation is altered by adding two bias terms to the scaled dot product of query and key before softmax. Moreover, GRPE [Park *et al.*, 2022] adopts relative positional encoding to realize tight integration of node-edge and node-spatial information. By learning the spatial relation embeddings and edge embeddings, spatial bias and edge bias are inserted into the attention score computation formula.

Type II GTs. Rong *et al.* [2020] design their own message passing GNN and use three such GNNs (with different parameters) to get query, key and value vectors respectively for the consequent self-attention. A great contribution of their work is the pre-training procedure, where they devise two new self-supervised tasks for molecule property prediction. Baek *et al.* [2021] focus on methods of graph pooling. Their graph multiset Transformer (GMT) takes Transformer as a pooling method and coarsens an original graph layer by layer. Attention of GMT uses two different GNNs for key and value, and by learning query seed vectors whose number is fewer than the number of real nodes, the graph is coarsened once in each layer. Strictly speaking, GMT is not a graph Transformer, but a GNN with a Transformer as its pooling method. To better capture long-range pairwise dependencies, Graph-Trans [Wu *et al.*, 2021] is proposed, as a combined model where a multi-layer Transformer is on the top of a multi-layer GNN (an example of the GNN+Transformer scheme as mentioned in Section 1). From the perspective of kernel methods, SAT [Chen *et al.*, 2022] utilizes a shared GNN to get query and key vectors and thus implement a particular kernel function as their own attention score formula; besides, it uses k-subgraph GNNs [Zhang and Li, 2021] (instead of k-subtree ones [Xu *et al.*, 2019]) as more expressive node information

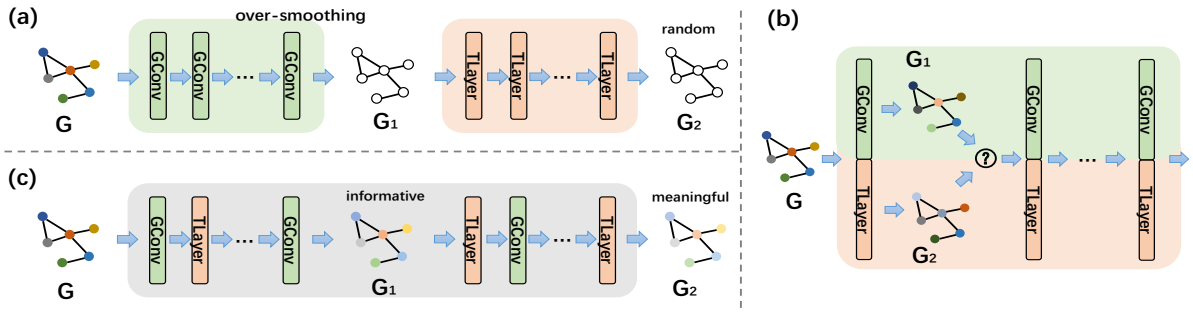

Figure 1: GNN+Transformer scheme (a): G_1 is the output of the k-th GConv and embeddings of the nodes may be the same due to over-smoothing, thus meaningless to TLayers behind. Parallel scheme (b): in each block, a same input (graph G) is fed into a GConv and a TLayer in parallel, and the outputs of them G_1 and G_2 are fused into one somehow, while simple fusing methods like summation or concatenation may limit the model capacity. Our interleaving scheme (c): TLayers alleviate the over-smoothing problem, thus intermediate node representations are informative after modeling local pattern via GConvs and global pattern via TLayers; meanwhile, there is no need to consider fusing information from different blocks.

extractors beyond the 1-WL test. Furthermore, Rampasek *et al.* [2022] propose GPS as a GT framework, including kinds of absolute/relative positional/structural encodings into GTs. As for their model architecture, they parallelize GNN and self-attention modules, and then get the sum of their outputs before the feed forward network (FFN) (an example of the parallel scheme as mentioned in Section 1).

In this work, we design a novel GT by interleaving GNN layers and Transformer layers to better leverage the advantages of both. Our proposed LGI-GT belongs to Type II GTs and also keeps loosely-coupled between the message passing and global self-attention.

3 Model Structure Analysis

Since convolution is typically used to extract information of a local patch in an image [LeCun *et al.*, 1989], graph convolution is considered as the similar operation, which is indeed a general scheme representing a class of local message passing methods rather than only GCN [Kipf and Welling, 2017]. A graph local operator refers to an instance of this scheme, usually with a specific message construction function followed by a specific aggregation manner, and we name it **GConv** for the rest of this paper.

The Transformer encoder layer (hereinafter referred to as **TLayer**) is used as the global operator, directly modeling dependencies of any two nodes via full-graph attention. Note that many Transformer models can be used as TLayer here, such as Performer [Choromanski *et al.*, 2021] and BigBird [Zaheer *et al.*, 2020].

The main advantage of our proposed LGI-GT over the other two constructing schemes is the interleaving manner where GConvs and TLayers are alternately arranged. Given a graph G, k GConvs and k TLayers, we compare the GNN+Transformer scheme, the parallel scheme and our interleaving scheme in Figure 1. We first show a defect of the GNN+Transformer scheme, as follows.

A graph Transformer in the GNN+Transformer scheme still has the risk of over-smoothing. See Figure 1 (a). Assume that k is just the depth, that makes a GNN stacked by GConvs incur over-smoothing. Therefore, $G_1 = GConv(GConv(\dots GConv(G)\dots))$ is already an over-smoothed graph, meaning that the features of all nodes in G_1 are basically the same. Since a Transformer stacked by k TLayers only depends on node features and handles the graph without considering node connectivity or edge features, G_1 as its input is of no meaning for it. As a result, the final output of the whole model $G_2 = TLayer(TLayer(\dots TLayer(G_1)\dots))$ is also meaningless, i.e., over-smoothing occurs and thus makes the model unable to perform well.

A graph Transformer in the parallel scheme needs a better fusion method other than simple summation. As for the parallel scheme, see Figure 1 (b), $G_1 = GConv(G)$ and $G_2 = TLayer(G)$ are computed independently and based on them local and global information are fused immediately, thus the aforementioned over-smoothing scenario may not happen. Nevertheless, it is yet to explore that which fusion method is suitable for the outputs of the parallel GConv and TLayer in each block. As an example, GPS chooses to sum them up, and then feeds the summation into an FFN. However, a simple summation without considering the significant difference of each part may limit the model capacity, so that this operation is too simple and may hurt the model performance.

Advantages of the LGI scheme. As mentioned above, in some cases, the GNN+Transformer models may fail to distinguish different nodes and thus be unable to well accomplish the learning task at hand. On the contrary, in the case where k GConvs and k TLayers are stacked in an interleaving manner (e.g., repeat 1 GConv + 1 TLayer for k times), TLayers can alleviate the over-smoothing problem. Node embeddings as the input of any TLayer are not only meaningful, but also fused with globally important information. Intuitively, while using this architecture, the intermediate node embeddings are informative and beneficial to downstream modules, as shown in Figure 1 (c). Additionally, since we arrange GConv and TLayer serially versus parallelly, there is no need to consider how to further handle outputs of these two modules, e.g., giving different weights to outputs of them if summation is used.

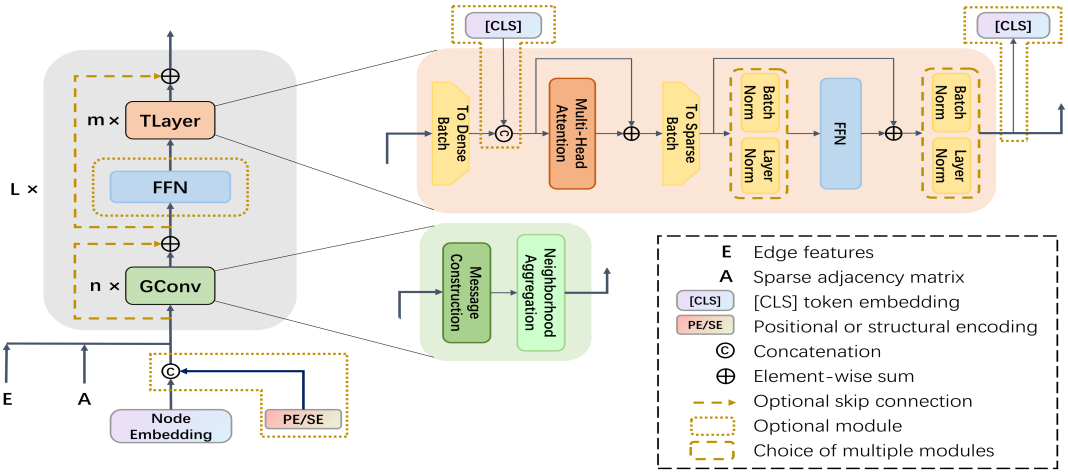

Figure 2: Overview of the architecture of our proposed graph Transformer with local and global operators interleaving.

4 Methodology

We first introduce the architecture of the proposed LGI-GT with a new algorithm to use the [CLS] token. To get a full-attention LGI-GT, we then propose a novel message passing module based on an attention mechanism, called EELA.

4.1 LGI-GT

Local and Global Operators Interleave

The LGI-GT architecture is shown in Figure 2, of which one block is named as an **LGI-Block**. The LGI-Block takes a graph as its input and outputs an isomorphic one with updated node embeddings. We formalize this procedure as follows:

$$\boldsymbol{X}^l = LGI\text{-}Block^{(l)}(\boldsymbol{X}^{l-1}, \boldsymbol{E}, \boldsymbol{A}), \tag{1}$$

where $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ and $\boldsymbol{E} \in \mathbb{R}^{|\mathcal{E}|\times F}$ denote adjacency matrix and F-dimensional edge features of a graph with N nodes and edge set $\mathcal{E}$, $LGI\text{-}Block^{(l)}$ denotes the l-th block of LGI-GT ($l = 1, 2, ..., L$), and $\boldsymbol{X}^l \in \mathbb{R}^{N\times F}$ denotes the intermediate embeddings of nodes and also the output of $LGI\text{-}Block^{(l)}$ except that $\boldsymbol{X}^0$ is the node features optionally concatenated with positional or structural encodings. For simplicity, we regard edge features as constants, but note that our method is a flexible and general framework, thus any GConv updating edge embeddings can be easily plugged into it and any PE/SE can be used by concatenating it to node or edge features.

$LGI\text{-}Block^{(l)}$ consists of GConvs and TLayers in order, regardless of an optional FFN. Thus Equation (1) can be decomposed into the following equations:

$$\hat{\boldsymbol{X}}^l = GConvs^{(l)}(\boldsymbol{X}^{l-1}, \boldsymbol{E}, \boldsymbol{A}), \tag{2}$$

$$\boldsymbol{X}^l = TLayers^{(l)}(\hat{\boldsymbol{X}}^l), \tag{3}$$

where $GConvs^{(l)}$ and $TLayers^{(l)}$ are just the two components in $LGI\text{-}Block^{(l)}$, as mentioned above, and $\hat{\boldsymbol{X}}^l \in \mathbb{R}^{N\times F}$ is the intermediate node embeddings after $GConvs^{(l)}$ and before $TLayers^{(l)}$.

Considering $GConvs^{(l)}$ has n layers and $TLayers^{(l)}$ has m as shown in Figure 2 (n and m are both small numbers such as 1 or 2), Equations (2) and (3) can be further decomposed, thus replaced by Equations (4) and (5):

$$\boldsymbol{H}_i^{(l)} = GConv_i^{(l)}(\boldsymbol{H}_{i-1}^{(l)}, \boldsymbol{E}, \boldsymbol{A}), \tag{4}$$

$$\hat{\boldsymbol{H}}_j^{(l)} = TLayer_j^{(l)}(\hat{\boldsymbol{H}}_{j-1}^{(l)}), \tag{5}$$

where $GConv_i^{(l)}$ is the i-th layer of $GConvs^{(l)}$ ($i = 1, 2, ..., n$), $\boldsymbol{H}_i^{(l)} \in \mathbb{R}^{N\times F}$ is the node intermediate representation within $GConvs^{(l)}$, $\boldsymbol{H}_0^{(l)} = \boldsymbol{X}^{l-1}$ and $\boldsymbol{H}_n^{(l)} = \hat{\boldsymbol{X}}^l$. Similarly, $TLayer_j^{(l)}$ is the j-th layer of $TLayers^{(l)}$ ($j = 1, 2, ..., m$), $\hat{\boldsymbol{H}}_j^{(l)} \in \mathbb{R}^{N\times F}$ is the node intermediate representation within $TLayers^{(l)}$, $\hat{\boldsymbol{H}}_0^{(l)} = \hat{\boldsymbol{X}}^l$ and $\hat{\boldsymbol{H}}_m^{(l)} = \boldsymbol{X}^l$. Note that, as shown in Figure 2, there are two batch transformations within each TLayer. Before and after the global self-attention are sparse-to-dense and dense-to-sparse, respectively.

To overcome shortcomings brought by depth, we design some skip connections within LGI-GT, e.g., at the beginning of the first GConv in each LGI-Block, we can set a skip connection to the beginning of the first TLayer.

A New Method to Propagate Embeddings of the [CLS] Token

We specially devise a procedure of propagating embeddings of the [CLS] token in LGI-GT, to guarantee in each block they skip all the local operators and only participate in global operators. Different from that in Graphormer [Ying *et al.*, 2021], GraphTrans [Wu *et al.*, 2021] and SAT [Chen *et al.*, 2022], [CLS] embeddings in our model are propagated in a skip manner. Specifically, the embeddings of [CLS] do not exist in any GConv, so that in forward pass, they keep constant within GConvs and are only updated within TLayers. When there is a learnable vector representing the [CLS] token, Equations (1), (3) and (5) should be modified to Equations (6), (7) and (8) for the special architecture of LGI-GT:

Algorithm 1 Updating the embeddings of [CLS]

Input: $\boldsymbol{CLS}^0$, sparse batching $\boldsymbol{X}^0$, $\boldsymbol{E}$, $\boldsymbol{A}$
Output: $\boldsymbol{CLS}^L$

for $l = 1, 2, ..., L$ **do**
  Calculate $\hat{\boldsymbol{X}}^l$ via Equation (2).
  $\hat{\boldsymbol{H}}_0^{(l)} = \hat{\boldsymbol{X}}^l$, $\boldsymbol{C}_0^{(l)} = \boldsymbol{CLS}^{l-1}$.
  for $j = 1, 2, ..., m$ **do**
    Let $\tilde{\boldsymbol{H}}_{j-1}^{(l)} = [\hat{\boldsymbol{H}}_{j-1}^{(l)}; \boldsymbol{C}_{j-1}^{(l)}]$.
    Transform $\tilde{\boldsymbol{H}}_{j-1}^{(l)}$ to the dense batch $\bar{\boldsymbol{H}}_{j-1}^{(l)}$.
    Calculate $\boldsymbol{O}_1$ via Equation (9).
    Transform $\boldsymbol{O}_1$ to the sparse batch $\hat{\boldsymbol{O}}_1$.
    Calculate $\boldsymbol{O}_2$ via Equation (10).
    $[\hat{\boldsymbol{H}}_j^{(l)}; \boldsymbol{C}_j^{(l)}] = \boldsymbol{O}_2$.
  end for
  $\boldsymbol{X}^l = \hat{\boldsymbol{H}}_m^{(l)}$, $\boldsymbol{CLS}^l = \boldsymbol{C}_m^{(l)}$.
end for
return $\boldsymbol{CLS}^L$

$$\boldsymbol{X}^l, \boldsymbol{CLS}^l = LGI\text{-}Block^{(l)}(\boldsymbol{X}^{l-1}, \boldsymbol{E}, \boldsymbol{A}, \boldsymbol{CLS}^{l-1}), \quad (6)$$

$$\boldsymbol{X}^l, \boldsymbol{CLS}^l = TLayers^{(l)}(\hat{\boldsymbol{X}}^{l-1}, \boldsymbol{CLS}^{l-1}), \quad (7)$$

$$\hat{\boldsymbol{H}}_j^{(l)}, \boldsymbol{C}_j^{(l)} = TLayer_j^{(l)}(\hat{\boldsymbol{H}}_{j-1}^{(l)}, \boldsymbol{C}_{j-1}^{(l)}), \quad (8)$$

where $\boldsymbol{CLS}^l \in \mathbb{R}^{1\times F}$ is the [CLS] embedding output by $LGI\text{-}Block^{(l)}$ (also the same output of $TLayers^{(l)}$) except that $\boldsymbol{CLS}^0$ is the original embedding of [CLS], whilst $\boldsymbol{C}_j^{(l)} \in \mathbb{R}^{1\times F}$ denotes the intermediate embedding of [CLS] output by $TLayer_j^{(l)}$, $\boldsymbol{C}_0^{(l)} = \boldsymbol{CLS}^{l-1}$ and $\boldsymbol{C}_m^{(l)} = \boldsymbol{CLS}^l$.

For completeness, we describe the global multi-head attention (MHA) and FFN equations regardless of the specific block or layer, thus ignoring l and j:

$$\boldsymbol{O}_1 = MHA(\bar{\boldsymbol{H}}), \quad (9)$$

$$\boldsymbol{O}_2 = FFN(\hat{\boldsymbol{O}}_1), \quad (10)$$

where $\bar{\boldsymbol{H}}, \boldsymbol{O}_1 \in \mathbb{R}^{BN_{max}\times F}$ are both densely batched node representations for B graphs, with N_{max} as the maximum node number, $\hat{\boldsymbol{O}}_1 \in \mathbb{R}^{N_{sum}\times F}$ is the sparse batch version of $\boldsymbol{O}_1$ with N_{sum} as the summation of node numbers, and $\boldsymbol{O}_2 \in \mathbb{R}^{N_{sum}\times F}$ is the output of FFN, also the output of the related TLayer. See Figure 2 for more details.

Algorithm 1 describes the procedure to update the embeddings of [CLS] , where the operator $[\cdot\,;\,\cdot]$ denotes concatenation along the node dimension.

4.2 Edge Enhanced Local Attention

We propose a new instance of GConv called EELA, using the message passing mechanism as well as local attention mechanism simultaneously. Figure 3 illustrates how a message is constructed in EELA. Edge feature matrix $\boldsymbol{E}$ is first concatenated to source node feature matrix $\boldsymbol{X_s}$ along the feature dimension and then get $\boldsymbol{K}$ and $\boldsymbol{V}$ via linear projection; meanwhile, $\boldsymbol{Q}$ is calculated only using target node feature matrix

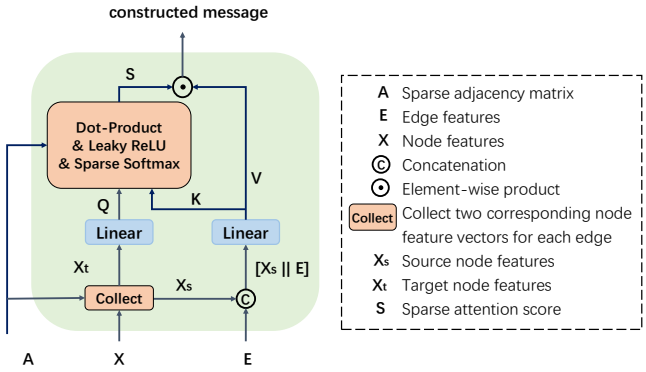

Figure 3: EELA message constructing procedure.

$\boldsymbol{X_t}$ via another linear layer. The whole procedure in terms of the central node i can be formalized as follows:

$$\hat{\alpha}_{i,j} = \frac{(\boldsymbol{W}_1\boldsymbol{x}_i)^\top(\boldsymbol{W}_2[\boldsymbol{x}_j \| \boldsymbol{e}_{i,j}])}{\sqrt{d}}, \quad (11)$$

$$\alpha_{i,j} = \frac{\exp(\hat{\alpha}_{i,j})}{\sum_{k\in\mathcal{N}(i)}\exp(\hat{\alpha}_{i,k})}, \quad (12)$$

$$\boldsymbol{x}'_i = \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\boldsymbol{W}_2[\boldsymbol{x}_j \| \boldsymbol{e}_{i,j}], \quad (13)$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{F\times F}$ is the query weight matrix, $\boldsymbol{W}_2 \in \mathbb{R}^{2F\times F}$ is the mutual weight matrix of key and value sharing a same linear module, d is the number of one head dimensions, $\boldsymbol{x}_j \in \mathbb{R}^{1\times F}$ and $\boldsymbol{e}_{i,j} \in \mathbb{R}^{1\times F}$ are the embedding vectors of node j and edge (i, j) respectively, $\boldsymbol{x}'_i \in \mathbb{R}^{1\times F}$ is the updated embedding vector of node i, $\mathcal{N}(i)$ is the 1-hop neighborhood of node i, and $[\cdot \| \cdot]$ represents concatenation along the feature dimension.

Different from GATConv (attention with too few paramters and without dot product) [Velickovic *et al.*, 2018] and TransformerConv (attention like the original Transformer) [Shi *et al.*, 2021], EELA not only uses linear projection to get $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$, but makes $\boldsymbol{V}$ identical to $\boldsymbol{K}$ and thus reduces the number of parameters while keeping the effectiveness of local attention.

5 Experiments

In this section, we first compared our method to the state-of-the-art baselines including both GNNs and GTs. To demonstrate the superiority of the LGI scheme over the GNN+Transformer scheme and the parallel scheme, we performed another comparison experiment, a depth study and a visualization study on these three GT constructing manners. In addition, we validated the effectiveness of the module EELA via an ablation study.

5.1 Comparison with the SOTA Methods

We conducted a comparison experiment on node-level and graph-level graph representation learning tasks using datasets from different sources for ensuring diversity.

Model	ZINC	PATTERN	CLUSTER	ogbg-molpcba	ogbg-code2
	MAE ↓	Accuracy ↑	Accuracy ↑	AP ↑	F1 score ↑
GCN	0.367 ± 0.011	71.89 ± 0.33	68.50 ± 0.98	0.2424 ± 0.0034	0.1595 ± 0.0018
GIN	0.526 ± 0.051	85.39 ± 0.14	64.72 ± 1.55	0.2703 ± 0.0023	0.1581 ± 0.0026
PNA	0.188 ± 0.004	–	–	0.2838 ± 0.0035	0.1570 ± 0.0032
DGN	0.168 ± 0.003	86.68 ± 0.03	–	0.2885 ± 0.0030	–
GIN-AK+	0.080 ± 0.001	86.85 ± 0.06	–	0.2930 ± 0.0044	–
CRaWl	0.085 ± 0.004	–	–	0.2986 ± 0.0025	–
SAN	0.139 ± 0.006	86.58 ± 0.04	76.69 ± 0.65	0.2765 ± 0.0042	–
GraphTrans	–	–	–	0.2761 ± 0.0029	0.1830 ± 0.0024
SAT	0.094 ± 0.008	86.85 ± 0.04	77.86 ± 0.10	–	0.1937 ± 0.0028
GPS	0.070 ± 0.004	86.69 ± 0.06	78.02 ± 0.18	0.2907 ± 0.0028	0.1894 ± 0.0024
LGI-GT (ours)	**0.069 ± 0.002**	**86.93 ± 0.04**	**78.19 ± 0.10**	**0.3040 ± 0.0029**	**0.1948 ± 0.0024**

Table 1: Comparison of LGI-GT to the SOTA methods on 5 benchmark datasets. The best results are highlighted in boldface, and lacking results for baselines in the original papers are marked as "–".

Datasets. Among all the datasets we tested on, ZINC, PATTERN, CLUSTER were from [Dwivedi *et al.*, 2020], whilst ogbg-molpcba and ogbg-code2 were from OGB [Hu *et al.*, 2020a]. Specifically, ZINC is used to do graph regression; PATTERN and CLUSTER target to classify nodes; ogbg-molpcba is for multi-task binary graph classification; ogbg-code2 related task is code summarization where code is represented by the abstract syntax trees processed as graphs, so it is also a graph-level task.

Baselines. There are two classes of baselines — GNNs and GTs. For GNNs, we used GCN [Kipf and Welling, 2017], GIN [Xu *et al.*, 2019], PNA [Corso *et al.*, 2020], DGN [Beaini *et al.*, 2021], GIN-AK+ [Zhao *et al.*, 2022] and CRaW1 [Tönshoff *et al.*, 2021]; for GTs, we used SAN [Kreuzer *et al.*, 2021], GraphTrans [Wu *et al.*, 2021], SAT [Chen *et al.*, 2022] and GPS [Rampasek *et al.*, 2022]. Among them, GraphTrans represents the GNN+Transformer scheme and GPS is an instance of the parallel scheme. Following [Rampasek *et al.*, 2022], here we did not compare with ensemble methods as well as models pre-trained on extra datasets.

Configurations of LGI-GT. On each dataset, we used the same number of hidden dimensions F and number of layers (or blocks) L as GPS. To exclude performance improvement brought by the advanced GNNs (e.g., PNA) or GNNs more expressive than the 1-WL test (e.g., GraphSNN [Wijesinghe and Wang, 2022] and ID-GNN [You *et al.*, 2021]), we only utilized the basic GINEConv [Xu *et al.*, 2019; Hu *et al.*, 2020b] and GCNConv [Kipf and Welling, 2017] as GConvs, except that EELA was employed for ogbg-molpcba to realize full-attention LGI-GT. As for PE/SE, only Random Walk Structural Encoding (RWSE) was used for ZINC, PATTERN and CLUSTER. To achieve a fair comparison, $m = n = 1$ were constant (never tuned) across all the datasets. As for the readout methods concerning the graph-level tasks, we used summation for ZINC, and `[CLS]` token (see Section 4.1) for ogbg-molpcba and ogbg-code2. Evaluation metrics and dataset splits were the same as in the original papers for each dataset. We took mean ± std of 10 runs with different random seeds as the final result.

Results. As shown in Table 1, all the statistics of the baselines are from [Rampasek *et al.*, 2022] or the original papers (some of GPS are from the github issues). From the results, we can see that the proposed LGI-GT performs consistently better than the SOTA approaches on all the 5 benchmark datasets, and the margins are relatively obvious.

5.2 The Effectiveness of Interleaving GConvs and TLayers

Besides on big datasets, we conducted another comparison experiment on four other smaller ones: NCI1, NCI109 [Wale *et al.*, 2008; Ivanov *et al.*, 2019], ogbg-molbbbp and ogbg-moltox21 [Hu *et al.*, 2020a].

Configurations. On each dataset, we used the same number of GConv, TLayer and hidden dimensions for different schemes. Particularly, we set $n = 2, m = 1$ for LGI-GT on NCI1, NCI109 and ogbg-moltox21 to try a different combination of n and m, while on ogbg-molbbbp we still set $n = 1$ as the size is too small. As Section 5.1, on ogbg-molbbbp and ogbg-tox21, we took mean ± std of 10 runs with different random seeds; while on NCI1 and NCI109 we took mean ± std of 20 runs following [Wu *et al.*, 2021].

Model	NCI1	NCI109	ogbg-molbbbp	ogbg-moltox21
	Accuracy ↑	Accuracy ↑	Accuracy ↑	Accuracy ↑
GNN	80.72 ± 2.03	81.70 ± 1.85	69.45 ± 1.36	73.38 ± 0.68
Transformer	65.46 ± 2.74	66.53 ± 2.15	64.49 ± 3.48	73.23 ± 0.57
GNN +Transformer	81.84 ± 2.23	82.79 ± 1.94	69.96 ± 2.50	75.73 ± 0.86
Parallel GT	**82.32 ± 1.79**	82.34 ± 1.89	69.10 ± 1.01	74.99 ± 0.56
LGI-GT	82.18 ± 1.90	**83.36 ± 1.89**	**70.54 ± 0.80**	**78.17 ± 0.57**

Table 2: Comparison experiment on 4 benchmark datasets, showing that neither GNN nor Transformer alone does not perform well enough and the LGI scheme delivers better results. The best results are highlighted in boldface.

Results. From Table 2, we can see that both GConv and TLayer are important for LGI-GT, and the LGI scheme is better than the GNN+Transformer and parallel schemes. On 3 out of 4 datasets, LGI-GT surpasses both of them, especially on ogbg-moltox21, while on NCI1 the accuracy of LGI-GT is a little lower than that of Parallel GT.

5.3 Depth Study of GTs

To further validate the effectiveness of interleaving GConvs and TLayers, we performed experiments on PATTERN to observe the performance with respect to (w.r.t.) model depth d, i.e., how the performance changed while models got deeper. Typically, we compared our LGI scheme with the GNN+Transformer scheme and the parallel scheme.

Configurations were the same with LGI-GT in Section 5.1 except for number of layers and constructing schemes (i.e., GNN+Transformer, parallel and LGI). As for the parallel scheme, we followed the model structure of GPS, taking summation of GConv and TLayer outputs before FFN in each block.

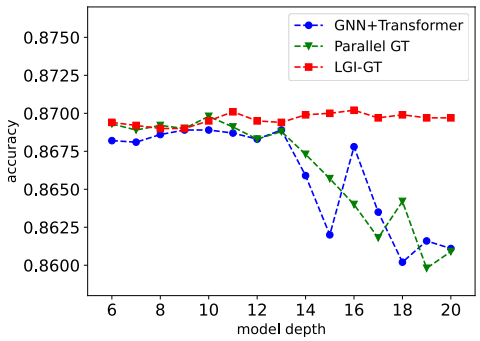

Figure 4: Test performance w.r.t. model depth.

Figure 4 shows the influence of the model depth with the GNN+Transformer scheme, parallel scheme and LGI scheme. It is easy to see that with the model becoming deeper, GNN+Transformer and Parallel GT are degraded and the performances tend to become worse overall, while LGI-GT keeps steady and the results stay in a high level. The reason why such difference between LGI-GT and the others exists is analyzed in Section 3. Hence, we can conclude that over-smoothing obviously limits GNN+Transformer, a simple summation feature fusion limits the parallel GT, but the LGI scheme is effective and robust to the model depth.

5.4 Model Interpretation

To better view the superiority of our LGI scheme over the other ones from the perspective of model interpretation, on ogbg-molpcba, we visualized the attention weights of the [CLS] node to all the real nodes in the same graph. Figure 5 shows two examples, where the first column displays the original molecules and the other columns from left to right are the visualization results of LGI-GT, GNN+Transformer and Parallel GT in turn. Apparently LGI-GT learns to put more attention on some important atoms like N, O and S, as well as atoms that connect different motifs, while the GNN+Transformer scheme and the parallel scheme fail to do both simultaneously. Hence, we can conclude that LGI-GT is good at handling structure information and focuses on the discriminative nodes.

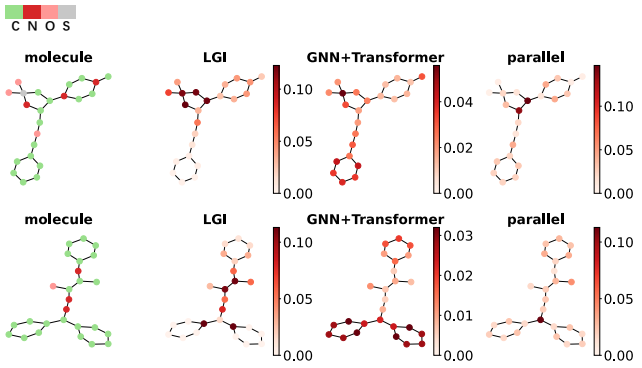

Figure 5: Visualization of the [CLS] node attention to the real nodes.

5.5 The Effectiveness of EELA

We tested our EELA module on ogbg-molpcba by comparing with two other attention-based message passing modules—TransformerConv [Shi *et al.*, 2021] and GATConv [Velickovic *et al.*, 2018].

Module	LGI-GT	GNN+Transformer	Parallel GT
	AP ↑	AP ↑	AP ↑
GATConv	0.2995 ± 0.0039	0.2523 ± 0.0059	0.2366 ± 0.0029
TransformerConv	0.3031 ± 0.0014	0.2402 ± 0.0062	0.2401 ± 0.0049
EELA	**0.3040 ± 0.0029**	**0.2681 ± 0.0056**	**0.2479 ± 0.0030**

Table 3: Ablation study to test the effectiveness of EELA for full-attention GTs in three constructing schemes on ogbg-molpcba.

Most configurations were the same with LGI-GT in Section 5.1 except for the GConv types and model structures. Particularly, following GPS [Rampasek *et al.*, 2022], we took an FFN after a simple summation to fuse outputs of GConvs and TLayers for the Parallel GT, and global average pooling was taken as its readout method. As shown in Table 3, apparently GTs with EELA outperform the counterparts with TransformerConv and GATConv in all three schemes, demonstrating that EELA is more suitable to construct full-attention GTs than other local attention-based message passing modules.

6 Conclusion

In this paper, we classify current graph Transformers into two types and analyze the advantages of our interleaving scheme over other GT constructing methods in terms of model structures of Type II GTs. More importantly, we propose LGI-GT, a graph Transformer interleaving message passing layers and Transformer encoder layers. Adapting to the LGI structure, we introduce a new algorithm to forward propagate embeddings of the [CLS] token in a skip manner. Additionally, we design a new graph local operator called EELA, which is used to construct full-attention GTs (including our LGI-GT) better than existing attention-based message passing modules. Empirically, we demonstrate that our LGI-GT outperforms other SOTA GNNs and GTs on both node-level and graph-level tasks, and through several ablation studies, we further validate the effectiveness of the LGI scheme and EELA.

Acknowledgments

This work was partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0100400, HY Project under Grant No. LZY2022033004, the Natural Science Foundation of Shandong Province under Grants No. ZR2020MF131 and No. ZR2021ZD19, Project of the Marine Science and Technology cooperative Innovation Center under Grant No. 22-05-CXZX-04-03-17, the Science and Technology Program of Qingdao under Grant No. 21-1-4-ny-19-nsh, and Project of Associative Training of Ocean University of China under Grant No. 202265007.

References

[Alon and Yahav, 2021] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *ICLR*, 2021.

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

[Beaini *et al.*, 2021] Dominique Beaini, Saro Passaro, Vincent Létourneau, William L. Hamilton, Gabriele Corso, and Pietro Lió. Directional graph networks. In *ICML*, pages 748–758. PMLR, 2021.

[Chen *et al.*, 2020] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, 2020.

[Chen *et al.*, 2022] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *ICML*, pages 3469–3489. PMLR, 2022.

[Choromanski *et al.*, 2021] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *ICLR*, 2021.

[Corso *et al.*, 2020] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *NeurIPS*, pages 13260–13271, 2020.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Dwivedi and Bresson, 2020] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *CoRR*, abs/2012.09699, 2020.

[Dwivedi *et al.*, 2020] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *CoRR*, abs/2003.00982, 2020.

[Elman, 1990] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272. PMLR, 2017.

[Hamilton *et al.*, 2017] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1024–1034, 2017.

[Hu *et al.*, 2020a] 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. *NeurIPS*, pages 22118–22133, 2020.

[Hu *et al.*, 2020b] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *ICLR*, 2020.

[Ivanov *et al.*, 2019] Sergei Ivanov, Sergei Sviridov, and Evgeny Burnaev. Understanding isomorphism bias in graph data sets. *CoRR*, abs/1910.12091, 2019.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[Klicpera *et al.*, 2019] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019.

[Kreuzer *et al.*, 2021] Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In *NeurIPS*, pages 21618–21629, 2021.

[LeCun *et al.*, 1989] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[Li *et al.*, 2018] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

[Morris *et al.*, 2019] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

[Oono and Suzuki, 2020] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *ICLR*, 2020.

[Park *et al.*, 2022] Wonpyo Park, Woong-Gi Chang, Donggeon Lee, Juntae Kim, and seung-won hwang. GRPE: Relative positional encoding for graph transformer. In *ICLR2022 Machine Learning for Drug Discovery*, 2022.

[Rampasek *et al.*, 2022] Ladislav Rampasek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *NeurIPS*, 2022.

[Rong *et al.*, 2020] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. In *NeurIPS*, 2020.

[Shi *et al.*, 2021] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In *IJCAI*, 2021.

[Tönshoff *et al.*, 2021] Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph learning with 1d convolutions on random walks. *CoRR*, abs/2102.08786, 2021.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

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

[Wale *et al.*, 2008] Nikil Wale, Ian A. Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.*, 14(3):347–375, 2008.

[Wijesinghe and Wang, 2022] Asiri Wijesinghe and Qing Wang. A new perspective on" how graph neural networks go beyond weisfeiler-lehman?". In *ICLR*, 2022.

[Wu *et al.*, 2021] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *NeurIPS*, pages 13266–13279, 2021.

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

[Ying *et al.*, 2021] 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 *NeurIPS*, pages 28877–28888, 2021.

[You *et al.*, 2021] Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *AAAI*, 2021.

[Zaheer *et al.*, 2020] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *NeurIPS*, pages 17283–17297, 2020.

[Zhang and Li, 2021] Muhan Zhang and Pan Li. Nested graph neural networks. In *NeurIPS*, pages 15734–15747, 2021.

[Zhao *et al.*, 2022] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *ICLR*, 2022.