ScaleGNN: Towards Scalable Graph Neural Networks via Adaptive High-order Neighboring Feature Fusion

Xiang Li
Ocean University of China
Qingdao, China
lixiang1202@stu.ouc.edu.cn

Haobing Liu
Ocean University of China
Qingdao, China
haobingliu@ouc.edu.cn

Jianpeng Qi
Ocean University of China
Qingdao, China
qijianpeng@ouc.edu.cn

Yanwei Yu*
Ocean University of China
Qingdao, China
yuyanwei@ouc.edu.cn

Yuan Cao
Ocean University of China
Qingdao, China
cy8661@ouc.edu.cn

Guoqing Chao
Harbin Institute of Technology
Harbin, China
guoqingchao@hit.edu.cn

Abstract

Graph Neural Networks (GNNs) have demonstrated strong performance across various graph-based tasks by effectively capturing relational information between nodes. These models rely on iterative message passing to propagate node features, enabling nodes to aggregate information from their neighbors. Recent research has significantly improved the message-passing mechanism, enhancing GNN scalability on large-scale graphs. However, GNNs still face two main challenges: over-smoothing, where excessive message passing results in indistinguishable node representations, especially in deep networks incorporating high-order neighbors; and scalability issues, as traditional architectures suffer from high model complexity and increased inference time due to redundant information aggregation. This paper proposes a novel framework for large-scale graphs named ScaleGNN that simultaneously addresses both challenges by adaptively fusing multi-level graph features. We first construct neighbor matrices for each order, learning their relative information through trainable weights through an adaptive high-order feature fusion module. This allows the model to selectively emphasize informative high-order neighbors while reducing unnecessary computational costs. Additionally, we introduce a High-order redundant feature masking mechanism based on a Local Contribution Score (LCS), which enables the model to retain only the most relevant neighbors at each order, preventing redundant information propagation. Furthermore, low-order enhanced feature aggregation adaptively integrates low-order and high-order features based on task relevance, ensuring effective capture of both local and global structural information without excessive complexity. Extensive experiments on real-world datasets demonstrate that our approach consistently outperforms state-of-the-art GNN models in both accuracy and computational efficiency.

*Yanwei Yu is the corresponding author.

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

Keywords

Scalable Graph Neural Network, Over-smoothing, Large-scale graph

ACM Reference Format:
Xiang Li, Haobing Liu, Jianpeng Qi, Yanwei Yu, Yuan Cao, and Guoqing Chao. 2018. ScaleGNN: Towards Scalable Graph Neural Networks via Adaptive High-order Neighboring Feature Fusion. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Graph Neural Networks (GNNs) have demonstrated remarkable success in various graph-based learning tasks, such as node classification, link prediction, and recommender systems [3, 19, 35, 36, 44]. By leveraging the inherent structure of graphs, GNNs propagate information through iterative message passing, allowing nodes to aggregate features from their neighbors. This mechanism enables GNNs to effectively capture complex relationships and structural dependencies. However, despite their success, existing GNNs face two fundamental challenges when applied to large-scale graphs [39]: over-smoothing and scalability issues.

Over-smoothing occurs as the number of message-passing layers increases, causing node representations to become overly similar and lose their discriminative power. This issue is particularly severe when incorporating high-order neighbors [8, 10, 15, 37, 40], as their contributions may become redundant or even introduce noise, leading to performance degradation. Several existing techniques, such as residual connections, skip connections, and decoupling propagation from feature transformation, have been proposed to alleviate over-smoothing. However, these methods either introduce additional computational complexity or fail to effectively balance local and global information. Scalability [1, 4, 18, 20, 30, 42, 43] is another major challenge, as many traditional GNN architectures struggle with high memory consumption and computational costs when applied to large-scale graphs. The primary reason is the exponential growth in the number of high-order neighbors, which leads to excessive information aggregation and redundant computations. Some recent works address scalability issues by adopting pre-computation-based techniques, mini-batch training, or sampling strategies. While these approaches improve efficiency, they often sacrifice performance due to incomplete neighborhood information.

To overcome these challenges, we propose a novel framework towards scalable graph neural network via adapt high-order neighboring feature fusion named ScaleGNN, that fuses multi-hop graph features to address the over-smoothing issue. Specifically, we introduce a learnable mechanism to construct and refine multi-hop neighbor matrices, allowing the model to adjust the relative importance of different neighborhood levels through trainable weight parameters. By regulating these weights, our approach selectively emphasizes the most informative high-order neighbors while minimizing the influence of less useful ones. Additionally, we introduce a task-aware feature fusion mechanism that adaptively integrates low-order and high-order features based on their relevance to the specific task. This ensures that our model effectively captures both local and global information without suffering from over-smoothing. Furthermore, we incorporate a selective neighbor filtering strategy based on a Local Contribution Score (LCS), which quantifies the relevance of high-order neighbors to the target node, ensuring that only the most useful neighbors contribute to feature aggregation. To validate the effectiveness of our approach, we conduct extensive experiments on multiple real-world graph datasets. The results demonstrate that our method consistently outperforms state-of-the-art GNN models in terms of both accuracy and efficiency, providing a scalable and robust solution to the over-smoothing problem in deep graph learning.

The main contributions of our work are as follows:

- We introduce a trainable mechanism to construct and refine multi-hop neighbor matrices, allowing the model to selectively retain informative high-order neighbors while suppressing redundant or irrelevant ones. This ensures an effective balance between local and global information aggregation.
- We propose a dynamic feature fusion mechanism that adaptively combines low-order and high-order features based on their relevance to the specific learning task, which allows the model to capture both local structure and long-range dependencies while mitigating over-smoothing.
- We incorporate a selective high-order neighbor filtering strategy based on a local contribution score (LCS) to enhance computational efficiency, reducing unnecessary computational overhead by focusing only on the most valuable high-order neighbors
- Extensive experiments on multiple real-world graph datasets demonstrate that our method consistently outperforms state-of-the-art GNN models in terms of both accuracy and efficiency, highlighting its scalability and practical applicability.

2 Related Work

2.1 Graph Neural Network

Graph Neural Networks (GNNs) are specialized neural networks for graph deep learning. GCN [16], an early model, performs message passing in each layer to aggregate 1-hop neighbor information, enriching vertex semantics. Stacking K GCN layers allows GCNs to integrate information from neighbors within K hops. GAT [27] employs attention mechanisms to prioritize important neighbors during message passing. GraphSAGE [9] enhances scalability for large graphs with neighbor sampling for mini-batch training. SGC [29] simplifies multilayer GNNs by removing non-linearities and weight matrices between layers, leading to faster processing without sacrificing accuracy. APPNP [7] introduces an improved propagation scheme inspired by personalized PageRank [23], ensuring linear complexity via PageRank approximation. S^2GC [45] uses a modified Markov Diffusion Kernel for propagation, balancing global and local vertex contexts. However, these GNNs show limitations in many real-world graphs due to challenges like heterophily [17, 24, 46] and heterogeneity. In this paper, we focus on the challenge of heterogeneity, specifically that the aforementioned GNNs are tailored for homogeneous graphs and overlook the characteristics of heterogeneous graphs, where vertices and edges can be of different types and contribute differently. HGNNs have been developed to address this by accommodating the properties of heterogeneous graphs.

2.2 Heterogeneous Graph Neural Networks

HGNNs can be classified into two categories: relation-wise and representation-wise styles. Relation-wise HGNNs identify neighbors based on different relations (meta-paths) and aggregate this information for vertex representations. HAN [28] uses hand-designed meta-paths and semantic attention to enhance GAT for neighbor aggregation. MAGNN [6] involves all vertices within a meta-path for better semantics. HetGNN [38] samples neighbors via random walks and aggregates same-type neighbors with Bi-LSTMs. Representation-wise HGNNs perform multiple iterations of message passing and vertex updates. R-GCN [26] extends GCN by using distinct transformation matrices for each 1-hop relation. RSHN [47] uses a coarsened line graph for global edge-type embeddings. HetSANN [11] extends GAT with type-specific attention values. HGT [13] applies heterogeneous mutual attention with type-specific parameters. Simple-HGN [21] extends GAT by incorporating vertex features and learnable edge-type embeddings. HINormer [22] utilizes a Graph Transformer with a broader aggregation mechanism. These HGNNs are end-to-end models, but can be impractical for large graphs due to resource-intensive message passing during training.

2.3 Scalable Graph Neural Networks

To address the efficiency challenges of HGNNs on large graphs, strategies like sampling [14, 33] and pre-computation-based methods have emerged. This paper focuses on pre-computation-based methods, which have shown superior performance and efficiency across datasets. One approach is building deep GNNs by decoupling feature propagation and non-linear transformation, enabling feature propagation without model parameter training. SGC [29]s eliminates nonlinearity between consecutive graph convolutional layers, improving scalability and efficiency. Following this, works like SIGN, S^2GC, GBP, and GAMLP have been proposed to further enhance SGC's performance while maintaining scalability. Another direction involves pre-computation-based scalable GNNs. NARS [34] mitigates high information loss by extracting multiple subgraphs based on sampled relation subsets and pre-computing on each subgraph rather than the entire graph. SeHGNN [32] exemplifies the relation-wise style by separately collecting neighbor information for target vertices across different relations within K hops, reducing information loss but compromising efficiency. As the number of relations grows with K hops, SeHGNN becomes

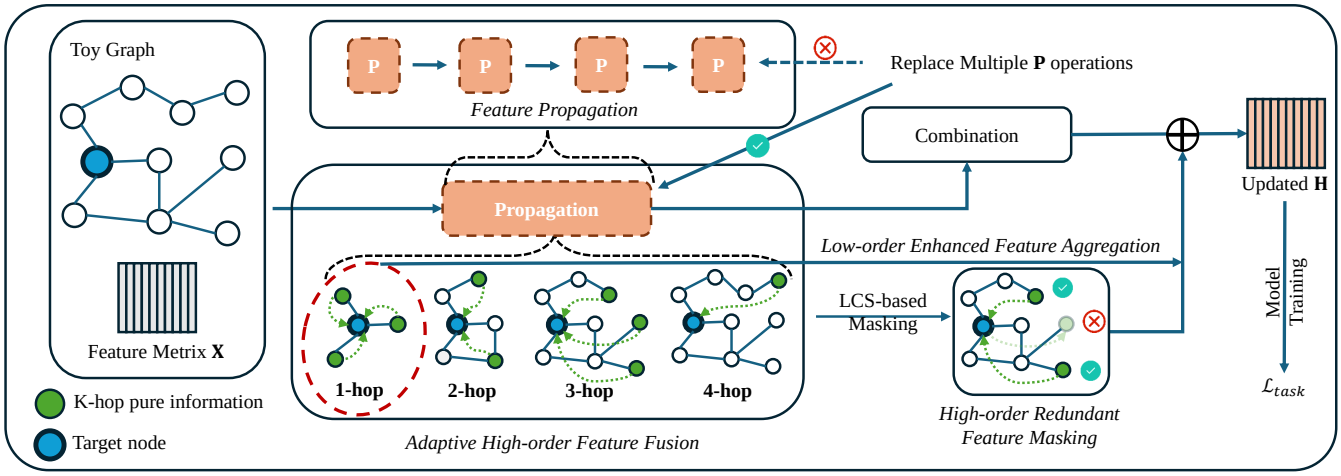


Figure 1: The overview of the proposed ScaleGNN.

inefficient, especially for $K > 2$ on large graphs. RpHGNN [12] combines the low information loss of the relation-wise style with the efficiency of the representation-wise style.

3 Preliminary

Generally, we consider an undirected large-scale graph as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\, \mathcal{X}\}$, where $\mathcal{V}$ is the collection of nodes, $\mathcal{E}$ is the collection of edges between the nodes, and $\mathcal{X} \in \mathbb{R}^{|\mathcal{V}|\times f}$ is the collection of node attributes, f is the size of node feature vector. C is the number of node classes, and $\mathbf{A}$ denotes the adjacency matrix of $\mathcal{G}$. $\mathbf{D} = diag(d_1, d_2, \ldots, d_{|\mathcal{V}|}) \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ represents the degree matrix of $\mathbf{A}$, where $d_i = \sum_{v_j \in \mathcal{V}} \mathbf{A}_{ij}$ denotes the degree of node v_i. Given the labeled node set $\mathcal{V}_l$, our goal is to predict the labels for nodes in unlabeled set $\mathcal{V}_u = \mathcal{V} - \mathcal{V}_l$ with the supervision of $\mathcal{V}_l$.

4 Methodology

In this section, we introduce ScaleGNN, an efficient end-to-end framework designed to address the scalability and performance challenges of GNNs. Our proposed ScaleGNN integrates several key innovations to handle issues such as over-smoothing, computational inefficiency, and redundant high-order information. The framework consists of three core components: (1) Adaptive High-order Feature Fusion, (2) Low-order Enhanced Feature Aggregation, and (3) High-order Redundant Feature Masking. These components are designed to work in synergy, allowing ScaleGNN to efficiently process large-scale graphs while maintaining effective information flow across different neighborhood orders. The outline of our ScaleGNN is shown in Fig. 1.

4.1 Adaptive High-order Feature Fusion

Traditional GNNs are based on message passing through fixed adjacency matrices, where the adjacency structure remains constant throughout the layers. This approach can lead to over-smoothing, where node representations converge and lose their distinctiveness as the network depth increases. Furthermore, as nodes aggregate information from higher-order neighbors, the feature similarity across nodes can grow excessively, reducing the expressiveness of the model.

To mitigate these issues, we propose a method for constructing distinct adjacency matrices for each neighborhood order. This allows us to isolate the relationships specific to each order, ensuring that each neighborhood level contributes uniquely to the node representation. We define the *pure* i-th order adjacency matrix as the difference between the $(i-1)$-th and i-th powers of the adjacency matrix:

$$\mathbf{A}_i = \mathbf{A}^i - \mathbf{A}^{i-1}, \quad i = 1, 2, 3, \ldots, K, \tag{1}$$

where $\mathbf{A}^i$ is the n-th power of the adjacency matrix, which captures the connections up to n hops, and $\mathbf{A}_i$ isolates the new relationships emerging specifically at the i-th order. Specifically, $\mathbf{A}_1$ is the adjacency matrix of the graph, $\mathbf{A}_0$ is the unit matrix.

Incorporating information from multiple orders can provide a more comprehensive view of the graph structure. However, the challenge lies in integrating these multiple orders in a way that preserves their distinct contributions. To address this, we introduce a learnable weight vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_K]$, where each α_i represents the importance of the i-th order adjacency matrix in the final aggregation:

$$\tilde{\mathbf{A}} = \sum_{i=1}^{K} \alpha_i \mathbf{A}_i, \quad \sum_{i=1}^{K} \alpha_i = 1. \tag{2}$$

These weights are learned through back-propagation, ensuring that the model dynamically adjusts the contribution of each neighborhood order. The use of the softmax function to optimize these weights ensures that the contributions across the different orders remain balanced. Finally, we obtain the adaptive high-order fused feature matrix $\mathbf{H} \in \mathbb{R}^{n\times d}$ as follows:

$$\mathbf{H} = \sigma(\tilde{\mathbf{A}} \cdot \mathcal{X} \cdot \mathbf{W}), \tag{3}$$

where d is the hidden embedding dimension, $\mathcal{X} \in \mathbb{R}^{n\times f}$ is the input feature matrix, and $\mathbf{W}$ is learnable weight matrix for the high-order features.

We use the model obtained here as the base version named ScaleGNN$_\text{b}$, which is relatively simple and efficient.

4.2 Low-order Enhanced Feature Aggregation

In many GNN models, the integration of low-order (local) and high-order (global) features is done in a single, often static, step. However, this can lead to high-order features dominating over low-order ones, which might diminish the model's ability to preserve fine-grained, local information. To address this, we propose to explicitly separate the low-order and high-order features before fusion.

First, we obtain the low-order feature matrix $\mathbf{H}_{\text{low}}$ and the high-order feature matrix $\mathbf{H}_{\text{high}}$:

$$\mathbf{H}_{\text{low}} = \sigma(\mathbf{A} \cdot \mathcal{X} \cdot \mathbf{W}_{\text{low}}), \quad \mathbf{H}_{\text{high}} = \sigma(\tilde{\mathbf{A}} \cdot \mathcal{X} \cdot \mathbf{W}_{\text{high}}), \tag{4}$$

where $\mathbf{W}_{\text{low}} \in \mathbb{R}^{f \times d}$ and $\mathbf{W}_{\text{high}} \in \mathbb{R}^{f \times d}$ are both learnable weight matrices for the low-order and high-order features, respectively. The activation function $\sigma(\cdot)$ introduces non-linearity to the transformation.

To ensure a balanced integration of local and global information, we introduce a learnable balancing factor β, which allows for dynamic adjustment of the contribution from each feature type. The final node representation $\mathbf{H}$ is computed as a weighted sum of the low-order and high-order features:

$$\mathbf{H} = \beta \cdot \mathbf{H}_{\text{low}} + (1 - \beta) \cdot \mathbf{H}_{\text{high}}, \tag{5}$$

where β is optimized during training to ensure that the model can adaptively balance the trade-off between local and global information, depending on the specific task and graph structure.

This separation and fusion process enhances the model's ability to capture both fine-grained, local details and broader, global patterns in the graph, ultimately leading to more expressive and robust node embeddings.

4.3 High-order Redundant Feature Masking

Higher-order neighbors introduce diverse, yet potentially redundant, information to the GNN. Without an effective masking mechanism, these neighbors can introduce noise, diluting the informative signal and impairing the model's ability to generate useful node embeddings. To address this, we propose a novel method for selecting the most relevant and masking other redundant high-order neighbors based on their local contribution to node representations.

We introduce the Local Contribution Score (LCS) to assign importance scores to high-order neighbors. The LCS is designed to quantify the relevance of each neighbor by considering both structural similarity and feature alignment. Specifically, for a node v_i and its k-th order neighbor v_j, the LCS score is computed as:

$$\text{LCS}(v, u, k) = \frac{\alpha_k (\mathbf{A}_k)_{vu} \cdot (x_v^\top x_u)}{\sqrt{d_v \cdot d_u}}, \tag{6}$$

where $(\mathbf{A}_k)_{vu}$ is the strength of the connection between nodes v and u in the k-hop adjacency matrix, x_v and x_u are the feature vectors of nodes v and u, respectively, and d_v and d_u represent the degrees of nodes v and u, which normalize the influence of highly connected nodes. α_k is the learnable parameter that controls the contribution of the k-th hop's adjacency matrix defined in Eq. 2.

The LCS allows us to mask redundant high-order neighbors and identify the most relevant neighbors by considering both local structure and feature similarity, enabling the model to selectively pass information from the most meaningful high-order neighbors.

Once the LCS is computed for each neighbor, we select the top-K neighbors with the highest LCS values for each node:

$$\mathbf{A}_k^{\text{filtered}} = \text{Top}_K(\mathbf{A}_k, \text{LCS}(u, v, k)). \tag{7}$$

This filtering process ensures that only the most informative high-order neighbors contribute to the node's final representation, effectively reducing noise and improving the stability of the embeddings.

4.4 Optimization Objective

To ensure the effectiveness of our proposed ScaleGNN, we design an optimization framework that consists of three key loss components: the primary task loss, the local contribution score (LCS) regularization, and the mask constraint on high-order information masking mechanism. These losses work together to enhance model performance, suppress noise from uninformative high-order neighbors, and maintain computational efficiency.

For node classification, we employ the standard cross-entropy loss as the primary task objective:

$$\mathcal{L}_{\text{task}} = -\sum_{i \in \mathcal{V}} y_i \log \hat{y}_i, \tag{8}$$

where y_i represents the ground truth label for node i, and $\hat{y}_i$ is the predicted probability distribution over classes.

To guide the model in selecting meaningful high-order neighbors, we introduce an LCS-based regularization term:

$$\mathcal{L}_{\text{LCS}} = \sum_{i=2}^{K} \sum_{(v,u) \in \mathcal{N}_i^{\text{selected}}} (1 - \text{LCS}(v, u))^2 . \tag{9}$$

where $\mathcal{N}_i^{\text{selected}}$ is the selected neighboring nodes which are not been masked. This term enforces high-order neighbors be selected based on their LCS values, reducing the influence of noisy or redundant neighbors.

To further mitigate the over-smoothing issue, we impose a mask constraint on high-order feature propagation:

$$\mathcal{L}_{\text{mask}} = \sum_{k=2}^{N} \|\tilde{\mathbf{A}}_k\|_1, \tag{10}$$

where $\tilde{\mathbf{A}}_k$ is the filtered high-order adjacency matrix, and $\|\cdot\|_1$ denotes the ℓ_1 norm. Combining all components, the final optimization objective is:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_1 \mathcal{L}_{\text{LCS}} + \lambda_2 \mathcal{L}_{\text{mask}}, \tag{11}$$

where λ_1 and λ_2 are hyperparameters that control the balance between regularization and the primary task.

This optimization framework ensures that ScaleGNN maintains a balance between classification performance, robust high-order neighbor aggregation, and computational efficiency.

4.5 Time Complexity Analysis

To evaluate the computational efficiency of ScaleGNN, we analyze its time complexity and compare it with existing scalable GNNs. The overall complexity consists of three main components: **preprocessing**, **training**, and **inference**.

Let n be the number of nodes, m be the number of edges, and f be the feature dimension, K be the maximum number of hops. a is the number of layers in MLP classifiers of SIGN, GAMLP, and GBP. S in NARS denotes the number of subgraphs, M in SeHGNN denotes the number of divided groups, and R in RpHGNN is the number of node classes. In our proposed ScaleGNN, we perform the following major computations:

- **Multi-hop Adjacency Construction**: Computing higher-order adjacency matrices requires $O(Km)$, where K is the maximum number of hops.
- **LCS-based Neighbor Filtering**: Computing the Local Contribution Score (LCS) involves pairwise feature similarity for selected neighbors, yielding a complexity of $O(K_s nf)$, where K_s is the number of selected high-hop neighbors.
- **Feature Aggregation and Training**: The GNN layer updates, including low-order and high-order feature aggregation, take $O(nf^2)$.

The final complexity per training iteration is approximately $O(Km + K_s nf + nf^2)$. Table 1 compares the time complexity of ScaleGNN with other scalable GNN architectures.

Table 1: Time complexity comparison of existing deep and scalable GNN models.

Method	Pre-processing	Training	Inference
S^2GC	$O(Kmf)$	$O(nf^2)$	$O(nf^2)$
SIGN	$O(Kmf)$	$O(anf^2)$	$O(anf^2)$
GAMLP	$O(Kmf)$	$O(anf^2)$	$O(anf^2)$
GBP	$O(Kmf + K\sqrt{m}\log n)$	$O(anf^2)$	$O(anf^2)$
NARS	$O(KnSf + m)$	$O(nf^2)$	$O(nf^2)$
SeHGNN	$O(KnMf + m)$	$O(nf^2)$	$O(nf^2)$
RpHGNN	$O(KnRf + m)$	$O(nf^2)$	$O(nf^2)$
ScaleGNN	$O(Km + K_s nf)$	$O(nf^2)$	$O(nf^2)$

Compared to traditional GNNs that suffer from excessive computation due to repeated feature propagation, ScaleGNN optimizes computational efficiency in two ways: (1) **Sparse high-hop neighbor selection**: By leveraging LCS-based filtering, ScaleGNN significantly reduces the number of unnecessary high-hop neighbors involved in feature propagation, improving computational efficiency. (2) **Lightweight aggregation**: Since ScaleGNN explicitly separates low-hop and high-hop features, it avoids redundant computations, leading to lower memory and runtime costs.

Empirical results on real-world datasets demonstrate that ScaleGNN achieves faster convergence and lower training time compared to existing model-simplification methods while maintaining high predictive accuracy.

5 Experiments

In this section, we evaluate the performance of our proposed method through extensive experiments and answer the following questions:

- **(RQ1)** Can ScaleGNN outperform the state-of-the-art GNN methods regarding predictive accuracy on real-world datasets?
- **(RQ2)** How does the efficiency of ScaleGNN compare to other baseline methods?
- **(RQ3)** How dose ScaleGNN mitigate the over-smoothing issue?
- **(RQ4)** What are the effects of different modules in ScaleGNN on performance?
- **(RQ5)** How does ScaleGNN achieves the trade-off between efficiency and accuracy?
- **(RQ6)** How do different hyperparameter settings affect the performance?

5.1 Experimental Settings

5.1.1 Datasets. We have evaluated the effectiveness of ScaleGNN using six real-world graph datasets, including Citeseer, Cora, Pubmed, and ogbn-arxiv, ogbn-products, ogbn-papers100M. The first three datasets (Citeseer, Cora, and Pubmed) are relatively small and have been adopted by existing researches [15], while the latter three (ogbn-arxiv, ogbn-products, and ogbn-papers100M) are large-scale heterogeneous graphs commonly used in scalable HGNN evaluations. The detailed description of these datasets is shown in the Table 2.

5.1.2 Baselines. We compare our method with three categories of GNN baseline methods as follows:

- **Traditional GNN methods**: R-GCN [26], HetGNN [38], HAN [28], MAGNN [6], Simple-HGN [21], HINormer [22].
- **Deep GNN methods**: SIGN [25], S^2GC [45], GBP [2], GAMLP [41], GRAND+ [5], LazyGNN [31].
- **Scalable GNN methods**: NARS [34], SeHGNN [32], RpHGNN [12].

Notice that most of traditional and deep GNN methods encounter out-of-memory (OOM) issues when dealing with large datasets (such as products and paper100M). As for R-GCN, while the original version presents OOM issues with large datasets, it is often used as a baseline for large-scale HGNNs. in an effort to adapt R-GCN for large datasets, we follow NARS' experimental setting and adopt the neighbor sampling strategy used by GraphSAGE [9]. Besides, scalable GNN methods NARS, SeHGNN, and RpHGNN, are both pre-computation-based methods which differ from other end-to-end models. Most of their model runtime is consumed in obtaining the pre-computation tensor for large-scale graphs, with only a tiny amount of time spent on training and inference.

5.1.3 Implementation Details. We implement all baseline methods according to their provided code. For three small datasets, we set the epoch number to 100 and select dropout rate, weight decay, and learning rate from {0.1, 0.2, 0.3, 0.4, 0.5, 0.6}, {1e-5, 1e-4, 1e-3} and {0.001, 0.005, 0.01, 0.05, 0.1}, respectively. For three large-scale datasets, we set the epoch number and learning rate to 1000 and 0.001 without weight decay. The embedding dimension d is chosen from {32, 64, 128, 256, 512}, and our method uses GCN as the message aggregator.

5.2 Performance Comparison (RQ1)

We evaluate the performance of our ScaleGNN and all baselines on six real-world datasets under the node classification task. The experimental results are shown in Table 3. The best results are highlighted in bold, and the second-best results are underlined. As we can see, our ScaleGNN achieves the optimal performance, significantly outperforming all baselines in both micro-F1 and macro-F1

Table 2: Statistical summaries of datasets.

Datasets	#Nodes	#Edges	#Features	#Classes	#Train/Val/Test	Description
Citeseer	3,327	4,732	3,703	6	120/500/1000	citation network
Cora	2,708	5,429	1,433	7	140/500/1000	citation network
Pubmed	19,717	44,338	500	3	60/500/1000	citation network
ogbn-arxiv	169,343	1,166,243	128	40	91K/30K/49K	citation network
ogbn-products	2,449,029	61,859,140	100	47	196K/49K/2,204K	co-purchasing network
ogbn-papers100M	111,059,956	1,615,685,872	128	172	1200k/200k/146k	citation network

Table 3: Performance comparison of all models on six real-world datasets. Mi-F1 and Ma-F1 are short for micro-F1 and macro-F1. Marker * indicates the results are statistically significant against the best-performed baselines (t-test with p-value < 0.01).

Method		Citeseer		Cora		Pubmed		ogbn-arxiv		ogbn-products		ogbn-paper100M	
		Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
Trad. GNN	RGCN	70.53	70.14	81.23	80.59	78.37	75.58	58.26	46.83	48.94	48.04	42.31	31.12
	HetGNN	70.55	70.13	81.20	80.57	78.40	75.66	OOM.	OOM.	OOM.	OOM.	OOM.	OOM.
	HAN	71.10	70.57	81.39	80.86	78.53	75.85	54.68	31.50	OOM.	OOM.	OOM.	OOM.
	MAGNN	71.22	70.69	81.57	80.98	78.55	76.01	OOM.	OOM.	OOM.	OOM.	OOM.	OOM.
	Simple-HGN	71.34	70.78	81.88	81.14	78.88	76.17	69.47	58.29	OOM.	OOM.	OOM.	OOM.
	HINormer	72.75	71.91	82.19	81.67	79.35	76.60	70.71	59.68	OOM.	OOM.	OOM.	OOM.
Deep GNN	SIGN	72.44	71.97	82.11	80.46	79.53	76.44	71.78	62.99	78.21	70.10	64.32	55.49
	S^2GC	72.78	72.13	82.45	82.09	79.56	76.67	71.88	63.23	77.09	69.54	64.63	55.90
	GBP	72.61	72.08	83.52	82.42	80.56	79.13	71.42	62.41	77.27	69.49	64.66	56.07
	GAMLP	72.59	72.06	82.31	80.32	79.11	76.15	71.88	63.15	80.34	71.37	64.53	55.90
	GRAND+	72.48	72.03	82.04	80.23	79.24	76.46	71.43	62.30	77.52	69.70	64.11	55.19
	LazyGNN	72.91	72.21	82.45	80.87	79.92	77.03	71.63	62.72	77.60	69.90	64.25	55.38
Scal. GNN	NARS	72.41	71.21	81.79	80.30	78.82	75.82	70.75	59.88	78.08	70.20	63.52	54.63
	SeHGNN	72.90	72.45	83.04	81.43	79.71	77.70	71.82	62.69	79.68	70.85	64.55	55.86
	RpHGNN	73.12	72.54	83.27	82.37	80.60	79.22	71.98	63.11	80.65	71.34	64.97	56.50
Ours	$ScaleGNN_b$	73.11	72.59	83.60	82.41	80.54	79.24	71.95	63.23	80.78	71.34	64.95	56.51
	ScaleGNN	**74.22***	**73.61***	**84.53***	**84.01***	**81.95***	**80.50***	**73.02***	**64.08***	**81.64***	**72.33***	**65.53***	**57.29***

metrics on six datasets, with specific emphasis on the improvement relative to the SOTA methods.

Traditional GNN methods work flawlessly on small-scale datasets, but most of them cannot be extended to large-scale graphs and are prone to OOM issues. For example, R-GCN, HAN, Simple-HGN, and HINormer can get experimental results on ogbn-arxiv, but can't run on larger-scale graphs such as ogbn-products and ogbn-paper100M. For deep GNN methods, they work on learning large-scale graph representations using deeper GNNs while mitigating the over-smoothing problem on large-scale graphs. A common issue with such methods is that they typically convolve dozens or even tens of GNN layers. while capturing the caveat features to some extent, it also greatly increases the computational complexity of the model. Pre-computation-based GNN approaches are distinguished from end-to-end models by the fact that they consume a significant amount of time in pre-computing the tensor of large-scale graphs, so that the models do not need to iteratively learn graph representations during training and inference.

It is worth mentioning that our basic model $ScaleGNN_b$, although retaining only the basic adaptive high-order feature fusion module, achieves very similar performance to one of the SOTA methods, RpHGNN, and runs efficiently in the far outperforms all baselines. ScaleGNN gains further performance gains despite being less efficient than $ScaleGNN_b$. Therefore, ScaleGNN and $ScaleGNN_b$ provide an important reference value for us to explore the trade-off between model accuracy and efficiency.

5.3 Efficiency Analysis (RQ2)

To evaluate the efficiency of our proposed methods, ScaleGNN and $ScaleGNN_b$, we compare their total runtime with several state-of-the-art baselines, including S^2GC, GBP, GAMLP, and RpHGNN, across three benchmark datasets: ogbn-arxiv, ogbn-products, and ogbn-papers100M. The results presented in Fig. 2 clearly demonstrate the superior efficiency of ScaleGNN and $ScaleGNN_b$ over all competing methods.

5.3.1 Computation Efficiency. As shown in the figure, $ScaleGNN_b$ achieves the lowest runtime across all datasets, followed closely by ScaleGNN. Specifically, on the large-scale ogbn-papers100M

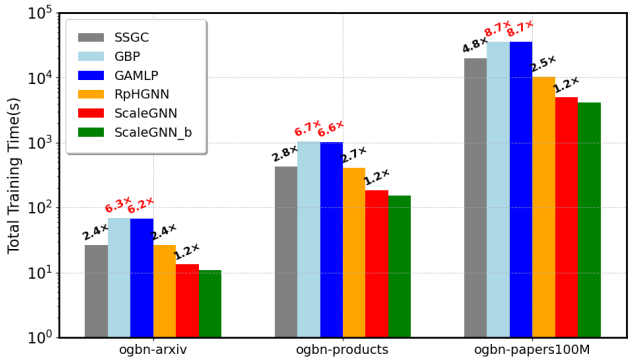


Figure 2: Efficiency comparison with SOTA GNN methods, and speedup analysis compared with our ScaleGNN$_b$.

dataset, ScaleGNN$_b$ completes the model running in only 4,110.2 seconds, significantly outperforming S^{2}GC (19,914.4s), GBP (35,706.1s), GAMLP (35,656.6s), and RpHGNN (10,399.8s). This trend is also consistently observed across ogbn-arxiv and ogbn-products datasets, where ScaleGNN$_b$ reduces runtime by at least 4.1× to 6.7× compared to the best-performing baseline.

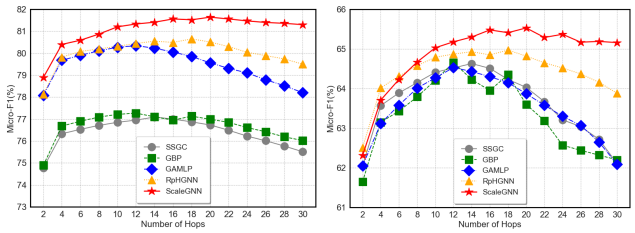
The significant reduction in runtime can be attributed to the efficient design of ScaleGNN, which optimizes neighborhood aggregation and selectively samples informative neighbors, thereby reducing computational overhead while preserving model performance. Moreover, the enhanced scalability of ScaleGNN$_b$ further refines this efficiency by introducing additional optimizations in feature propagation and memory management.

5.3.2 Contrastive Speedup Analysis. To better illustrate the efficiency advantage, we annotate each baseline method with its relative speedup factor compared to ScaleGNN$_b$. It is evident that methods like GBP and GAMLP exhibit substantially higher computational costs, with runtime exceeding 6× that of ScaleGNN$_b$ in certain cases. Even RpHGNN, which demonstrates competitive efficiency, remains 2.5× to 3.1× slower than ScaleGNN$_b$ on large datasets. These results highlight the capability of ScaleGNN to effectively mitigate the scalability limitations of existing methods. By reducing redundant computations and dynamically adjusting the information flow, our approach enables substantial efficiency improvements without compromising accuracy. The consistent speedup across diverse datasets further underscores the robustness of our design, making it particularly suitable for large-scale graph learning tasks.

In summary, ScaleGNN and ScaleGNN$_b$ achieve remarkable computational efficiency, substantially reducing runtime compared to existing approaches. The combination of selective neighbor aggregation, adaptive feature diffusion, and memory-efficient propagation mechanisms enables our methods to handle large-scale graphs with minimal computational overhead. These findings underscore the practical applicability of ScaleGNN in real-world scenarios where the model efficiency is a critical concern.

5.4 Mitigating Over-smoothing Issue (RQ3)

Over-smoothing is a well-known issue in deep graph neural networks, where node representations become indistinguishable as the number of propagation layers increases. This problem is particularly severe in large-scale graphs such as ogbn-papers100M, where excessive feature mixing leads to performance degradation. To evaluate the effectiveness of our approach in addressing this challenge, we compare ScaleGNN with several SOTA baselines, including S^{2}GC, GBP, GAMLP, and RpHGNN, across different propagation depths.



(a) ogbn-products **(b) ogbn-paper100M**

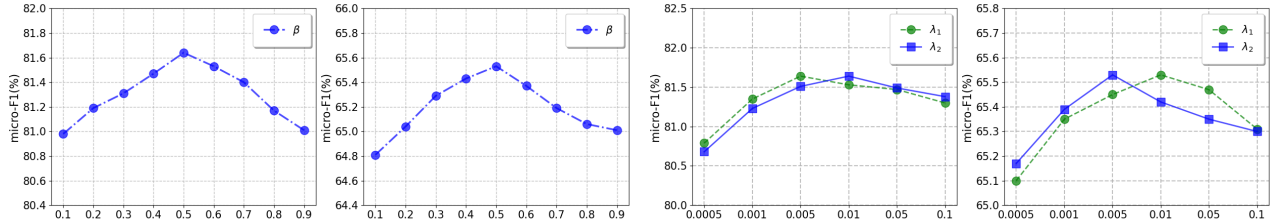
Figure 3: Experimental results of our model over SOTA GNN models w.r.t. the number of hops.

Fig. 5 shows that traditional GNN methods experience significant performance drops as the number of propagation layers increases. For example, S^{2}GC, GBP, and GAMLP achieve their peak performance at moderate depths (around 8-12 layers) but degrade rapidly beyond this point. This phenomenon occurs because deeper layers cause excessive aggregation, leading to the loss of discriminative node features. In contrast, ScaleGNN exhibits a more stable performance curve. Even at greater depths (*e.g.*, 30 layers), ScaleGNN maintains a high micro-F1 score, outperforming all baselines. This robustness demonstrates that ScaleGNN effectively preserves structural and feature information, preventing the collapse of node representations.

The key to ScaleGNN's resilience against over-smoothing lies in its adaptive feature aggregation mechanism. Unlike conventional approaches that uniformly aggregate neighborhood information, ScaleGNN dynamically balances local and global features, ensuring that useful information is retained while mitigating excessive smoothing. Specifically, our ScaleGNN Selectively incorporates higher-order neighbors with controlled influence, preventing overmixing of features. It employs adaptive diffusion mechanisms to regulate information propagation, ensuring a balance between short-term and long-term dependencies. ScaleGNN utilizes a local contribution score (LCS) to prioritize informative neighbors, reducing noise and redundancy in feature aggregation. These strategies allow ScaleGNN to maintain a richer set of node representations, even in deep architectures, making it particularly effective for large-scale graphs.

5.5 Ablation Study (RQ4)

To evaluate the effectiveness of each component in ScaleGNN, we further conduct ablation studies on different variants. Specifically, we generate three variants as follows:



(a) ogbn-products (b) ogbn-paper100M (c) ogbn-products (d) ogbn-paper100M

Figure 4: The parameter sensitivity experimental results.

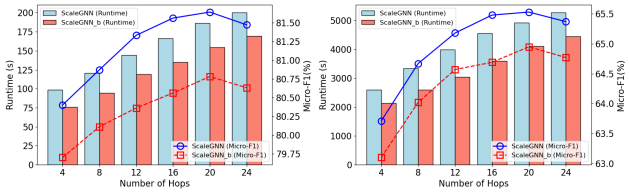
- ***w/o Ada*** removes the adaptive high-order feature fusion module and employs the deep GNN method S^2GC [45] to obtain the graph representation.
- ***w/o Low*** eliminates the low-order enhanced feature aggregation and only employs the adaptive high-order feature fusion.
- ***w/o Mask*** removes the high-order redundant feature masking mechanism.

Table 4: The comparison of ScaleGNN and its variants on micro-F1 and macro-F1 (Mi-F1 and Ma-F1 for short) metrics.

Dataset	ogbn-arxiv		ogbn-products		ogbn-paper100M	
Metrics	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
w/o Ada	71.25	62.55	80.28	70.79	64.19	55.87
w/o Low	72.07	63.27	80.30	71.57	64.97	56.56
w/o Mask	71.49	62.70	80.42	71.04	64.45	55.98
ScaleGNN	**73.02**	**64.08**	**81.64**	**72.33**	**65.53**	**57.29**

The results demonstrate that each component is vital for improving ScaleGNN's performance. ***w/o Ada*** variant experiences the largest performance drop across all datasets, with reductions of 1.77% in Micro-F1 and 1.53% in Macro-F1 on ogbn-arxiv, 1.16% and 1.54% on ogbn-products, and 1.34% and 1.42% on ogbn-paper100M, highlighting the importance of adaptive high-order feature fusion for capturing structural dependencies and preventing information loss. ***w/o Low*** variant also shows notable degradation, especially on ogbn-arxiv and ogbn-paper100M, with Micro-F1 dropping by 0.95% and 0.56%, respectively, indicating that low-order enhanced feature aggregation is crucial for maintaining local information. ***w/o Mask*** variant sees moderate degradation, with Micro-F1 decreasing over 1% on ogbn-arxiv, ogbn-products, and ogbn-paper100M, showing the effectiveness of high-order redundant feature masking in reducing redundancy and preventing excessive feature mixing.

Overall, ScaleGNN consistently outperforms all ablated versions, confirming the significant contribution of each module to the model's success. Adaptive high-order feature fusion is the most influential in improving performance, followed by low-order enhanced feature aggregation and high-order redundant feature masking. The combination of these mechanisms enables ScaleGNN to achieve superior results on various large-scale graph datasets.



(a) ogbn-products (b) ogbn-paper100M

Figure 5: The trade-off between efficiency and accuracy on large-scale graph datasets.

5.6 Trade-off Analysis: Performance vs. Efficiency (RQ5)

In this section, we evaluate the trade-off between efficiency and accuracy for ScaleGNN and ScaleGNN$_b$ by analyzing their runtime and Micro-F1 scores across different hop settings.

The runtime analysis reveals that ScaleGNN incurs a higher computational cost than ScaleGNN$_b$ across all hop settings. As the number of hops increases, both methods show rising runtimes, reflecting the increased complexity of message passing over larger neighborhoods. However, ScaleGNN consistently requires more computational resources, suggesting that its modeling choices introduce additional overhead.

In terms of performance, ScaleGNN consistently outperforms ScaleGNN$_b$ in Micro-F1 scores. Both methods benefit from increasing hops, as it helps capture structural and semantic relationships. However, beyond a certain point, performance gains diminish due to redundant or noisy information from distant nodes.

These results highlight the trade-off between accuracy and efficiency. While ScaleGNN achieves higher Micro-F1 scores, it comes with a significantly higher computational cost. In contrast, ScaleGNN$_b$ offers a more efficient alternative with a slight performance compromise. This suggests that model selection should be based on practical constraints: ScaleGNN is ideal for accuracy-focused tasks, while ScaleGNN$_b$ is better for resource-constrained environments. Balancing efficiency and accuracy remains crucial in large-scale graph learning.

5.7 Parameter Sensitivity Analysis (RQ6)

5.7.1 The effect of hyperparameters β. We conduct a sensitivity analysis of β, which balances low-order and high-order feature aggregations, by varying its value from 0.1 to 0.9 and evaluating the Micro-F1 score (%). The results in Fig. 4 show that performance improves as β increases, peaking at $\beta = 0.5$. Beyond this point, further

increases in β lead to a gradual decline, likely due to overemphasizing high-order information, which may introduce noise or redundancy. These findings highlight the importance of β to strike an optimal balance between local and global information aggregation.

5.7.2 The effect of hyperparameters λ_1 and λ_2. We investigate the sensitivity of λ_1 and λ_2 in the final loss function by varying their values within $\{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}$ and evaluating the Micro-F1 score (%). As shown in Fig. 4, λ_1 is more sensitive, peaking at $\lambda_1 = 0.005$ and $\lambda_1 = 0.01$, with performance declining as λ_1 increases. This suggests that moderate regularization improves performance, while excessive regularization harms it. On the other hand, λ_2 shows a more stable trend, optimizing at $\lambda_2 = 0.01$, with minimal fluctuations, indicating that it primarily enhances model stability. These results highlight the need for careful hyperparameter tuning: λ_1 requires precise adjustment for optimal performance, while λ_2 ensures robustness.

6 Conclusion

We propose a novel scalable GNN model named **ScaleGNN**, tackling over-smoothing and scalability challenges in large-scale graphs. ScaleGNN adaptively aggregates informative high-order neighbors while suppressing redundancy through a Local Contribution Score (LCS)-based masking mechanism. Extensive experiments show that ScaleGNN outperforms state-of-the-art deep and scalable GNNs in both accuracy and efficiency. Future work includes extending it to heterogeneous graphs and optimizing large-scale training.

References

[1] Hao Chen, Yuanchen Bei, Qijie Shen, Yue Xu, Sheng Zhou, Wenbing Huang, Feiran Huang, Senzhang Wang, and Xiao Huang. 2024. Macro graph neural networks for online billion-scale recommender systems. In *Proceedings of the ACM on Web Conference 2024*. 3598–3608.

[2] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. *NeurIPS* 33 (2020), 14556–14566.

[3] Zi Chen, Bo Feng, Long Yuan, Xuemin Lin, and Liping Wang. 2023. Fully Dynamic Contraction Hierarchies with Label Restrictions on Road Networks. *Data Science and Engineering* 8, 3 (2023), 263–278.

[4] Mucong Ding, Tahseen Rabbani, Bang An, Evan Wang, and Furong Huang. 2022. Sketch-GNN: Scalable graph neural networks with sublinear training complexity. *NeurIPS* 35 (2022), 2930–2943.

[5] Wenzheng Feng, Yuxiao Dong, Tinglin Huang, Ziqi Yin, Xu Cheng, Evgeny Kharlamov, and Jie Tang. 2022. Grand+: Scalable graph random neural networks. In *WWW*. 3248–3258.

[6] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *WWW*. 2331–2341.

[7] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* (2018).

[8] Kaiqi Gong, Xiao Song, Wenxin Li, and Senzhang Wang. 2024. HN-GCCF: High-order neighbor-enhanced graph convolutional collaborative filtering. *Knowledge-Based Systems* 283 (2024), 111122.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *NeurIPS* 30 (2017).

[10] Liancheng He, Liang Bai, Xian Yang, Hangyuan Du, and Jiye Liang. 2023. High-order graph attention network. *Information Sciences* 630 (2023), 222–234.

[11] Huiting Hong, Hantao Guo, Yucheng Lin, Xiaoqing Yang, Zang Li, and Jieping Ye. 2020. An attention-based graph neural network for heterogeneous structural learning. In *AAAI*, Vol. 34. 4132–4139.

[12] Jun Hu, Bryan Hooi, and Bingsheng He. 2024. Efficient heterogeneous graph learning via random projection. *IEEE TKDE* (2024).

[13] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *WWW*. 2704–2710.

[14] Xunqiang Jiang, Tianrui Jia, Yuan Fang, Chuan Shi, Zhe Lin, and Hui Wang. 2021. Pre-training on large-scale heterogeneous graph. In *SIGKDD*. 756–766.

[15] Di Jin, Yingli Gong, Zhiqiang Wang, Zhizhi Yu, Dongxiao He, Yuxiao Huang, and Wenjun Wang. 2022. Graph neural network for higher-order dependency networks. In *WWW*. 1622–1630.

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

[17] Bingheng Li, Erlin Pan, and Zhao Kang. 2024. Pc-conv: Unifying homophily and heterophily with two-fold filtering. In *AAAI*, Vol. 38. 13437–13445.

[18] Jianxin Li, Hao Peng, Yuwei Cao, Yingtong Dou, Hekai Zhang, S Yu Philip, and Lifang He. 2021. Higher-order attribute-enhancing heterogeneous graph neural networks. *IEEE TKDE* 35, 1 (2021), 560–574.

[19] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Zebra: When temporal graph neural networks meet temporal personalized PageRank. *VLDB* 16, 6 (2023), 1332–1345.

[20] Zhijun Liu, Chao Huang, Yanwei Yu, Baode Fan, and Junyu Dong. 2020. Fast attributed multiplex heterogeneous network embedding. In *CIKM*. 995–1004.

[21] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. 2021. Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks. In *SIGKDD*.

[22] Qiheng Mao, Zemin Liu, Chenghao Liu, and Jianling Sun. 2023. Hinormer: Representation learning on heterogeneous information networks with graph transformer. In *WWW*. 599–610.

[23] Lawrence Page. 1999. *The PageRank citation ranking: Bringing order to the web*. Technical Report.

[24] Erlin Pan and Zhao Kang. 2023. Beyond homophily: Reconstructing structure for graph-agnostic clustering. In *ICML*. PMLR, 26868–26877.

[25] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198* 7 (2020), 15.

[26] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*. Springer, 593–607.

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[28] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *WWW*. 2022–2032.

[29] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

[30] Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. 2022. Nodeformer: A scalable graph structure learning transformer for node classification. *NeurIPS* 35 (2022), 27387–27401.

[31] Rui Xue, Haoyu Han, MohamadAli Torkamani, Jian Pei, and Xiaorui Liu. 2023. Lazygnn: Large-scale graph neural networks via lazy propagation. In *ICML*. PMLR, 38926–38937.

[32] Xiaocheng Yang, Mingyu Yan, Shirui Pan, Xiaochun Ye, and Dongrui Fan. 2023. Simple and efficient heterogeneous graph neural network. In *AAAI*, Vol. 37. 10816–10824.

[33] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*. 974–983.

[34] Lingfan Yu, Jiajun Shen, Jinyang Li, and Adam Lerer. 2020. Scalable graph neural networks for heterogeneous graphs. *arXiv preprint arXiv:2011.09679* (2020).

[35] Le Yu, Leilei Sun, Bowen Du, Tongyu Zhu, and Weifeng Lv. 2022. Label-enhanced graph neural network for semi-supervised node classification. *IEEE TKDE* 35, 11 (2022), 11529–11540.

[36] Pengyang Yu, Chaofan Fu, Yanwei Yu, Chao Huang, Zhongying Zhao, and Junyu Dong. 2022. Multiplex heterogeneous graph convolutional network. In *SIGKDD*.

[37] Zeyuan Zeng, Qinke Peng, Xu Mou, Ying Wang, and Ruimeng Li. 2023. Graph neural networks with high-order polynomial spectral filters. *IEEE TNNLS* (2023).

[38] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *SIGKDD*. 793–803.

[39] Fanjin Zhang, Xiao Liu, Jie Tang, Yuxiao Dong, Peiran Yao, Jie Zhang, Xiaotao Gu, Yan Wang, Evgeny Kharlamov, Bin Shao, et al. 2022. Oag: Linking entities across large-scale heterogeneous knowledge graphs. *IEEE TKDE* 35, 9 (2022), 9225–9239.

[40] Heng-Kai Zhang, Yi-Ge Zhang, Zhi Zhou, and Yu-Feng Li. 2024. HONGAT: Graph Attention Networks in the Presence of High-Order Neighbors. In *AAAI*, Vol. 38. 16750–16758.

[41] Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. 2022. Graph attention multi-layer perceptron. In *SIGKDD*.

[42] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. 2018. Billion-scale network embedding with iterative random projection. In *ICDM*. IEEE, 787–796.

[43] Weichen Zhao, Tiande Guo, Xiaoxi Yu, and Congying Han. 2023. A learnable sampling method for scalable graph neural networks. *Neural Networks* 162 (2023), 412–424.

[44] Shanna Zhong, Jiahui Wang, Kun Yue, Liang Duan, Zhengbao Sun, and Yan Fang. 2023. Few-shot relation prediction of knowledge graph via convolutional neural network with self-attention. *Data Science and Engineering* 8, 4 (2023), 385–395.

[45] Hao Zhu and Piotr Koniusz. 2021. Simple spectral graph convolution. In *ICLR*.

[46] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. 2021. Graph neural networks with heterophily. In *AAAI*, Vol. 35. 11168–11176.

[47] Shichao Zhu, Chuan Zhou, Shirui Pan, Xingquan Zhu, and Bin Wang. 2019. Relation structure-aware heterogeneous graph neural network. In *ICDM*. IEEE, 1534–1539.