GNFormer: Structral Transformer for Large Graphs

Shuyang Fang[1], Qian Xie [2], Jingfeng Li[2], Pengfei Lin[2], Runheng Lin[2]

[1]School of Informatics, Xiamen University
[2]Artificial Intelligence Research Institute, Xiamen University
23020241154390, 36920241153265, 36920241153228, 36920241153235, 36920241153236

Abstract

Graph Transformers adapt the concept of traditional transformers to the domain of graphs, significantly harnessing the advantages of the transformer's self-attention mechanism for effective node feature extraction and updating. However, Graph Transformers also face scalability challenges due to the quadratic complexity of the self-attention mechanism, making their application to large graphs problematic. Our research primarily focuses on representing large graphs, aiming to address the scalability issues of Graph Transformer models. First, We propose GNFormer, a Graph Transformer model that incorporates structural and positional encodings, using single-layer global attention to enhance processing efficiency and introducing a stochastic mini-batch training method to accommodate large-scale graph data. Second, we validate GNFormer's effectiveness on medium and large graph datasets, demonstrating its superiority in node classification tasks over baseline models. Additional comparative tests confirm the significance of its modules and overall efficiency in time and resource usage.

Introduction

Since the Transformer(Vaswani et al. 2017) was introduced, it has proven effective in processing sequential data, and its architecture has been widely adopted in the field of natural language processing.(Devlin 2018)Subsequently, researchers have continued to investigate the potential applications of the Transformer in various fields. In the field of computer vision, various adaptations of the Transformer have also demonstrated strong performance in processing non-sequential data, such as images. These successful implementations have spurred the advancement of Transformers in the graph domain, now recognized as Graph Transformers (GTs).

Graph neural networks, the classical models for processing graphs, excel at capturing local structural information. However, they typically struggle with long-range dependencies between nodes and are susceptible to phenomena such as over-smoothing(Rong et al. 2020) and over-squashing(Alon and Yahav 2021). Over-smoothing occurs when an increase in the number of neural network layers and iterations leads to node representations becoming overly consistent and non-discriminatory. Over-squashing occurs as long-distance message passing results in the compression of numerous node features into the limited vector space of a single node. These issues impair the efficient representation of node features and consequently restrict the application of graph neural networks to more intricate graph processing tasks.

In recent years, employing the Transformer architecture for graph data has emerged as a promising approach to address these challenges. The Graph Transformer effectively captures long-range dependencies among nodes via its self-attention mechanism and strategically places attention on various nodes, thereby mitigating the effects of over-smoothing and over-squashing to a certain extent. However, Graph Transformer has been mostly applied to small-scale graph data for graph classification tasks, such as molecular classification(Ying et al. 2021; Hussain, Zaki, and Subramanian 2021; Dwivedi and Bresson 2020). However, deploying Graph Transformers on large-scale graph data presents significant challenges.

On the one hand, a primary challenge is the development of reasonable methods for embedding graph data. Due to graph data's irregularity, encoding it effectively is crucial for its proper utilization by the Transformer architecture, which necessitates the comprehensive extraction of nodes' semantic information without sacrificing the intricate structural and positional details inherent in the graphs. On the other hand, addressing the substantial computational costs associated with the self-attention mechanism remains a significant issue. When processing large-scale graphs, the time and space complexity of existing Transformers increases quadratically.

Two primary strategies exist for utilizing graph structural information: the first involves employing various positional encoding methods to integrate this information into node representations. This approach transforms graph data into sequential data enriched with structural information, thereby facilitating processing by the Transformer architecture. Common positional encoding methods, such as Laplace positional encoding(Dwivedi and Bresson 2020) and random walk positional encoding(Dwivedi et al. 2021), have proven to be both simple and effective. Additionally, some positional encoding methods are trainable, such as the Katz position encoding implemented in the DGT(Park

et al. 2022). Another innovative approach involves combining GNN with the Transformer architecture, which merges node representations from both models in various ways, an example being SAT(Li et al. 2021). Typically, merging GNN with Transformer architectures proves more effective than merely employing positional encodings.

Regarding the quadratic complexity arising from the Transformer's self-attention mechanism, solutions can be broadly categorized into two types. One approach involves sampling the entirety of graph nodes to reduce the number of nodes processed by the model; this type of Graph Transformer is termed a sparse Graph Transformer, exemplified by Gophormer(Zhao et al. 2021) and NAGphormer(Chen et al. 2022). The other method entails the adoption of linear Graph Transformers, which simplify the computation of self-attention, as demonstrated by Nodeformer(Wu et al. 2023b) and Difformer(Wu et al. 2023a).

Our work proposes a model that can be extended to large graphs for node feature representation, and our main contributions can be summarized as follows:

- We develop GNFormer, a Graph Transformer model that integrates structural encoding, positional encoding, and single-layer attention.
- We demonstrate the effectiveness and feasibility of GNFormer through empirical evaluations on medium and large-scale graph datasets.

Related Work

Graph Neural Networks. GNNs(Kipf and Welling 2016) have advanced deep learning for graph-structured data by enabling efficient extraction of both node- and graph-level features. Graph Convolutional Networks (GCNs) aggregate information from neighboring nodes, but struggle to distinguish the relative importance of neighbors, limiting their ability to capture complex node relationships. To address this, Graph Attention Networks (GATs)(Veličković et al. 2018) assign variable weights to neighbors, improving flexibility in learning node dependencies. However, GATs increase computational overhead due to attention mechanisms. Jumping Knowledge Networks (JKNet)(Xu et al. 2018) further enhance GNNs by allowing flexible selection of information from multiple layers, adapting to different neighborhood sizes for central and peripheral nodes, improving representation based on local structures. Additionally, Structure-aware Interactive Graph Neural Network (SIGN)(Park et al. 2022) enhances molecular interaction modeling, using specialized layers to capture spatial and interaction-based features in protein-ligand binding affinity.

Positional Encoding. Positional encoding methods include Laplacian positional encoding(Dwivedi and Bresson 2020), Weisfeiler-Lehman encoding, and random walk encoding(Dwivedi et al. 2021). Laplacian encoding rely on the graph's Laplacian matrix, which reflects the topological structure of the graph. Through spectral decomposition of the Laplacian matrix, these methods extract eigenvectors, providing features that represent node positions. Such features are further used to encode positional information, enhancing node representations within the graph. Weisfeiler-Lehman encoding, derived from the Weisfeiler-Lehman algorithm, focus on detecting isomorphic structures by iteratively aggregating neighborhood information over multiple rounds, thereby classifying and labeling nodes. These labels are then utilized for positional encoding. Besides, some other positional encoding methods take distance into consideration(You, Ying, and Leskovec 2019), such as shortest path distances between nodes. This method captures the structural relationships between nodes from a global perspective. Additionally, there is also research on trainable positional encodings(Dwivedi et al. 2022).

Graph Transformers. Graph Transformers have shown significant promise for handling long-range dependencies in graph-structured data. Graphormer(Ying et al. 2021) adopts a Transformer-like architecture, integrating graph-specific features such as node centrality and spatial relationships to improve performance on large-scale graphs. Similarly, GraphTrans(Wu et al. 2022) merges GNNs with self-attention to learn global graph-level relationships, incorporating a novel readout mechanism. NodeFormer(Liu et al. 2023) addresses the challenge of scaling Transformer models to large graphs by implementing efficient message passing and random feature mapping, reducing the computational complexity inherent in attention-based models.

Given the challenges of scaling graph models to large datasets, GNFormer combines the scalability of Transformers with the structural advantages of GNNs. By incorporating structural encodings and attention mechanisms, GNFormer efficiently captures both long-range dependencies and local patterns in large graphs, making it particularly suitable for graph-level tasks. It also reduces the computational overhead commonly associated with attention mechanisms, offering a more scalable solution for processing complex graph structures.

Proposed Solution

Overview.

We presents a Graph Transformer model named GNFormer, which is scalable to large-scale graph scenarios and closely integrated with Graph Neural Networks. The detailed architecture is shown in Figure 1. GNFormer consists of three core modules: the structural encoding module, the positional encoding module, and the single-layer global attention module. The computational process of the model can be represented by the following equation 1 and 2:

$$\mathbf{Z}^{(0)} = (1 - w_s - w_p)\,\mathbf{X}^{(0)} + w_s\mathbf{X}^{(s)} + w_p\mathbf{X}^{(P)} \quad (1)$$

$$\mathbf{Z} = \mathrm{SGA}(\mathbf{Z}^{(0)}) \quad (2)$$

Where $\mathbf{X}^{(s)}$ and $\mathbf{X}^{(P)}$ represent the outputs of the structural encoding module and positional encoding module, respectively, w_s and w_p are the corresponding weight parameters, and $\mathrm{SGA}(\cdot)$ denotes the single-layer global attention module. $\mathbf{X}^{(0)}$, $\mathbf{X}^{(s)}$, and $\mathbf{X}^{(P)}$ are fused in a certain proportion to obtain the node features with added structural and positional biases, serving as the input to the single-layer global attention module, with the final output denoted as $\mathbf{Z}$. The model performs node classification tasks, outputting a sequence of length equal to the number of node categories,

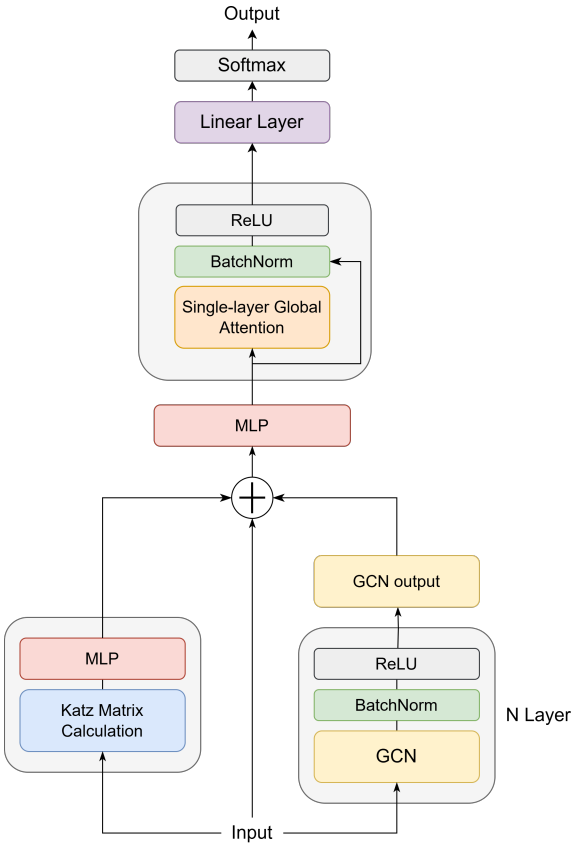

Figure 1: GNFormer Model Architecture.

indicating the likelihood of nodes being predicted as various categories.

Structural Encoding.

The structural encoding module is based on the GCN model and stacks N graph convolutional layers according to the task requirements. This module generates a vector identical in dimension to the node features, containing structural bias information of the graph data.The input of the structural encoding module consists of the raw features of the nodes $X^{(0)}$ and the adjacency matrix A. The output dimensionality of the structural encoding $X^{(s)}$ is consistent with the raw features of the nodes. This moduls is defined as follows:

$$X^{(s)} = \mathrm{SE}\left(X^{(0)}, A\right) \tag{3}$$

Positional Encoding.

The position encoding module is designed to extract positional bias information from graph data, which is used to learn the positional relationships between nodes.

This module employs Katz positional encoding, comprising a Katz matrix computation layer and an MLP. The construction of Katz positional encoding is inspired by the Katz index, which is calculated based on the collection of all possible paths. Specifically, it involves the direct summation of the path sets, with an exponential decay factor applied according to the length of the paths to assign greater weight to shorter paths. This can be represented as follows:

$$\widetilde{\boldsymbol{A}} = \sum_{k=1}^{\infty} \beta^{k-1} \boldsymbol{A}^k \tag{4}$$

where β is the decay coefficient, and A is the adjacency matrix of the graph. The i-th row of the adjacency matrix represents the connectivity of the i-th node with all other nodes. After weighted calculation, the resulting i-th row vector corresponds to the i-th node. By transposing the row vector corresponding to a node and using it as the input for a Multilayer Perceptron (MLP), we obtain the Katz positional encoding:

$$\mathrm{KatzPE}(v_i) = \mathrm{MLP}\left(\widetilde{\mathbf{A}}[v_i]^{\mathrm{T}}\right) \tag{5}$$

Single-layer Global Attention.

The calculation of global attention is based on the same principle as self attention in Transformer, but without Softmax, it is a simpler yet more effective way of calculating attention. Removing this step can greatly simplify the calculation of attention. The computation of single-layer global attention can be described as follows:

$$\boldsymbol{Q} = f_Q(\boldsymbol{Z}^{(0)}), ...\widetilde{\boldsymbol{Q}} = \frac{Q}{||\boldsymbol{Q}||_{\mathcal{F}}} \tag{6}$$

$$K = f_K(\mathbf{Z}^{(0)}), \cdots \widetilde{K} = \frac{K}{||K||_F} \tag{7}$$

$$V = f_V(\mathbf{Z}^{(0)}) \tag{8}$$

$$\boldsymbol{D} = diag\left(\mathbf{1} + \frac{1}{N}\widetilde{\boldsymbol{Q}}(\widetilde{\boldsymbol{K}}^{\top}\mathbf{1})\right) \tag{9}$$

$$\mathbf{Z} = \alpha \boldsymbol{D}^{-1}\left[\boldsymbol{V} + \frac{1}{N}\widetilde{\boldsymbol{Q}}(\widetilde{\boldsymbol{K}}^{\top}\boldsymbol{V})\right] + (1-\alpha)\boldsymbol{Z}^{(0)} \tag{10}$$

Here, $Z^{(0)}$ represents the node feature matrix that has been augmented with structural and positional biases, with dimensions [N, D], where N is the number of nodes and D is the dimensionality of the node features. The terms f_q, f_k and f_v denote linear layers that are used to construct the corresponding matrices for Q, K, and V. The notation $||\cdot||_{\mathcal{F}}$ indicates the Frobenius norm, which is employed to normalize the matrices. The symbol $\mathbf{1}$ signifies an N-dimensional vector of ones. The matrix D serves as a diagonal matrix for regularization purposes, ensuring that there are no zeros or very small values on its diagonal. The matrix Z is composed of two parts: the first part enables the model to capture the influence of other nodes, while the second part retains the information of the central node. These two components are combined through a hyperparameter, which effectively forms a residual connection.

Experiments

To validate the effectiveness of our method, we conducted extensive experiments using datasets of various sizes.

Experiment Setup

Datasets. First, we utilize three medium-scale graph datasets and one large-scale dataset, namely Cora(Sen et al. 2008), Squirrel, Chameleon(Rozemberczki, Allen, and Sarkar 2021), and ogbn-arxiv(Hu et al. 2020). For each dataset, we standardize the graph data format during the preprocessing stage, encompassing edge indices, edge features, node features, the total number of nodes, as well as indices for the training, validation, and test sets. The specifics of the introduction and preprocessing for each dataset are detailed in Table 1.

The Squirrel and Chameleon datasets are heterogeneous web-page networks from Wikipedia, with nodes as pages and edges as hyperlinks, classified by traffic. The ogbn-arxiv dataset is a CS Arxiv citation network, with nodes representing papers and edges representing citations. It aims to predict paper subject areas across 40 domains, using a refined partitioning method to resolve node overlap issues.

Dataset	Node Count	Edge Count	Feature Dimension
Cora	2,708	5,278	1,433
Squirrel	2,223	46,998	2,089
Chameleon	890	8,854	2,325
ogbn-arxiv	169,343	1,166,243	128

Table 1: Detailed statistics of the datasets.

Baselines. To demonstrate the performance of our proposed model on the node classification task, we compare GNFormer with SOTA methods. We select the following models as the baseline for comparison based on different scale of graphs: (1) On medium-sized graphs, we utilize graph neural network-based models such as GCN, GAT, SGC, JKNet, and SIGN, as well as graph transformer-based models Graphormer, GraphTrans, and Nodeformer as benchmarks. We use three medium-sized graph datasets, Cora, Squirrel, and Chameleon, for training, prediction, and evaluation. (2) On large graphs, we use MLP and the graph transformer-based model Nodeformer that can be extended to large graphs as benchmarks. We use the large-scale graph dataset ogbn-arxiv for training, prediction, and evaluation.

Implementation Details. We built the model based on Python and the deep learning framework PyTorch, and performed model training and experimental testing on a GPU server. The experimental equipment configuration information used in this experiment is shown in Table 1.

Table 2: Experimental Equipment Configuration

Computational Framework	Specifications
Python	3.7.16
PyTorch	1.12.1
CUDA	10.1
CPU	Intel Gold 5220R
GPU	A100
Operating System	CentOS 7.9

Experiment Result

The results of our proposed method, as well as the other baseline models, are presented in Table 1 for medium-scale and large-scale graph datasets. Our model outperforms all other methods across all metrics. Based on these experimental findings, we draw the following conclusions:

1) **Medium-scale graph datasets**: The performance of various models on the Cora dataset is generally good, among which the GNFormer model performs best with an average accuracy of 83.70%, which is about 2.1% higher than the standard GCN model. On the Squirrel dataset, the performance of all models generally declines, but GNFormer still maintains the lead, reaching an average accuracy of 44.84%. The results on the Chameleon dataset further verify the strong performance of the GNFormer model, with an average accuracy of 46.56%, ahead of all other models. This may indicate that the GNFormer model has obvious advantages in processing medium-scale graph data, especially when the data has a complex topological structure or feature distribution. It should be noted that the two variants of Graphormer, GraphormerSMALLER and GraphormerULTRASS_SMALL, show poor results on these three datasets, while GraphormerSMALL exhibits memory overflow (OOM) problems, indicating that the model may be overfitting due to its relatively complex structure and is not efficient in memory usage, which may become a problem when processing larger-scale graph data. Overall, these results provide concrete evidence to validate GNFormer as a powerful learner for node category prediction. Compared with other advanced GNNs and graph transformers, GNFormer is highly competitive in most cases.

2) **Large-scale graph datasets**: Compared with traditional MLP models and Nodeformer, GNFormer shows significant performance improvements. Specifically, the accuracy of the MLP model is 55.5%, while the accuracy of Nodeformer is 59.90%, and GNFormer reaches 66.07%, far exceeding the former two. This result highlights the advantage of GNFormer in processing large-scale graph data. GNFormer combines structural encoding, position encoding modules, and simple single-layer global attention. Experiments have shown that such an architecture can handle the dependencies of distant nodes on large graphs well without huge memory and time overhead.

Ablation Study

Effect of Components To validate the design choices in our proposed framework, we perform an ablation experiment by removing three components individually: Structure encoding module (GNFormer w/o SE), position encoding module (GNFormer w/o PE), and single-layer global attention module (GNFormer w/o SGA). The results are presented in Table 1. We observe that the GNFormer model achieves the best results overall. On the Cora dataset, the performance of the GNFormer w/o PE and GNFormer w/o SGA variants is slightly worse than the default GNFormer model, while the GNFormer w/o SE variant, which removes the structure encoding module, has the largest performance degradation, by about 12%. On the Chameleon dataset, the performance of

Table 3: Performance on Medium-Scale Graph Datasets

Model	Cora	Squirrel	Chameleon
GCN	81.6 ± 0.4	38.6 ± 1.8	41.3 ± 3.0
GAT	83.0 ± 0.7	35.6 ± 2.1	39.2 ± 3.1
SGC	80.1 ± 0.2	39.3 ± 2.3	39.0 ± 3.3
JKNet	81.8 ± 0.5	39.4 ± 1.6	39.4 ± 3.8
SIGN	82.1 ± 0.3	40.7 ± 2.5	41.7 ± 2.2
GraphormerSMALLER	75.8 ± 1.1	40.9 ± 2.5	41.9 ± 2.8
GraphormerULTRASS_SMALL	74.2 ± 0.9	39.9 ± 2.4	41.3 ± 2.8
GraphTransSMALL	80.7 ± 0.9	41.0 ± 2.8	42.8 ± 3.3
GraphTransULTRASS_SMALL	81.7 ± 0.6	40.6 ± 2.4	42.2 ± 2.9
Nodeformer	82.2 ± 0.9	38.5 ± 1.5	34.7 ± 4.1
GNFormer	83.70 ± 0.48	44.84 ± 2.13	46.56 ± 3.15

Table 4: Performance on Large-Scale Graph Dataset

Model	ogbn-arxiv
MLP	55.50 ± 0.23
Nodeformer	59.90 ± 0.42
GNFormer	66.07 ± 0.11

GNFormer w/o SE and GNFormer w/o PE is significantly worse than that of GNFormer. On the ogbn-arxiv dataset, there is no significant difference between the performance of GNFormer w/o PE and GNFormer, but the performance of GNFormer w/o SE and GNFormer w/o SGA is significantly reduced.

Table 5: Results of Ablation Experiments

Model	Cora	Chameleon	ogbn-arxiv
default	83.70 ± 0.48	46.56 ± 3.15	66.07 ± 0.11
w/o SE	71.26 ± 2.66	44.72 ± 2.83	57.18 ± 0.19
w/o PE	83.18 ± 0.42	43.51 ± 4.83	66.19 ± 0.11
w/o SGA	83.12 ± 0.46	46.72 ± 2.34	65.70 ± 0.30

Structural and Position Encoding Analysis The structural encoding module in GNFormer significantly impacts model performance; its removal substantially degrades performance. This module leverages graph neural networks to extract structural bias from graph data, as validated by ablation studies. Conversely, the position encoding module, which employs the Katz matrix to derive positional information from the adjacency matrix, offers minimal performance gains. This could be attributed to the redundancy with the structural encoding module, which also relies on the adjacency matrix but integrates node features for a more comprehensive extraction of graph data's latent information. The position encoding module's sole reliance on adjacency matrix-derived positional information is insufficient for robust graph embedding, underscoring the necessity to incorporate node feature embeddings.

Efficiency Study

We conducted a comparative analysis of the efficiency of GNFormer against other graph Transformer models by evaluating their training time, inference time, and GPU memory consumption on the Cora dataset.

As presented in Table 1, GNFormer outperforms Graphormer and GraphTrans in all three metrics, indicating its superior efficiency. This efficiency is achieved without compromising the model's performance, thereby highlighting the lightweight design of the GNFormer architecture. The inclusion of the Katz position coding module and the single-layer global attention mechanism significantly contributes to this efficiency. The Katz position coding leverages GPU acceleration for position encoding, while the single-layer global attention mechanism streamlines the model by reducing the complexity of attention operations across layers and heads. These features exemplify GNFormer as a simple, effective, and scalable graph Transformer model, making it suitable for deployment in large-scale graph scenarios.

Table 6: Comparison of the efficiency on Cora

Model	Training time (ms)	Inference time (ms)	GPU usage (GB)
Graphormer	563.5	537.1	5.0
GraphTrans	160.4	40.2	3.8
GNFormer	76.7	65.8	1.0

Conclusion

We introduce GNFormer, a novel architecture that integrates graph transformers (GTs) with graph pooling to achieve efficient node classification. Our approach tackles two prevalent issues associated with traditional GTs: the interference from noisy distant neighbors and the quadratic growth in computational complexity relative to the number of nodes. Through rigorous testing on 13 diverse graph datasets, we demonstrate that GNFormer not only surpasses current GTs but also outperforms other graph neural networks. Despite these impressive results, there is still scope for enhancement in GNFormer. For instance, future work could focus on: 1) devising an effective strategy to synergize the proposed local pooling augmented attention with global pooling augmented attention, and 2) integrating additional techniques to bolster performance on large-scale graph datasets.

References

Alon, U.; and Yahav, E. 2021. On the Bottleneck of Graph Neural Networks and its Practical Implications. In *International Conference on Learning Representations*.

Chen, J.; Gao, K.; Li, G.; and He, K. 2022. NAGphormer: Neighborhood Aggregation Graph Transformer for Node Classification in Large Graphs. *CoRR*, abs/2206.04910.

Devlin, J. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dwivedi, V. P.; and Bresson, X. 2020. A Generalization of Transformer Networks to Graphs.

Dwivedi, V. P.; Joshi, C. K.; Luu, A. T.; Laurent, T.; Bengio, Y.; and Bresson, X. 2022. Benchmarking Graph Neural Networks. arXiv:2003.00982.

Dwivedi, V. P.; Luu, A. T.; Laurent, T.; Bengio, Y.; and Bresson, X. 2021. Graph Neural Networks with Learnable Structural and Positional Representations.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33: 22118–22133.

Hussain, M. S.; Zaki, M. J.; and Subramanian, D. 2021. Edge-augmented Graph Transformers: Global Self-attention is Enough for Graphs.

Kipf, T. N.; and Welling, M. 2016. Semi-Supervised Classification with Graph Convolutional Networks.

Li, S.; Zhou, J.; Xu, T.; Huang, L.; Wang, F.; Xiong, H.; Huang, W.; Dou, D.; and Xiong, H. 2021. Structure-aware Interactive Graph Neural Networks for the Prediction of Protein-Ligand Binding Affinity.

Liu, C.; Zhan, Y.; Ma, X.; Ding, L.; Tao, D.; Wu, J.; and Hu, W. 2023. Gapformer: Graph Transformer with Graph Pooling for Node Classification. 2196–2205.

Park, J.; Yun, S.; Park, H. J.; Kang, J.; Jeong, J.; Kim, K. H.; Ha, J. W.; and Kim, H. J. 2022. Deformable Graph Transformer. *ArXiv*, abs/2206.14337.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*.

Rozemberczki, B.; Allen, C.; and Sarkar, R. 2021. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2): cnab014.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine*, 29(3): 93–93.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. *arXiv*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. arXiv:1710.10903.

Wu, Q.; Yang, C.; Zhao, W.; He, Y.; Wipf, D.; and Yan, J. 2023a. DIFFormer: Scalable (Graph) Transformers Induced by Energy Constrained Diffusion. *arXiv e-prints*.

Wu, Q.; Zhao, W.; Li, Z.; Wipf, D.; and Yan, J. 2023b. NodeFormer: A Scalable Graph Structure Learning Transformer for Node Classification. arXiv:2306.08385.

Wu, Z.; Jain, P.; Wright, M. A.; Mirhoseini, A.; Gonzalez, J. E.; and Stoica, I. 2022. Representing Long-Range Context for Graph Neural Networks with Global Attention.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K. I.; and Jegelka, S. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. *arXiv*.

Ying, C.; Cai, T.; Luo, S.; Zheng, S.; and Liu, T. Y. 2021. Do Transformers Really Perform Bad for Graph Representation?

You, J.; Ying, R.; and Leskovec, J. 2019. Position-aware Graph Neural Networks. arXiv:1906.04817.

Zhao, J.; Li, C.; Wen, Q.; Wang, Y.; Liu, Y.; Sun, H.; Xie, X.; and Ye, Y. 2021. Gophormer: Ego-Graph Transformer for Node Classification.