Electronic Research Archive

https://www.aimspress.com/journal/era

ERA, 33(2): 1093–1119.
DOI: 10.3934/era.2025049
Received: 24 November 2024
Revised: 05 February 2025
Accepted: 18 February 2025
Published: 26 February 2025

Research article

Dual-branch graph Transformer for node classification

Yong Zhang[1], Jingjing Song[1,*], Eric C.C. Tsang[2] and Yingxing Yu[1]

[1] School of Computer, Jiangsu University of Science and Technology, Zhenjiang 212100, China

[2] Faculty of Information Technology, Macau University of Science and Technology, Macau, China

* **Correspondence:** Email: songjingjing108@163.com.

Abstract: As an emerging architecture, graph Transformers (GTs) have demonstrated significant potential in various graph-related tasks. Existing GTs are mainly oriented to graph-level tasks and have proved their advantages, but they do not perform well in node classification tasks. This mainly comes from two aspects: (1) The global attention mechanism causes the computational complexity to grow quadratically with the number of nodes, resulting in substantial resource demands, especially on large-scale graphs; (2) a large number of long-distance irrelevant nodes disperse the attention weights and weaken the focus on local neighborhoods. To address these issues, we proposed a new model, dual-branch graph Transformer (DCAFormer). The model divided the graph into clusters with the same number of nodes by a graph partitioning algorithm to reduce the number of input nodes. Subsequently, the original graph was processed by graph neural network (GNN) to obtain outputs containing structural information. Next, we adopted a dual-branch architecture: The local branch (intracluster Transformer) captured local information within each cluster, reducing the impact of long-distance irrelevant nodes on attention; the global branch (intercluster Transformer) captured global interactions across clusters. Meanwhile, we designed a hybrid feature mechanism that integrated original features with GNN outputs and separately optimized the construction of the query (Q), key (K), and value (V) matrices of the intracluster and intercluster Transformers in order to adapt to the different modeling requirements of two branches. We conducted extensive experiments on 8 benchmark node classification datasets, and the results showed that DCAFormer outperformed existing GTs and mainstream GNNs.

Keywords: graph Transformer; graph neural networks; node classification; dual-branch; graph partitioning

1. Introduction

Graph data is widely used in many practical applications, such as social networks [1], bioinformatics [2], recommendation systems [3], and knowledge graphs [4]. This type of data is characterized by complex relationships between nodes and edges, which are difficult to be effectively processed by traditional machine learning methods. In this context, GNNs [5–8], as a representative graph learning method, have attracted widespread attention in recent years. Due to the message-passing mechanism, GNNs can effectively capture the topology and node features of the graph and then learn high-quality representations. Although GNNs perform well in handling graph data, they still face some inherent limitations. For example, as the depth of the network increases, GNNs tend to suffer from problems such as over-smoothing [9] and over-squashing [10], which lead to a gradual convergence of the features of different nodes, making it difficult to distinguish the relationships between distant nodes. This restricts the receptive field of GNNs mainly to shallow neighbors, limiting their ability to capture global information. To address these problems, researchers have started to focus on Transformer [11], a model that has shown superior performance in processing Euclidean data (e.g., natural language [12] and images [13]). Its powerful global modeling capabilities and parallel computing advantages have led to its significant success in several fields [14, 15]. Transformer can naturally construct a fully connected graph and capture the complex interactions between nodes through a global attention mechanism, which provides a powerful solution to overcome the local receptive field limitations of GNNs.

Nevertheless, the unique topology and node features of graph data prevent Transformer from encoding such data directly. To address this, some recent studies [16–18] have proposed to design unique positional encodings for each node to promote the application of Transformer on graph data, giving rise to GTs. These models have demonstrated outstanding performance in a growing number of graph tasks [19, 20]. Existing GTs are mainly used for graph-level tasks, with graphs containing only a small number of nodes. In contrast, developing GTs suitable for node classification on graphs with a large number of nodes remains a challenging problem. The reasons are as follows: (1) The receptive field of GTs is global, which leads to the computational complexity of the self-attention mechanism being quadratic with respect to the number of nodes. This results in a very high calculation overhead on large-scale graphs, making GTs difficult to scale effectively. (2) The attention mechanism of GTs is based on global interactions between nodes. In graphs with a large number of nodes, the attention weights will be spread over a vast number of irrelevant nodes, leading to the blurring of attention [21].

In recent years, some studies have applied GTs to node classification. Kuang et al. [21] introduced a two-view architecture that combines GNNs with graph coarsening methods to simultaneously capture local and remote information, and reduced the computational effort of Transformer on large-scale graphs. Zhao et al. [22] proposed sampling ego-graphs as input to the Transformer to reduce computational complexity. At the same time, they introduced a proximity-enhanced attention mechanism to capture fine-grained structural bias. Derived from previous work, Zhang et al. [23] optimized the node sampling strategy and proposed a hierarchical attention scheme with graph coarsening to capture long-range interactions. Liu et al. [24] employed a graph pooling technique to reduce the number of nodes while supplementing neighborhood and structural information through adjacency matrix and positional encodings, balancing the performance and efficiency of the model. Chen et al. [25] focused on self-attention only on multi-hop aggregations for each node's embedding vector, providing scalability for mini-batch training of GTs and achieving good results.

On the one hand, although these studies reduce the computational complexity through techniques such as sampling, graph coarsening, and graph pooling, their methods inevitably lead to the loss of some original node information. Meanwhile, in order to preserve the graph structural information in attention computation, these studies introduce positional encoding, which further increases the calculation overhead. Therefore, how to reduce computational complexity while preserving the structural and original node information remains an urgent challenge.

On the other hand, most existing studies handle local and global information by mixing them within a single Transformer branch. Although this hybrid modeling approach simplifies the model structure, it easily leads to information interference and attention dispersion: The details of the local information may be obscured by the overall features of the global information, while the global information capture may be weakened by the complexity of local features. This not only limits the precise modeling of local and global information, but also weakens their expected synergy. Therefore, decoupling local and global information and modeling them independently is a more effective strategy.

To address these issues, we propose leveraging a dual-branch architecture. Dual-branch architectures have been widely validated in various domains. For instance, in image processing, they effectively improve the performance of tasks such as deraining and deblurring through fine-grained and coarse-grained feature modeling [26–28]. In long-tailed learning, collaborative optimization between branches significantly mitigates class imbalance between head and tail categories [29–31]. In scene graph generation, hierarchical learning across fine-grained and coarse-grained branches has improved semantic label modeling [32, 33].

Inspired by these works, we propose a DCAFormer that not only reduces the computational complexity of the Transformer but also effectively preserves the information and structure of the original graph, enabling it to handle both local and global information. First, we utilize a graph partitioning algorithm (e.g., METIS [34]) to divide the graph into multiple clusters, each containing a smaller number of nodes and, thus, reducing the number of nodes input to the Transformer. Next, a GNN processes the entire original graph to generate node representations that incorporate graph structural information. Based on this, we design a dual-branch Transformer module, where the local branch (intracluster Transformer) models local information within clusters, and the global branch (intercluster Transformer) captures global interactions across clusters. To adapt to the different modeling needs of the two branches, we optimize the construction of the query (Q), key (K), and value (V) matrices to align with the attention mechanism of each branch: in the local branch, the query and key matrices are computed from GNN representations of the intracluster nodes, incorporating local structural information, while the value matrix is computed from the original features of intracluster nodes, preserving fundamental node attributes. In the global branch, the query matrix is computed from GNN representations of all nodes to guide intercluster attention, while the key and value matrices are obtained by pooling the GNN representations and original features of each cluster to enhance global information modeling. This construction ensures that different branches focus on different granularities of information, avoiding the limitations of existing methods that either rely solely on original features or GNN-generated node representations for the construction of query, key, and value matrices [35]. Additionally, it eliminates the need for complex k-subtree or k-subgraph GNN extractors [36]. This design limits the scope of attention calculation through a branching structure, effectively preventing irrelevant nodes from diluting attention weights while ensuring precise local modeling and reasonable global relationship capture. Finally, we compare DCAFormer with 20 methods on 8 benchmark node classification datasets to verify the effectiveness of this method. Our main contributions are summarized as follows:

- We propose DCAFormer, an innovative dual-branch graph Transformer. This model combines intracluster Transformers with intercluster Transformers to effectively process local and global information in the graph and reduce the computational complexity of the Transformer.
- We propose a hybrid feature mechanism that optimizes the construction of the query (Q), key (K), and value (V) matrices separately for the dual-branch architecture, adapting to the distinct modeling requirements of different branches.
- Compared with 20 existing methods, our model shows good performance on the node classification task.

2. Related works

2.1. GNNs

GNNs, as classic methods for processing graph data, primarily learn node representations by aggregating the information of neighboring nodes, of which typical representatives include graph convolutional network (GCN) [5] and graph attention network (GAT) [6]. GCN captures local structural information of the graph through convolution operations, while GAT uses a self-attention mechanism to dynamically weight neighbor information. Both models have shown strong performance on various graph tasks. However, due to the problems of over-smoothing and over-squashing, these models are ineffective in capturing deep graph structural information. To address this problem, a series of deep GNNs have been developed. Klicpera et al. [37] used personalized PageRank [38] to propagate information. Chen et al. [39] used residual connections [40] and identity mapping [41] techniques to retain shallow features, thereby alleviating the impacts of over-smoothing and over-squashing. Additionally, most GNNs rely on the adjacency matrix as input, which makes them inefficient when processing large-scale graphs, prompting the emergence of scalable GNNs. Hamilton et al. [7] reduced the number of nodes during training by randomly sampling neighbor nodes. Zeng et al. [42] performed sampling at the GNN layer to enhance scalability. Feng et al. [43] used submatrix approximation to accelerate propagation operations. Although both sampling and approximation methods can reduce training costs, they may lead to information loss, which limits their performance on large-scale graphs.

2.2. Graph Transformers

In recent years, Transformer has garnered widespread attention in graph tasks due to its global attention mechanism. Compared with GNNs, many GTs have demonstrated superior performance in processing graph data. Dwivedi et al. [44] first introduced the Transformer architecture to graph tasks and improved the graph representation by using Laplacian eigenvectors to represent the positional encoding of nodes. Subsequently, Kreuzer et al. [16] further proposed utilizing the full spectrum information of the Laplacian matrix as positional encoding to make the structural information between nodes more accurate. Additionally, Ying et al. [18] introduced a spatial encoding method based on the shortest path between nodes, incorporating structural similarity as an attention bias, thereby strengthening the attention matrix's ability to capture graph structure. Nevertheless, these methods all employ global attention computation for all node pairs, and their complexity is quadratic to the number of nodes, which is unacceptable for large-scale graphs. To this end, recent studies have begun to consider the development of GTs suitable for large-scale graphs. Rampášek et al. [45] designed a model that mixes the GNN layer with the Transformer layer to capture local and global information at the same time and reduce the complexity to linear. Shirzad et al. [46] proposed a sparse attention mechanism, which further reduces the amount of computation by selecting some edges to perform attention operations between nodes. Chen et al. [25] introduced a model based on mini-batch training, which achieved good results by aggregating multi-hop neighbor information and performing attention mechanisms. However, this approach may not be able to handle long-distance dependencies, and it uses complex Laplace position encoding, which increases the computational cost. Fu et al. [47] improved it by introducing virtual connections to enhance the capture of long-range dependencies between nodes. Although the Laplace matrix is replaced with personalized PageRank, which reduces computational complexity, the virtual connection technology increases the dimension of node features and still has a considerable computational cost.

2.3. Dual-branch architecture

The dual-branch architecture has been widely applied in various domains, including image processing, long-tailed learning, and scene graph generation. In deraining tasks, Zhang et al. [26] employed a dual-branch structure to separately process rain streaks and raindrops, and utilized a dual-attention-in-attention mechanism to enhance feature learning within each branch. Zhang et al. [27] leveraged a dual-branch framework to integrate multi-view information and semantic information, improving the performance of removing rain streaks. In deblurring tasks, a survey by Zhang et al. [28] found that the dual-branch architecture effectively enhances image restoration performance by separately modeling local details and global contextual information. In long-tailed learning, Zhou et al. [29] adopted a bilateral-branch network that utilizes conventional sampling and rebalanced sampling strategies to enhance the stability of head classes and the discriminability of tail classes. The SimCal framework [30] employs a bi-level class balanced sampling approach to alleviate classification head bias, significantly improving long-tailed instance segmentation performance. Guo et al. [31] further introduced a dual-branch network with cross-branch loss, ensuring synergy between uniform sampling and rebalanced sampling, leading to superior results on long-tailed datasets. In scene graph generation, Zhou et al. [32] applied a coarse-grained branch to learn expertise and robust features of head predicates, while a fine-grained branch focuses on tail predicates, mitigating the long-tailed issue in scene graph generation. Zheng et al. [33] utilized the hierarchy guided feature learning strategy to learn better region features of both the coarse-grained level and the fine-grained level, capturing relationships between predicate labels, significantly improving predicate classification and overall scene graph generation performance. In summary, the dual-branch architecture effectively addresses information modeling challenges in multiple tasks through division of labor and collaboration. It provides new ideas for decoupling learning and collaborative optimization of local and global features.

3. Preliminaries

Given an undirected graph $G = (V, E)$, it has n nodes and e edges, where V represents the set of nodes and E represents the set of edges. Let $X = \{x_1, x_2, ..., x_n\} \in \mathbb{R}^{n\times d}$ denote the feature matrix of the nodes and d represents the feature dimension of each node. The topology information of the graph is described by the adjacency matrix $A \in \{0, 1\}^{n\times n}$, which defines the mutual connection between nodes, $A_{ij} = A_{ji}$. D is the degree matrix of A, which is a diagonal matrix, D_{ii} denotes the degree of the i-th node, and $D_{ii} = \sum_j A_{ij}$. $\widetilde{A} = A + I_n$ is the adjacency matrix with self-loops, I_n is the identity matrix of n-order, and $\widetilde{D} = D + I_n$.

3.1. GNNs architecture

GNNs effectively capture local information in the graph structure by aggregating the feature information of each node and its neighborhood. Most GNNs utilize a message-passing mechanism to transmit node features to their neighboring nodes, and iteratively aggregate the representations of first-order or higher-order neighbors to expand the receptive field of nodes. Taking GCN as an example, GCN aggregates the feature information of nodes through graph convolution operation with the following formula:

$$H^{(k)} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(k-1)}W^{(k)}), \tag{3.1}$$

where $H^{(k)}$ is the node feature matrix of the k-th layer, with $H^{(0)} = X$, and $W^{(k)}$ is the trainable weight matrix of the k-th layer. $\sigma(\cdot)$ is an activation function, such as ReLU. A GCN model with a total of K layers can perform message-passing at each layer through Eq (3.1) to generate the output representations $H^{(K)}$ of the nodes, where the receptive field of each node covers its K-hop neighbors. This message passing mechanism is also applicable to other GNN models.

3.2. Transformer architecture

The encoder of Transformer is composed of multiple identical layers stacked together, each containing two crucial modules: multi-head self-attention (MHA) and a position-wise feed-forward network (FFN).

The MHA module is the core component of Transformer, which aggregates global information by calculating the correlation (i.e., attention score) between each token in the input sequence and all other tokens. For simplicity, we use single-head self-attention to describe it. Assume that the self-attention module has an input $H \in \mathbb{R}^{n\times h}$, where n is the number of tokens and h is the hidden dimension. The self-attention module first projects H into three subspaces, denoted as Q, K, and V:

$$Q = HW^{(Q)}, K = HW^{(K)}, V = HW^{(V)}, \tag{3.2}$$

where $W^{(Q)} \in \mathbb{R}^{h\times h}$, $W^{(K)} \in \mathbb{R}^{h\times h}$, and $W^{(V)} \in \mathbb{R}^{h\times h}$ are the projection matrices. Next, the attention score is calculated by taking the dot product of the query matrix Q and the key matrix K. Finally, the output representation H' is obtained by weighted aggregation using the attention score and the value matrix V:

$$H' = \text{softmax}(\frac{QK^T}{\sqrt{h}})V, \tag{3.3}$$

Eq (3.3) can be directly extended to MHA, which is widely used in practice. However, the complexity of attention is $O(n^2h)$, resulting in low computational efficiency, which is a major bottleneck of Transformer.

The FFN module typically consists of two fully connected layers and a nonlinear activation function. It can be represented by the following formula:

$$\mathrm{FFN}(H') = \sigma(H'W_1 + b_1)W_2 + b_2, \tag{3.4}$$

where W_1 and W_2 are trainable weight matrices, b_1 and b_2 are bias terms, and $\sigma(\cdot)$ is a nonlinear activation function, such as ReLU. This layer structure further enhances the representation capability of each node.

4. DCAFormer

In this section, we introduce the framework of DCAFormer, as shown in Figure 1. The model aims to reduce the computational complexity of Transformer while effectively capturing both local and global information by combining graph partitioning, GNN processing, and a dual-branch Transformer structure. Specifically, we first apply a graph partitioning algorithm to divide the graph G into different clusters. Then, the GNN-based module processes graph G to obtain an output representation containing graph structural information. We further design a dual-branch Transformer architecture that encodes node representations from a local perspective and a global perspective. Finally, the output representations of the dual-branch are fused to obtain node features that integrate local and global information.

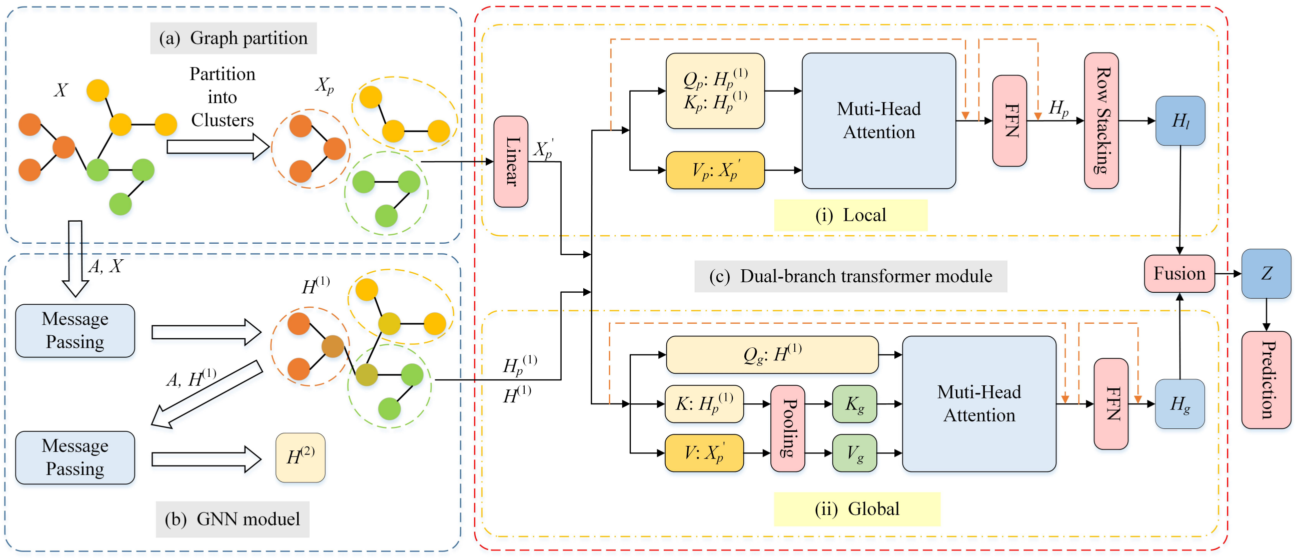

Figure 1. The framework of DCAFormer.

4.1. Graph partition

Traditional GTs utilize a self-attention mechanism to capture information between arbitrary pairs of nodes, which is computationally expensive, especially when dealing with large-scale graph data. To alleviate this problem, we adopt the METIS algorithm to partition the original graph $G = (V, E)$. METIS is widely recognized for its efficiency in partitioning large graphs. METIS is a multilevel graph partitioning algorithm comprising three main stages: (1) Coarsening, which reduces the original graph to a compact multilevel graph by gradually merging adjacent nodes; (2) partitioning, which performs K-way partitioning on the most compact graph and optimizes edge-cut weights (e.g., minimizing intercluster edges) to generate an initial partition; and (3) uncoarsening, which the partition results are progressively projected back to the original graph size, with local optimization applied to improve partition quality further. Ultimately, METIS partitions the original graph into a set of P clusters as $\mathcal{P} = \left\{G_p\right\}$, where each cluster G_p is a subgraph of the original graph. These clusters satisfy the following conditions: $G_p = (V_p, E_p)$, $\bigcup G_p = G$, and $\bigcap G_p = \emptyset$, where $p \in P$, $V_p \subset V$, and $E_p \subset E$. The node features in cluster G_p are represented by $X_p \in \mathbb{R}^{\frac{n}{p} \times d}$. This partitioning approach divides the original graph into smaller subgraphs, with each subgraph containing significantly fewer nodes compared to the original graph, while preserving as much of the structure of the original graph as possible. The partitioned subgraphs serve as input to the subsequent model, effectively reducing the overall computational burden on the Transformer layer.

4.2. GNN module

Graph data has complex structural relationships, which is different from traditional sequential data. However, when Transformer performs attention calculations, it mainly processes feature information without directly considering the structure of the graph. Therefore, before inputting graph data into Transformer, it is usually necessary to introduce additional encoding mechanisms to embed the structural information of the graph. Most existing GTs design positional or structural encodings for each node so that the model can perceive the relative position or structural relationship between nodes.

To simplify this process and capture the structural information of the graph more effectively, we choose GNN to directly generate node representations containing structural information. GNN can naturally embed the local structural information of the graph by aggregating neighborhood information for each node, thereby producing richer node features. Specifically, we apply two layers of GNN to the original graph, generating the output representation $H^{(1)} \in \mathbb{R}^{n \times h}$ of the first layer and $H^{(2)} \in \mathbb{R}^{n \times c}$ of the second layer, respectively, where c is the number of node categories. $H^{(1)}$ will be used as input for the subsequent model, while $H^{(2)}$ will be used to compute the loss function of GNN. Also to reduce the computational effort of the Transformer layer, we consider partitioning the node representation processed by GNN. Since the computation of the GNN does not change the structure of the graph, we can reuse the clusters from the original graph partitioning without performing a new partitioning step. The difference is that the node features in each cluster will be replaced by $H_p^{(1)} \in \mathbb{R}^{\frac{n}{p} \times h}$. $H_p^{(1)}$ will also be used as the input of the subsequent model.

4.3. Dual-branch Transformer module

When processing graph data, the traditional Transformer model may cause the attention of the target node to be dispersed by a large number of remote nodes due to its global attention mechanism, thus causing the attention of its neighboring domain nodes. To address this problem, many existing GTs attempt to fuse local and global attention information by improving the attention mechanism or designing a new model structure to ensure that the model can effectively focus on local details while still capturing global features.

In contrast, we optimize the construction of the query, key, and value matrices in the attention mechanism and propose a dual-branch Transformer architecture. This architecture uses both original features and GNN-processed node representations to participate in the computation of the query, key, and value matrices, and captures the local and global information in the graph through two branches, local and global, respectively. The local branch focuses on local information within each cluster, while the global branch is responsible for integrating global features between clusters.

4.3.1. Local branch

The goal of the local branch is to capture the information from the local neighborhood between nodes within each cluster while avoiding the distraction caused by distant unrelated nodes. By partitioning the graph into several clusters, nodes within each cluster typically exhibit strong local correlations, and the local branch can focus on such relationships to achieve more refined feature representation.

To achieve this goal, we utilize Transformer within each cluster to compute attention between all node pairs. Inspired by the work of Chen et al. [36], which generates structure-aware node representations by extracting k-hop subgraph information through GNN, we redesign the query, key, and value matrices (Q_p, K_p, V_p) in the intracluster Transformer.

First, the GNN module processes the entire graph to generate node representations $H^{(1)}$ that contain structural information. Then, the graph is divided into multiple clusters, and the node representations $H_p^{(1)}$ of each cluster are selected to compute Q_p and K_p:

$$Q_p = H_p^{(1)} W_p^{(Q)}, K_p = H_p^{(1)} W_p^{(K)}, \tag{4.1}$$

where $W_p^{(Q)}$, $W_p^{(K)} \in \mathbb{R}^{h \times h}$ are the trainable weights of the projection layer. By constructing Q_p and K_p with $H_p^{(1)}$ incorporating the graph structure information, the computation of attention weights fully integrates the graph structural information, thereby enabling more precise modeling of the relationships of the local neighborhood between nodes.

Next, for the value matrix V_p, we first use multi-layer perceptron (MLP) to project the original features X_p of each cluster to obtain X'_p, and then compute V_p based on X'_p:

$$X'_p = \text{MLP}(X_p), V_p = X'_p W_p^{(V)}, \tag{4.2}$$

where $W_p^{(V)} \in \mathbb{R}^{h \times h}$ is a trainable weight of the projection layer. Constructing V_p using the original features X_p instead of $H_p^{(1)}$ helps preserve the original feature information of the nodes, which contributes to generating more comprehensive node representations through the attention mechanism.

Subsequently, the hidden representation H_p of each cluster is updated via the intracluster Transformer:

$$H_p = \mathrm{FFN}(\mathrm{softmax}(\frac{Q_p K_p^T}{\sqrt{h}})V_p), \tag{4.3}$$

where FFN denotes a feed-forward neural network. In the experiments, the intracluster Transformer is set to one layer, and Wu et al. [48] demonstrates that single layer attention also has good performance. Residual connections and layer normalization [49] are applied in each attention block and FFN block. Through the above design, the hidden representation H_p of each cluster not only effectively integrates the local structural information of nodes but also retains the original features of nodes, enhancing the ability to model relationships within clusters.

Finally, the hidden representations H_p of all clusters are stacked row-wise to obtain the final node representations H_l of the local branch.

4.3.2. Global branch

The goal of the global branch is to capture the global information interactions between different clusters, complementing the local branch, which focuses only on intracluster relationships. Inspired by the work of Liu et al. [24], we introduce the generation of cluster-level features and the intercluster Transformer to efficiently model long-range dependencies.

First, we perform average pooling on the GNN output representations $H_p^{(1)}$ and the projected original features X'_p within each cluster to obtain the cluster-level features $\overline{H}_p^{(1)} \in \mathbb{R}^{1\times h}$ and $\overline{X}'_p \in \mathbb{R}^{1\times h}$:

$$\overline{H}_p^{(1)} = \mathrm{Pooling}(H_p^{(1)}), \overline{X}'_p = \mathrm{Pooling}(X'_p), \tag{4.4}$$

where Pooling$(\cdot)$ denotes the average pooling operation. Next, we stack the cluster-level features of P clusters row-wise to obtain matrices $\overline{H}_P^{(1)} \in \mathbb{R}^{P\times h}$ and $\overline{X}'_P \in \mathbb{R}^{P\times h}$, which are used in the intercluster Transformer. This cluster-level feature generation approach compresses intracluster information through pooling while retaining the global information of each cluster, providing the foundation for subsequent intercluster interactions.

Second, we construct the query, key, and value matrices (Q_g, K_g, V_g) for the intercluster Transformer:

$$Q_g = H^{(1)}W_g^{(Q)}, K_g = \overline{H}_P^{(1)}W_g^{(K)}, V_g = \overline{X}'_P W_g^{(V)}, \tag{4.5}$$

where $W_g^{(Q)}$, $W_g^{(K)}$, and $W_g^{(V)} \in \mathbb{R}^{h\times h}$ are the trainable weights of the projection layer. Similar to the intracluster Transformer, Q_g and K_g are constructed using the GNN output representations, while V_g is constructed based on the original features. Q_g contains the node features of the entire graph, integrates the complete structural information of the graph, and effectively guides the generation of intercluster attention weights. K_g and V_g are derived solely from the cluster-level features, utilizing the compressed feature representations to reduce computational complexity while preserving the essential information of each cluster. This intercluster attention mechanism is designed to efficiently capture intercluster interactions while ensuring computational efficiency, and its effectiveness has been demonstrated in similar works [24].

Finally, we update the global representations of nodes H_g through the intercluster Transformer:

$$H_g = \mathrm{FFN}(\mathrm{softmax}(\frac{Q_g K_g^T}{\sqrt{h}})V_g). \tag{4.6}$$

Similar to the local branch, the Transformer for global branch is set to a single layer, and residual connections and layer normalization are applied in each attention block and feedforward block.

Through the intercluster attention mechanism, the global branch captures intercluster global interaction information, effectively enhancing the model's ability to handle long-range dependencies.

4.4. *Feature fusion*

In order to fully utilize local and global information, we perform weighted fusion on the outputs of the two branches. We first transform the dimensions of H_l and H_g through linear transformations $f_1(\cdot)$ and $f_2(\cdot)$, respectively, as shown in $Z_l = f_1(H_l)$ and $Z_g = f_2(H_g)$, Z_l and $Z_g \in \mathbb{R}^{n\times c}$. Then, we apply a weighted fusion on Z_l and Z_g to obtain the final node representation Z:

$$Z = (1 - \alpha)Z_l + \alpha Z_g, \tag{4.7}$$

where α is a hyperparameter used to balance the contribution of local and global information.

4.5. *Inference and optimization*

The downstream task of DCAFormer is node classification, which aims to predict the class of nodes from the test set and obtain the output vector of the model through the softmax function:

$$\widehat{Y} = \text{softmax}(Z), \tag{4.8}$$

and $\widehat{Y}$ is the class prediction representation of n nodes. In order to optimize the model, we do not directly calculate the cross-entropy loss for $\widehat{Y}$. Instead, we calculate the class distribution probabilities and cross-entropy loss separately for the GNN output $H^{(2)}$, the local branch output Z_l, and the global branch output Z_g, respectively:

$$\begin{gathered}\widehat{Y}_{gn} = \text{softmax}(H^{(2)}), \widehat{Y}_l = \text{softmax}(Z_l), \widehat{Y}_g = \text{softmax}(Z_g),\\ \mathcal{L}_{gn} = \mathcal{L}(Y, \widehat{Y}_{gn}), \mathcal{L}_l = \mathcal{L}(Y, \widehat{Y}_l), \mathcal{L}_g = \mathcal{L}(Y, \widehat{Y}_g),\end{gathered} \tag{4.9}$$

where $\mathcal{L}(\cdot,\cdot)$ denotes the cross-entropy loss function [50], Y represents the ground truth label, and $\widehat{Y}_{gn}$, $\widehat{Y}_l$, and $\widehat{Y}_g$ are all predicted labels. The final loss function $\mathcal{L}$ is composed of the individual losses:

$$\mathcal{L} = \mathcal{L}_{gn} + \mathcal{L}_l + \mathcal{L}_g. \tag{4.10}$$

This approach ensures that the features from different branches are fully optimized for their respective tasks, allowing local and global information to complement each other more effectively, thereby improving the final classification performance.

4.6. *Computational complexity*

To evaluate the efficiency of DCAFormer, we conduct a detailed analysis of its computational complexity. First, in the GNN module, we employ a two-layer GCN in our experiments, whose computational complexity is $O(2eh)$. Second, the local branch consists of a Transformer and an FFN block, with a computational complexity of $O(\frac{n^2h}{P} + 2nh^2)$; while the global branch also includes a Transformer and an FFN block, with a computational complexity of $O(nhP + 2nh^2)$. Thus, the total computational complexity of DCAFormer is $O(\frac{n^2h}{P} + nhP + 4nh^2 + 2eh)$, i.e., $O(nh(\frac{n}{P} + P + 4h + 2\frac{e}{n}))$. In contrast, the computational complexity of the standard Transformer (including an FFN) is $O(n^2h + 2nh^2)$, i.e., $O(nh(n + 2h))$. Following, we need to compare the sizes of $\frac{n}{P} + P + 2h + 2\frac{e}{n}$ and n. In actual graph data, usually $2\frac{e}{n} \ll n$, which can be ignored; while the number of partitions P and hidden dimension h are usually between tens and hundreds. Thus, in small-scale graph data with a small number of nodes and edges, $\frac{n}{P} + P + 2h + 2\frac{e}{n} < n$ holds. As the number of nodes n increases, $\frac{n}{P} + P + 2h + 2\frac{e}{n} \ll n$ will gradually hold, which proves that DCAFormer has a certain efficiency in processing large-scale graph data.

5. Experiments

5.1. Experimental setup

5.1.1. Datasets

We conduct experiments on 8 widely used graph datasets, which vary in scale: 6 small-scale datasets and 2 relatively large-scale datasets, covering multiple domains including citation networks, collaboration networks, and co-purchase networks. For small-scale datasets, we use PubMed, CoraFull, Computer, Photo, CS, and Physics from PyTorch Geometric[1] [51], with 60%/20%/20% random splits for training/validation/test sets [23]. For large-scale datasets, we select Ogbn-Arxiv and Ogbn-Products from the Open Graph Benchmark (OGB)[2] [52], with dataset splits following the public settings provided by OGB. Table 1 summarizes the specific information about each dataset, where "Homo." denotes the proportion of edges connecting nodes with the same label [53].

Table 1. The detailed dataset statistics.

	PubMed	CoraFull	Computer	Photo	CS	Physics	Ogbn-Arxiv	Ogbn-Products
# Nodes	19717	19793	13752	7650	18333	34493	169343	2449029
# Edges	44338	126842	491722	238162	163788	495924	1166343	61859140
# Features	500	8710	767	745	6805	8415	128	100
# Classes	3	70	10	8	15	5	40	47
Homo.	0.79	0.57	0.80	0.85	0.83	0.91	0.63	0.81

5.1.2. Baseline

To validate the effectiveness of our proposed DCAFormer, we compare it with the following 20 baselines models, categorized into two groups: 9 GNNs and 11 GTs.

GNNs include:

- GCN [5]: GCN uses graph convolutional layers to learn node embeddings by aggregating information from neighbors.
- GAT [6]: GAT introduces attention mechanisms to weight the importance of neighboring nodes' features during aggregation.
- GraphSAGE [7]: GraphSAGE generates node embeddings by sampling and aggregating features from neighboring nodes, enabling scalable inductive learning for large graphs.

[1] `https://github.com/pyg-team/pytorch_geometric`
[2] `https://ogb.stanford.edu/docs/nodeprop/#ogbn-arxiv`

- APPNP [37]: APPNP combines graph convolution with personalized PageRank propagation to effectively capture both local and global graph structures.
- GPRGNN [54]: GPRGNN adapts PageRank propagation to learn node features and graph topology, handling both homogeneous and heterogeneous graphs.
- GraphSAINT [42]: GraphSAINT uses a sampling-based method for inductive graph learning, allowing efficient training on large-scale graphs.
- GRAND+ [43]: GRAND+ introduces a scalable GNN framework that efficiently handles large-scale graphs using a generalized forward push algorithm for propagation.
- tunedGNN [55]: tunedGNN reassesses classic GNN models and shows that with proper hyperparameter tuning, they outperform newer Graph Transformer models in node classification.
- PCNet [56]: PCNet unifies homophily and heterophily in graph data by introducing a two-fold filtering mechanism for more effective node classification.

GTs include:

- GT [44]: GT generalizes the transformer architecture to arbitrary graphs by incorporating neighborhood-based attention and Laplacian-based positional encoding to capture the graph's structural information.
- SAN [16]: SAN introduces spectral attention to enhance graph transformers, enabling better capture of graph structures through spectral properties.
- Graphormer [18]: Graphormer improves graph representation learning by incorporating structural information into transformer models for graph data.
- GraphGPS [45]: GraphGPS combines positional encoding, local message-passing, and global attention mechanisms to create scalable and powerful graph transformers.
- Exphormer [46]: Exphormer uses sparse transformers with expander graphs, achieving linear complexity and enabling scalability to large graphs.
- NAGformer [25]: NAGformer introduces a tokenized graph transformer for node classification in large graphs, improving scalability and performance.
- VCR-Graphormer [47]: VCR-Graphormer uses personalized PageRank tokenization and virtual connections to enable efficient mini-batch training for graph transformers, capturing both local and global structural information.
- CoBFormer [57]: CoBFormer mitigates the over-globalizing issue in graph transformers by using a bi-level structure and collaborative training to effectively capture both local and global information.
- Polynormer [58]: Polynormer is a polynomial-expressive graph transformer that achieves linear time complexity, enabling efficient processing of large graphs with strong expressive power
- GOAT [59]: GOAT is a global transformer designed for large-scale graphs, enabling adaptive learning of homophily and heterophily relationships between nodes.
- SGFormer [48]: SGFormer simplifies graph transformers with a single-layer attention model, achieving efficient information propagation on large graphs with linear complexity.

5.1.3. Implementation details

To comprehensively evaluate the models, we conduct 10 trials with random seeds for each model and report the mean accuracy and standard deviation. For the baseline models, we follow the hyperparameter configurations suggested in their original papers. All models are trained using the Adam optimizer [60]. For DCAFormer, the learning rate is set within {0.005, 0.01}, the hidden dimension is in the range of (64, 512), and the weight decay range between (0, 5×10^{-3}). The GNN module uses GCN with a dropout rate of 0.5, while the Transformer module has a dropout rate of 0.1. The implementation of DCAFormer is based on Python (3.8.0), PyTorch (1.10.1), and PyTorch Geometric (2.5.3). All experiments are conducted on a Linux server equipped with an A100-PCIE-40 GB GPU.

5.2. Node classification performance

5.2.1. Comparison on small-scale datasets

We compare DCAFormer with 17 different models to evaluate their accuracy. The experimental results are shown in Table 2, where the best results are highlighted in bold black, and OOM indicates out-of-memory errors.

From the results in Table 2, we can observe that:

Table 2. Comparison of all models in terms of mean accuracy ± stdev (%) on small-scale datasets.

Method	PubMed	CoraFull	Computer	Photo	CS	Physics
GCN	86.54 ± 0.12	61.76 ± 0.14	89.65 ± 0.52	92.70 ± 0.20	92.92 ± 0.12	96.18 ± 0.07
GAT	86.32 ± 0.16	64.47 ± 0.18	90.78 ± 0.13	93.87 ± 0.11	93.61 ± 0.14	96.17 ± 0.08
APPNP	88.43 ± 0.15	65.16 ± 0.28	90.18 ± 0.17	94.32 ± 0.14	94.94 ± 0.07	96.54 ± 0.07
GPRGNN	89.34 ± 0.25	67.12 ± 0.31	89.32 ± 0.29	94.49 ± 0.14	95.13 ± 0.09	96.85 ± 0.08
GraphSAINT	88.96 ± 0.16	67.85 ± 0.21	90.22 ± 0.15	91.72 ± 0.13	94.41 ± 0.09	96.43 ± 0.05
GRAND+	88.64 ± 0.09	71.37 ± 0.11	88.74 ± 0.11	94.75 ± 0.12	93.92 ± 0.08	96.47 ± 0.04
tunedGNN	89.72 ± 0.50	71.88 ± 0.55	93.25 ± 0.14	96.10 ± 0.46	**96.17 ± 0.06**	97.19 ± 0.05
PCNet	89.77 ± 0.35	70.93 ± 0.37	90.50 ± 0.15	95.02 ± 0.26	95.96 ± 0.09	97.30 ± 0.23
GT	88.79 ± 0.12	61.05 ± 0.38	91.18 ± 0.17	94.74 ± 0.13	94.64 ± 0.13	97.05 ± 0.05
Graphormer	OOM	OOM	OOM	92.74 ± 0.14	94.64 ± 0.13	OOM
SAN	88.22 ± 0.15	59.01 ± 0.34	89.93 ± 0.16	94.86 ± 0.10	94.51 ± 0.15	96.83 ± 0.18
GraphGPS	88.94 ± 0.16	55.76 ± 0.23	91.19 ± 0.54	95.06 ± 0.13	93.93 ± 0.15	97.12 ± 0.19
Exphormer	89.52 ± 0.54	69.09 ± 0.72	91.59 ± 0.31	95.27 ± 0.42	95.77 ± 0.15	97.16 ± 0.13
NAGphormer	89.70 ± 0.19	71.51 ± 0.13	91.22 ± 0.14	95.49 ± 0.11	95.75 ± 0.09	97.34 ± 0.03
CoBFormer	89.47 ± 0.24	71.51 ± 0.38	90.42 ± 0.53	94.97 ± 0.26	94.95 ± 0.42	96.85 ± 0.07
VCR-Graphormer	89.77 ± 0.15	71.67 ± 0.10	91.75 ± 0.15	95.53 ± 0.14	95.37 ± 0.04	97.34 ± 0.04
Polynormer	89.88 ± 0.18	72.02 ± 0.25	93.68 ± 0.21	**96.46 ± 0.26**	95.53 ± 0.16	97.27 ± 0.08
DCAFormer	**89.95 ± 0.13**	**72.20 ± 0.25**	**94.00 ± 0.08**	95.85 ± 0.10	95.32 ± 0.06	**97.37 ± 0.05**

(1) Compared to various GNNs, DCAFormer shows significant improvement on most datasets, which is mainly due to its dual-branch module that can simultaneously capture local and global information of the graph. Traditional GNNs are typically effective at aggregating local neighborhood information but struggle to capture long-range dependencies. DCAFormer captures long-range dependencies through intercluster attention, thereby making up for the limitations of traditional

GNNs. Additionally, compared to the recent GNN model PCNet, DCAFormer demonstrates a clear advantage. Although PCNet captures both homogeneous and heterogeneous information, it has limitations in extracting homogeneous information, which restricts its performance. In contrast, DCAFormer can effectively extract homogeneous information through both GNN module and dual-branch Transformer module, thereby enhancing its overall performance.

(2) Compared to other GTs, DCAFormer exhibits superior performance on most datasets. This improvement can be attributed to DCAFormer's optimization in the Transformer: by combining GNN output with original features, it optimizes the construction of query, key, and value matrices, providing richer graph structural information. In contrast, conventional GTs usually rely on positional or structural encodings to supplement graph structural information. This approach may be insufficient to fully capture the complex relationships between nodes, which is effectively compensated by the design of DCAFormer. Although the recently proposed polynomial Transformer model, Polynormer, has shown certain advantages, it incorporates graph structure information as polynomial coefficients, which may lead to insufficient utilization of structural information. In comparison, DCAFormer integrates graph structure information more effectively, resulting in superior performance on multiple datasets.

Overall, DCAFormer achieves state-of-the-art performance on 4 datasets and performs competitively on the Photo and CS datasets, demonstrating its effectiveness in node classification tasks.

5.2.2. Comparison on large-scale datasets

To verify the scalability of DCAFormer, we conduct experiments on two large-scale datasets and compare it with 10 baseline models. Since the graph in Ogbn-Products is too large for full-batch training on a GPU, we adopt the random partitioning method proposed by Wu et al. [48] for mini-batch training. The experimental results are shown in Table 3, where the best results are highlighted in bold black, and OOM indicates out-of-memory errors.

From the results in Table 3, we can observe that:

(1) DCAFormer is capable of running on large-scale graphs, while some baseline models fail to complete training due to memory constraints. This demonstrates the good scalability of DCAFormer for handling large graph tasks.

(2) On the Ogbn-Arxiv dataset, DCAFormer significantly outperforms all baseline models, showing strong advantages. On the Ogbn-Products dataset, DCAFormer outperforms most baseline models and approaches optimal performance. This highlights the effectiveness of DCAFormer on large-scale graphs.

(3) Compared to small-scale datasets, DCAFormer achieves more significant performance improvements on large-scale datasets. This could be because large-scale graphs provide Transformer with richer feature and structural information, allowing them to fully leverage their global modeling capabilities. Furthermore, DCAFormer's dual-branch architecture effectively integrates local and global information, further enhancing its performance.

Table 3. Comparison of all models in terms of mean accuracy ± stdev (%) on large-scale datasets.

Method	Ogbn-Arxiv	Ogbn-Products
GCN	71.74 ± 0.29	75.64 ± 0.21
GAT	72.01 ± 0.20	79.45 ± 0.59
GraphSAGE	71.49 ± 0.27	78.29 ± 0.16
GPRGNN	71.10 ± 0.12	79.76 ± 0.59
GraphGPS	70.97 ± 0.41	OOM
Exphormer	72.44 ± 0.28	OOM
NAGphormer	70.13 ± 0.55	73.55 ± 0.21
SGFormer	72.63 ± 0.13	74.16 ± 0.31
GOAT	72.41 ± 0.40	**82.00 ± 0.43**
CoBFormer	72.16 ± 0.39	78.15 ± 0.07
DCAFormer	**73.04 ± 0.07**	81.68 ± 0.49

5.3. Efficiency of DCAFormer

5.3.1. Efficiency comparison between DCAFormer and standard Transformer

To validate the efficiency of DCAFormer described in Section 4.6, we compare it to the standard Transformer (ST) on node classification task. Specifically, we evaluate classification accuracy (%), training time per epoch (s), and GPU memory consumption (GB) for both models. ST includes a multi-head attention module and an FFN module. To ensure a fair comparison, the multi-head attention in ST uses only a single layer of single-head attention and does not process the input such as position encodings. Table 4 presents the comparison results.

Table 4. Efficiency comparison between ST and DCAFormer.

Dataset	Acc. (%)		Train/Epoch (s)		Mem. (GB)	
	ST	DCAFormer	ST	DCAFormer	ST	DCAFormer
PubMed	84.72	**89.95**	0.1891	**0.0184**	7.71	**0.36**
Computer	83.39	**94.00**	0.0944	**0.0267**	3.79	**1.07**
Photo	91.29	**95.85**	0.0930	**0.0203**	1.20	**0.56**
CS	93.91	**95.32**	0.2494	**0.0323**	7.10	**1.17**
Physics	—	**97.37**	OOM	**0.0315**	OOM	**1.62**
Ogbn-Arxiv	—	**73.04**	OOM	**0.1797**	OOM	**6.17**
Ogbn-Products	—	**81.68**	OOM	**3.1751**	OOM	**11.74**

From the results in Table 4, we can observe that:

(1) DCAFormer outperforms ST in node classification accuracy on all datasets, indicating that ST is difficult to process graph data directly. ST cannot effectively utilize graph structural information, which is crucial for node classification tasks.

(2) Compared with ST, DCAFormer significantly reduces training time and GPU memory consumption. Specifically, on the PubMed and CS datasets, the training time is reduced by 90% and 87%, respectively, and the GPU memory consumption is decreased by 95% and 84%, which proves the efficiency of DCAFormer. In addition, on larger-scale datasets like Ogbn-Arxiv and Ogbn-Products, ST is unable to complete training due to memory constraints, while DCAFormer is able to run smoothly, further proving its efficiency in processing large-scale graphs, which is also consistent with the conclusion in Section 4.6.

5.3.2. Efficiency comparison between DCAFormer and other models

In addition, to further verify the efficiency of DCAFormer, we also compare it with other widely used models, including two classic GNNs (GCN, GAT) and two latest GTs (NAGphormer, CoBFormer). The results are shown in Table 5. It is worth noting that the number of attention heads for GAT is set to 8, while the other models use the default parameter configurations recommended in their respective papers.

From the results in Table 5, we can observe that:

(1) GCN, as a classic GNN model, has relatively low time and GPU memory consumption. Although GAT does not consume much training time, its attention mechanism increases memory overhead. Despite DCAFormer incorporating a GNN module, its time and memory consumption do not increase significantly, which reflects the efficiency of the model design.

(2) NAGphormer is designed as a mini-batch training model, and its tokenization mechanism effectively reduces memory consumption, but at the same time increases training time. CoBFormer employs graph partitioning and a bi-level global attention mechanism, demonstrating higher efficiency. In contrast, DCAFormer is comparable to both in terms of training time and memory consumption.

Table 5. Efficiency comparison between DCAFormer and other models.

Method	PubMed		CoraFull		Computer	
	Train/Epoch (s)	Mem. (GB)	Train/Epoch (s)	Mem. (GB)	Train/Epoch (s)	Mem. (GB)
GCN	0.0069	0.16	0.0187	1.35	0.0143	1.05
GAT	0.0098	0.31	0.0309	3.09	0.0230	2.12
NAGphormer	0.3440	0.60	17.0219	1.13	0.2096	0.42
CoBFormer	0.0257	0.17	0.0276	0.85	0.0279	0.31
DCAFormer	0.0184	0.36	0.0491	1.79	0.0267	1.07
	Photo		CS		Physics	
GCN	0.0100	0.55	0.0135	1.00	0.0128	1.52
GAT	0.0143	1.10	0.0194	1.91	0.0203	3.06
NAGphormer	0.1029	0.10	1.4450	1.70	49.3211	0.98
CoBFormer	0.0258	0.16	0.0260	0.62	0.0337	1.41
DCAFormer	0.0203	0.56	0.0323	1.17	0.0315	1.62

5.4. Ablation study and analysis

In this section, we conduct ablation experiments on the critical modules of DCAFormer, aiming to further analyze the contribution of each module to the model performance.

5.4.1. Ablation of dual-branch Transformer module

This module includes not only the local and global branches but also the construction of Q, K, and V matrices within the Transformer. To evaluate the contribution of each component in this module, we design the following model variants:

- DCAFormer-Lo: This variant retains only the local branch for node classification.
- DCAFormer-Gl: This variant retains only the global branch for node classification.
- DCAFormer-G: This variant constructs the Q, K, and V matrices in Transformer using only the GNN output.
- DCAFormer-X: This variant constructs the Q, K, and V matrices in Transformer using only the original features.

The ablation results of this module are shown in Table 6. From the results in Table 6, we can observe that:

(1) DCAFormer-Lo outperforms DCAFormer-Gl on all datasets, but neither performs as well as the full DCAFormer. This indicates that it is more crucial to capture intracluster local information than intercluster global information in node classification tasks. However, combining local and global information can further enhance classification performance, highlighting the synergistic effect of the dual-branch Transformer module.

Table 6. Performance of various variants of DCAFormer (%).

Method	PubMed	CoraFull	Computer	Photo	CS	Physics	Ogbn-Arxiv
DCAFormer-Lo	89.94	71.47	93.42	95.78	95.17	97.28	72.87
DCAFormer-Gl	82.06	64.53	93.37	95.53	95.06	97.27	70.46
DCAFormer-G	89.88	72.19	93.66	95.80	94.95	97.33	72.90
DCAFormer-X	84.04	57.64	92.50	92.81	89.62	92.63	65.74
DCAFormer	**89.95**	**72.20**	**94.00**	**95.85**	**95.32**	**97.37**	**73.04**

(2) The performance of DCAFormer-Lo is close to that of the DCAFormer, demonstrating that noly the local branch is capable of effectively representing graph data. Nevertheless, integrating global information can still improve the node classification results, suggesting that the two types of information are complementary and collectively enhance the overall performance of the model.

(3) The performance of DCAFormer-G is slightly lower than DCAFormer, indicating that using the GNN output alone to construct the Q, K, and V matrices can effectively leverage graph structural information and improve model performance. However, without the original features, the model's performance cannot reach its optimal level.

(4) The performance of DCAFormer-X is significantly lower than DCAFormer, demonstrateing that using only the original features to construct the Q, K, and V matrices cannot fully utilize graph structural information. When using Transformer for node classification, graph structural information is crucial as it can help the model capture the dependencies between nodes.

(5) DCAFormer constructs the Q and K matrices using GNN output, capturing graph structural information and effectively calculating the attention between nodes. At the same time, it uses the original features to construct the V matrix, preserving the original node information. Combining GNN-generated features and original features maximizes the information utilization in graph data, thereby improving the accuracy of node classification.

5.4.2. Ablation of GNN module

To explore the effects of different GNNs on the performance of DCAFormer, we replace the GCN in the module with other commonly used GNNs, including GAT, APPNP, GPRGNN, and GCNII, and conduct comparative experiments. All replacement models are implemented with a two-layer structure to ensure a fair comparison in terms of model complexity. The experimental results are shown in Table 7.

From the results in Table 7, we can observe that:

(1) Replacing GCN with GPRGNN and GCNII improves performance on all datasets, while using GAT and APPNP also show better performance on most datasets. This indicates that the GNN module in DCAFormer has a certain flexibility and can be compatible with other GNNs to further enhance performance.

(2) Combined with the results in Table 2, it can be found that GAT, APPNP, and GPRGNN outperform GCN on their own, so the performance of the base GNN model directly affects the overall performance of DCAFormer. In other words, the better the base GNN model performs in node classification tasks, the greater the improvement in DCAFormer's performance.

(3) Regardless of the GNNs used, DCAFormer consistently outperforms using these models alone. This demonstrates that the overall framework of DCAFormer can fully utilize the strengths of the base models, further optimizing the node classification performance.

Table 7. Performance of GNN module using different models (%).

Method	PubMed	CoraFull	Computer	Photo	CS	Physics
GCN	89.95	72.20	94.00	95.85	95.32	97.37
GAT	89.61	72.61	94.07	**96.80**	94.89	97.45
APPNP	89.83	74.40	92.76	96.51	95.40	97.41
GPRGNN	**91.43**	**74.65**	94.12	96.69	**96.45**	**97.80**
GCNII	90.18	73.57	**94.57**	96.41	95.40	97.55

5.4.3. Ablation of graph partitioning algorithms

Table 8. Performance of DCAFormer using different graph partitioning algorithms(%).

Method	PubMed	CoraFull	Computer	Photo	CS	Physics
METIS	89.95	72.20	94.00	95.85	95.32	97.37
K-means	89.76	71.46	93.82	95.75	95.31	97.43
S^3GC	**90.10**	**72.78**	**94.36**	95.85	95.35	**97.47**
CDC	90.05	72.50	94.29	**96.11**	**95.46**	97.43

This subsection explores the impact of different graph partitioning methods on the performance of DCAFormer. We replace the default graph partitioning algorithm, METIS, with other clustering methods, including a classical method, K-means [61], and two recent learning-based clustering methods, S^3GC [62] and CDC [63]. K-means is a popular clustering algorithm that partitions data into K groups by minimizing intracluster variance. S^3GC combines contrastive learning with GNNs and node features to learn clusterable representations. CDC is an efficient clustering framework for complex data that combines graph filtering and adaptive anchors to handle various data types with linear complexity. The experimental results are shown in Table 8.

From the results in Table 8, we can observe that:

(1) K-means performs slightly worse than METIS across all datasets. This may be because K-means relies solely on features for clustering and does not consider the structural information of the graph. In contrast, METIS partitions the graph based on its topology, allowing the partitioning results to better preserve adjacency relationships, thereby providing more reasonable subgraph divisions and enhancing model performance.

(2) S^3GC and CDC outperform METIS across all datasets. This may be due to their ability to adaptively learn more reasonable cluster structures rather than relying on static graph partitioning rules. A well-structured clustering strategy is crucial for DCAFormer's information modeling, making learning-based partitioning methods significantly improve the overall model performance.

5.5. *Parameter study*

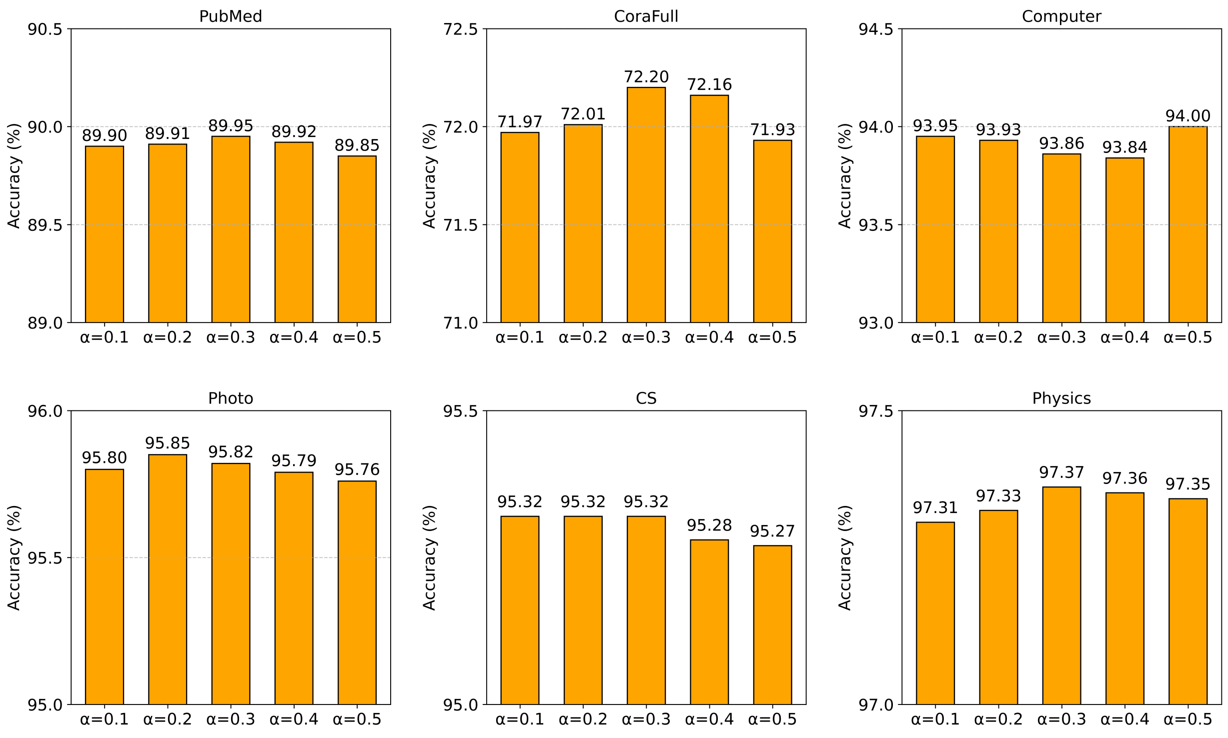

Figure 2. Performance of DCAFormer for different α.

In this section, we analyze two critical hyperparameters in DCAFormer: The weight coefficient α used to fuse the outputs of the two branches (see Eq (4.7)) and the number of graph partitions P.

5.5.1. Analysis of weight coefficient α

In this subsection, we discuss the impact of the weight coefficient α on the model performance. Based on the results of the ablation study in Table 6, the local branch outputs outperform those of the global branch. Therefore, we empirically select a range of α values {0.1, 0.2, 0.3, 0.4, 0.5} and observe their effect on the model's performance. The experimental results are shown in Figure 2.

As can be seen in Figure 2, with the change of α, the overall accuracy changes more smoothly, and the fluctuation of each dataset is relatively small. This demonstrates that the model exhibits good robustness to the choice of α. Most datasets achieve their best performance when α is between 0.2 and 0.3, indicating that an appropriate balance between local and global information fusion is crucial for improving model performance.

5.5.2. Analysis of number of graph partitions P

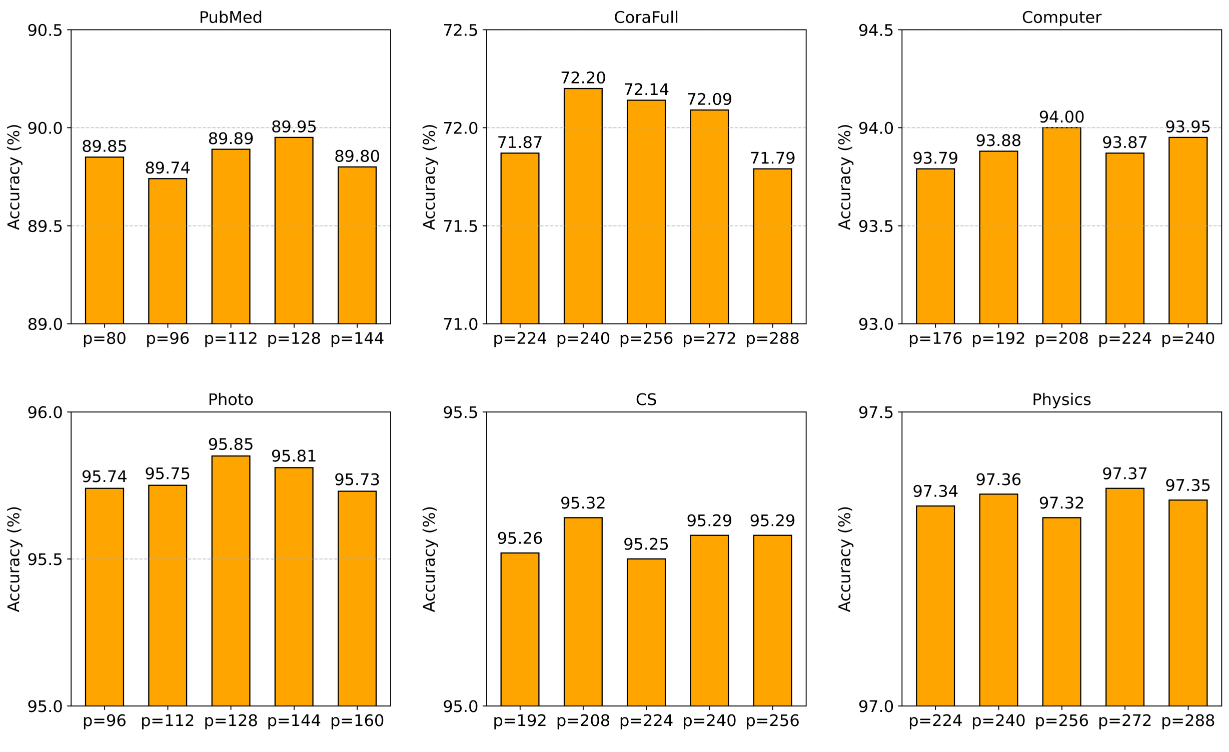

Figure 3. Performance of DCAFormer for different P.

In this subsection, we explore the impact of different graph partition numbers on the model performance and report the results in Figure 3. In theory, if P is too small, each cluster will contain too many nodes, increasing the computational burden; if P is too large, the partition will be too fine, reducing partition quality and potentially introducing noise. Therefore, we choose P close to the square root of the number of nodes or its multiple, i.e., close to $\sqrt{n}$ or $m \times \sqrt{n}$, where n represents the number of nodes and m is a constant. This partition results in clusters with a more appropriate node size, effectively reducing the computational burden while ensuring partition quality.

From Figure 3, it can be observed that on the Physics dataset, variations in P have minimal impact on model performance, suggesting a lower dependency on graph partitioning, whereas on the other datasets, the optimal P is crucial for achieving peak performance. Nonetheless, even without choosing the optimal P, DCAFormer generally outperforms other models, validating its robustness under different graph partition numbers.

6. Conclusions

In this paper, we propose a dual-branch graph Transformer model, DCAFormer, for the node classification task. The model divides the graph into subgraphs by graph partitioning and utilizes the dual-branch Transformer module to extract local and global information, respectively, thereby effectively reducing the computational complexity of Transformer and reducing the noise interference caused by long-distance irrelevant nodes. Additionally, we propose a hybrid feature approach to optimize the construction of the query, key, and value matrices for Transformers of different branches in order to better adapt to their specific needs. Experiments on several datasets of different sizes show that DCAFormer outperforms existing representative GNNs and GTs in terms of performance.

In future work, we plan to further optimize DCAFormer to improve its efficiency on larger-scale graph data and generalize it to more graph tasks, such as graph classification. In addition, there are still some areas where the model can be improved. First, we will explore adaptive or learning-based partitioning methods to improve the quality of graph partitioning and enhance the flexibility and adaptability of the model. Second, how to effectively integrate the information of heterogeneous graphs is also one of the improvement directions, which will help us better cope with the complex relationships between different types of nodes and edges, thereby expanding the scope of application of the model. Furthermore, designing a more effective information interaction mechanism between the two branches could help improve prediction accuracy and robustness.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work is supported by the Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX23_3911), Macao Science and Technology Development Fund (0036/2023/RIA1, 0037/2024/RIA1) and Jiangsu Kaite Auto Parts Co., Ltd. (MUST Code: 1330).

Conflict of interest

The authors declare there is no conflicts of interest.

References

1. W. Q. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. L. Tang, et al., Graph neural networks for social recommendation, in *Proceedings of the 2019 World Wide Web Conference*, (2019), 417–426. https://doi.org/10.1145/3308558.3313488

2. N. Xu, P. H. Wang, L. Chen, J. Tao, J. Z. Zhao, MR-GNN: Multi-resolution and dual graph neural network for predicting structured entity interactions, in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, (2019), 3968–3974. https://doi.org/10.24963/ijcai.2019/551

3. P. Y. Zhang, Y. C. Yan, X. Zhang, L. C. Li, S. Z. Wang, F. R. Huang, et al., TransGNN: Harnessing the collaborative power of transformers and graph neural networks for recommender systems, in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, (2024), 1285–1295. https://doi.org/10.1145/3626772.3657721

4. S. X. Ji, S. R. Pan, E. Cambria, P. Marttinen, S. Y. Philip, A survey on knowledge graphs: Representation, acquisition, and applications, *IEEE Trans. Neural Networks Learn. Syst.*, **33** (2021), 494–514. https://doi.org/10.1109/TNNLS.2021.3070843

5. T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, preprint, arXiv:1609.02907. https://doi.org/10.48550/arXiv.1609.02907

6. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, preprint, arXiv:1710.10903. https://doi.org/10.48550/arXiv.1710.10903

7. W. Hamilton, Z. T. Ying, J. Leskovec, Inductive representation learning on large graphs, in *Advances in Neural Information Processing Systems* (eds. I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett), Curran Associates, Inc., **30** (2017).

8. X. T. Yu, Z. M. Liu, Y. Fang, X. M. Zhang, Learning to count isomorphisms with graph neural networks, in *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, **37** (2023), 4845–4853. https://doi.org/10.1609/aaai.v37i4.25610

9. Q. M. Li, Z. C. Han, X. M. Wu, Deeper insights into graph convolutional networks for semi-supervised learning, in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, **32** (2018), 3538–3545. https://doi.org/10.1609/aaai.v32i1.11604

10. J. Topping, F. Di Giovanni, B. P. Chamberlain, X. W. Dong, M. M. Bronstein, Understanding over-squashing and bottlenecks on graphs via curvature, preprint, arXiv:2111.14522. https://doi.org/10.48550/arXiv.2111.14522

11. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, et al., Attention is all you need, in *Advances in Neural Information Processing Systems* (eds. I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett), Curran Associates, Inc., **30** (2017).

12. J. Devlin, M. W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, **1** (2019), 4171–4186. https://doi.org/10.18653/v1/N19-1423

13. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. H. Zhai, T. Unterthiner, et al., An image is worth 16×16 words: Transformers for image recognition at scale, in *Proceedings of the 9th International Conference on Learning Representations*, 2021.

14. Y. H. Liu, M. Ott, N. Goyal, J. F. Du, M. Joshi, D. Q. Chen, et al., RoBERTa: A robustly optimized bert pretraining approach, preprint, arXiv:1907.11692. https://doi.org/10.48550/arXiv.1907.11692

15. Z. Liu, Y. T. Lin, Y. Cao, H. Hu, Y. X. Wei, Z. Zhang, et al., Swin transformer: Hierarchical vision transformer using shifted windows, in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, (2021), 10012–10022. https://doi.org/10.1109/ICCV48922.2021.00986

16. D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, P. Tossou, Rethinking graph transformers with spectral attention, in *Advances in Neural Information Processing Systems* (eds. M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang and J. Wortman Vaughan), Curran Associates, Inc., **34** (2021), 21618–21629.

17. Y. Ye, S. H. Ji, Sparse graph attention networks, *IEEE Trans. Knowl. Data Eng.*, **35** (2023), 905–916. https://doi.org/10.1109/TKDE.2021.3072345

18. C. X. Ying, T. L. Cai, S. J. Luo, S. X. Zheng, G. L. Ke, D. He, et al., Do transformers really perform bad for graph representation?, in *Advances in Neural Information Processing Systems* (eds. M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang and J. Wortman Vaughan), Curran Associates, Inc., **34** (2021), 28877–28888.

19. E. X. Min, R. F. Chen, Y. T. Bian, T. Y. Xu, K. F. Zhao, W. B. Huang, et al., Transformer for graphs: An overview from architecture perspective, preprint, arXiv:2202.08455. https://doi.org/10.48550/arXiv.2202.08455

20. A. Shehzad, F. Xia, S. Abid, C. Y. Peng, S. Yu, D. Y. Zhang, et al., Graph Transformers: A survey, preprint, arXiv:2407.09777. https://doi.org/10.48550/arXiv.2407.09777

21. W. R. Kuang, W. Zhen, Y. L. Li, Z. W. Wei, B. L. Ding, *Coarformer: Transformer for large graph via graph coarsening*, 2021. Available from: https://openreview.net/forum?id=fkjO_FKVzw

22. J. N. Zhao, C. Z. Li, Q. L. Wen, Y. Q. Wang, Y. M. Liu, H. Sun, et al., Gophormer: Ego-graph transformer for node classification, preprint, arXiv:2110.13094. https://doi.org/10.48550/arXiv.2110.13094

23. Z. X. Zhang, Q. Liu, Q. Y. Hu, C. K. Lee, Hierarchical graph transformer with adaptive node sampling, in *Advances in Neural Information Processing Systems* (eds. S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho and A. Oh), Curran Associates, Inc., **35** (2022), 21171–21183.

24. C. Liu, Y. B. Zhan, X. Q. Ma, L. Ding, D. P. Tao, J. Wu, et al., Gapformer: Graph transformer with graph pooling for node classification, in *Proceedings of the 32nd International Joint Conference on Artificial Intelligence*, (2023), 2196–2205. https://doi.org/10.24963/ijcai.2023/244

25. J. S. Chen, K. Y. Gao, G. C. Li, K. He, NAGphormer: A tokenized graph transformer for node classification in large graphs, preprint, arXiv:2206.04910. https://doi.org/10.48550/arXiv.2206.04910

26. K. H. Zhang, D. X. Li, W. H. Luo, W. Q. Ren, Dual attention-in-attention model for joint rain streak and raindrop removal, *IEEE Trans. Image Process.*, **30** (2021), 7608–7619. https://doi.org/10.1109/TIP.2021.3108019

27. K. H. Zhang, W. H. Luo, Y. J. Yu, W. Q. Ren, F. Zhao, C. S. Li, et al., Beyond monocular deraining: Parallel stereo deraining network via semantic prior, *Int. J. Comput. Vision*, **130** (2022), 1754–1769. https://doi.org/10.1007/s11263-022-01620-w

28. K. H. Zhang, W. Q. Ren, W. H. Luo, W. S. Lai, B. Stenger, M. H. Yang, et al., Deep image deblurring: A survey, *Int. J. Comput. Vision*, **130** (2022), 2103–2130. https://doi.org/10.1007/s11263-022-01633-5

29. B. Y. Zhou, Q. Cui, X. S. Wei, Z. M. Chen, BBN: Bilateral-branch network with cumulative learning for long-tailed visual recognition, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, (2020), 9716–9725. https://doi.org/10.1109/CVPR42600.2020.00974

30. T. Wang, Y. Li, B. Y. Kang, J. N. Li, J. H. Liew, S. Tang, et al., The devil is in classification: A simple framework for long-tail instance segmentation, in *Computer Vision–ECCV 2020: 16th European Conference*, **12359** (2020), 728–744. https://doi.org/10.1007/978-3-030-58568-6_43

31. H. Guo, S. Wang, Long-tailed multi-label visual recognition by collaborative training on uniform and re-balanced samplings, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (2021), 15084–15093. https://doi.org/10.1109/CVPR46437.2021.01484

32. Y. Zhou, S. Y. Sun, C. Zhang, Y. K. Li, W. L. Ouyang, Exploring the hierarchy in relation labels for scene graph generation, preprint, arXiv:2009.05834. https://doi.org/10.48550/arXiv.2009.05834

33. C. F. Zheng, L. L. Gao, X. Y. Lyu, P. P. Zeng, A. El Saddik, H. T. Shen, Dual-branch hybrid learning network for unbiased scene graph generation, *IEEE Trans. Circuits Syst. Video Technol.*, **34** (2024), 1743–1756. https://doi.org/10.1109/TCSVT.2023.3297842

34. G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM J. Sci. Comput.*, **20** (1998), 359–392. https://doi.org/10.1137/S1064827595287997

35. Y. Rong, Y. T. Bian, T. Y. Xu, W. Y. Xie, Y. Wei, W. B. Huang, et al., Self-supervised graph transformer on large-scale molecular data, in *Advances in Neural Information Processing Systems* (eds. H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan and H. Lin), **33** (2020), 12559–12571.

36. D. X. Chen, L. O'Bray, K. Borgwardt, Structure-aware transformer for graph representation learning, in *Proceedings of the 39th International Conference on Machine Learning*, PMLR, **162** (2022), 3469–3489.

37. J. Klicpera, A. Bojchevski, S. Günnemann, Predict then propagate: Graph neural networks meet personalized pagerank, preprint, arXiv:1810.05997. https://doi.org/10.48550/arXiv.1810.05997

38. L. Page, S. Brin, R. Motwani, T. Winograd, *The PageRank Citation Ranking: Bringing Order to the Web.*, 1998. Available from: http://ilpubs.stanford.edu:8090/422/

39. M. Chen, Z. W. Wei, Z. F. Huang, B. L. Ding, Y. L. Li, Simple and deep graph convolutional networks, in *Proceedings of the 37th International Conference on Machine Learning*, **119** (2020), 1725–1735. https://doi.org/10.48550/arXiv.2007.02133

40. K. M. He, X. Y. Zhang, S. Q. Ren, J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (2016), 770–778. https://doi.org/10.1109/CVPR.2016.90

41. M. Hardt, T. Y. Ma, Identity matters in deep learning, preprint, arXiv:1611.04231. https://doi.org/10.48550/arXiv.1611.04231

42. H. Q. Zeng, H. K. Zhou, A. Srivastava, R. Kannan, V. Prasanna, Graphsaint: Graph sampling based inductive learning method, preprint, arXiv:1907.04931. https://doi.org/10.48550/arXiv.1907.04931

43. W. Z. Feng, Y. X. Dong, T. L. Huang, Z. Q. Yin, X. Cheng, E. Kharlamov, et al., Grand+: Scalable graph random neural networks, in *Proceedings of the 31st ACM Web Conference*, (2022), 3248–3258. https://doi.org/10.1145/3485447.3512044

44. V. P. Dwivedi, X. Bresson, A generalization of transformer networks to graphs, preprint, arXiv:2012.09699.

45. L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, D. Beaini, Recipe for a general, powerful, scalable graph transformer, in *Proceedings of the 36th Annual Conference on Neural Information Processing Systems*, **35** (2022), 14501–14515. https://doi.org/10.48550/arXiv.2205.12454

46. H. Shirzad, A. Velingker, B. Venkatachalam, D. J. Sutherland, A. K. Sinop, Exphormer: Sparse transformers for graphs, in *Proceedings of the 40th International Conference on Machine Learning*, **202** (2023), 31613–31632.

47. D. Q. Fu, Z. G. Hua, Y. Xie, J. Fang, S. Zhang, K. Sancak, et al., VCR-Graphormer: A mini-batch graph transformer via virtual connections, in *Proceedings of the 12th International Conference on Learning Representations*, 2024.

48. Q. T. Wu, W. T. Zhao, C. X. Yang, H. R. Zhang, F. Nie, H. T. Jiang, et al., SGFormer: Simplifying and empowering transformers for large-graph representations, in *Advances in Neural Information Processing Systems* (eds. A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt and S. Levine), Curran Associates, Inc., **36** (2023), 64753–64773.

49. J. L. Ba, J. R. Kiros, G. E. Hinton, Layer normalization, preprint, arXiv:1607.06450. https://doi.org/10.48550/arXiv.1607.06450

50. P. T. De Boer, D. P. Kroese, S. Mannor, R. Y. Rubinstein, A tutorial on the cross-entropy method, *Ann. Oper. Res.*, **134** (2005), 19–67. https://doi.org/10.1007/s10479-005-5724-z

51. J. Zhu, Y. J. Yan, L. X. Zhao, M. Heimann, L. Akoglu, D. Koutra, Beyond homophily in graph neural networks: Current limitations and effective designs, in *Advances in Neural Information Processing Systems* (eds. H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan and H. Lin), **33** (2020), 7793–7804.

52. W. H. Hu, M. Fey, M. Zitnik, Y. X. Dong, H. Y. Ren, B. W. Liu, et al., Open graph benchmark: Datasets for machine learning on graphs, in *Advances in Neural Information Processing Systems* (eds. H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan and H. Lin), **33** (2020), 22118–22133. https://doi.org/10.48550/arXiv.2005.00687

53. M. Fey, J. E. Lenssen, Fast graph representation learning with pytorch geometric, preprint, arXiv:1903.02428. https://doi.org/10.48550/arXiv.1903.02428

54. E. Chien, J. H. Peng, P. Li, O. Milenkovic, Adaptive universal generalized pagerank graph neural network, preprint, arXiv:2006.07988. https://doi.org/10.48550/arXiv.2006.07988

55. Y. K. Luo, L. Shi, X. M. Wu, Classic gnns are strong baselines: Reassessing gnns for node classification, preprint, arXiv:2406.08993. https://doi.org/10.48550/arXiv.2406.08993

56. B. H. Li, E. L. Pan, Z. Kang, PC-Conv: Unifying homophily and heterophily with two-fold filtering, in *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI, **38** (2024), 13437–13445. https://doi.org/10.1609/aaai.v38i12.29246

57. Y. J. Xing, X. Wang, Y. B. Li, H. Huang, C. Shi, Less is more: On the over-globalizing problem in graph transformers, preprint, arXiv:2405.01102. https://doi.org/10.48550/arXiv.2405.01102

58. C. H. Deng, Z. C. Yue, Z. R. Zhang, Polynormer: Polynomial-expressive graph transformer in linear time, preprint, arXiv:2403.01232. https://doi.org/10.48550/arXiv.2403.01232

59. K. Z. Kong, J. H. Chen, J. Kirchenbauer, R. K. Ni, C. B. Y. Bruss, T. Goldstein, GOAT: A global transformer on large-scale graphs, in *Proceedings of the 40st International Conference on Machine Learning*, **202** (2023), 17375–17390.

60. D. P. Kingma, J. L. Ba, Adam: A method for stochastic optimization, preprint, arXiv:1412.6980. https://doi.org/10.48550/arXiv.1412.6980

61. J. MacQueen, Some methods for classification and analysis of multivariate observations, in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability/University of California Press*, **1** (1967), 281–297.

62. F. Devvrit, A. Sinha, I. Dhillon, P. Jain, S^{3}GC: Scalable self-supervised graph clustering, in *Advances in Neural Information Processing Systems* (eds. S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho and A. Oh), **35** (2022), 3248–3261.

63. Z. Kang, X. T. Xie, B. H. Li, E. L. Pan, CDC: A simple framework for complex data clustering, *IEEE Trans. Neural Networks Learn. Syst.*, (2024), 1–12. https://doi.org/10.1109/TNNLS.2024.3473618