Gapformer: Graph Transformer with Graph Pooling for Node Classification

Chuang Liu[1*], **Yibing Zhan**[2], **Xueqi Ma**[3], **Liang Ding**[2],
Dapeng Tao[4,5], **Jia Wu**[6] and **Wenbin Hu**[1†]
[1]School of Computer Science, Wuhan University, Wuhan, China
[2]JD Explore Academy, JD.com, China
[3]School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia
[4]School of Computer Science, Yunnan University, Kunming, China
[5]Yunnan Key Laboratory of Media Convergence, Kunming, China
[6]School of Computing, Macquarie University, Sydney, Australia
{chuangliu, hwb}@whu.edu.cn, zhanyibing@jd.com, xueqim@student.unimelb.edu.au,
liangding.liam@gmail.com, dptao@ynu.edu.cn, jia.wu@mq.edu.au

Abstract

Graph Transformers (GTs) have proved their advantage in graph-level tasks. However, existing GTs still perform unsatisfactorily on the node classification task due to 1) the overwhelming unrelated information obtained from a vast number of irrelevant distant nodes and 2) the quadratic complexity regarding the number of nodes via the fully connected attention mechanism. In this paper, we present Gapformer, a method for node classification that deeply incorporates Graph Transformer with Graph Pooling. More specifically, Gapformer coarsens the large-scale nodes of a graph into a smaller number of pooling nodes via local or global graph pooling methods, and then computes the attention solely with the pooling nodes rather than all other nodes. In such a manner, the negative influence of the overwhelming unrelated nodes is mitigated while maintaining the long-range information, and the quadratic complexity is reduced to linear complexity with respect to the fixed number of pooling nodes. Extensive experiments on 13 node classification datasets, including homophilic and heterophilic graph datasets, demonstrate the competitive performance of Gapformer over existing Graph Neural Networks and GTs.

1 Introduction

Graph Neural Networks (GNNs), which are based on local message-passing [Kipf and Welling, 2017], have achieved notable success in a variety of applications, including biology [Xu *et al.*, 2019b] and recommendation [Zhang *et al.*, 2019]. In contrast to GNNs, Graph Transformers (GTs) allow each node in a graph to directly attend to all other nodes, which enables the aggregation of information from arbitrary

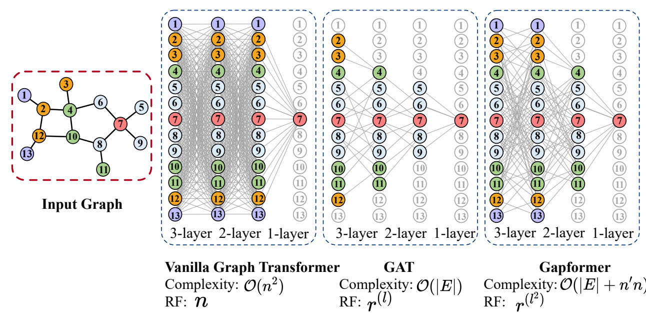

Figure 1: Complexity and receptive fields (RF) in Vanilla Graph Transformer (GT), GAT, and our Gapformer. Here, n denotes the number of nodes in the original graph, $|E|$ denotes the number of edges, and n' denotes the number of pooling nodes, which is constant and significantly smaller than n. $r^{(l)}$ denotes the number of the l-hop neighboring nodes, where l is the number of layers. The vanilla GT has the maximum receptive field, which comes with a quadratic complexity. In comparison, our Gapformer is computationally efficient, meanwhile maintaining a large receptive field size.

nodes. Along with normalization and residual connection, GTs overcome the deficiencies of GNNs in dealing with oversmoothing [Rong *et al.*, 2020b], over-squashing [Alon and Yahav, 2021], heterophily [Zhu *et al.*, 2020], and long-range dependencies [Zhang *et al.*, 2022].

However, existing GTs [Dwivedi and Bresson, 2021; Kreuzer *et al.*, 2021; Ying *et al.*, 2021] are exploited primarily for graph-level tasks (*e.g.*, graph classification) with a small number of nodes in a graph. Developing GTs for node classification, where the number of nodes in a graph is relatively large (up to around one million), remains a challenging proposition for the following two reasons. First, the quadratic computational complexity $\mathcal{O}(n^2)$ of self-attention in vanilla GTs, in regards to the number of nodes, inhibits their application to node classification in real-world scenarios. Second, vanilla GTs calculate the full connected attention and aggregate messages from arbitrary nodes, including numerous irrelevant nodes; this results in ambiguous atten-

*This work was done when Chuang Liu worked as an intern at JD Explore Academy.
†Corresponding Author

tion weights and the aggregation of noise information from incorrectly correlated nodes.

Only a few existing works have attempted to consider GTs for node classification. GT-sparse [Dwivedi and Bresson, 2021] and SAN [Kreuzer *et al.*, 2021] confine the receptive field of each node to its 1-hop neighboring nodes; as a result, expressiveness is sacrificed when important interactions are multiple hops away, especially in the large-scale graph correspondingly requiring a large receptive field. LiteGT [Chen *et al.*, 2021a], DGT [Park *et al.*, 2022], and DET [Guo *et al.*, 2022] propose selecting important nodes for attention using certain specific, fixed node sampling strategies, which may result in the selection of uninformative nodes. GraphGPS [Rampasek *et al.*, 2022] and TokenGT [Kim *et al.*, 2022] directly adopt efficient approximations [Choromanski *et al.*, 2021; Beltagy *et al.*, 2020] in Transformers [Vaswani *et al.*, 2017] to improve the efficiency of vanilla GTs; nevertheless, they neglect unique characteristics of graph data and tend to yield dense attention, causing an enormous amount of noise messages to be aggregated from irrelevant nodes.

In light of the above analysis, we propose Gapformer, which combines Graph Transformer with Graph Pooling, to capture long-range dependencies and improve the efficiency of vanilla GTs. In vanilla GTs, self-attention converts nodes into queries and keys/values, after which each query attends to all the keys. Specifically, self-attention involves computing the inner product between the *query* and *key* vectors to generate attention scores. These scores are then used to perform a weighted aggregation of *value* vectors. To reduce the complexity of the dense inner product, Gapformer first utilizes graph pooling to group key and value nodes into a smaller number of pooling nodes. For graph pooling, we propose two types of strategies to compress the original graph efficiently and effectively: 1) global graph pooling, which condenses the entire graph into several significant pooling nodes; 2) local graph pooling, which compresses the k-hop neighboring nodes of each query node into the corresponding pooling nodes. Subsequently, each query node interacts with pooling keys (fewer in number) and generates representation with the pooling values. In conclusion, our Gapformer transforms the fully connected attention in vanilla GTs into a sparse attention schema by decreasing the number of attended tokens via graph pooling.

As shown in Figure 1, Gapformer has the following advantages. 1) Gapformer enables a larger attention field per node and thus allows to compute multi-hop correlations via graph pooling between each node and its corresponding disconnected nodes. 2) Since the number of pooling nodes is significantly smaller than that of nodes in the original graph, the computational complexity of Gapformer only increases linearly with the number of nodes in a graph, hence making Gapformer suitable for processing large-scale datasets in the node classification task.

Our main contributions are summarized as follows:

1. We propose Gapformer, a deeper combination of Transformer and Graph Neural Networks. Specifically, Gapformer utilizes Graph Pooling to group the attended nodes of each node into pooling nodes (fewer in number) and computes its attention using only the pooling nodes. This design mitigates the overwhelming unrelated information and quadratic complexity issues associated with GTs while preserving long-range interactions.
2. We conduct extensive experiments to compare Gapformer with 20 GNN and GT baseline models in the node classification task on 13 real-world graph datasets, including homophilic and heterphilic datasets. Experimental results consistently validate the effectiveness and efficiency of our proposed Gapformer.

2 Related Work

Graph Pooling. Graph Neural Networks (GNNs) [Kipf and Welling, 2017; Hamilton *et al.*, 2017] are networks that perform on graph domain. As an essential component of GNNs, Graph Pooling condenses the input graph with node representations into a smaller graph or a holistic graph-level representation. There are two main types of designs proposed for graph pooling: flat and hierarchical. Flat pooling directly generates a graph-level representation in one step, mostly by taking the average or sum over all node embeddings as the graph representation [Duvenaud *et al.*, 2015]. On the other hand, hierarchical pooling gradually coarsens a graph into a smaller one using either node clustering pooling [Ying *et al.*, 2018; Bianchi *et al.*, 2020] or node drop pooling [Gao and Ji, 2019; Lee *et al.*, 2019; Liu *et al.*, 2023]. Node clustering requires significant computational resources to cluster nodes into clusters, while node drop selects a subset of nodes from the original graph to construct a coarsened version that is more efficient and suitable for large-scale graphs [Lee *et al.*, 2019]. For further details, please refer to [Liu *et al.*, 2022b].

Graph Transformers. In recent years, many Transformer variants have been successfully applied to graph modeling, displaying competitive or even superior performance on many tasks when compared to GNNs. Dwivedi *et al.* [2021] were the first to extend the transformer architecture to graphs and propose position encoding [Ding *et al.*, 2020] for nodes in a graph. Subsequently, Kreuzer *et al.* [2021] enabled the position encoding by making it learnable, and further divided the fully connected edges into true edges and virtual edges. There are many other existing GTs [Rong *et al.*, 2020a; Chen *et al.*, 2021b; Wu *et al.*, 2021; Hussain *et al.*, 2022; Chen *et al.*, 2022; Nguyen *et al.*, 2022] and the applications of GTs [Xu *et al.*, 2019a; Zhu *et al.*, 2021; Zhu *et al.*, 2022; Cai *et al.*, 2022], for a more detailed introduction, please refer to the recent reviews of GTs [Rampasek *et al.*, 2022; Min *et al.*, 2022]. However, the above methods are mostly designed for graph-level tasks due to the time and memory constraints imposed by the self-attention layer, which requires $\mathcal{O}(n^2)$ complexity. Therefore, several works [Zhao *et al.*, 2021; Choromanski *et al.*, 2022; Guo *et al.*, 2022; Park *et al.*, 2022; Wu *et al.*, 2022] have been proposed to make graph transformers more scalable and efficient, but they still suffer from some issues such as long-range information loss or noise aggregation.

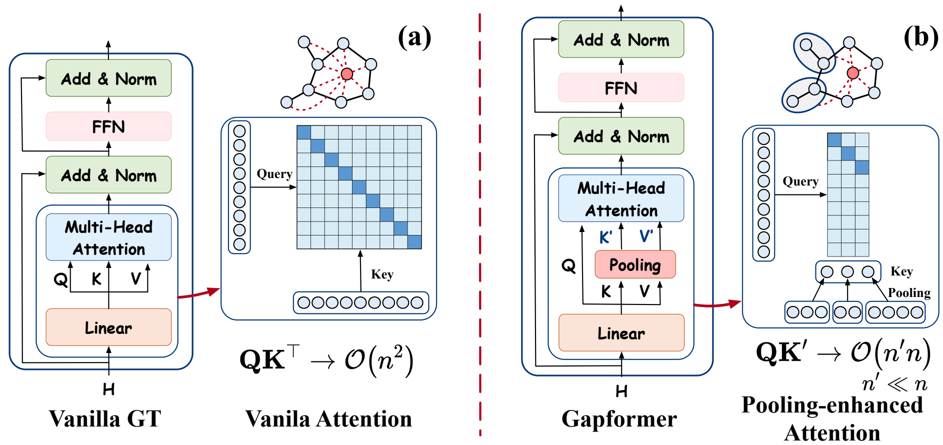

Figure 2: Comparison of Vanilla Graph Transformer (GT) and our Gapformer. (a) One core of GT models is the self-attention layer, which computes the pairwise inner product between the input node tokens $\mathbf{Q}$ and $\mathbf{K}$. (b) Before calculating self-attention, Gapformer utilizes the graph pooling operation to coarsen the key ($\mathbf{K}$) and value ($\mathbf{V}$) vectors into the pooling key ($\mathbf{K}^{'}$) and value ($\mathbf{V}^{'}$) vectors, respectively, which decreases the number of attended tokens ($n \to n'$).

3 Methodology

3.1 Preliminaries

Notations. A graph $\mathcal{G}$ can be represented by an adjacency matrix $\mathbf{A} \in \{0, 1\}^{n\times n}$ and a node feature matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, where n is the number of nodes, d is the dimension of the node features, and $\mathbf{A}[i, j] = 1$ if there exits an edge between node v_i and node v_j, otherwise, $\mathbf{A}[i, j] = 0$.

Graph Pooling. Let a graph pooling operator be defined as any function Pooling that maps a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to a new pooled graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$:

$$\mathcal{G}' = \text{Pooling}(\mathcal{G}), \tag{1}$$

where $|\mathcal{V}'| < |\mathcal{V}|$. The main objective of graph pooling is to decrease the number of nodes in a graph while maintaining its semantic information.

Transformer. The vanilla Transformer [Vaswani *et al.*, 2017] consists of two essential parts: a multi-head self-attention (MHA) module and a position-wise feed-forward network (FFN). To build a deeper model, a residual connection [He *et al.*, 2016] is employed to each module, followed by a layer normalization (LN) [Ba *et al.*, 2016]. The self-attention mechanism calculates attention scores by taking the inner product of query vectors ($\mathbf{Q}$) and key vectors ($\mathbf{K}$). It then uses these scores to aggregate value vectors ($\mathbf{V}$) in a weighted manner, resulting in contextualized representations, that is,

$$\mathbf{Q} = \mathbf{H}\mathbf{W}^Q, \mathbf{K} = \mathbf{H}\mathbf{W}^K, \mathbf{V} = \mathbf{H}\mathbf{W}^V; \tag{2}$$

$$\mathbf{H}' = \text{softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{d'}}\right)\mathbf{V}, \tag{3}$$

where $\mathbf{W}^Q \in \mathbb{R}^{d\times d'}, \mathbf{W}^K \in \mathbb{R}^{d\times d'}$, and $\mathbf{W}^V \in \mathbb{R}^{d\times d'}$ are projection matrices, $\mathbf{H} = \left[\boldsymbol{h}_1^\top, \ldots, \boldsymbol{h}_n^\top\right] \in \mathbb{R}^{n\times d}$ denotes the input matrix of node embeddings, $\mathbf{H}' \in \mathbb{R}^{n\times d'}$ is the output matrix, and d' is the output hidden dimension. Note that, Equation (3) denotes the single-head self-attention module, which can straightforwardly generalize to MHA.

3.2 Proposed Method: Gapformer

In this section, we present the model architecture of Gapformer. First, we introduce the base architecture of Gapformer and its core module; that is, the attention enhanced with graph pooling (AGP). We then provide a comprehensive description of AGP from both global and local perspectives.

Architecture

As shown in Figure 2 (a), self-attention in vanilla GT calculates the dot product between each pair of nodes after projection ($\mathbf{QK}^\top$). Therefore, the computation of full self-attention comes with potential noises from long-distance neighbors and $\mathcal{O}(n^2)$ complexity, limiting its capacity to analyze large-scale graphs. Graph pooling manages to reduce the number of graph nodes while maintaining semantic information. This encourages our employment of graph pooling to overcome the deficiencies of GTs. To our knowledge, no efforts have been made to integrate GTs and graph pooling.

Attention Enhanced with Graph Pooling. In light of the above analysis, Gapformer uses a sparse attention schema based on graph pooling to replace the full self-attention mechanism. Specifically, as shown in Figure 2 (b), Gapformer first produces query, key, and value matrices (Linear Module); that is,

$$\widetilde{\mathbf{Q}} = \mathbf{H}\mathbf{W}^{\widetilde{Q}}, \widetilde{\mathbf{K}} = \mathbf{H}\mathbf{W}^{\widetilde{K}}, \widetilde{\mathbf{V}} = \mathbf{H}\mathbf{W}^{\widetilde{V}}. \tag{4}$$

We define the query vector of node v_i as $\widetilde{\boldsymbol{q}}_i$, while its corresponding key and value matrices in the vanilla self-attention are $\widetilde{\mathbf{K}} \in \mathbb{R}^{n\times d'}$ and $\widetilde{\mathbf{V}} \in \mathbb{R}^{n\times d'}$, respectively. Subsequently, we apply graph pooling to compress $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{V}}$, which is defined as follows:

$$\overline{\mathbf{K}}_{\mathcal{S}(i)} = \text{Pooling}\left(\tilde{\mathbf{K}}\right); \tag{5}$$

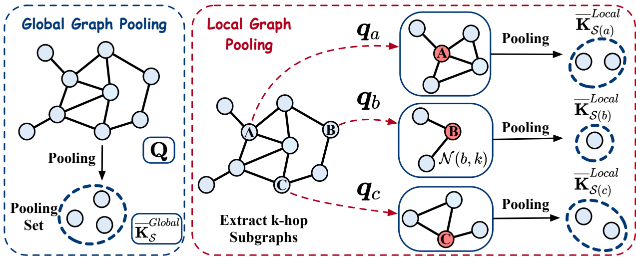

Figure 3: Two types of graph pooling methods to enhance attention in Gapformer. Here, $\mathbf{q}_a$ is the query vector of node v_a, and $\mathcal{N}(a,k)$ refers to the neighbor set within k hops of node v_a. *Left*: each node in the original graph attends with all nodes in the pooling set ($\mathcal{S}$). *Right*: each node attends with nodes in its own pooling set, which is generated from its k-hop neighbors.

$$\overline{\mathbf{V}}_{\mathcal{S}(i)} = \text{Pooling}\left(\tilde{\mathbf{V}}\right), \tag{6}$$

where $\overline{\mathbf{K}}_{\mathcal{S}(i)} \in \mathbb{R}^{n' \times d'}$ and $\overline{\mathbf{V}}_{\mathcal{S}(i)} \in \mathbb{R}^{n' \times d'}$ are the compressed key and value matrices of node v_i respectively, and the size of pooled node sets $\mathcal{S}(i)$ for node v_i is n', significantly smaller than n. Pooling$(\cdot)$ refers to graph pooling, which is discussed in detail in the next section. The attention enhanced with graph pooling is then calculated as:

$$\boldsymbol{h}_i = \text{softmax}\left(\alpha \tilde{\boldsymbol{q}}_i^T \overline{\mathbf{K}}_{\mathcal{S}(i)}\right) \overline{\mathbf{V}}_{\mathcal{S}(i)}^T, \tag{7}$$

where α is a constant scalar ($\alpha = \frac{1}{\sqrt{d'}}$).

Following [Guo *et al.*, 2022; Zhao *et al.*, 2021], we also maintain the message-passing with the neighboring nodes. The process is defined as follows:

$$\boldsymbol{z}_i = \text{softmax}\left(\alpha \tilde{\boldsymbol{q}}_i^T \tilde{\mathbf{K}}_{\mathcal{N}(i)}\right) \tilde{\mathbf{V}}_{\mathcal{N}(i)}^T, \tag{8}$$

where $\tilde{\mathbf{K}}_{\mathcal{N}(i)}$ and $\tilde{\mathbf{V}}_{\mathcal{N}(i)}$ are the key and value matrices of neighboring nodes, respectively. Therefore, the node v_i's final output of single attention module (AGP) in Gapformer is calculated as follows:

$$\boldsymbol{h}'_i = \boldsymbol{h}_i + \beta * \boldsymbol{z}_i, \tag{9}$$

where β is a balanced hyper-parameter which controls the combination for the attention enhanced with graph pooling and the neighboring attention.

Other Modules. In addition to AGP, Gapformer also contains layer normalization (LN$(\cdot)$) applied after the multi-head self-attention (MHA$(\cdot)$) and the feed-forward blocks (FFN$(\cdot)$), as illustrated in Figure 2. We formalize the Gapformer layer as below:

$$h'^{(l)} = \text{LN}\left(\text{MHA}\left(h^{(l-1)}\right)\right) + h^{(l-1)}; \tag{10}$$

$$h^{(l)} = \text{LN}\left(\text{FNN}\left(h'^{(l)}\right)\right) + h'^{(l)}. \tag{11}$$

As with most GT methods, our Gapformer also adopts the positional encodings (PEs), *i.e.*, Laplacian eigenvectors encodings (LapPE) [Dwivedi and Bresson, 2021; Kreuzer *et al.*, 2021] and random-walk positional encodings (RWPE) [Dwivedi *et al.*, 2022].

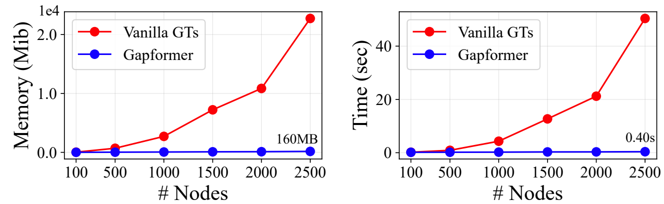

Figure 4: Running time and GPU memory of the full self-attention in vanilla GTs and sparse self-attention in our proposed Gapformer. We evaluate the performance of Gapformer on the synthetic datasets. The time and memory usages of Gapformer scale linearly with the number of nodes, unlike the full self-attention mechanism in vanilla GTs whose values scale exponentially.

Two Types of Attention Enhanced with Graph Pooling

In this section, we discuss how to implement the attention enhanced with graph pooling from the global and local views.

Attention Enhanced with Global Graph Pooling (AGP-G). AGP-G reduces the number of attended nodes by compressing the original nodes into new pooling nodes in smaller sizes. Intuitively, all information in a graph is compressed into the new pooling nodes. As shown in Figure 3, given node features $\mathbf{H} \in \mathbb{R}^{n \times d}$ with their adjacency information $\mathbf{A} \in \{0,1\}^{n \times n}$, we first construct new keys and values using graph pooling operations, as follows:

$$\overline{\mathbf{K}}_{\mathcal{S}}^{Global} = \text{Pooling}\left(\mathbf{H}\mathbf{W}^{\tilde{K}}, \mathbf{A}\right); \tag{12}$$

$$\overline{\mathbf{V}}_{\mathcal{S}}^{Global} = \text{Pooling}\left(\mathbf{H}\mathbf{W}^{\tilde{V}}, \mathbf{A}\right). \tag{13}$$

Then, as shown in the left part of Figure 3, each node ($\mathbf{Q}$) in the original graph attends to the pooling nodes ($\overline{\mathbf{K}}_{\mathcal{S}}^{Global}$) in the new sets $\mathcal{S}$ to generate attention scores. For the pooling operation (Pooling$(\cdot)$), we empirically explore several different pooling methods [Liu *et al.*, 2022b] to perform compressions, including flat pooling methods (*e.g.*, the mean and max pooling), and trainable pooling mechanisms (*e.g.*, Set-Pool [Vinyals *et al.*, 2016] and SAGPool [Lee *et al.*, 2019]).

Attention Enhanced with Local Graph Pooling (AGP-L). Different from AGP-G, AGP-L works by compressing the neighbor information of each node. Specifically, as shown in the right part of Figure 3, for each node (*i.e.*, each query $\mathbf{Q}_i$), we execute graph pooling on its neighboring nodes extracted from its k-hop subgraphs. Formally, the new keys and values for each node (*i.e.*, node v_i) are generated by

$$\overline{\mathbf{K}}_{\mathcal{S}(i)}^{Local} = \text{Pooling}\left(\tilde{\mathbf{K}}_{\mathcal{N}(i,k)}\right); \tag{14}$$

$$\overline{\mathbf{V}}_{\mathcal{S}(i)}^{Local} = \text{Pooling}\left(\tilde{\mathbf{V}}_{\mathcal{N}(i,k)}\right), \tag{15}$$

where $\mathcal{N}(i,k)$ refers to the neighbor set within k hops of node v_i. The nodes in a graph then perform attention with those in the corresponding pooled sets. Formally, the output of the AGP-L for node v_i is calculated by Eq. (7). Note that the operation of extracting k-hop subgraphs can be performed

	Cora	Citeseer	Pubmed	DBLP	CS	Physics	Photo	CoraFull	ogbn-arxiv	Cornell	Texas	Wisconsin	Actor
# Nodes	2,708	3,327	19,717	17,716	18,333	34,493	7,650	19,793	169,343	183	183	251	7,600
# Edges	5,429	4,732	44,338	105,734	81,894	247,962	119,081	126,842	1,166,343	280	195	466	26,752
Homo.	0.83	0.72	0.79	0.70	0.83	0.91	0.85	0.57	0.63	0.30	0.11	0.21	0.22

Table 1: Statistics of benchmark datasets.

in the preprocessing stage without consuming additional resources in the training stage. Finally, as with the global pooling, we empirically explore several different pooling methods for use in performing compression ($\mathrm{Pooling}(\cdot)$), including the flat pooling methods (*e.g.*, the mean and max pooling) and trainable pooling mechanisms (*e.g.*, SetPool [Vinyals *et al.*, 2016] and SAGPool [Lee *et al.*, 2019]).

3.3 Merits of Gapformer

Reducing Computational Complexity. We first analyze the complexity of Gapformer. The computational complexity of the attention enhanced with graph pooling (Eq. (7)) is $\mathcal{O}(n'n)$. Since n' is a constant and usually much smaller than n, the computational complexity can be simplified as $\mathcal{O}(n)$. Moreover, the computational complexity of the neighboring attention (Eq. (8)) is $\mathcal{O}(|E|)$. Therefore, the overall complexity of Gapformer is $\mathcal{O}(n + |E|)$. To illustrate the superiority of Gapformer, we conduct experiments on synthetic datasets. The results in Figure 4 demonstrate that the time and memory usages of Gapformer do indeed scale linearly with the number of nodes, unlike the full self-attention mechanism that scales exponentially, which enables the application of Gapformer to extremely large-scale datasets.

Reducing the Ratio of Noisy Connections. In most existing graph transformer models, each node aggregates information from all nodes in a graph, which provides the global receptive field. This approach may pose a challenge for node classification tasks since the aggregated information could potentially contain noise and irrelevant data that is not useful for the target node. Consequently, this can hinder the model's ability to perform effectively. To address this issue, our proposed Gapformer modifies the standard full self-attention to a sparse schema, which helps greatly reduce the ratio of noisy connections, thereby enhancing the capacity of graph transformer-based models in node classification.

Handling Long-range Dependency. As shown in Figure 1, the receptive field of Gapformer is flexible and ranges from linear growth to exponential growth. Thus, it requires fewer layers than traditional GNN models to capture long-distance connections. This is a remarkable benefit for the case when significant correlations are multiple hops away.

4 Experiments

4.1 Experimental Settings

Datasets. We employ a total of 13 real-world datasets, including nine homophilic graph datasets (Cora, Citeseer, Pubmed, DBLP, CoraFull, CS, Physics, Photo, and ogbn-arxiv) and four heterophilic graph datasets (Cornell, Texas, Wisconsin, and Actor), involving diverse domains (citation, co-authorship, co-purchase, and web pages) and sizes (ogbn-arxiv is a large-scale dataset). The dataset statistics are summarized in Table 1. Please note that in reference to [Zhu *et al.*, 2020], **Homo.** refers to the ratio of edges linking nodes with identical labels. We use different training, validation, and test splits for various datasets. Specifically, for Cora, Citeseer, and Pubmed datasets we follow the (48%/32%/20%) split as proposed in [Pei *et al.*, 2020]. The same splits used by [Zhu *et al.*, 2020] and [Liu *et al.*, 2022a] are adopted for the four heterophilic graph datasets. For all other datasets, we randomly split them into 60%/20%/20% training/validation/test sets following [Zhang *et al.*, 2022]. All the adopted graph datasets, except ogbn-arxiv, can be downloaded from PyTorch Geometric (PyG) [Fey and Lenssen, 2019] [1], and obgn-arxiv can be downloaded from Open Graph Benchmark (OGB) [2].

Baseline. To demonstrate the effectiveness of our proposed method, we compare Gapformer with the following 20 baselines: **(I) 7 standard GCN-based models:** GCN [2017], GatedGCN [2016], APPNP [2019], GCNII [2020], GAT [2018], GATv2 [2022], and SuperGAT [2021]; **(II) 5 heterophilic-graph-oriented models:** MLP [2015], MixHop [2019], FAGCN [2021], H2GCN [2020], and GPRGNN [2021]; **(III) 8 transformer-based models for graphs:** GT-sparse [Dwivedi and Bresson, 2021], SAN [2021], Graphormer [2021], UniMP [2021], LiteGT [2021a], DET [2022], NAGformer [2023], and ANS-GT [2022]. The last five transformer-based models are efficient graph transformer models.

Implementation Details. We assess the effectiveness of our proposed model by measuring its accuracy in node classification. To ensure reliability, we conduct 10 trials for each model using random seeds. We utilize Adam optimizer for GCN-based and heterophily-based methods, while Adamw is adopted for all graph transformer-based models. Each method and dataset are run for 200 epochs, with the test accuracy reported based on the epoch that achieves the highest validation accuracy. For ease of tuning work, we set some hyperparameters: dropout at 0.5, weight decay at $5e^{-4}$, position encoding dimension at 20, and hidden dimension within $\{64, 128, 256\}$. Our implementation of Gapformer is developed using Python (3.7.0), Pytorch (1.11.0), and Pytorch Geometric (2.2.0). All experiments are conducted on a Linux server with two NVIDIA A100s.

4.2 Overall Performance

We evaluate the effectiveness of the proposed model in terms of accuracy. For each model and dataset, we conduct 10 tri-

[1] https://github.com/pyg-team/pytorch_geometric

[2] https://ogb.stanford.edu/docs/nodeprop/#ogbn-arxiv

	Cora	Citeseer	Pubmed	DBLP	Photo	Physics	CS	CoraFull	ogbn-arxiv
GCN-based methods									
GCN [Kipf and Welling, 2017]	86.92$_{\pm 1.33}$	76.13$_{\pm 1.51}$	87.01$_{\pm 0.62}$	85.13$_{\pm 0.44}$	85.94$_{\pm 1.18}$	95.38$_{\pm 0.20}$	89.11$_{\pm 0.70}$	24.49$_{\pm 0.47}$	70.40$_{\pm 0.10}$
GatedGCN [Li *et al.*, 2016]	85.49$_{\pm 1.32}$	74.94$_{\pm 1.68}$	86.19$_{\pm 0.46}$	**85.50**$_{\pm 0.57}$	57.84$_{\pm 14.6}$	95.89$_{\pm 0.21}$	89.94$_{\pm 2.24}$	49.59$_{\pm 7.57}$	62.71$_{\pm 1.76}$
APPNP [Gasteiger *et al.*, 2019]	**87.75**$_{\pm 1.30}$	**76.53**$_{\pm 1.61}$	86.52$_{\pm 0.61}$	85.22$_{\pm 0.56}$	84.71$_{\pm 1.25}$	95.04$_{\pm 0.31}$	87.49$_{\pm 0.48}$	20.61$_{\pm 0.78}$	70.20$_{\pm 0.16}$
GCNII [Chen *et al.*, 2020]	86.08$_{\pm 2.18}$	74.75$_{\pm 1.76}$	85.98$_{\pm 0.61}$	83.26$_{\pm 0.49}$	67.06$_{\pm 1.74}$	94.88$_{\pm 0.32}$	84.23$_{\pm 0.78}$	9.10$_{\pm 0.62}$	69.78$_{\pm 0.16}$
GAT [Veličković *et al.*, 2018]	87.34$_{\pm 1.14}$	75.75$_{\pm 1.86}$	85.37$_{\pm 0.56}$	83.86$_{\pm 0.44}$	87.13$_{\pm 1.00}$	95.14$_{\pm 0.28}$	88.53$_{\pm 0.54}$	25.32$_{\pm 1.43}$	67.56$_{\pm 0.12}$
GATv2 [Brody *et al.*, 2022]	87.25$_{\pm 0.89}$	75.72$_{\pm 1.30}$	85.75$_{\pm 0.55}$	84.96$_{\pm 0.47}$	81.52$_{\pm 3.23}$	95.02$_{\pm 0.32}$	88.46$_{\pm 0.61}$	31.62$_{\pm 0.71}$	68.84$_{\pm 0.13}$
SuperGAT [Kim and Oh, 2021]	87.22$_{\pm 1.24}$	75.41$_{\pm 1.78}$	85.30$_{\pm 0.52}$	83.64$_{\pm 0.40}$	85.83$_{\pm 1.29}$	95.11$_{\pm 0.26}$	88.11$_{\pm 0.43}$	23.52$_{\pm 0.85}$	66.99$_{\pm 0.07}$
Heterophily-based methods									
MLP [LeCun *et al.*, 2015]	70.32$_{\pm 2.68}$	68.64$_{\pm 1.98}$	86.46$_{\pm 0.35}$	72.54$_{\pm 0.95}$	88.66$_{\pm 0.85}$	95.12$_{\pm 0.26}$	92.99$_{\pm 0.51}$	53.63$_{\pm 0.96}$	52.63$_{\pm 0.12}$
MixHop [Sami *et al.*, 2019]	84.47$_{\pm 1.37}$	72.04$_{\pm 1.49}$	88.44$_{\pm 0.47}$	82.23$_{\pm 0.65}$	93.24$_{\pm 0.59}$	96.34$_{\pm 0.22}$	93.88$_{\pm 0.63}$	56.66$_{\pm 1.19}$	70.83$_{\pm 0.30}$
H2GCN [Zhu *et al.*, 2020]	83.48$_{\pm 2.29}$	75.16$_{\pm 1.48}$	88.86$_{\pm 0.45}$	83.10$_{\pm 0.27}$	91.56$_{\pm 0.70}$	96.28$_{\pm 0.13}$	94.02$_{\pm 0.31}$	50.38$_{\pm 0.82}$	68.29$_{\pm 0.67}$
FAGCN [Bo *et al.*, 2021]	85.17$_{\pm 1.08}$	75.60$_{\pm 2.37}$	87.71$_{\pm 0.44}$	83.80$_{\pm 0.47}$	87.53$_{\pm 0.75}$	95.86$_{\pm 0.12}$	91.82$_{\pm 0.54}$	30.14$_{\pm 0.45}$	66.12$_{\pm 0.02}$
GPRGNN [Chien *et al.*, 2021]	86.82$_{\pm 1.15}$	75.45$_{\pm 1.40}$	86.83$_{\pm 0.48}$	84.97$_{\pm 0.64}$	92.27$_{\pm 0.44}$	96.06$_{\pm 0.21}$	93.60$_{\pm 0.36}$	64.11$_{\pm 0.80}$	68.28$_{\pm 0.21}$
Graph Transformer-based methods									
SAN [Kreuzer *et al.*, 2021]	81.91$_{\pm 3.42}$	69.63$_{\pm 3.76}$	81.79$_{\pm 0.98}$	–	94.17$_{\pm 0.65}$	96.83$_{\pm 0.18}$	94.16$_{\pm 0.36}$	45.61$_{\pm 5.25}$	69.17$_{\pm 0.15}$
Graphormer [Ying *et al.*, 2021]	67.71$_{\pm 0.78}$	73.30$_{\pm 1.21}$	OOM	OOM	85.20$_{\pm 4.12}$	OOM	OOM	OOM	OOM
LiteGT [Chen *et al.*, 2021a]	80.62$_{\pm 2.69}$	69.09$_{\pm 2.03}$	85.45$_{\pm 0.69}$	–	–	OOM	92.16$_{\pm 0.44}$	56.86$_{\pm 0.69}$	OOM
UniMP [Shi *et al.*, 2021]	84.18$_{\pm 1.39}$	75.00$_{\pm 1.59}$	88.56$_{\pm 0.32}$	84.25$_{\pm 0.42}$	92.49$_{\pm 0.47}$	**96.82**$_{\pm 0.13}$	94.20$_{\pm 0.34}$	**67.93**$_{\pm 0.56}$	**73.19**$_{\pm 0.18}$
DET [Guo *et al.*, 2022]	86.30$_{\pm 1.41}$	75.37$_{\pm 1.41}$	86.28$_{\pm 0.44}$	84.96$_{\pm 0.39}$	91.44$_{\pm 0.49}$	96.30$_{\pm 0.18}$	93.34$_{\pm 0.31}$	67.12$_{\pm 0.93}$	55.70$_{\pm 0.30}$
NAGphormer [Chen *et al.*, 2023]	85.77$_{\pm 1.35}$	73.69$_{\pm 1.48}$	87.87$_{\pm 0.33}$	–	**94.64**$_{\pm 0.60}$	96.66$_{\pm 0.16}$	**95.00**$_{\pm 0.14}$	66.75$_{\pm 0.79}$	–
ANS-GT [Zhang *et al.*, 2022]	86.71$_{\pm 1.45}$	74.57$_{\pm 1.51}$	**89.76**$_{\pm 0.46}$	85.19$_{\pm 0.47}$	94.41$_{\pm 0.62}$	96.22$_{\pm 0.15}$	94.64$_{\pm 0.24}$	61.66$_{\pm 1.85}$	–
Gapformer (w/o GP)	81.69$_{\pm 2.03}$	70.90$_{\pm 3.05}$	87.35$_{\pm 0.51}$	83.54$_{\pm 0.48}$	94.06$_{\pm 0.81}$	96.68$_{\pm 0.14}$	93.62$_{\pm 0.72}$	54.95$_{\pm 1.37}$	70.20$_{\pm 0.21}$
Gapformer (AGP-G)	**87.37**$_{\pm 0.76}$	**76.21**$_{\pm 1.47}$	**88.98**$_{\pm 0.46}$	**85.50**$_{\pm 0.43}$	**94.81**$_{\pm 0.45}$	**97.10**$_{\pm 0.12}$	**95.13**$_{\pm 0.40}$	**68.22**$_{\pm 0.70}$	**71.90**$_{\pm 0.19}$
Gapformer (AGP-L)	87.04$_{\pm 1.14}$	75.24$_{\pm 1.44}$	88.49$_{\pm 0.44}$	85.31$_{\pm 0.49}$	92.34$_{\pm 0.63}$	96.42$_{\pm 0.20}$	94.48$_{\pm 0.36}$	67.59$_{\pm 0.66}$	71.70$_{\pm 0.33}$

Notations: 1) Gapformer (w/o GP) refers to Gapformer without graph pooling, which is also the GT-sparse baseline model [Dwivedi and Bresson, 2021]. 2) The results of Graphormer are taken from NAGphormer [2023] and Specformer [2023]. 3) The full-version of GT [2021] and SAN [2021] is OOM even on the small-scale datasets. 4) Another recent graph Transformer, SAT [Chen *et al.*, 2022], is not considered, as it reports OOM even on the small-scale datasets.

Table 2: Experimental results for the node classification task on eight common datasets (mean accuracy (%) and standard deviation over 10 different runs). Red: the best performance per dataset. Blue: the second best performance per dataset. OOM denotes out-of-memory.

als with random seeds, and then take the mean accuracy and standard deviation, which are reported in Tables 2 and 3.

Performance on Homophilic Graphs. From the results in Table 2, we can observe: **1)** Gapformer achieves the state-of-the-art performance on five datasets and competitive performance on three datasets, which demonstrates the effectiveness of our proposed method. Gapformer has a significant advantage over its variant that does not include the graph pooling module, referred to as Gapformer (w/o GP). **2)** Compared with GCN-based methods, Gapformer performs better on graphs with more nodes (*e.g.*, Photo, Physics, and CS). This is likely because local message-passing based GCN methods neglect long-range dependencies, but our Gapformer enables to learn more informative node representations from the multi-hop neighborhoods, which is a remarkable benefit for the bigger graphs, where the required receptive field is large [Guo *et al.*, 2022; Park *et al.*, 2022; Wu *et al.*, 2022]. **3)** The performance of Gapformer surpasses that of graph transformer-based methods on the small-scale datasets (*e.g.*, Cora and Citeseer). The reason may be that vanilla GTs with full connected attention (*e.g.*, Graphormer) and sampling-based GTs (*e.g.*, LiteGT [Chen *et al.*, 2021a] and DET [Guo *et al.*, 2022]) both introduce more noises from massive unrelated nodes. However, GCN-based models perform better than GT-based methods on small-scale datasets. This is likely because, on small-scale datasets, local information is more important. Moreover, GTs, including our Gapformer, have more parameters than GCN-based methods, meaning that they may suffer from over-fitting on small datasets. **4)** Our Gapformer can be applied to large-scale graphs, such as ogbn-arxiv, while some other transformer-based methods cannot be applied to such graphs due to their poor scalability. We have noticed that Graphormer [Ying *et al.*, 2021] and LiteGT [Chen *et al.*, 2021a] encounter out-of-memory errors, even when processing small graphs. This highlights the need for a graph Transformer that can scale effectively to handle large-scale graphs.

Performance on Heterophilic Graphs. Table 3 summarizes the results of models on heterophilic graphs. From these results, we can make the following observations: **1)** GCN-

	Cornell	Texas	Wisconsin	Actor
GCN-based methods				
GCN [Kipf and Welling, 2017]	$45.67_{\pm 7.96}$	$60.81_{\pm 8.03}$	$52.55_{\pm 4.27}$	$28.73_{\pm 1.17}$
GatedGCN [Li *et al.*, 2016]	$72.70_{\pm 5.33}$	$75.40_{\pm 4.26}$	$81.37_{\pm 3.31}$	$35.13_{\pm 1.10}$
APPNP [Gasteiger *et al.*, 2019]	$41.35_{\pm 7.15}$	$61.62_{\pm 5.37}$	$55.29_{\pm 3.90}$	$29.42_{\pm 0.81}$
GCNII [Chen *et al.*, 2020]	$44.32_{\pm 5.81}$	$58.91_{\pm 4.32}$	$52.54_{\pm 7.32}$	$25.40_{\pm 0.97}$
GAT [Veličković *et al.*, 2018]	$47.02_{\pm 7.66}$	$62.16_{\pm 4.52}$	$57.45_{\pm 3.51}$	$28.33_{\pm 1.13}$
GATv2 [Brody *et al.*, 2022]	$50.27_{\pm 8.97}$	$60.54_{\pm 4.55}$	$52.74_{\pm 3.96}$	$28.79_{\pm 1.47}$
SuperGAT [Kim and Oh, 2021]	$43.51_{\pm 6.55}$	$59.99_{\pm 4.64}$	$53.52_{\pm 4.64}$	$28.08_{\pm 1.03}$
Heterophily-based methods				
MLP [LeCun *et al.*, 2015]	$71.62_{\pm 5.57}$	$77.83_{\pm 5.24}$	$82.15_{\pm 6.93}$	$33.26_{\pm 0.91}$
MixHop [Sami *et al.*, 2019]	$76.48_{\pm 2.97}$	**83.24**$_{\pm 4.48}$	**85.48**$_{\pm 3.06}$	$34.92_{\pm 0.91}$
H2GCN [Zhu *et al.*, 2020]	$75.40_{\pm 4.09}$	$79.73_{\pm 3.25}$	$77.57_{\pm 4.11}$	**36.18**$_{\pm 0.45}$
FAGCN [Bo *et al.*, 2021]	$67.56_{\pm 5.26}$	$75.67_{\pm 4.68}$	$75.29_{\pm 3.06}$	$32.13_{\pm 1.33}$
GPRGNN [Chien *et al.*, 2021]	**76.76**$_{\pm 2.16}$	**81.08**$_{\pm 4.35}$	$82.66_{\pm 5.62}$	$35.30_{\pm 0.80}$
Graph Transformer-based methods				
SAN [Kreuzer *et al.*, 2021]	$50.85_{\pm 8.54}$	$60.17_{\pm 6.66}$	$51.37_{\pm 3.08}$	$27.12_{\pm 2.59}$
UniMP [Shi *et al.*, 2021]	$66.48_{\pm 12.5}$	$73.51_{\pm 8.44}$	$79.60_{\pm 5.41}$	$35.15_{\pm 0.84}$
DET [Guo *et al.*, 2022]	$41.35_{\pm 7.45}$	$56.76_{\pm 4.98}$	$54.90_{\pm 6.56}$	$28.94_{\pm 0.64}$
NAGphormer [Chen *et al.*, 2023]	$56.22_{\pm 8.08}$	$63.51_{\pm 6.53}$	$62.55_{\pm 6.22}$	$34.33_{\pm 0.94}$
Gapformer (w/o GP)	$61.89_{\pm 5.85}$	$70.54_{\pm 4.75}$	$75.29_{\pm 5.12}$	$33.86_{\pm 0.79}$
Gapformer (AGP-G)	**77.57**$_{\pm 3.43}$	$80.27_{\pm 4.01}$	**83.53**$_{\pm 3.42}$	**36.90**$_{\pm 0.82}$
Gapformer (AGP-L)	$76.22_{\pm 2.65}$	$79.73_{\pm 5.16}$	$82.15_{\pm 2.22}$	$36.47_{\pm 1.02}$

Notations: 1) Gapformer (w/o GP) refers to Gapformer without graph pooling, which is also the GT-sparse baseline model [Dwivedi and Bresson, 2021]. 2) The official codes of LiteGT [Chen *et al.*, 2021a] and ANS-GT [2022] fail to handle the above four heterophilic datasets.

Table 3: Experimental results for the node classification task on four heterophilic datasets (mean accuracy (%) and standard deviation over 10 different runs). Red: the best performance per dataset. Blue: the second best performance per dataset.

based models exhibit relatively inferior performance on heterophilic graphs. This is because most GCNs utilize directly connected nodes for aggregation even in heterophilic graphs. **2)** Surprisingly, transformer-based models show poor performance, which implies that GTs fail to filter out irrelevant messages. **3)** Instead, our proposed Gapformer achieves superior performance on heterophilic graph datasets. In particular, Gapformer significantly outperforms transformer-based baselines. This phenomenon is probably because Gapformer summarizes the graph structure or neighbor information from a global view instead of a similarity view.

4.3 Further Discussions

Efficiency of Gapformer. To validate the efficiency of Gapformer, we compare its training cost in terms of running time (s) and GPU memory (MB) with such representative methods as GT [Dwivedi and Bresson, 2021], SAN [Kreuzer *et al.*, 2021], and ANS-GT [Zhang *et al.*, 2022]. The results are summarized in Table 4. From these results, we can observe that Gapformer with AGP-G shows high efficiency compared with all existing GTs, especially when dealing with large-scale graphs. Moreover, Gapformer with AGP-L incurs lower memory costs compared to vanilla GTs, although its time cost is high, comparable to NAGphormer [Chen *et al.*, 2023] and ANS-GT [Zhang *et al.*, 2022].

	Cora		Photo	
	Memory (MB)	Training Time (s)	Memory (MB)	Training Time (s)
GAT [Veličković *et al.*, 2018]	1,672	2.64	2,189	17.27
GT-Full [Dwivedi and Bresson, 2021]	13,375	48.80	OOM	OOM
SAN-Sparse [Kreuzer *et al.*, 2021]	2,936	16.64	4,878	82.77
SAN-Full [Kreuzer *et al.*, 2021]	13,410	372.94	OOM	OOM
LiteGT [Chen *et al.*, 2021a]	4,414	23.96	OOM	OOM
UniMP [Shi *et al.*, 2021]	1,861	4.85	2,437	20.88
DET [Guo *et al.*, 2022]	1,961	11.93	4,827	222.73
NAGphormer [Chen *et al.*, 2023]	1,879	12.02	1923	1,936.22
ANS-GT [Zhang *et al.*, 2022]	1,909	805.89	1,883	19,709.21
Gapformer (w/o GP)	1,827	3.81	2,725	7.89
Gapformer (AGP-G)	1,829	5.56	2,727	9.40
Gapformer (AGP-L)	1,843	40.52	6,983	3,038.38

Table 4: Comparison of training time and GPU memory costs of Gapformer to graph transformer-based models. GT-Full and SAN-Full denote the dense self-attention versions of GT and SAN, respectively. OOM denotes out-of-memory.

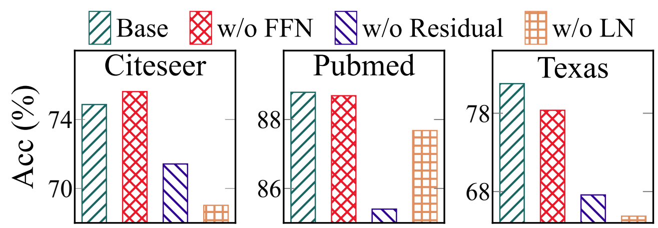

Figure 5: Ablation: Components of Graph Transformer architecture.

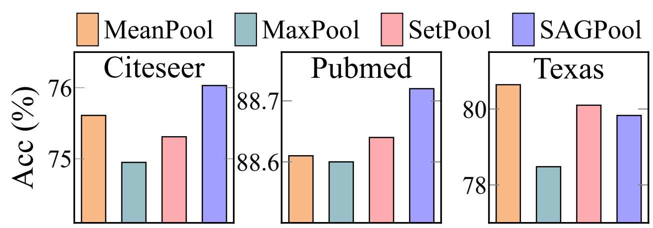

Figure 6: Performance of Gapformer with different pooling types.

Ablation on Transformer Components and Pooling Type. We next study the effects of the three components of the transformer and different pooling types. We conduct experiments with AGP-G on three graph datasets. Please note that, apart from the selected components, all other parts remain identical to the complete model. We can observe in Figure 5 that the performance drops after the LayerNorm and Residual components are removed. However, the performance increases after removing the FFN module, which indicates that this module may cause over-fitting. Moreover, from the results in Figure 6, we can determine that SAGPool [Lee *et al.*, 2019] performs better than other simple pooling methods. This encourages our search for more effective and efficient pooling methods to improve the performance of Gapformer.

Impact of Number of Layers and Balance Parameter. We analyzed the effects of l and β using AGP-G on three

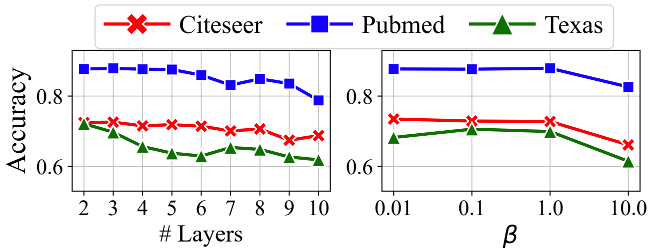

Figure 7: Performance of Gapformer with different parameters.

graph datasets (Citeseer, Pubmed, and Texas) and presented the results in Figure 7. Specifically, we investigated how the number of layers impacts node classification performance. Our findings indicate that as l increases from low to high values, test accuracy decreases due to over-fitting. Additionally, when β is relatively small, our model's accuracy curve remains smooth indicating less sensitivity to hyper-parameters.

5 Conclusion

We propose Gapformer, which combines Graph Transformers (GTs) with Graph Pooling for efficient node classification. Our Gapformer addresses the two main issues of existing GTs: potential noises from long-distance neighbors and the quadratic computational complexity in regards to the number of nodes. Extensive experiments on 13 graph datasets demonstrate that Gapformer outperforms existing GTs and Graph Neural Networks. Despite its competitive performance, Gapformer still has room for improvement. For instance, 1) devising an effective manner to combine the proposed local pooling enhanced attention and global pooling enhanced attention, and 2) incorporating useful techniques to further enhance the performance on large-scale graph datasets.

Acknowledgments

This work was supported in part by the Natural Science Foundation of China (Nos. 61976162, 82174230, 62002090, 62172354), Artificial Intelligence Innovation Project of Wuhan Science and Technology Bureau (No. 2022010702040070), and Science and Technology Major Project of Hubei Province (Next Generation AI Technologies) (No. 2019AEA170). Dr Wu is partially supported by ARC Projects LP210301259 and DP230100899.

References

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

[Ba *et al.*, 2016] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

[Beltagy *et al.*, 2020] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

[Bianchi *et al.*, 2020] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *ICML*, 2020.

[Bo *et al.*, 2021] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *AAAI*, 2021.

[Bo *et al.*, 2023] Deyu Bo, Chuan Shi, Lele Wang, and Renjie Liao. Specformer: Spectral graph neural networks meet transformers. In *ICLR*, 2023.

[Brody *et al.*, 2022] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *ICLR*, 2022.

[Cai *et al.*, 2022] Weishan Cai, Wenjun Ma, Jieyu Zhan, and Yuncheng Jiang. Entity alignment with reliable path reasoning and relation-aware heterogeneous graph transformer. In *IJCAI*, 2022.

[Chen *et al.*, 2020] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, 2020.

[Chen *et al.*, 2021a] Cong Chen, Chaofan Tao, and Ngai Wong. Litegt: Efficient and lightweight graph transformers. In *CIKM*, 2021.

[Chen *et al.*, 2021b] Jianwen Chen, Shuangjia Zheng, Ying Song, Jiahua Rao, and Yuedong Yang. Learning attributed graph representation with communicative message passing transformer. In *IJCAI*, 2021.

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

[Chen *et al.*, 2023] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. NAGphormer: A tokenized graph transformer for node classification in large graphs. In *ICLR*, 2023.

[Chien *et al.*, 2021] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *ICLR*, 2021.

[Choromanski *et al.*, 2021] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, et al. Rethinking attention with performers. In *ICLR*, 2021.

[Choromanski *et al.*, 2022] Krzysztof Choromanski, Han Lin, Haoxian Chen, Tianyi Zhang, Arijit Sehanobish, Valerii Likhosherstov, Jack Parker-Holder, Tamas Sarlos, Adrian Weller, and Thomas Weingarten. From block-toeplitz matrices to differential equations on graphs: towards a general theory for scalable masked transformers. In *ICML*, 2022.

[Ding *et al.*, 2020] Liang Ding, Longyue Wang, and Dacheng Tao. Self-attention with cross-lingual position representation. In *ACL*, 2020.

[Duvenaud *et al.*, 2015] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*, 2015.

[Dwivedi and Bresson, 2021] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *AAAI Workshop*, 2021.

[Dwivedi *et al.*, 2022] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *ICLR*, 2022.

[Fey and Lenssen, 2019] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop*, 2019.

[Gao and Ji, 2019] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *ICML*, 2019.

[Gasteiger *et al.*, 2019] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Combining neural networks with personalized pagerank for classification on graphs. In *ICLR*, 2019.

[Guo *et al.*, 2022] Lingbing Guo, Qiang Zhang, and Huajun Chen. Unleashing the power of transformer for graphs. *arXiv:2202.10581*, 2022.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[Hussain *et al.*, 2022] Md Shamim Hussain, Mohammed J. Zaki, and Dharmashankar Subramanian. Global self-attention as a replacement for graph convolution. In *SIGKDD*, 2022.

[Kim and Oh, 2021] Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *ICLR*, 2021.

[Kim *et al.*, 2022] Jinwoo Kim, Dat Tien Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. In *NeurIPS*, 2022.

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

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

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, pages 436–444, 2015.

[Lee *et al.*, 2019] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *ICML*, 2019.

[Li *et al.*, 2016] Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. Gated graph sequence neural networks. In *ICLR*, 2016.

[Liu *et al.*, 2022a] Chuang Liu, Xueqi Ma, Yinbing Zhan, Liang Ding, Dapeng Tao, Bo Du, Wenbin Hu, and Danilo Mandic. Comprehensive graph gradual pruning for sparse training in graph neural networks. *arXiv:2207.08629*, 2022.

[Liu *et al.*, 2022b] Chuang Liu, Yibing Zhan, Chang Li, Bo Du, Jia Wu, Wenbin Hu, Tongliang Liu, and Dacheng Tao. Graph pooling for graph neural networks: Progress, challenges, and opportunities. *arXiv:2204.07321*, 2022.

[Liu *et al.*, 2023] Chuang Liu, Yibing Zhan, Xueqi Ma, Dapeng Tao, Bo Du, and Wenbin Hu. Masked graph auto-encoder constrained graph pooling. In *ECML PKDD*, 2023.

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

[Nguyen *et al.*, 2022] Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. Universal graph transformer self-attention networks. In *WWW*, 2022.

[Park *et al.*, 2022] Jinyoung Park, Seongjun Yun, Hyeonjin Park, Jaewoo Kang, Jisu Jeong, Kyung-Min Kim, Jung-woo Ha, and Hyunwoo J Kim. Deformable graph transformer. *arXiv:2206.14337*, 2022.

[Pei *et al.*, 2020] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2020.

[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.*, 2020a] 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.

[Rong *et al.*, 2020b] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

[Sami *et al.*, 2019] Sami, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *ICML*, 2019.

[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.

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

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

[Vinyals *et al.*, 2016] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2016.

[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. In *NeurIPS*, 2021.

[Wu *et al.*, 2022] Qitian Wu, Wentao Zhao, Zenan Li, David Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. In *NeurIPS*, 2022.

[Xu *et al.*, 2019a] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S. Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. Graph contextualized self-attention network for session-based recommendation. In *IJCAI*, 2019.

[Xu *et al.*, 2019b] Nuo Xu, Pinghui Wang, Long Chen, Jing Tao, and Junzhou Zhao. Mr-gnn: Multi-resolution and dual graph neural network for predicting structured entity interactions. In *IJCAI*, 2019.

[Ying *et al.*, 2018] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[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*, 2021.

[Zhang *et al.*, 2019] Jiani Zhang, Xingjian Shi, Shenglin Zhao, and Irwin King. Star-gcn: Stacked and reconstructed graph convolutional networks for recommender systems. In *IJCAI*, 2019.

[Zhang *et al.*, 2022] Zaixi Zhang, Qi Liu, Qingyong Hu, and Chee-Kong Lee. Hierarchical graph transformer with adaptive node sampling. In *NeurIPS*, 2022.

[Zhao *et al.*, 2021] Jianan Zhao, Chaozhuo Li, Qianlong Wen, Yiqi Wang, Yuming Liu, Hao Sun, Xing Xie, and Yanfang Ye. Gophormer: Ego-graph transformer for node classification. *arXiv:2110.13094*, 2021.

[Zhu *et al.*, 2020] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *NeurIPS*, 2020.

[Zhu *et al.*, 2021] Yiran Zhu, Xing Xu, Fumin Shen, Yanli Ji, Lianli Gao, and Heng Tao Shen. Posegtac: Graph transformer encoder-decoder with atrous convolution for 3d human pose estimation. In *IJCAI*, 2021.

[Zhu *et al.*, 2022] Yangfu Zhu, Linmei Hu, Xinkai Ge, Wanrong Peng, and Bin Wu. Contrastive graph transformer network for personality detection. In *IJCAI*, 2022.