electronics

MDPI

Article

Towards Pure High-Order Aggregation: Rethinking Multi-Hop Neighborhood Learning in Graph Convolutional Networks

Chaochao Hu and Zhaohui Zhang *

School of Information and Intelligent Science, Donghua University, Shanghai 201620, China; 1229161@mail.dhu.edu.cn

* Correspondence: zhzhang@dhu.edu.cn

Abstract

Graph convolutional networks (GCNs) have achieved remarkable success in graph representation learning, yet they remain limited by over-smoothing and insufficient utilization of high-order topology. Existing high-order GCNs exploit multi-hop neighbors but ignore the purity of high-order neighborhoods: the high-order graphs they construct contain duplicated and fictional edges, which cause feature redundancy and false topological semantics. In this paper, we propose a pure high-order graph convolutional network (PHGCN) grounded in pure high-order neighborhoods. We first analyze how duplicated and fictional edges arise from powers of the adjacency matrix, and then design a graph pure high-order projection (GPHP) algorithm that eliminates both types of invalid edges. On this basis, we construct a multi-path GCN architecture with an attention-based fusion module to learn and combine features from pure high-order neighborhoods of different orders. Experiments on seven graph classification benchmarks (IMDB-B, IMDB-M, MUTAG, PROTEINS, NCI1, DD, and COLLAB) show that PHGCN achieves the best average ranking among the compared models, and ablation studies show that each component contributes to the final performance.

Keywords: graph neural network; graph convolutional network; high-order information; pure high-order projection

1. Introduction

Graph-structured data is widely deployed in numerous real-world scenarios, including social networks [1], biological networks, recommendation systems [2], and computer vision [3]. Such data can represent complex systems by encoding topological relationships between nodes and attribute characteristics of nodes and edges. However, conventional deep learning algorithms cannot directly process such non-Euclidean data. To overcome this challenge, researchers have developed numerous graph representation learning and embedding methods, including graph decomposition [4], deep walking [5], and graph neural networks (GNNs) [6], which aggregate and update node and edge features through message passing to generate embeddings for downstream tasks.

As a dominant family of graph embedding frameworks, GNNs [7], especially GCN [8], have achieved remarkable success on various graph learning tasks. Nevertheless, traditional shallow GCNs face two inherent limitations: severe over-smoothing [9] and insufficient utilization of high-order topological information [10]. Restricted to single-hop low-order neighborhood aggregation, standard GCNs fail to capture latent multi-hop dependencies and deep structural semantics embedded in graph data. In fact, high-order

Check for updates

Academic Editor: Domenico Ursino

Received: 24 August 2026
Revised: 7 September 2026
Accepted: 8 September 2026
Published: 9 September 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

neighborhood information contains abundant complementary structural features that are crucial for distinguishing node heterogeneity and improving model generalization; for instance, the pattern that "the enemies of enemies are friends" in social networks [11]. Existing studies have verified that excavating high-order neighborhood connections can substantially enhance node representation quality, demonstrating the necessity of high-order topological modeling for graph learning [12,13].

To compensate for the limitations of low-order feature learning, various high-order GCN (HGCN) methods have been proposed to capture deep structural information by aggregating multi-hop neighborhood features. Current mainstream high-order modeling strategies mainly include equal-weight multi-order feature fusion, such as MixHop [14] and SoGCN [15]; selective partial high-order neighborhood utilization, such as EvenNet [12], SkipGNN [16], and k-hop GNN [11]; and adaptive high-order feature aggregation, such as ADC [17], AGDN [18], HGAT [19], and NAGphormer [20]. These methods effectively break through the bottleneck of single-hop aggregation and improve the feature expression ability of GCNs to a certain extent, as demonstrated by KPGNN [21]. However, nearly all existing HGCN methods ignore the structural purity of high-order neighborhood graphs, a critical topological factor that restricts model performance improvement.

Specifically, high-order neighborhood graphs constructed by conventional multi-hop aggregation strategies contain substantial duplicated edges and fictional edges. Duplicated edges trigger repeated accumulation of low-order shallow features, causing severe feature redundancy. Meanwhile, fictional edges introduce non-existent false topological relationships, interfering with the model's perception of real graph structures and blurring the topological boundary between low-order and high-order neighborhoods. The coexistence of these two structural impurities undermines the independence and diversity of multi-order neighborhood components. The model is forced to learn redundant overlapping information iteratively and cannot mine effective latent high-order structural features, which severely limits the representation learning capability and performance upper bound of the existing high-order GCNs.

To address the structural impurity defects of traditional high-order neighborhood graphs, this paper proposes a pure high-order graph convolutional network (PHGCN) based on pure high-order neighborhood projection. First, we conduct a detailed theoretical analysis of the generation mechanism and negative effects of duplicated and fictional edges in high-order topological graphs. Then, a graph pure high-order projection (GPHP) algorithm is designed to eliminate invalid duplicated edges and fictional edges, obtaining pure and independent multi-order neighborhood topologies (as shown in Figure 1). On this basis, a multi-path graph feature-learning architecture and an adaptive multi-order feature fusion module are established to realize hierarchical learning and complementary aggregation of pure high-order features. Theoretical derivations support the rationality of the proposed method. Experiments on seven graph classification benchmarks show that PHGCN achieves the best average ranking among the compared models.

The contributions of this paper are summarized as follows:

1. A theoretical analysis is conducted to reveal that high-order neighborhoods extracted by the existing methods are impure and contaminated with fictional edges and duplicated edges.
2. A pure high-order graph convolutional network is proposed to effectively aggregate pure high-order neighborhood information for graph learning, which integrates a graph attention mechanism to adaptively learn the importance of different pure high-order neighborhoods.
3. Theoretical validation shows that the GPHP algorithm extracts pure high-order neighborhood structures by removing fictional and duplicated edges.

4. PHGCN achieves the best average ranking compared with standard GCN and SOTA algorithms on seven graph classification benchmark datasets.

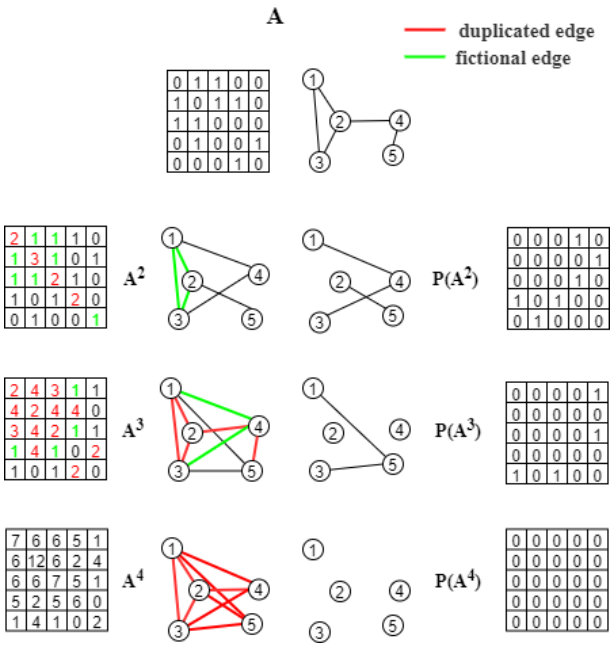


Figure 1. The difference between traditional high-order neighborhood graphs and pure high-order neighborhood graphs. In the traditional second-order neighborhood graph, fictional edges (1-2), (2-3), (3-1) exist; the traditional third-order neighborhood graph contains fictional edges (1-4), (3-4) and duplicated edges (1-2), (2-3), (3-1), (2-4), (4-5); and the traditional fourth-order neighborhood graph has duplicated edges (1-2), (1-3), (1-4), (1-5), (2-3), (2-4), (2-5), (3-4), (3-5).

2. Related Work

For high-order or multi-order GCNs, related approaches are classified into two main categories: methods that preserve the original graph structure and methods that modify the original graph structure.

2.1. GCNs with Unchanged Original Structure

In GCNs that retain the original graph structure, node representations are usually obtained directly via GCN-based node aggregation to capture high-order neighborhood graphs. Abu-El-Haija et al. [14] propose to compute neighborhoods of different orders first and then perform full linear mixing of multi-order neighborhood information in each message-passing step to capture multi-order dependencies and learn corresponding feature representations. Their feature update formula is as follows:

$$H^{(l+1)} = \sigma\left(\Big\|_{k=0}^{K} \hat{A}^k H^{(l)} W^{(l,k)}\right), \tag{1}$$

where $\hat{A}$ is the normalized adjacency matrix with self-loops, $H^{(l)}$ is the feature matrix of layer l, $W^{(l,k)}$ is the weight matrix associated with the k-order neighborhood at layer l, $\Big\|_{k=0}^{K}$ concatenates the $K+1$ order-specific components from order 0 to the maximum order K, and $\sigma(\cdot)$ is the ReLU activation function defined in Equation (18).

Wang et al. [15] argue that graph convolutional networks incorporating high-order neighborhood information exhibit stronger expressive power than standard GCNs, but such models incur higher computational complexity. To address this issue, they propose a multi-layer second-order graph convolutional network to replace existing single-hop aggregation, multi-hop propagation, and skip-connection models for capturing comparable high-order neighborhood information. The second-order convolution formula is as follows:

$$H^{(l+1)} = \sigma\left(\hat{A}H^{(l)}W^{(l)} + \hat{A}^2H^{(l)}W^{(l)}\right), \tag{2}$$

where $\hat{A}$ is the normalized adjacency matrix (including self-loops), $H^{(l)}$ is the feature matrix of layer l, $W^{(l)}$ is the weight matrix of layer l, and $\sigma(\cdot)$ is the ReLU activation function defined in Equation (18).

Zhao et al. [17] assume that neighborhood information of different orders has varying importance and propose an adaptive diffusion convolution strategy that automatically learns the optimal neighborhood size from data. This design ensures that high-order neighborhood information captured by each convolutional layer differs, which facilitates better graph representation learning. Based on this framework, Sun et al. [18] propose two scalable weighting mechanisms, namely step-by-step attention and step-by-step convolution, to capture high-order neighborhood information. He et al. [19] further develop a high-order graph attention network built upon the graph attention network, which adaptively aggregates feature information from high-order neighborhoods via an attention mechanism. Chen et al. [20] propose NAGphormer, which aggregates hop-wise neighborhood information into token sequences and enables node classification on large graphs in mini-batches. The attention-based aggregation formula is as follows:

$$h_i' = \sigma\left(\sum_{k=1}^{K}\sum_{j\in\mathcal{N}^{(k)}(i)}\alpha_{ij}^{(k)}W^{(k)}h_j\right), \tag{3}$$

where $\mathcal{N}^{(k)}(i)$ is the k-order neighborhood of node i, K is the maximum hop count of high-order neighborhoods, $\alpha_{ij}^{(k)}$ is the attention weight between node i and node j in the k-order neighborhood, h_j is the input feature vector of node j, $W^{(k)}$ is the weight matrix shared by all nodes in the k-order neighborhood, h_i' is the updated feature vector of node i, and $\sigma(\cdot)$ is the ReLU activation function defined in Equation (18).

2.2. GCNs with Modified Original Structure

In GCNs that alter the original graph structure, new graph structures are typically generated based on the topological connections of the input graph to derive high-order neighborhood graphs. Lei et al. [12] propose EvenNet, a spectral GCN based on even-polynomial graph filters. EvenNet outperforms full-order models in generalization across homogeneous and heterogeneous graphs, which indicates that ignoring odd-hop neighborhood information can improve model performance. More recently, Bo et al. [22] have proposed Specformer, which learns the coefficients of the spectral filter with a transformer and further improves the expressive power of spectral graph convolutional networks. The spectral filter formula of EvenNet is as follows:

$$H' = U\left(\sum_{k=0}^{K}\theta_k\Lambda^{2k}\right)U^\top H, \tag{4}$$

where U and Λ are the eigenvector matrix and eigenvalue matrix of the normalized Laplacian, respectively, H and H' are the input and output feature matrices, θ_k is the coefficient of the even polynomial, and K is the maximum order of the polynomial filter. As in the previous equations, K denotes the maximum hop count of the neighborhoods considered by the model.

Huang et al. [16] note that existing GCN models neglect the similarity between non-directly connected nodes, and propose SkipGNN. SkipGNN improves model performance by aggregating not only direct interaction information but also second-order interaction information, i.e., second-order neighborhood information. Nikolentzos et al. [11] propose

k-hop GNN, which updates node representations by aggregating information from both immediate neighbors and k-order neighbors. The aggregation formula is as follows:

$$H = \mathrm{ReLU}\left(\hat{A}XW_1 + \hat{A}_2XW_2\right), \tag{5}$$

where $\hat{A}$ is the normalized adjacency matrix with self-loops, $\hat{A}_2$ is the normalized adjacency matrix of the second-order neighborhood, X is the input feature matrix, W_1 and W_2 are learnable weight matrices, and $\mathrm{ReLU}(\cdot)$ is the rectified linear unit activation function.

Feng et al. [21] introduce the KPGNN framework. In this framework, peripheral subgraphs of each order are generated based on high-order neighborhoods, and k-hop information propagation is enhanced using the peripheral subgraph information for each order.

However, the high-order neighborhood graphs used in the aforementioned models contain partial information from low-order neighborhood graphs, which causes the model to repeatedly learn low-order information. To overcome this limitation, a new model that exploits pure high-order neighborhoods is proposed in this paper.

3. The Limitations of Existing High-Order Neighborhood Extraction Methods

To illustrate the generation of duplicated edges and fictional edges during the transformation from the first-order matrix to higher-order neighborhood matrices, the graph-theoretic interpretation of adjacency matrix powers must be considered. Specifically, an element in the adjacency matrix power denotes the number of paths of length k from node i to node j. Analyses are performed separately for graphs with node self-loops and graphs without node self-loops. The adjacency matrix power is defined as the matrix power of the first-order adjacency matrix of the graph:

$$A^k = \underbrace{A \cdot A \cdots A}_{k \text{ times}}, \quad (A^k)_{ij} = \#\{\text{paths of length } k \text{ from } i \text{ to } j\}, \tag{6}$$

where A is the first-order adjacency matrix of the graph.

3.1. Node with a Self-Loop

A self-loop means $A_{ii} = 1$ (a node has an edge connected to itself). The graph-theoretic meaning of adjacency matrix multiplication remains the counting of paths, but self-loops introduce cyclic paths.

3.1.1. From First-Order Adjacency Matrix A to Second-Order Adjacency Matrix A^2

The second-order adjacency matrix is calculated as follows:

$$(A^2)_{ij} = \sum_{t=1}^{n} A_{it}A_{tj}, \tag{7}$$

where n is the number of nodes in the graph, and A contains self-loops ($A_{ii} = 1$).

3.1.2. Generation of Duplicated Edges

As shown in Figure 2, if the node i has a self-loop ($A_{ii} = 1$), then A^2 will contain A^2_{ii}; that is, A^2 includes the path of direct edge + (self-loop + direct edge), resulting in A^2_{ij}, forming a duplicated edge (edge weight increases). For example, if node 1 has a self-loop ($A_{11} = 1$), then A^2_{11}, and A^2_{12}, so $A^2_{12} > 0$; this process increases edge weights and forms duplicated edges.

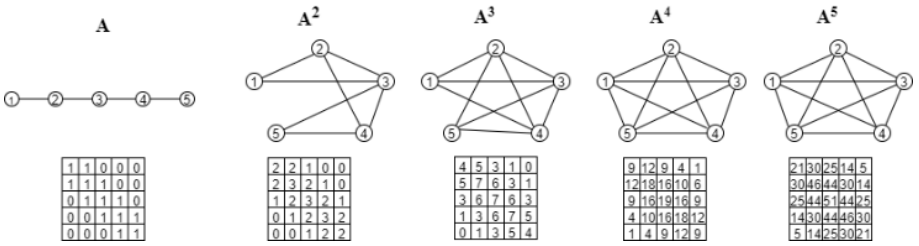


Figure 2. Higher-order neighborhood graph with self-loop graph.

3.1.3. Generation of Fictional Edges

Cyclic paths induced by self-loops create new fictional edges. For example, as shown in Figure 2, node 1 can reach itself via a self-loop (a path of length 2) or other nodes via a combination of self-loops and direct edges, which forms fictional edges that coexist with direct edges and two-step paths. The number of such fictional edges can be quantified as follows:

$$E_f = \sum_{i,j} \mathbb{I}\Big\{ (A^k)_{ij} > 0 \text{ and } (A^{k-1})_{ij} = 0 \Big\}, \tag{8}$$

where $\mathbb{I}\{\cdot\}$ is the indicator function (1 if the condition is true, 0 otherwise), and E_f is the number of fictional edges.

3.1.4. From Second-Order Adjacency Matrix A^2 to k-Order Adjacency Matrix A^k ($k \geq 2$)

The k-order adjacency matrix is calculated recursively, as follows:

$$A^k = A^{k-1} \cdot A, \tag{9}$$

Due to the presence of self-loops, during the calculation process, there must be $A_{ii} = 1$ or $A^2_{ii} = 1$, which will be retained A^2_{ij} and will further accumulate the number of paths.

3.1.5. Generation of Duplicated Edges

The k-order adjacency matrix A^k including the "path + direct edge" or "self-loop + self-loop + direct edge" combinations from A^{k-1}, and the self-loop structure will retain the original newly generated connection paths, causing the number of paths to be much larger than the corresponding value in A^{k-1}, forming duplicated edges (the weights continue to change). The weight increase in duplicated edges is as follows:

$$\Delta w_{ij} = (A^k)_{ij} - (A^{k-1})_{ij}, \tag{10}$$

3.1.6. Generation of Fictional Edges

Multi-step cyclic paths originating from self-loops generate additional fictional edges that correspond exclusively to paths of length k and lack direct or shorter paths. As shown in Figure 2, high-order neighborhood graphs gradually become fully connected with multi-step cycles. In fully connected graphs, all nodes obtain proportional information during graph convolution, leading to over-smoothing.

3.2. Node Without a Self-Loop

As shown in Figure 3, assuming that the node set of the graph is $\{1, 2, 3, 4, 5\}$, the first-order neighborhood matrix A satisfies the following:

$$A_{ij} = \begin{cases} 1, & (i,j) \in E \\ 0, & \text{otherwise} \end{cases}, \tag{11}$$

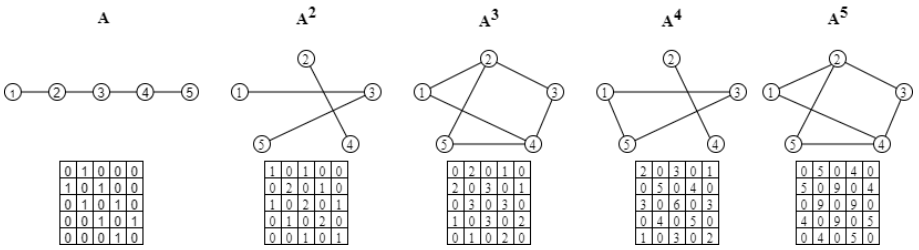


Figure 3. Higher-order neighborhood graphs without self-cycles.

3.2.1. From First-Order Adjacency Matrix A to Second-Order Adjacency Matrix A^2

The multiplication rule for adjacency matrices is $(A^2)_{ij} = \sum_s A_{is}A_{sj}$, where s indexes the intermediate node through which the length-2 path passes. Its graph theory meaning is the number of paths from node i to j via a path of length 2.

3.2.2. Generation of Duplicated Edges

If i and j are directly connected in A (i.e., $A_{ij} = 1$), but in A^2, $A^2_{ij} > 0$, it means that there is a path of length 2 that overlaps the direct edge, forming a duplicated edge. The proportion of duplicated edges in the second-order neighborhood is as follows:

$$\rho = \frac{\left|\left\{(i,j) : A_{ij} > 0, (A^2)_{ij} > 0\right\}\right|}{\left|\left\{(i,j) : (A^2)_{ij} > 0\right\}\right|}, \tag{12}$$

For example, path 2-1-2 is a path of length 2, corresponding A^2_{22}, forming a duplicated edge. In practice, in a graph, A, without a self-loop, if two nodes have a common adjacent node, the number of paths of length 2 is A^2_{ij}. If the value is greater than 0 and $A_{ij} = 0$, it is a new two-path edge; if $A^2_{ij} > 0$ and $A_{ij} = 1$, it is a duplicated edge (but when there is no self-loop, the length of the direct edge is 1 and does not overlap with the path of length 2. The actual duplicated edges are more reflected in the accumulation of edge weights (number of paths)).

3.2.3. Generation of Fictional Edges

If $A^2_{ij} > 0$ but $A_{ij} = 0$, then these edges are fictional edges of pure high-order neighborhoods (only exist in paths of length 2, no direct edges).

3.2.4. From Second-Order Adjacency Matrix A^2 to k-Order Adjacency Matrix A^k

The k-order adjacency matrix is defined as $A^k = A^{k-1}A$. Its graph theory meaning is the number of paths from node i to j via a path of length k.

3.2.5. Generation of Duplicated Edges

A duplicated edge is formed if A^k contains a combination of a path of length k and an edge of length 1 that already exists in A^{k-1}, or if a path of length k overlaps a shorter path (of length 1 or $k-1$). For example, for the path 1-2-3-2, the actual value of A^3_{12} is the accumulation of the number of paths. If a certain path number already has a corresponding structure in A^{k-1}, it is the weight change of the duplicated edge.

3.2.6. Generation of Fictional Edges

If $(A^k)_{ij} > 0$ but $(A^{k-1})_{ij} = 0$, it is a pure fictional edge. For example, in a graph, G, the edge structure contains more crossed paths, and the edge 1-2, 2-3, 3-4, 4-5 are fictional edges.

In short, the power of the adjacency matrix generates duplicated edges (changes in edge weights, i.e., Δw_{ij}, the sum of the number of paths) through the accumulation of the number of paths, and fictional edges (connections without lower-order paths and

direct edges) through the exclusive connections of higher-order paths; the existence of a self-loop amplifies the weight change of duplicated edges and increases the number of fictional edges.

Thus, a high-order neighborhood must contain duplicated and fictional edges, which inevitably cause information confusion and must be removed.

4. Pure High-Order Graph Convolutional Network

4.1. Model Framework

This paper introduces the pure high-order graph convolutional network (PHGCN), which uses the graph pure high-order projection algorithm to derive pure high-order neighborhood graphs from the input graph. As shown in Figure 4, the framework of PHGCN comprises three key modules: pure high-order neighborhood graph generation, multi-path graph feature learning, and multi-order neighborhood feature fusion.

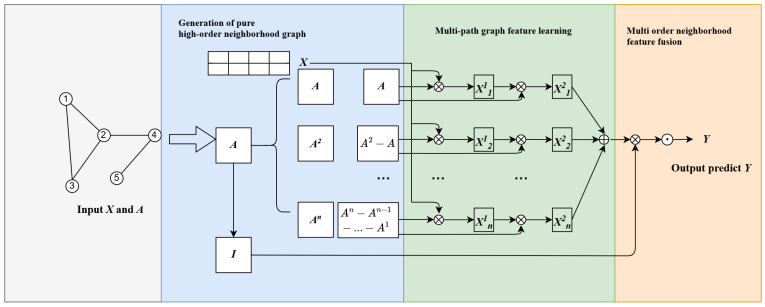


Figure 4. Schematic diagram of PHGCN.

The pure high-order neighborhood graph generation module uses the GPHP algorithm to derive pure neighborhood graphs of different orders from the original graph structure. The multi-path graph feature-learning module employs two-layer graph convolutional networks to learn features from different pure neighborhood graphs, with non-shared parameters across paths. The multi-order neighborhood feature fusion module refines graph topology using learned features to improve the representation of node relationships and supports iterative updates of graph edge weights.

The overall forward propagation formula of PHGCN is as follows:

$$Y = \mathrm{Softmax}\Big(\mathrm{Fusion}\Big(\{\mathrm{GCN}_k(X, P_k(A))\}_{k=1}^{K}\Big)W\Big), \tag{13}$$

where $P_k(A)$ is the pure k-order neighborhood matrix obtained by the GPHP algorithm, GCN_k is the k-th path GCN model, Fusion is the multi-order feature fusion function, and W is the output layer weight matrix.

4.2. Graph Pure High-Order Projection Algorithm

In the first module, the graph pure high-order projection (GPHP) algorithm is proposed to compute corresponding pure high-order neighborhood graphs from the original graph structure. A neighborhood matrix is called pure when it contains no self-loops and no edge that already appears in a lower-order neighborhood.

Figure 5 illustrates the four steps of the GPHP algorithm on an example graph for $k = 2$. Step 1 (path counting): For the adjacency matrix A of the original graph, the second-order neighborhood A^2 is obtained by computing the matrix square of the adjacency matrix. An entry $(A^2)_{ij}$ counts the number of length-2 paths between nodes i and j through an intermediate node, which is consistent with the interpretation of adjacency-matrix powers given in Section 3.

Step 2 (binarization): Because the entries of A^2 are path counts that can exceed 1, the sign$(\cdot)$ function is applied to binarize the matrix. The sign function maps every positive element to 1 and every non-positive element to 0:

$$\text{sign}(x) = \begin{cases} 1, & x > 0 \\ 0, & \text{otherwise} \end{cases}, \tag{14}$$

Step 3 (removal of lower-order edges): To ensure that the second-order neighborhood does not contain edges of the first-order neighborhood, the first-order adjacency matrix A is directly subtracted from the normalized second-order neighborhood sign(A^2):

$$C^{(2)} = \text{sign}\left(A^2\right) - A, \tag{15}$$

Because both A and sign(A^2) are binary matrices, each entry of $C^{(2)}$ is 1 (a candidate pure second-order edge), 0 (no length-2 path), or -1 (an edge that already exists in the first-order neighborhood). Step 4 (removal of negative entries and self-loops): the negative entries generated in Step 3 are removed by applying the sign$(\cdot)$ function again, and the diagonal entries, which correspond to closed walks and self-loops, are set to zero. Formally, the diagonal-zeroing operator $D(\cdot)$ sets every diagonal entry of its argument to zero, i.e., $[D(M)]_{ij} = M_{ij}$ for $i \neq j$ and $[D(M)]_{ii} = 0$. This is the degree constraint $D(\text{temp}) = \text{diag}(\text{temp}) = 0$ mentioned in the original derivation: at the node-degree level, the constraint removes self-loops so that no node is connected to itself. The pure second-order neighborhood is therefore $P_2(A) = \text{sign}\big(D\big(\text{sign}(A^2)\big) - A\big)$.

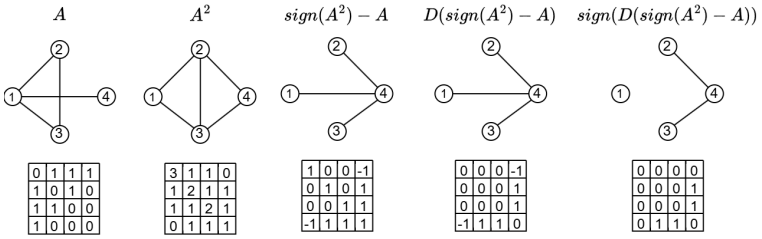


Figure 5. Illustration of the GPHP algorithm on an example graph. The panels show the graph structures obtained at successive steps of the algorithm, from the original graph to the final pure second-order neighborhood. These panels present example results on a graph rather than an algorithmic flowchart.

For an arbitrary order $k \geq 2$, the same procedure is applied recursively: the binarized k-th-order neighborhood is used as the basis, all lower-order pure neighborhoods are subtracted, and the result is binarized once more. The GPHP algorithm can thus be formally expressed as follows:

$$P_k(A) = \text{sign}\left(D\left(\text{sign}\left(A^k\right)\right) - \sum_{i=1}^{k-1} P_i(A)\right), \quad P_1(A) = A, \tag{16}$$

where $P_1(A) = A$ is the first-order pure neighborhood (the original adjacency matrix), $D(\cdot)$ is the diagonal-zeroing operator defined above, sign$(\cdot)$ is the sign function defined in Equation (14), and $P_i(A)$ is the pure i-order neighborhood matrix. Because every lower-order edge is subtracted before the final sign operation, the resulting matrix contains only edges that appear in the k-th-order neighborhood and in none of the lower-order neighborhoods; Section 6.2 proves this purity property formally. Based on the GPHP algorithm, PHGCN breaks through the limitation of traditional graph neural networks that only focus on first-order neighborhoods and can capture the long-distance dependencies of

nodes (such as the indirect influence of "friends of friends" and "enemies of enemies are friends" in social networks), which is essential for long-range graph reasoning [13].

4.3. Multi-Path Graph Feature-Learning Module

After obtaining pure high-order neighborhood graphs, PHGCN needs to learn the characteristics of the different pure high-order neighborhood graphs.

As shown in Figure 6, the multi-path graph feature-learning module is an important component in PHGCN for feature extraction of multi-order pure high-order neighborhood graphs. Pure high-order neighborhood graphs contain rich node features, edge relationships, and structural information at different levels. The multi-path graph feature-learning module uses a multi-path parallel approach to extract graph features from multiple branches at the same time, and then fuses the extracted features from multiple channels to capture the complex patterns and intrinsic information of the graph more comprehensively and accurately, improving the performance of subsequent tasks.

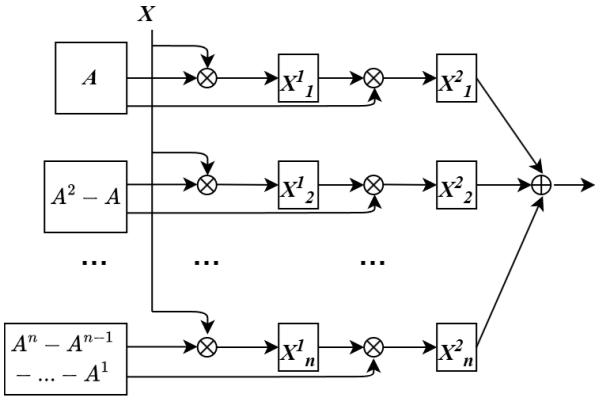


Figure 6. Schematic diagram of the multi-path graph feature-learning module.

The module takes the node feature matrix and the adjacency relationships of the graph as input. In Figure 6, A is the adjacency matrix of the graph (describing the connection relationship between nodes), $P_k(A)$ is the matrix obtained by performing different transformations on the adjacency matrix using the GPHP algorithm (k represents the order of the neighbors). Each path's GCN layer uses the following formula for feature aggregation:

$$H^{(l+1,k)} = \sigma\left(P_k(A)\, H^{(l,k)}\, W^{(l,k)}\right), \tag{17}$$

where $P_k(A)$ is the normalized pure k-order adjacency matrix, $H^{(l,k)}$ is the feature matrix of the k-th path at layer l, and $W^{(l,k)}$ is the weight matrix of the k-th path at layer l.

Multi-path parallel branches: There are multiple parallel branches in the graph, and each branch is responsible for a feature extraction path. Taking a certain branch as an example, the first step is to perform a graph convolution operation (usually matrix multiplication or related feature interaction operation) with the corresponding transformed matrix (such as $P_k(A)$). This step is to aggregate node features based on specific neighbor relationships and obtain a preliminary combination of neighbor information and node features that the branch focuses on. Then, through modules such as W_1 and W_2 (which are different feature transformation layers, such as linear transformation, nonlinear activation, convolution operations, etc.), the aggregated features are further processed to transform and enhance features, and extract more abstract and differentiated feature representations. The nonlinear activation uses the ReLU function:

$$\sigma(x) = \max(0, x), \tag{18}$$

Modules such as W_1 and W_2 of different branches use different parameters to extract different features.

Feature fusion part: The features processed by each branch are finally brought together through $\|$ (for feature splicing and fusion operation), and the features extracted in parallel from multiple channels and describing the graph from different angles are integrated into a comprehensive feature representation for use by the multi-order neighborhood feature fusion module. The concatenation operation is as follows:

$$H = H_1^{(L)} \| H_2^{(L)} \| \cdots \| H_K^{(L)}, \tag{19}$$

where L is the number of layers of each path GCN, and $\|$ denotes the concatenation operation along the feature dimension.

This fusion can integrate multiple aspects of information, avoid the limitations of a single feature extraction path, and enable the model to understand graph data more comprehensively. Multi-perspective feature extraction: Through multi-path parallelism, graph features can be extracted from multiple perspectives (such as local neighbors, long-distance dependencies, different structural patterns, etc.) at the same time. Compared with single-path methods, information capture is more comprehensive. Flexibility and adaptability: Different branches can be customized for different task requirements and different graph data characteristics, which enhances the adaptability of the module to different scenarios. After fusing multi-path features, it can provide richer and more discriminative features for downstream tasks, effectively improving the accuracy of graph classification tasks.

In short, the multi-path graph feature-learning module fully mines the multi-dimensional information of graph data through "multi-path extraction + fusion", which is one of the key components to improve the performance of graph neural networks. Based on the multi-path graph feature-learning module, PHGCN can extract features from neighborhoods of different orders at the same time, avoiding the problem of insufficient information in single-order neighborhoods and enhancing the model's ability to express graph structures.

4.4. Multi-Order Neighborhood Feature Fusion Module

Based on the multi-order neighborhood feature fusion module, PHGCN adaptively integrates multi-path features through a learnable fusion method, so that the model can adjust the attention to different-order neighborhoods according to task requirements.

As shown in Figure 7, the multi-order neighborhood feature fusion module plays a key role in the PHGCN model. It aims to integrate information from different-order neighborhoods of nodes to more comprehensively capture the structural and semantic features of graph data. PHGCN adopts an attention-based weighted fusion strategy, and the fusion formula is as follows:

$$H = \sum_{k=1}^{K} a_k H_k^{(L)}, \tag{20}$$

where a_k is the attention weight of the k-th order pure neighborhood feature, satisfying $\sum_{k=1}^{K} a_k = 1$ and $a_k \geq 0$.

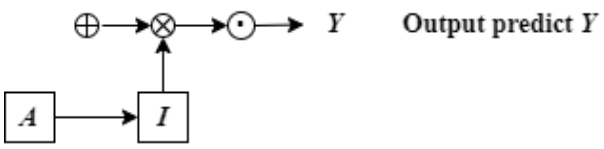


Figure 7. Schematic diagram of the multi-order neighborhood feature fusion module.

The attention weight a_k is learned through a two-layer neural network:

$$a_k = \frac{\exp\left(w_2\,\sigma\left(w_1\bar{h}_k + b_1\right) + b_2\right)}{\sum_{t=1}^{K}\exp\left(w_2\,\sigma\left(w_1\bar{h}_t + b_1\right) + b_2\right)}, \tag{21}$$

where $\bar{h}_k$ is the graph-level average feature vector of the k-th path, w_1, w_2 are learnable weight vectors, b_1, b_2 are learnable bias terms, $\exp(\cdot)$ is the exponential function, and $\sigma(\cdot)$ is the ReLU activation function defined in Equation (18). The softmax denominator in Equation (21) normalizes the weights so that $\sum_{k=1}^{K} a_k = 1$ and $a_k \geq 0$.

In the graph structure, the attributes and classification of nodes often not only depend on their own characteristics, but are also closely related to the information of surrounding neighbor nodes. Moreover, the degree of influence of neighbors of different orders on nodes and the type of information they provide are also different. First-order neighbors are directly connected and provide the most direct local information; high-order neighbors can reflect the structural position and potential relationships of nodes in a larger range. The multi-order neighborhood feature fusion module generates a more representative and differentiated node feature representation by comprehensively considering the features of different-order neighborhoods to support subsequent graph analysis tasks. The module receives feature information from neighborhoods of different orders as input.

As shown in Figures 6 and 7, the nodes pointed by multiple arrows represent the aggregation of features extracted from neighborhoods of different orders. These features may be obtained through different graph convolution operations or neighborhood aggregation methods. For example, in graph convolutional networks, the first-order, second-order and other different-order neighborhood information of nodes can be extracted through multi-layer convolution operations. Fusion operation: The $\|$ in the figure usually indicates a feature fusion operation. Common fusion methods include feature splicing and weighted summation. Feature splicing: We connect the feature vectors of different order neighborhoods in the sequence to form a longer feature vector. This method can completely retain the feature information of each order neighborhood, allowing subsequent models to independently learn the importance of different order features. For example, if the dimension of the first-order neighborhood feature vector is d, and the dimension of the second-order neighborhood feature vector is d, the dimension of the spliced feature vector is $2d$. According to the importance of neighborhood features of different orders, different weights are assigned to them, and then weighted summation is performed to obtain the fused features. These weights can be learned through model training, allowing the model to automatically adjust the contribution of different orders of neighborhood information to the final feature representation. Output transformation: The $\sigma(\cdot)$ operation in the figure is usually a linear transformation (such as matrix multiplication) or a transformation layer with a nonlinear activation function (such as ReLU). The function of this step is to further transform and map the fused features, convert them into a feature space suitable for downstream tasks, and enhance the expression ability of the features.

$$H_{\text{out}} = \sigma(H_{\text{fusion}}\,W_t + b_t), \tag{22}$$

where W_t is the transformation weight matrix, b_t is the bias term, and $\sigma(\cdot)$ is the ReLU activation function defined in Equation (18).

For example, for graph classification tasks, this step can map features to dimensions related to the number of categories, which facilitates subsequent classification predictions through the softmax function. The final output H_{out} is the node feature representation used for prediction after multi-order neighborhood feature fusion and transformation.

By fusing multi-order neighborhood features, PHGCN can obtain richer information than using only first-order neighborhoods, better describe the structural role and semantic

information of nodes in the graph, and improve the model's ability to understand complex graph structures. This module has a lot of versatility and is suitable for a variety of graph analysis tasks. Whether it is node-level classification and regression tasks, or graph-level classification and generation tasks, multi-order neighborhood feature fusion can provide more discriminative feature representation, thereby improving task performance. It can be easily combined with other graph learning modules (such as attention mechanism module, graph pooling module, etc.) to further enhance the model's expressive ability and adaptability to different graph data.

To summarize, the multi-order neighborhood feature fusion module provides more powerful and comprehensive feature representation for graph data analysis and modeling by integrating information from different-order neighborhoods, and it is an important part of graph neural networks to improve model performance.

4.5. Time and Space Complexity Analysis

Time complexity: In the common two-layer GCN model, the main computing operation is a two-layer graph convolution layer. The time complexity of each layer of graph convolution layer is usually expressed as $O(\|E\|d + nd^2)$, and the total time complexity is $O(\|E\|d + nd^2)$. In the PHGCN model, the calculation operations are: 1. K high-order neighborhood extraction, that is, K times multiplication A^k and K times subtraction $A^k - A^{k-1}$, with a time complexity of $O(Kn^2)$; 2. K times two-layer graph convolution operations of path branches, with a time complexity of $O(Kmd + Knd^2)$; and 3. high-order neighborhood feature aggregation operations $O(Knd)$. The total time complexity of the model is as follows:

$$T = O\left(Kn^2\right) + O\left(Kmd + Knd^2\right), \tag{23}$$

Space complexity: In the common two-layer GCN model, the main space occupied is to store the neighborhood matrix A, node features X, and node embedding H, and the space complexity is $O(n^2) + O(nd)$. In the PHGCN model, the space occupied is: 1. pure high-order neighborhood matrices $O(Kn^2)$; 2. the space occupied by two-layer graph convolution operations of each branch $O(Knd)$; and 3. the space occupied by high-order neighborhood feature aggregation operations $O(Knd)$. The total space complexity of the model is as follows:

$$S = O\left(Kn^2\right) + O(Knd), \tag{24}$$

To sum up, compared with the general GCN model, PHGCN's time complexity and space complexity are approximately K times those of the standard GCN when K is fixed.

5. Experimental Study

This section evaluates the proposed PHGCN model. We first compare PHGCN with previous SOTA models on five and seven graph classification benchmarks, respectively. We then investigate the impact of the maximum order, and finally conduct ablation studies to isolate the contribution of each component.

5.1. Dataset

In the experiments, we evaluate the proposed PHGCN model on seven graph classification benchmarks selected from the TUDataset [23]: the scientific collaboration dataset COLLAB, two social network datasets (IMDB-BINARY and IMDB-MULTI), and four bioinformatics datasets (MUTAG, PROTEINS, DD, and NCI1). Table 1 reports the statistics of these datasets, including the number of graphs and classes, average number of nodes, and feature dimensions.

Table 1. The statistics of seven benchmark datasets.

	IMDB-B	IMDB-M	MUTAG	PROTEINS	NCI1	DD	COLLAB
Graphs	1000	1500	188	1113	4110	1178	5000
Classes	2	3	2	2	2	2	3
Nodes (avg)	19.8	13.0	17.9	39.1	29.8	284.3	74.5
Feature dimensions	65	59	7	3	22	82	369

5.2. Baseline

Two popular GCN backbone networks are used as baselines: GCN [8] and GIN [24]. We also compare with SOTA graph-level models, including StructPool [25], GMT [26], TopKPool [27], ASAP [28], MinCutPool [29], SEP-G [30], GPS [31], AF-GCN [32], AUGCL [33], TRGC-Net [34], AMD-GCL [35], and EDA-GCL [36]. In addition, we compare them with SOTA high-order GCNs, including k-hop-GNN [11], KPGNN [21], MixHop [14], ADC [17], and SoGCN [15].

5.3. Implementation Details

Ten-fold cross-validation is performed on all datasets: each dataset is randomly partitioned into ten folds of approximately equal size; in each round, nine folds are used for training and the remaining fold for testing, and the procedure is repeated until every fold has been used once as the test fold. The average accuracy and standard deviation across the ten test folds are reported. The accuracy is calculated as follows:

$$\text{Accuracy} = \frac{N_c}{N} \times 100\%, \tag{25}$$

where N_c is the number of correctly classified graphs, and N is the total number of graphs.

For model configuration, we list the hyperparameters below, following the inductive-learning setting of [21]. The learning rate is set to 0.001, the hidden size is set to 256, the batch size is set to 64, the weight decay is set to 0.001, the activation function is the ReLU function and the random dropout rate is set to 0.3. We optimize the network using the Adam optimizer, and $\mathcal{L}$ is the cross-entropy loss function:

$$\mathcal{L} = -\frac{1}{B} \sum_{g=1}^{B} \sum_{c=1}^{C} y_{gc} \log p_{gc}, \tag{26}$$

where B is the batch size, C is the number of categories, y_{gc} is the one-hot label of the g-th graph, and p_{gc} is the predicted probability. All models are trained with the Adam optimizer for 100 epochs.

All experiments are run on a Linux server with two NVIDIA RTX 3090 GPUs.

Regarding computational cost, PHGCN is more demanding than a standard two-layer GCN because the GPHP algorithm computes matrix powers and the architecture runs K parallel GCN branches; the detailed time and space complexity analysis is given in Section 4.5. The matrix-power step of the GPHP algorithm dominates the cost for datasets with larger graphs, such as DD and COLLAB.

5.4. Comparison with SOTA

The proposed PHGCN is compared with other SOTA models on five and seven benchmark datasets, respectively. Two evaluation protocols are used, and we state them explicitly to keep the comparison transparent. In Table 2, the accuracy values of the pooling-, transformer-, and contrastive-learning-based baselines are taken from the original publications and therefore follow the experimental protocols of their respective papers; this comparison is indicative of reference performance levels rather than a controlled head-

to-head evaluation. In Table 3, the high-order GCN baselines were trained and evaluated under the same ten-fold cross-validation protocol and the same data splits as PHGCN (Section 5.3), wherever the models could be executed in our environment; entries marked with OOT or OOM indicate baselines that could not complete the evaluation within the time or memory limits. The average ranking in Table 3 is therefore computed from results obtained under a shared protocol. The inclusion criteria are as follows: Table 2 covers the baselines whose published results are available on the five commonly used benchmarks (IMDB-B, COLLAB, MUTAG, PROTEINS, and NCI1), whereas Table 3 covers the high-order GCN baselines that can be evaluated on all seven benchmarks, including DD and IMDB-M. Missing results are marked with "–".

Table 2 reports the classification accuracy (%) of the compared models on five benchmark datasets.

Within the reference-level comparison of Table 2, the proposed PHGCN reports the highest mean accuracy in each of the five benchmark datasets among the methods listed in Table 2, achieving accuracies of 80.00 ± 2.14, 83.76 ± 2.17, 89.97 ± 8.29, 80.32 ± 2.30 and 81.24 ± 1.33 on IMDB-B, COLLAB, MUTAG, PROTEINS and NCI1, respectively. Because these baseline values follow the evaluation protocols of their respective original papers rather than a shared protocol, we treat this comparison as an indication of reference performance levels and do not conclude that PHGCN clearly outperforms the compared methods. Under the shared ten-fold cross-validation protocol described above, Table 3 reports a controlled comparison between PHGCN and the high-order GCN baselines, which is discussed below.

Table 2. Graph classification accuracies on five benchmarks (%).

Models	Refs.	IMDB-B	COLLAB	MUTAG	PROTEINS	NCI1
GCN	ICLR'17	73.26 ± 0.46	80.59 ± 0.27	69.50 ± 1.78	73.24 ± 0.73	76.29 ± 1.79
GIN	ICLR'19	72.78 ± 0.86	78.19 ± 0.63	81.39 ± 1.53	71.46 ± 1.66	80.00 ± 1.40
StructPool	ICLR'20	72.06 ± 0.64	77.27 ± 0.51	79.50 ± 0.75	75.16 ± 0.86	78.64 ± 1.53
GMT	ICLR'21	73.48 ± 0.76	80.74 ± 0.54	83.44 ± 1.33	75.09 ± 0.59	76.35 ± 2.62
TopKPool	ICML'19	71.58 ± 0.95	77.58 ± 0.85	67.61 ± 3.36	70.48 ± 1.01	67.02 ± 2.25
ASAP	AAAI'20	72.81 ± 0.50	78.64 ± 0.50	77.83 ± 1.49	73.92 ± 0.63	71.48 ± 0.42
MinCutPool	PMLR'20	72.65 ± 0.75	80.87 ± 0.34	79.17 ± 1.64	74.72 ± 0.48	74.25 ± 0.86
SEP-G	PMLR'22	74.12 ± 0.56	81.28 ± 0.15	85.56 ± 1.09	76.42 ± 0.39	78.35 ± 0.33
GPS	NeurIPS'22	74.00 ± 4.25	OOT	86.81 ± 5.08	73.80 ± 3.66	72.16 ± 1.91
AF-GCN	InfSci'23	71.86	79.89	–	74.21	–
AUGCL	TNNLS'24	–	–	87.70 ± 1.19	75.76 ± 0.39	79.53 ± 0.48
TRGC-Net	TPAMI'26	–	–	84.61 ± 6	58.94 ± 3	–
AMD-GCL	TNNLS'26	–	–	89.44 ± 0.51	75.80 ± 0.36	78.62 ± 0.31
EDA-GCL	AAAI'26	74.70 ± 3.38	75.56 ± 1.75	79.47 ± 4.97	69.54 ± 3.99	52.36 ± 1.97
PHGCN	ours	**80.00 ± 2.14**	**83.76 ± 2.17**	**89.97 ± 8.29**	**80.32 ± 2.30**	**81.24 ± 1.33**

'–' means the results were not reported in the original paper; 'OOT' and 'OOM' denote failures due to time and memory limits, respectively. For TRGC-Net, the reported standard deviations retain the precision of the original paper. We bolded the best result.

Table 3. Graph classification accuracies on seven benchmarks (%).

Models	IMDB-B	COLLAB	MUTAG	PROTEINS	NCI1	DD	IMDB-M	AR
k-hop-GNN	73.80 ± 4.77	OOT	81.96 ± 9.11	74.75 ± 3.78	70.24 ± 2.85	74.96 ± 4.00	46.60 ± 6.33	5.1
KPGNN	79.30 ± 2.90	OOT	**90.68 ± 6.14**	78.79 ± 2.85	**81.33 ± 1.35**	OOM	54.87 ± 3.63	3
MixHop	77.90 ± 2.12	83.74 ± 1.84	77.28 ± 11.68	71.06 ± 2.46	68.13 ± 1.41	OOM	51.87 ± 5.10	5
ADC	78.30 ± 2.49	83.62 ± 1.86	88.33 ± 7.34	80.11 ± 2.76	81.17 ± 1.81	80.26 ± 1.82	55.00 ± 3.09	2.6
SoGCN	78.20 ± 2.06	83.60 ± 2.30	84.01 ± 7.68	78.87 ± 3.12	78.71 ± 0.68	80.22 ± 3.42	52.67 ± 2.26	3.7
PHGCN	**80.00 ± 2.14**	**83.76 ± 2.17**	89.97 ± 8.29	**80.32 ± 2.30**	81.24 ± 1.33	**81.24 ± 2.62**	**55.47 ± 1.95**	**1.3**

'OOT' and 'OOM' denote failures due to time and memory limits, respectively. The average ranking (AR) is computed over the datasets with reported results. We bolded the best result.

Existing graph learning methods present obviously divergent performance on different datasets. Traditional pooling methods such as TopKPool yield relatively low accuracy values on most test sets. Some recent contrastive learning models show prominent advantages on the MUTAG dataset yet unsatisfactory results on other benchmarks, while PHGCN maintains relatively stable performance across these benchmarks.

The standard deviation of prediction results varies greatly among comparative algorithms. Models like GPS, TRGC-Net and EDA-GCL possess large numerical fluctuation ranges, reflecting weak prediction stability. By contrast, PHGCN retains acceptable error fluctuation while reporting the highest mean accuracy in Table 2, striking a favorable balance between prediction accuracy and result robustness within this reference-level comparison.

Against advanced methods published in top-tier conferences and journals from 2022 to 2026, PHGCN also delivers competitive performance in the reference-level comparison. Its reported accuracy is higher than those reported for SEP-G, GPS, AUGCL and AMD-GCL on the datasets for which these methods are listed in Table 2. Given the protocol differences explained above, however, we avoid claiming that PHGCN clearly outperforms these methods. These observations are consistent with the effectiveness of the designed high-order graph convolution mechanism in capturing intrinsic graph structural features.

Overall, the results indicate that PHGCN performs competitively in the graph classification tasks considered in this study. The shared-protocol comparison and the ablation studies reported below provide further support for the effectiveness of the proposed method.

Table 3 reports the classification accuracy (%) and the average ranking (AR) of different HGCN models on seven graph classification benchmark datasets.

PHGCN achieves competitive or optimal classification accuracy across most of the seven graph benchmark datasets. On the IMDB-B dataset, its accuracy reaches 80.00 ± 2.14, the highest among all compared graph neural network methods. For COLLAB benchmark, PHGCN obtains 83.76 ± 2.17 and stays at the leading level among all candidates. On MUTAG, the recorded accuracy 89.97 ± 8.29 is second only to KPGNN and distinctly higher than those of k-hop-GNN and MixHop.

Methods including ADC and PHGCN present relatively small standard deviation values, reflecting steady prediction performance during repeated experiments. The standard deviation of PHGCN on the IMDB-M dataset is merely 1.95, which is the minimum value among all involved models. Large deviation values appear in the partial results of k-hop-GNN and MixHop, indicating obvious fluctuation in prediction outcomes on corresponding tasks.

The compared models differ notably in computational overhead. KPGNN and MixHop encounter out-of-memory failure on the DD dataset, while k-hop-GNN cannot complete valid computation on the COLLAB dataset. ADC and SoGCN complete all benchmarks with moderate resource consumption.

Model performance varies markedly across datasets. All methods achieve relatively high accuracy in MUTAG and COLLAB, while accuracy declines notably on IMDB-M. KPGNN achieves prominent results on small-scale datasets yet exceeds the memory limit on the large DD graphs. k-hop-GNN generally shows inferior overall performance compared with the more recent multi-order graph convolution approaches.

The average ranking reflects the overall performance of a model across all datasets, where a smaller value indicates better performance. The average ranking of PHGCN is 1.3, which is the best (lowest) among all compared models. The average rankings of k-hop-GNN, KPGNN, MixHop, ADC, and SoGCN are 5.1, 3, 5, 2.6, and 3.7, respectively, all of which are higher (i.e., worse) than that of PHGCN. This result confirms that PHGCN

delivers the most stable and outstanding overall performance on the seven benchmark datasets and surpasses all SOTA HGCNs in comprehensive performance.

In summary, PHGCN achieves the best average ranking across the seven graph classification benchmarks, although it is not the single best method on every dataset; for example, KPGNN reports higher mean accuracy in MUTAG and NCI1.

5.5. PHGCN Performance with Different Orders

Table 4 presents the classification accuracy of the PHGCN with different orders on seven graph classification benchmark datasets under different maximum orders.

Table 4. Graph classification accuracies with different order of PHGCN (%).

Order	IMDB-B	COLLAB	MUTAG	PROTEINS	NCI1	DD	IMDB-M
1	78.70 ± 2.49	83.46 ± 1.83	86.78 ± 8.83	79.14 ± 3.75	74.67 ± 2.03	79.55 ± 2.88	55.33 ± 2.11
2	78.70 ± 2.65	83.22 ± 2.20	87.31 ± 7.82	79.59 ± 3.55	76.72 ± 1.29	79.37 ± 2.13	54.67 ± 2.83
3	78.20 ± 1.94	83.52 ± 2.01	86.75 ± 7.85	79.24 ± 2.79	78.78 ± 1.27	79.37 ± 2.40	55.20 ± 2.90
4	78.80 ± 2.44	83.74 ± 2.53	88.42 ± 10.73	79.87 ± 2.71	80.19 ± 1.12	80.98 ± 2.92	54.93 ± 2.46
5	78.90 ± 3.01	83.34 ± 2.12	**89.97 ± 8.29**	79.87 ± 2.71	80.78 ± 1.05	80.73 ± 1.90	54.67 ± 2.88
6	**80.00 ± 2.14**	**83.76 ± 2.17**	89.42 ± 8.81	**80.32 ± 2.30**	**81.24 ± 1.33**	**81.24 ± 2.62**	**55.47 ± 1.95**

We bolded the best result.

The overall classification accuracy tends to improve as the model order increases. On the IMDB-B dataset, accuracy rises steadily from 78.70 ± 2.49% at order 1 to a peak of 80.00 ± 2.14% at order 6. On the NCI1 dataset, performance improves continuously with the increasing order, from 74.67 ± 2.03% to 81.24 ± 1.33%, forming a stable ascending trend.

Different benchmarks exhibit distinct sensitivities to the model order. MUTAG reaches its highest recorded accuracy at order 5, although the difference from order 6 (89.97 ± 8.29 vs. 89.42 ± 8.81) is small relative to the standard deviation, while DD and PROTEINS reach the highest accuracy under order 6 configuration. The fluctuation of the IMDB-M results remains limited across all order settings, indicating a weak correlation between prediction performance and the order in this dataset.

Experimental stability also varies with the order setting, as reflected by the standard deviations. Higher-order settings generally yield smaller standard deviations on most datasets, indicating more stable and convergent results. Some high-order settings enlarge the deviations, implying that an excessive order may occasionally introduce fluctuation.

The performance improvement slows markedly when the order enters a high range. Obvious accuracy gains are observed in the low-to-medium order range, while the improvement becomes mild when the order exceeds four. Several datasets even show slight performance degradation after reaching the optimal order, which reveals the existence of a performance saturation boundary for order optimization.

These observations provide practical guidance for selecting the maximum order. Because the gains saturate beyond order 4 on most datasets while the computational cost grows as $O(Kn^2)$ (Section 4.5), we recommend selecting K by validation rather than by exhaustive search: under the ten-fold cross-validation protocol of Section 5.3, the order with the best average validation accuracy can be chosen, and the search can be stopped when increasing K by one yields a marginal gain below a small threshold (e.g., 0.5 percentage points). The optimal order is also related to the topology of the datasets. Datasets with larger average graphs, such as DD (284.3 average nodes) and COLLAB (74.5 average nodes), benefit from higher orders, whereas small and dense graphs, such as MUTAG and IMDB-M, saturate earlier or remain nearly unchanged. A quantitative characterization of the relation between the optimal order and structural statistics such as the graph diameter and density is left for future work, as discussed in Section 7.2.

In summary, the maximum order affects the classification performance of PHGCN on most datasets, with IMDB-M being a notable exception where the results remain nearly unchanged. High-order settings (especially order 6) yield better classification accuracy in most datasets, which is reminiscent of the six-degrees-of-separation phenomenon [37] and supports the value of high-order neighborhood information [13].

5.6. Ablation Studies

To isolate the contribution of each component of PHGCN, we conduct four ablation studies on the seven benchmark datasets: (1) pure high-order neighborhood extraction, (2) the sign-based normalization in the GPHP algorithm, (3) the multi-path graph feature-learning module, and (4) the multi-order neighborhood feature fusion module. All variants follow the experimental protocol described in Section 5.3, and the full model is the same as the PHGCN row in Tables 2 and 3.

5.6.1. Effect of Pure High-Order Neighborhoods

Table 5 compares PHGCN with a variant in which the pure high-order neighborhoods are replaced by the traditional high-order neighborhoods obtained directly from the powers of the adjacency matrix, so that duplicated and fictional edges are retained. Removing the purity constraint decreases accuracy in all seven datasets. The largest drops occur on MUTAG (from 89.97 ± 8.29 to 86.78 ± 9.14) and NCI1 (from 81.24 ± 1.33 to 77.88 ± 1.81), and the smallest on COLLAB (0.28 percentage points) and DD (0.34 percentage points). The standard deviations of the two variants remain comparable, which indicates that the gains reflect the purity of high-order neighborhoods rather than random fluctuation.

Table 5. Graph classification accuracies of PHGCN with and without pure high-order neighborhoods (%).

Pure High-Order	IMDB-B	COLLAB	MUTAG	PROTEINS	NCI1	DD	IMDB-M
w	**80.00 ± 2.14**	**83.76 ± 2.17**	**89.97 ± 8.29**	**80.32 ± 2.30**	**81.24 ± 1.33**	**81.24 ± 2.62**	**55.47 ± 1.95**
w/o	79.00 ± 2.57	83.48 ± 2.23	86.78 ± 9.14	79.77 ± 3.27	77.88 ± 1.81	80.90 ± 2.35	54.20 ± 2.67

w: with pure high-order neighborhoods; w/o: without, i.e., traditional high-order neighborhoods. We bolded the best result.

5.6.2. Effect of Sign-Based Normalization

Table 6 compares PHGCN with a variant that skips the sign normalization in the GPHP algorithm so that the path counts in the neighborhood matrices directly determine edge weights. Using sign normalization improves accuracy in all seven datasets, with the largest gains in MUTAG (4.27 percentage points) and NCI1 (3.87 percentage points), followed by IMDB-B (1.30) and IMDB-M (1.34). These results indicate that the sign operation prevents the path-count accumulation of adjacency-matrix powers from inflating edge weights and biasing feature aggregation.

Table 6. Graph classification accuracies of PHGCN with and without sign normalization (%)

Sign	IMDB-B	COLLAB	MUTAG	PROTEINS	NCI1	DD	IMDB-M
w	**80.00 ± 2.14**	**83.76 ± 2.17**	**89.97 ± 8.29**	**80.32 ± 2.30**	**81.24 ± 1.33**	**81.24 ± 2.62**	**55.47 ± 1.95**
w/o	78.70 ± 2.37	83.34 ± 1.96	85.70 ± 8.77	79.59 ± 3.58	77.37 ± 1.51	80.56 ± 2.10	54.13 ± 2.27

w: with sign normalization; w/o: without sign normalization. We bolded the best result.

5.6.3. Effect of the Multi-Path Graph Feature-Learning Module

Table 7 compares PHGCN with a serial variant in which the neighborhood orders share one pipeline: the output of the low-order neighborhood convolution is used as the input of

the higher-order neighborhood convolution instead of being processed in parallel branches. The multi-path module improves accuracy in all seven datasets, with the largest gains in NCI1 (3.28 percentage points) and MUTAG (2.66 percentage points). Parallel processing therefore preserves order-specific features that the serial pipeline mixes together.

Table 7. Graph classification accuracies of PHGCN with and without the multi-path graph feature-learning module (%).

Multi-Path	IMDB-B	COLLAB	MUTAG	PROTEINS	NCI1	DD	IMDB-M
w	**80.00 ± 2.14**	**83.76 ± 2.17**	**89.97 ± 8.29**	**80.32 ± 2.30**	**81.24 ± 1.33**	**81.24 ± 2.62**	**55.47 ± 1.95**
w/o	78.80 ± 2.79	82.94 ± 2.06	87.31 ± 7.82	79.77 ± 3.40	77.96 ± 2.19	80.82 ± 2.50	54.33 ± 2.72

w: with the multi-path module; w/o: without, i.e., all orders are processed in series. We bolded the best result.

5.6.4. Effect of the Multi-Order Neighborhood Feature Fusion Module

Table 8 compares PHGCN with a variant that replaces the attention-based fusion with a simple addition of the multi-order features. Attention-based fusion improves accuracy in all seven datasets, with gains ranging from 0.09 (PROTEINS) to 1.61 percentage points (MUTAG). The smaller gains on PROTEINS suggest that the multi-path features already carry comparable discriminative information on this dataset, while the fusion still matters on the remaining benchmarks.

Table 8. Graph classification accuracies of PHGCN with and without the multi-order neighborhood feature fusion module (%).

Feature Fusion	IMDB-B	COLLAB	MUTAG	PROTEINS	NCI1	DD	IMDB-M
w	**80.00 ± 2.14**	**83.76 ± 2.17**	**89.97 ± 8.29**	**80.32 ± 2.30**	**81.24 ± 1.33**	**81.24 ± 2.62**	**55.47 ± 1.95**
w/o	78.90 ± 2.62	83.00 ± 2.09	88.36 ± 8.74	80.23 ± 2.88	80.56 ± 1.54	80.48 ± 3.24	55.00 ± 2.48

w: with attention-based fusion; w/o: without, i.e., simple addition. We bolded the best result.

Overall, every component contributes to the final performance, and the contributions are complementary. The purity of high-order neighborhoods and the sign normalization provide the largest gains, which is consistent with the theoretical analysis: duplicated edges cause feature redundancy, and fictional edges distort topological semantics. The multi-path structure and the attention-based fusion further improve the model by preserving order-specific features and adapting their weights. Because the standard deviations remain comparable across all variants and the gains are consistent in direction in all seven datasets, the results are consistent with the contribution of each component. These differences are reported descriptively, without significance testing.

6. Theoretical Analysis

6.1. Relationship Between PHGCN and Traditional GCN

Traditional GCN aggregates only first-order neighborhood information, and its feature update formula is as follows:

$$H^{(l+1)} = \sigma\left(\hat{A}\, H^{(l)}\, W^{(l)}\right), \tag{27}$$

where $\hat{A}$ is the normalized adjacency matrix with self-loops. In contrast, PHGCN expands the aggregation scope to multi-order pure neighborhoods, and its feature-learning process can be regarded as a generalized form of traditional GCN. When PHGCN uses only the k-th-order pure neighborhood (i.e., a single path), the model degrades to the following:

$$H^{(l+1)} = \sigma\left(P_k(A)\, H^{(l)}\, W^{(l)}\right), \tag{28}$$

This is consistent with the traditional GCN form, verifying that traditional GCN is a special case of PHGCN. Moreover, PHGCN introduces multi-path parallel learning and adaptive feature fusion, which enhance the expressive power of the model compared with the single-path aggregation of traditional GCN.

6.2. Purity Proof of GPHP Algorithm

It is necessary to prove that the neighborhood graph obtained by the GPHP algorithm is pure, i.e., the k-th order pure neighborhood $P_k(A)$ does not contain any edges from lower-order pure neighborhoods ($P_1(A), P_2(A), \ldots, P_{k-1}(A)$).

Proof by induction:

1. Base case ($k = 1$): The first-order pure neighborhood $P_1(A)$ is the original adjacency matrix, which has no lower-order neighborhoods, so the purity holds.
2. Inductive hypothesis: Assume that for all $i < k$, the pure neighborhood $P_i(A)$ is pure, i.e., $E(P_i(A)) \cap E(P_j(A)) = \emptyset$ for any $i \neq j$ (the intersection of edge sets is empty).
3. Inductive step: For the k-th order pure neighborhood, according to the GPHP formulation in Equation (16), $P_k(A) = \text{sign}\left(D\left(\text{sign}(A^k)\right) - \sum_{i=1}^{k-1} P_i(A)\right)$. The subtraction operation removes all edges of the lower-order pure neighborhoods from the k-th-order neighborhood matrix. The subsequent sign$(\cdot)$ function retains only non-negative elements, ensuring that the final $P_k(A)$ contains no edges from $P_1(A)$ to $P_{k-1}(A)$. Thus, $P_k(A)$ is pure, and the induction is complete.

7. Discussion

7.1. Advantages of PHGCN

The experimental results show that PHGCN achieves the best average ranking (1.3) among all compared HGCNs on the seven benchmarks. Three properties of the model explain this advantage. First, the GPHP algorithm removes duplicated and fictional edges, so the neighborhoods of different orders remain independent and complementary, preventing the model from repeatedly learning low-level features. Second, the multi-path architecture learns order-specific representations in parallel and preserves their diversity. Third, the attention-based fusion module adaptively weights each order according to the task, avoiding manually set fusion weights.

7.2. Limitations and Future Directions

The experiments also reveal the boundaries of these advantages. The performance gains saturate when the order exceeds four on most datasets, and the optimal order is dataset-dependent: COLLAB performs best at orders 4 and 6, and MUTAG at order 5, indicating that the ideal neighborhood range depends on the dataset structure. High-order aggregation is also associated with the over-smoothing phenomenon, in which node representations become increasingly similar as the propagation range grows [9]. Some high-order settings also enlarge the standard deviation, as on MUTAG (89.97 ± 8.29), which suggests that an excessive order may introduce fluctuation on small datasets. In addition, the matrix-power computation in GPHP becomes expensive on dense graphs, and the memory and time limits observed for the baselines on DD and COLLAB indicate the practical cost boundary of high-order modeling.

Three limitations bound the current study. First, the GPHP algorithm computes matrix powers, whose cost grows with graph density; sparse-matrix or approximate computation is needed for very dense graphs, and the saturation of the gains beyond order 4 observed

in Section 5.5 indicates that only a few orders are usually needed in practice. Second, the maximum order is selected by validation-based tuning. To reduce this tuning cost, an adaptive order-selection mechanism can be adopted: the search stops when the marginal gain of increasing K by one falls below a small threshold on the validation folds, or the order is predicted from structural statistics such as the average path length and the clustering coefficient. Quantifying how the optimal order relates to the graph diameter and density is a valuable direction for future work. Third, this paper evaluates graph classification only; extending PHGCN to node classification, link prediction, and graph generation, and combining it with multi-order pooling to reduce nodes and edges on large graphs, are natural next steps.

8. Conclusions

This paper proposes a PHGCN to address the critical structural impurities of traditional HGCNs, including fictional and duplicated edges in high-order neighborhood graphs. We first theoretically analyze the neighborhood purity problem to reveal the inherent defects of existing high-order GCN methods. To eliminate invalid topological interference, a GPHP algorithm is designed to extract pure and independent high-order neighborhood topologies. Furthermore, we construct a multi-path feature-learning module and an attention-adaptive fusion module to effectively exploit complementary pure high-order structural information for high-quality graph representation learning.

Theoretical analyses show that PHGCN is a generalized form of conventional GCNs and that the GPHP algorithm ensures the structural purity of high-order neighborhoods. Experiments on seven graph classification benchmarks show that PHGCN achieves the best average ranking among the compared models.

Author Contributions: Conceptualization, C.H. and Z.Z.; Methodology, C.H. and Z.Z.; Software, C.H.; Validation, C.H.; Formal analysis, C.H.; Investigation, Z.Z.; Resources, Z.Z.; Data curation, C.H. and Z.Z.; Writing–original draft, C.H.; Writing–review & editing, C.H. and Z.Z.; Visualization, Z.Z.; Supervision, Z.Z.; Project administration, Z.Z.; Funding acquisition, Z.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Natural Science Foundation of Shanghai, grant number 24ZR1403300, and the National Natural Science Foundation of China, grant number 62477006.

Data Availability Statement: The TUDataset benchmarks analyzed in this study are publicly available at https://chrsmrrs.github.io/datasets/ (accessed on 6 September 2026). The source code is available at https://anonymous.4open.science/r/PHGCN-CODE (accessed on 6 September 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chen, X.; Jian, Y.; Ke, L.; Qiu, Y.; Chen, X.; Song, Y.; Wang, H. A Deep Semantic-Aware Approach for Cantonese Rumor Detection in Social Networks with Graph Convolutional Network. *Expert Syst. Appl.* **2024**, *245*, 123007. [CrossRef]
2. Chen, Q.; Wang, Z.; Yan, M.; Yan, H.; Lin, X.; Zhou, J. DeFedGCN: Privacy-Preserving Decentralized Federated GCN for Recommender System. *IEEE Trans. Serv. Comput.* **2025**, *18*, 729–742. [CrossRef]
3. Chen, C.; Wu, Y.; Dai, Q.; Zhou, H.-Y.; Xu, M.; Yang, S.; Han, X.; Yu, Y. A Survey on Graph Neural Networks and Graph Transformers in Computer Vision: A Task-Oriented Perspective. *IEEE Trans. Pattern Anal. Mach. Intell.* **2024**, *46*, 10297–10318. [CrossRef] [PubMed]
4. Liu, C.; Yu, L.; Han, K.; Yin, G. GAM-GCN: Decomposing Graph Convolutional Networks via Generalized Additive Models. In Proceedings of the 2024 IEEE International Conference on Knowledge Graph (ICKG), Abu Dhabi, United Arab Emirates, 11–12 December 2024; pp. 202–210. [CrossRef]
5. Kang, J.; Zhang, J.; Chen, L.; Zhang, H.; Zhuo, L. RWGCN: Random Walk Graph Convolutional Network for Group Activity Recognition. *Appl. Intell.* **2025**, *55*, 368. [CrossRef]
6. Corso, G.; Stark, H.; Jegelka, S.; Jaakkola, T.; Barzilay, R. Graph Neural Networks. *Nat. Rev. Methods Prim.* **2024**, *4*, 17. [CrossRef]

7. Cini, A.; Marisca, I.; Zambon, D.; Alippi, C. Graph Deep Learning for Time Series Forecasting. *ACM Comput. Surv.* **2025**, *57*, 1–34. [CrossRef]
8. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In Proceedings of the International Conference on Learning Representations (ICLR), Toulon, France, 24–26 April 2017.
9. Jin, Y.; Zhu, X. Oversmoothing Alleviation in Graph Neural Networks: A Survey and Unified View. *Knowl. Inf. Syst.* **2025**, *67*, 11259–11285. [CrossRef]
10. Li, Q.; Han, Z.; Wu, X.-M. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018*; AAAI Press: Palo Alto, CA, USA, 2018; Volume 32, pp. 1006–1013. [CrossRef]
11. Nikolentzos, G.; Dasoulas, G.; Vazirgiannis, M. K-Hop Graph Neural Networks. *Neural Netw.* **2020**, *130*, 195–205. [CrossRef] [PubMed]
12. Lei, R.; Wang, Z.; Li, Y.; Ding, B.; Wei, Z. EvenNet: Ignoring Odd-Hop Neighbors Improves Robustness of Graph Neural Networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 28 November–9 December 2022*; Neural Information Processing Systems Foundation, Inc. (NeurIPS): San Diego, CA, USA, 2022; Volume 35, pp. 4694–4706. [CrossRef]
13. Dwivedi, V.P.; Rampášek, L.; Galkin, M.; Parviz, A.; Wolf, G.; Luu, A.T.; Beaini, D. Long Range Graph Benchmark. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track, New Orleans, LA, USA, 28 November–9 December 2022.
14. Abu-El-Haija, S.; Perozzi, B.; Kapoor, A.; Alipourfard, N.; Lerman, K.; Harutyunyan, H.; Ver Steeg, G.; Galstyan, A. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In Proceedings of the International Conference on Machine Learning (ICML), Long Beach, CA, USA, 10–15 June 2019; pp. 21–29.
15. Wang, P.; Wang, Y.; Lin, H.; Shi, J. SoGCN: Second-Order Graph Convolutional Networks. *arXiv* **2021**. [CrossRef]
16. Huang, K.; Xiao, C.; Glass, L.M.; Zitnik, M.; Sun, J. SkipGNN: Predicting Molecular Interactions with Skip-Graph Networks. *Sci. Rep.* **2020**, *10*, 21092. [CrossRef] [PubMed]
17. Zhao, J.; Dong, Y.; Ding, M.; Kharlamov, E.; Tang, J. Adaptive Diffusion in Graph Neural Networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Virtual, 6–14 December 2021*; Neural Information Processing Systems Foundation, Inc. (NeurIPS): San Diego, CA, USA, 2021; Volume 34, pp. 23321–23333.
18. Sun, C.; Zhang, M.; Hu, J.; Gu, H.; Chen, J.; Yang, M. Adaptive Graph Diffusion Networks: Compact and Expressive GNNs with Large Receptive Fields. *Artif. Intell. Rev.* **2025**, *58*, 107. [CrossRef]
19. He, L.; Bai, L.; Yang, X.; Du, H.; Liang, J. High-Order Graph Attention Network. *Inf. Sci.* **2023**, *630*, 222–234. [CrossRef]
20. Chen, J.; Gao, K.; Li, G.; He, K. NAGphormer: A Tokenized Graph Transformer for Node Classification in Large Graphs. In Proceedings of the International Conference on Learning Representations (ICLR), Kigali, Rwanda, 1–5 May 2023.
21. Feng, J.; Chen, Y.; Li, F.; Sarkar, A.; Zhang, M. How Powerful Are K-Hop Message Passing Graph Neural Networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 28 November–9 December 2022*; Neural Information Processing Systems Foundation, Inc. (NeurIPS): San Diego, CA, USA, 2022; Volume 35, pp. 4776–4790. [CrossRef]
22. Bo, D.; Shi, C.; Wang, L.; Liao, R. Specformer: Spectral Graph Neural Networks Meet Transformers. In Proceedings of the International Conference on Learning Representations (ICLR), Kigali, Rwanda, 1–5 May 2023.
23. Morris, C.; Kriege, N.M.; Bause, F.; Kersting, K.; Mutzel, P.; Neumann, M. TUDataset: A Collection of Benchmark Datasets for Learning with Graphs. In Proceedings of the ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020), Virtual, 17 July 2020.
24. Xu, K.; Hu, W.; Leskovec, J.; Jegelka, S. How Powerful Are Graph Neural Networks? In Proceedings of the International Conference on Learning Representations (ICLR), New Orleans, LA, USA, 6–9 May 2019.
25. Yuan, H.; Ji, S. StructPool: Structured Graph Pooling via Conditional Random Fields. In Proceedings of the International Conference on Learning Representations (ICLR), Virtual, 26–30 April 2020.
26. Baek, J.; Kang, M.; Hwang, S.J. Accurate Learning of Graph Representations with Graph Multiset Pooling. In Proceedings of the International Conference on Learning Representations (ICLR), Virtual, 3–7 May 2021.
27. Gao, H.; Ji, S. Graph U-Nets. In Proceedings of the International Conference on Machine Learning (ICML), Long Beach, CA, USA, 10–15 June 2019; Volume 97, pp. 2083–2092.
28. Ranjan, E.; Sanyal, S.; Talukdar, P.P. ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations. In *Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020*; AAAI Press: Palo Alto, CA, USA, 2020; Volume 34, pp. 5470–5477. [CrossRef]
29. Bianchi, F.M.; Grattarola, D.; Alippi, C. Spectral Clustering with Graph Neural Networks for Graph Pooling. In Proceedings of the International Conference on Machine Learning (ICML), Virtual, 12–18 July 2020; Volume 119, pp. 874–883.
30. Wu, J.; Chen, X.; Xu, K.; Li, S. Structural Entropy Guided Graph Hierarchical Pooling. In Proceedings of the International Conference on Machine Learning (ICML), Baltimore, MD, USA, 17–23 July 2022; Volume 162, pp. 24017–24030.

31. Rampášek, L.; Galkin, M.; Dwivedi, V.P.; Luu, A.T.; Wolf, G.; Beaini, D. Recipe for a General, Powerful, Scalable Graph Transformer. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 28 November–9 December 2022*; Neural Information Processing Systems Foundation, Inc. (NeurIPS): San Diego, CA, USA, 2022; Volume 35, pp. 14501–14515. [CrossRef]
32. Pang, S.; Zhang, K.; Wang, G.; Lin, J.C.-W.; Wang, F.; Meng, X.; Wang, S.; Zhang, Y. AF-GCN: Completing Various Graph Tasks Efficiently via Adaptive Quadratic Frequency Response Function in Graph Spectral Domain. *Inf. Sci.* **2023**, *623*, 469–480. [CrossRef]
33. Niu, C.; Pang, G.; Chen, L. Affinity Uncertainty-Based Hard Negative Mining in Graph Contrastive Learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2024**, *35*, 11681–11691. [CrossRef] [PubMed]
34. Soussia, M.; Salsabila, G.A.; Mahjoub, M.A.; Rekik, I. Reservoir-Based Graph Convolutional Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2026**, *48*, 7875–7889. [CrossRef] [PubMed]
35. Chen, C.; Wei, X.; Zhang, Y.; Ye, W. Adversarial Augmentation with Maximum Discrepancy for Graph Contrastive Learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2026**, *37*, 3559–3573. [CrossRef] [PubMed]
36. Chen, C.; Wei, X.; Yang, J.; Wang, C.; Fu, Y.; Zhang, Y.; Sun, X.; Fan, R.; Ye, W. Edge Self-Adversarial Augmentation Enhances Graph Contrastive Learning Against Neighborhood Inconsistency. In *Proceedings of the AAAI Conference on Artificial Intelligence, Singapore, 20–27 January 2026*; AAAI Press: Palo Alto, CA, USA, 2026; Volume 40, pp. 20005–20013. [CrossRef]
37. Milgram, S. The Small World Problem. *Psychol. Today* **1967**, *2*, 60–67.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.