Author/s:
Ghanbari Malkhalifeh, Mohammadreza

Title:
Towards Graph Neural Networks for Node Influence Analysis

Date:
2025

Persistent Link:
https://hdl.handle.net/11343/361522

# Towards Graph Neural Networks for Node Influence Analysis

by

Mohammadreza Ghanbari

ORCID: 0000-0001-9125-9461

A thesis submitted in total fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Engineering and IT
**THE UNIVERSITY OF MELBOURNE**

November 2025

THE UNIVERSITY OF MELBOURNE

# *Abstract*

Faculty of Engineering and IT

Doctor of Philosophy

by Mohammadreza Ghanbari
ORCID: 0000-0001-9125-9461

Graph Convolutional Networks (GCNs) are among the most widely used models for learning on graph-structured data. These deep neural networks are specifically designed to process the irregular and non-Euclidean nature of graphs, which makes them highly effective for classification and prediction tasks across various domains. A fundamental aspect of GCNs—and Graph Neural Networks (GNNs) in general—is the influence of nodes, which plays a crucial role in key operations such as message passing, graph pooling, and node ranking. Despite their successes, existing GCN-based models still face significant challenges. One major limitation in message passing mechanisms is their inability to effectively capture information from distant nodes, as traditional message-passing mechanisms primarily rely on local neighborhood aggregation. Additionally, models overlook the node features when ranking node importance, that cause suboptimal performance in node scoring tasks which require a deeper understanding of both structure and features. Furthermore, existing graph pooling techniques often treat all nodes equally and consequently fail to prioritize the most influential ones which limits their ability to preserve essential substructures during graph downsampling. To address these challenges, this thesis introduces three novel models—NACFormer, FadiGNN, and CentralityPool—each designed to improve a specific aspect of GNN-based learning.

Firstly, NACFormer is introduced to address the issue of limited information propagation in message-passing mechanisms. Traditional models update node embeddings using only locally connected neighbors which suffer from resembling node representations as the number of layers grows. NACFormer mitigates this by integrating the Transformer model and graph coarsening into the message passing mechanism, allowing distant node information to be effectively captured. The model jointly learns embeddings from multi-head attention mechanisms and a GCN trained on a coarsened version of the graph. This multi-scale approach not only improves node classification accuracy but also enhances the model's adaptability to diverse graph structures, as demonstrated by extensive experiments on real-world datasets.

Secondly, FadiGNN is proposed to tackle the problem of identifying important nodes in a graph. Existing methods either disregard node features or rely on supervised learning, which is impractical due to the lack of ground truth for node importance. FadiGNN overcomes these limitations by incorporating both node features and graph topology in an unsupervised manner. The model integrates GCNs and an adapted personalized PageRank algorithm, coupled with a carefully designed loss function to ensure convergence. Extensive experiments on node classification and active learning scenarios demonstrate that FadiGNN significantly outperforms state-of-the-art models by improving classification accuracy by an average of 7.62%.

Finally, CentralityPool is developed to enhance graph classification by refining graph pooling techniques. Existing pooling methods fail to differentiate between critical and non-critical nodes, and lead to information loss and reduced classification performance. CentralityPool employs hierarchical pooling based on centrality measures, including Personalized PageRank, Katz Centrality, Total Communicability, and Eigenvector Centrality, to selectively retain the most important nodes. To mitigate the computational burden of these centrality calculations, efficient approximation strategies are integrated, ensuring scalability without compromising accuracy. Experiments on eight benchmark datasets show that CentralityPool improves graph classification accuracy by up to 4% on average while maintaining competitive performance in large-scale node classification tasks.

Together, these three contributions advance the state of GNNs by addressing key challenges in node embedding propagation, unsupervised node importance ranking, and graph pooling. The proposed models demonstrate superior performance across diverse real-world applications.

# Declaration of Authorship

I, Mohammadreza Ghanbari, declare that this thesis titled, 'Towards Graph Neural Networks for Node Influence Analysis' and the work presented in it are my own. I confirm that:

- The thesis comprises only my original work towards the Doctor of Philosophy except where indicated in the preface;
- due acknowledgement has been made in the text to all other material used; and
- the thesis is fewer than the maximum word limit in length, exclusive of tables, maps, bibliographies and appendices as approved by the Research Higher Degrees Committee.

Signed: _____

Date: _____

# Preface

This thesis draws upon material from several articles I have authored, currently in various stages of the publishing process. They include:

- **Chapter 3:** Ghanbari, M., Bagloee, SA., Qi, J., and Sarvi, M. Multi-Scale Node Neighborhood Aggregation Via Graph Coarsening and Transformer. In International Joint Conference on Neural Networks (IJCNN), 1-8, 2025.
- **Chapter 4:** Ghanbari, M., Bagloee, SA., Qi, J., and Sarvi, M. Feature-Aware Unsupervised Detection of Important Nodes in Graphs. In International Conference on Advanced Data Mining and Applications, 98-113, 2024.
- **Chapter 5:** Ghanbari, M., Bagloee, SA., Qi, J., and Sarvi, M. CentralityPool: Centrality-aware Hierarchical Graph Pooling. IEEE Transactions on Network Sciences and Engineering, 1-13, 2025.

Other works during my PhD, not included in this thesis:

- Ghanbari, M., Bagloee, SA., Aye, Z. M., and Sarvi, M. Evaluation of incremental deep learning approach for real-time traffic prediction. In Australasian Transport Research Forum, 44th, 2023, Perth, Western Australia, Australia.
- Ghanbari, M., Bagloee, SA., Qi, J., and Sarvi, M. Evaluating Node Importance in Transportation Networks Using Graph Neural Networks and Communicability Centrality. In IEEE International Conference on Intelligent Transportation Systems (ITSC), 2025.

# *Acknowledgements*

I would like to express my deepest gratitude to my primary supervisor, Prof. Majid Sarvi, for his unwavering support, insightful guidance, and invaluable expertise throughout this research. His encouragement and constructive feedback have been instrumental in shaping my work. I am also sincerely grateful to my co-supervisors, A/Prof. Qi, and Dr. Asadi for their feedback and thoughtful insights that have greatly refined my approach.

I am profoundly thankful to my family and friends for their unwavering encouragement, love, and patience throughout this journey. Their belief in me has been a constant source of strength.

I also extend my appreciation to my colleagues and fellow students for their stimulating discussions, valuable insights, and continuous support. Their collaboration has enriched my research experience. Additionally, I am grateful to the professionals in this field who have generously shared their knowledge and guidance, particularly during the early stages of this research.

This work is the result of collective effort, and I am truly appreciative of the contributions, support, and encouragement of all those mentioned above.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In today's digital age, we are surrounded by an overwhelming flow of information. Every interaction on social media, every sensor in our smart devices, and every online transaction contributes to a vast and ever-growing sea of data. This explosion of information offers incredible opportunities for insights and innovation, however, it also presents significant challenges in processing and understanding complex data structures [4, 5]. On the other hand, a substantial portion of information is inherently structured as graphs that captures the intricate relationships and interdependencies among various entities. Graph data structures consist of nodes (representing entities) and edges (depicting relationships between these entities), which makes them particularly suitable for modeling intricate systems. For instance, in social networks, nodes can represent users, while edges represent friendships or interactions [6]. In biological networks, nodes represent proteins, with edges indicating interactions [7]. This structural representation allows for a more detailed understanding of the data while facilitates advanced analyses.

Graph representation learning is a field within machine learning that focuses on transforming graph-structured data into low-dimensional vector embeddings to effectively learn the structural and feature information of graphs [8–11]. By learning these embeddings, models can better understand and exploit the complex relationships inherent in graphs and leads to improved performance in various applications.

To harness the full potential of graph-structured data, Graph Neural Networks (GNNs) have revolutionized the field of graph representation learning as a powerful tool. GNNs belong to the class of deep learning algorithms while specifically designed to operate on graph datasets to effectively handle the dependencies and relationships between nodes [12–14]. GNNs facilitates the application of graph learning and enable tasks such as node classification [15], graph classification [16], node ranking [17] link prediction [18–20], clustering [21, 22], and community detection [23, 24]. Together, these tasks form the backbone of many real-world GNN applications and highlight the importance of developing

Among these tasks, node classification, graph classification, and node ranking are especially important due to their broad applicability across domains. Node classification supports applications such as fraud detection, citation analysis, and user behavior modeling [15, 25–28]. Graph classification is fundamental for tasks like molecular property prediction, and protein function analysis [16, 29–31]. Node ranking, on the other hand,

is critical for influence maximization, recommendation, and identifying key actors in complex networks [17, 32]. Together, these tasks form the backbone of many real-world GNN applications and highlight the importance of developing.

In node classification, the task is to assign a category or label to each node in a graph by leveraging both its individual features and the information embedded in its connections. Modern GNN methods use message passing to aggregate information from connected nodes, which enables more accurate predictions [15, 25, 33, 34].

In graph classification, the goal is to predict a single label for an entire graph rather than individual nodes. The standard pipeline involves learning node embeddings and then applying a pooling or readout mechanism, such as hierarchical pooling, to generate a coarsened graph-level representation [16, 29, 35, 36].

In node ranking, the task is to compute an importance score for each node and order the nodes based on their influence or relevance within the graph. Unlike node classification, which produces discrete labels, node ranking yields continuous scores that reflect structural roles such as centrality, connectivity, or impact on information flow [17, 32].

Across all of these tasks including the node classification, graph classification, and node ranking, a common foundation is the ability to understand how influential each node is within the graph. Node-influence analysis helps to identify which nodes carry the most important structural or informational features. This provides an essential way to find and select these nodes for use in the message-passing mechanism [37], the pooling process [38], and node ranking tasks. By recognizing influential nodes, models can learn more meaningful representations, preserve key substructures, and achieve better performance across all three tasks.

Central to their effectiveness in node classification task, is the message-passing mechanism [37, 39], which enables nodes to iteratively exchange information with their neighbors. This iterative process allows each node to update its representation embedding by aggregating messages (information) received from its neighbors and result in capturing both local and global structural information within the graph. The ability to integrate node features as well as structural topology through message passing positions makes GNNs a powerful tools for analyzing and extracting insights from complex graph-structured data [40–44].

**Challenge 1.** Although GNNs with their message passing mechanism have achieved remarkable performance in node classification tasks, they face significant difficulties in capturing long-range node information within a graph. The challenge lies in the inherent limitation of message passing, which primarily focuses on local neighborhoods, and makes it difficult to preserve and propagate information from distant nodes. GNNs suffer from the over-smoothing problem as more layers are added. This phenomenon causes the embeddings of the nodes to become indistinguishable, which reduces the ability of the model to differentiate between the nodes [45, 46]. As a result, traditional GNNs struggle to effectively model relationships between distant node pairs, which limits their applicability in tasks that require a deeper understanding of global graph structure.

In node ranking methods, nodes within a graph are assigned scores based on their relative importance. These methods can be broadly classified into two categories: traditional approaches and learning-based models. Learning-based models use learning algorithms such as GNNs to integrate both structural information and node information to improve ranking performance [17, 32].

**Challenge 2.** These learning-based approaches are generally supervised, which requires labeled data for training. The challenge arises from the absence of ground truth labels, as there are no predefined labels available [47].

In graph classification task, graph pooling operations are a key components [2, 38, 48]. These techniques are used to reduce the size of a graph during the learning by aggregating node information and aim to preserve its essential structural and most informative nodes. This process enables the model to handle larger graphs efficiently and improve generalization by focusing on the most informative nodes or substructures. Therefore, accurately identifying and prioritizing important nodes in graph classification is essential to improve model performance and preserve critical structural information.

**Challenge 3.** Although GNNs with pooling techniques have demonstrated exceptional performance in tasks such as graph classification, current pooling methods often treat all node information irrespective of their importance, which leads to the loss of critical graph substructures and key node information, and ultimately reducing the model's ability to capture meaningful patterns [49]. The challenge lies in developing graph pooling methods that can effectively differentiate between important and less relevant nodes.

Building on the reasons and challenges mentioned above, this thesis seeks to advance graph learning methods across three primary tasks: node classification, graph classification, and node ranking, with a particular focus on node influence analysis. A central aspect of the methodology presented in this work is the use of centrality techniques [50, 51], which provide a powerful tool for analysing the importance of nodes within a graph. By integrating centrality measures into the analysis, this research aims to reframe GNNs through the lens of centrality, to enhance our understanding of node importance and its impact on graph-based tasks.

The contributions are grounded in recognizing and deeply examining existing gaps and are accomplished through detailed evaluations focused on nodes analysis evaluation.

## 1.1 Research Questions

In this research we are looking for proper answers to the below questions:

- **Q1.** How to identify and effectively employ distant neighbors, along with other critical nodes, to update a node's embedding in the graph filtering process? This question addresses the challenge of selecting and incorporating distant and critical nodes in the graph filtering (message passing) mechanism to improve node embeddings and enhance model performance.
- **Q2.** How can nodes in a graph be effectively ranked using an unsupervised GNN-based approach? This question focuses on developing a GNN-based model capable of scoring nodes within a graph in an unsupervised manner, addressing the challenge posed by the absence of labeled data.
- **Q3.** How to consider influence of important nodes in a graph classification task to effectively reduce the graph's size while preserving essential information? This question explores methods for incorporating the influence of the most informative nodes into hierarchical graph pooling approaches to improve both accuracy and computational efficiency.

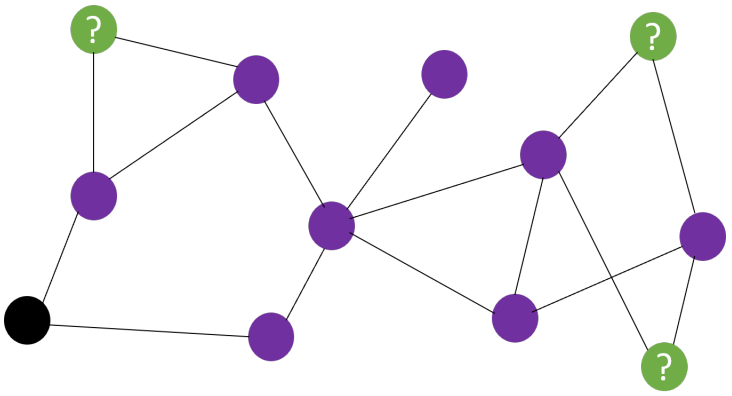



FIGURE 1.1: Effect of green points in predicting the value of black point

## 1.2 Research Aims and Objectives

To address the research questions, our goal is to propose novel architectures that can efficiently identify informative nodes within a graph for key tasks, including graph filtering, graph pooling, and node ranking.

Figure 1.2 illustrate the research objective components.

The first objective is based on improving the GNN architectures ability to capture and integrate information from both nearby and distant nodes. Traditional message-passing frameworks (as the core component of GNN models) struggle with oversmoothing phenomena that cause weakening their ability to represent complex graph structures. This work aims to address these limitations by enabling finding, selecting, and adding the influence distant nodes information and ultimately enhancing the performance of GNNs in graph-based learning tasks.

The second objective is to design a learning-based framework that ranks node importance by combining centrality techniques with GNNs. Instead of relying solely on plain structural centrality measures, which are fixed, non-learnable, and ignore node features—the goal is to build an unsupervised, feature-aware model that updates node representations during learning. This allows the framework to produce accurate, and adaptable, node importance scores across different graphs.

The third objective is to develop a centrality-driven graph pooling approach for the task of graph classification. By integrating centrality measures into pooling processes,

the goal is to enable effective node selection and feature aggregation to preserve key structural sub graph and remain scalable for large, real-world graphs.

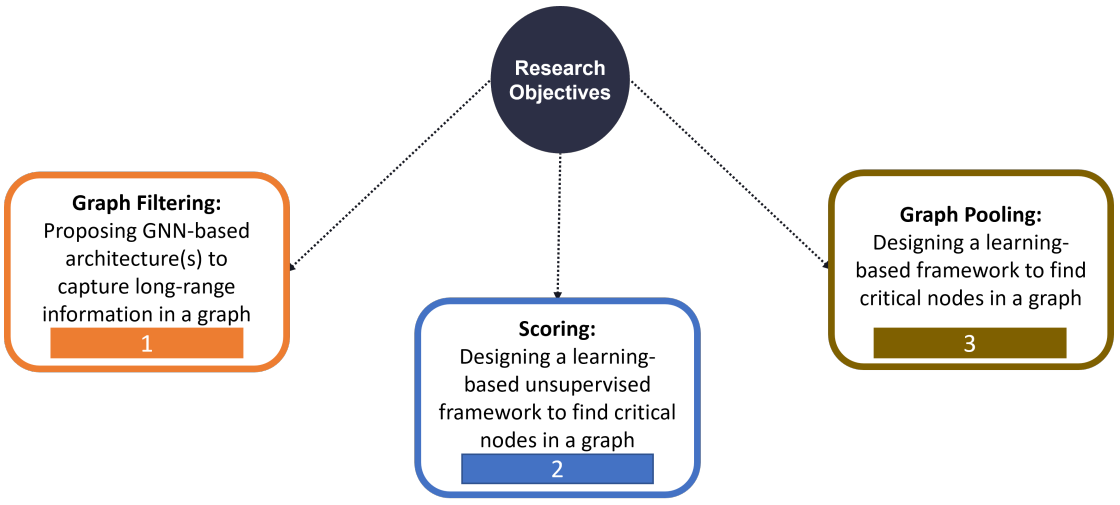



FIGURE 1.2: Research Objectives

## 1.3 Research Contributions

In this thesis, our contribution lies in the development of novel architectures that enhance the identification of informative nodes within a graph. These architectures are specifically designed to improve key tasks such as graph filtering, graph pooling, and node ranking. Our contribution lies in the design and integration of GNNs architectures with a focus on node analysis techniques, such as centrality techniques, to enhance their performance and ability to capture important node information. The key innovations and their impacts are briefly discussed as follows.

### 1.3.1 Graph Filtering: NACFormer

To address **Q1** and effectively aggregate information from both near and distant nodes, we developed NACFormer.

NACFormer enhances GNNs by integrating a multi-head attention mechanism, which allows it to selectively aggregate information from both local and distant nodes. This approach mitigates the oversmoothing problem by ensuring that node features retain their uniqueness even after multiple propagation steps. Unlike conventional GCNs that

rely solely on fixed neighborhood aggregation, NACFormer dynamically adjusts the importance of different nodes during message passing by enhancing information flow across the graph.

Additionally, NACFormer employs graph coarsening, a technique that reduces the complexity of large-scale graphs while preserving their essential structures. By constructing a set of coarsened graphs, NACFormer efficiently captures multi-scale dependencies and enables more effective long-range information propagation. This representation not only reduces computational overhead but also enhances the model's ability to generalize across diverse graph structures.

Experimental results demonstrate that NACFormer and its version NACFormer$_{MS}$ outperform state-of-the-art models in node classification tasks and show a superior capability in capturing both local and global structural information. The model's ability with improved representation learning makes it a promising advancement in the field of GNNs. By addressing key limitations of traditional message passing mechanisms, NACFormer paves the way for more effective graph-based learning in applications such as node classification.

### 1.3.2 Node Scoring: FadiGNN

To address **Q2**, and enabling centrality techniques specifically Personalized PageRank to effectively handle both node features and graph structures in node scoring tasks, we developed FadiGNN.

FadiGNN introduces a hybrid approach that combines GNNs with centrality techniques and ensure a more accurate and adaptable ranking of node importance. By integrating node features into the centrality computation process, it effectively bridges the gap between feature-based and topology-based importance assessment. Unlike traditional centrality measures that rely solely on structural properties, FadiGNN's learning-based approach allows for greater flexibility across different datasets and applications.

Furthermore, FadiGNN incorporates the approximation for centrality measures which makes it computationally efficient while preserving the effectiveness of centrality measures. It also benefits from an appropriately designed loss function that is compatible

with the message-passing iteration mechanism and ensures stable and efficient convergence during training. It leads to a better optimization of node representations to reflect their importance more accurately.

Empirical evaluations across various application scenarios, including node classification and active learning, demonstrate that FadiGNN consistently outperforms existing models by accurately identifying influential nodes and providing accurate rankings. Its ability to adapt to different centrality techniques makes it a powerful tool for applications such as influence analysis.

### 1.3.3 Graph Pooling: CentralityPool

To address **Q3**, we introduce CentralityPool that use centrality techniques as pooling layers to retains the most influential nodes and reducing computational complexity while preserving essential graph properties. Additionally, incorporating centrality into convolution layers for the node classification task broadens the perspective.

Unlike traditional pooling methods, CentralityPool employs four widely recognized centrality techniques—Personalized PageRank, Total Communicability, Katz Centrality, and Eigenvector Centrality— individually to guide node selection during downsampling. This approach ensures that the most important nodes and subgraphs contribute to the final representation and leads to improved classification accuracy. CentralityPool extends its applicability by providing results on both hierarchical and global pooling mechanisms, which allows more flexible and adaptable for graph learning.

In addition to its role in graph pooling, we individually employ these centrality measures directly into convolution layers to enhance node classification by incorporating importance awareness into feature aggregation. Also, we employ their approximation strategies that significantly improve computational manageability and scalability to efficiently process over billion nodes/edges datasets.

Empirical results across multiple benchmarks for graph classification demonstrate that CentralityPool outperforms state-of-the-art methods by effectively preserving key structural information during the pooling process. Furthermore, its application in node classification shows superior performance and highlighting the benefits of incorporating centrality techniques in convolution mechanisms. By conducting experiments on large-scale

datasets, CentralityPool exhibited a strong scalability capability which makes it a practical solution for handling complex real-world graphs while maintaining efficiency and accuracy.

## 1.4 Thesis Organisation

The remainder of this thesis is structured as follows. Chapter 2 provides the necessary preliminaries and background. Chapter 3 introduces Nacformer: Multi-Scale Node Neighborhood Aggregation Via Graph Coarsening and Transformer, which is a carefully designed framework to incorporate distant node information based on the Transformer model and graph coarsening algorithm. Chapter 4 presents FadiGNN: Feature-Aware Unsupervised Detection of Important Nodes in Graphs, where we designed a trainable model that incorporates node feature information as well as topology structure in an unsupervised manner to score the nodes within a graph. Chapter 5 proposes CentralityPool: Centrality-aware Hierarchical Graph Pooling, a node-dropping hierarchical graph pooling approach that uses centrality measures to identify and retain the most important nodes. Finally, in Chapter 6 we summarizes the key findings of this thesis, discuss its implications, and outline potential directions for future work.

# Chapter 2

# Background

In this Chapter, we first introduce the fundamental principles of graphs, with a focus on GNNs. To address the raised questions in the context of GNNs, it is essential to first understand the fundamental principles that define these models. We begin by introducing the concept of graphs and reviewing and examining the literature on GNNs based on various methodologies and approaches that have been developed and implemented over time. This review will provide a comprehensive understanding of the current state of research on GNNs. Following this, we will focus on their application in prediction and classification tasks. By analyzing existing GNN models, we aim to identify their strengths and limitations.

## 2.1 Basic Concepts

In this section, we first introduce the fundamental principles of graphs, which serve as a powerful framework for representing and analyzing complex relationships between entities. A graph consists of nodes, which represent individual entities, and edges, which represent the relationships or interactions between these entities.

### 2.1.1 Graphs

Graph theory is a branch of mathematics that focuses on the study of graphs, which are structures used to model pairwise relationships between objects. In graph theory, a graph is composed of nodes (or vertices) and edges that connect pairs of nodes. The study of graph theory involves exploring the various properties and types of graphs, as well as the relationships between the nodes within these structures [52, 53].

**Graph.** A graph is mathematically represented as $G = \{V, E\}$, where $V = \{v_1, \ldots, v_N\}$ denotes a set of nodes, with $N = |V|$ indicating the total number of nodes. The set $E = \{e_1, \ldots, e_M\}$ represents the edges, with $M$ being the total number of edges. An edge $e_{ij} \in E$ exists if and only if node $v_i$ is connected to node $v_j$. This means that there is a direct connection or relationship between these two nodes.

An equivalent representation of the graph $G = \{V, E\}$ is called adjacency matrix and is indicated with $\mathbf{A}$. The adjacency matrix $\mathbf{A}$ is a square matrix that characterizes the connectivity between the nodes in the graph. Each element in the adjacency matrix

indicates whether a pair of nodes is connected by an edge. The use of the adjacency matrix provides a convenient and efficient way to store and manipulate graph data, especially when performing computational operations. This matrix representation is crucial as it allows for the easy identification of connections and relationships within the graph.

**Adjacency Matrix.** For a given graph $G = \{V, E\}$, the corresponding adjacency matrix is denoted as $\mathbf{A} \in \{0, 1\}^{N \times N}$,

$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if there is an edge between } v_i \text{ and } v_j \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

The following matrix is the corresponding adjacency matrix of Figure 2.1.

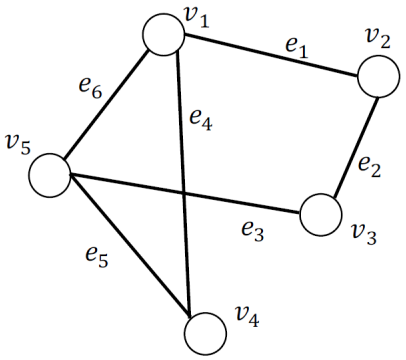



FIGURE 2.1: A graph with 5 nodes and 6 edges

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{2.2}$$

Despite its easy of use, it has several limitations, especially for large-scale and complex graph applications. Its quadratic space complexity $O(N^2)$ makes it inefficient for storing sparse graphs that leads to high memory usage and computational overhead. Additionally, it only captures direct connections between nodes and lacks higher-order structural information, which causes the requirement of extra computations. Moreover, adjacency

matrices are not invariant to node ordering, making them unsuitable for permutation-independent learning tasks, and they do not inherently encode node or edge features, which require additional feature matrices. Due to these limitations, more efficient representations such as graph Laplacians are preferred in modern graph-based learning and optimization tasks.

**Graph Laplacian.** The graph Laplacian is a fundamental matrix in spectral graph theory which is used to analyze the structure of a graph and its connectivity properties. Given a graph $G = (V, E)$ with an adjacency matrix $\mathbf{A}$ and a degree matrix $\mathbf{D}$, the combinatorial Laplacian is defined as: $\mathbf{L} = \mathbf{D} - \mathbf{A}$. This matrix captures important structural information: its eigenvalues reveal properties such as graph connectivity, while its eigenvectors form the basis for spectral clustering and graph-based learning. A key property of $\mathbf{L}$ is that it is positive semi-definite, meaning that all of its eigenvalues are non-negative. The smallest eigenvalue is always zero, and the number of zero eigenvalues corresponds to the number of connected components in the graph. A normalized Laplacian, defined as $L_{\text{sym}} = I - D^{-1/2}AD^{-1/2}$, is often used in GCN to ensure numerical stability and improve feature propagation.

**Types of Graphs.** Generally, there are two primary categories of graphs: directed graphs and undirected graphs. In directed graphs, also known as digraphs, the edges have a direction. It means that each edge points from one node to another that indicates a one-way relationship. This is commonly represented with arrows showing the direction of the connection between nodes. Directed graphs are particularly useful for modeling scenarios where relationships are not reciprocal, such as one-way streets in traffic networks or hierarchical structures. On the other hand, undirected graphs consist of edges that do not have a direction which indicates a bidirectional relationship between nodes. In undirected graphs, the connections are mutual It means that if node $a$ is connected to node $b$, then node $b$ is also connected to node $a$. This type of graph is useful for modeling situations where relationships are inherently two-way, such as friendships in social networks or undirected pathways in transportation systems. Hence, for an undirected graph, its corresponding adjacency matrix is symmetric. Both directed and undirected graphs have extensive applications in various fields. Understanding the fundamental concepts and classifications of graphs is essential for analyzing and solving complex problems that involve interconnected entities [54].

**Common Graph Properties.** Graphs have several properties that are critical for understanding their structure and behavior. These properties are used in a variety of graph-based algorithms and applications. For instance, **Connectivity** refers to the degree to which nodes in the graph are connected. A graph is said to be connected if there is a path between every pair of nodes, which means no node is isolated. For undirected graphs, this property ensures that information can flow throughout the network. If a graph is not connected, it consists of multiple connected components, and each of the components is a subgraph where any two nodes are connected by paths. **Centrality** is a measure of the importance of a node within a graph. Several centrality measures exist, where each captures different aspects of importance. Since centrality measures are a key component of the methodology used in this thesis (two chapters), we place greater emphasis on them and provide a more detailed explanation.

### 2.1.2 Centrality Measures

Centrality measures are essential tools for identifying the most important or influential nodes within a graph [50, 51, 55–57]. These measures assess the relative importance of nodes based on their position and connectivity within the network, where each offers a unique ways to quantify and capture node significance. In this thesis, we consider some of the most well-known centrality measures as follows:

- **Degree Centrality [58, 59]:** Degree centrality evaluates the importance of a node based on the number of immediate neighbors it is directly connected to. This measure is intuitive and computationally simple which makes it a common baseline in network analysis. Nodes with higher degrees are assumed to be more influential because they have more direct access to other nodes. However, degree centrality offers only a local perspective since it does not consider how influential those neighbors are. As a result, it may overlook nodes that play important roles beyond their immediate links.

- **Eigenvector Centrality [57, 60]:** Eigenvector centrality builds upon degree centrality by assigning higher importance to nodes connected to other highly ranked nodes. Instead of merely counting connections, it propagates influence through the network and gives greater weight to nodes embedded in structurally

important regions. This makes it more reflective of broader network topology and capable of capturing hierarchical influence patterns.

- **Katz Centrality [56, 61]:** Katz centrality generalizes eigenvector centrality by incorporating both direct and indirect paths of all lengths, weighted by a decay factor. This ensures that even distant nodes contribute to the score, but with progressively diminishing influence. This formulation overcomes the limitations of eigenvector centrality by assigning non-zero scores to isolated or peripheral nodes and makes it more stable in real-world graphs where disconnected components are common. It offers a more flexible representation of influence by integrating long-range information.

- **Betweenness Centrality [55, 58]:** Betweenness centrality focuses on identifying nodes that frequently lie on the shortest paths between pairs of other nodes. These nodes act as bridges that control communication and information flow within the network. However, betweenness centrality is computationally expensive and sensitive to small structural changes, and it does not account for alternative longer paths that may also contribute to influence.

- **Total Communicability Centrality [62]:** Total Communicability evaluates how effectively a node can spread information through all possible walks in the graph, not just shortest paths or immediate neighbors. By using the matrix exponential, it aggregates contributions from walks of every length and with shorter walks weighted more heavily. This provides a rich, global representation of influence across complex networks. Compared to eigenvector centrality, it captures a broader spectrum of structural connectivity but is more computationally demanding and require approximations for large-scale graphs.

- **Personalized PageRank [63, 64]:** Personalized PageRank centrality is a variation of the standard PageRank algorithm that personalizes importance scores based on a given preference vector. This makes node importance dependent not only on graph structure but also on contextual relevance or user-defined interests.

## 2.2 Graph Neural Networks

Graph Neural Networks (GNNs) are a specialized type of neural networks designed to process and analyze data that is structured as a graph, which consists of nodes (representing entities) and edges (representing relationships or interactions between these entities). Unlike traditional neural networks that operate on grid-like or sequential data, such as images or text, GNNs are capable of using the irregular, interconnected nature of graphs to learn meaningful representations. This ability makes them highly effective for a wide range of applications, including social network analysis [65–67], biological modeling [7, 68–70], recommendation systems [41, 71], citation graphs [72–74], transportation network [75–77], and etc.

Two of the primary tasks in GNNs are node classification and graph classification.

- **Node classification** involves predicting labels for individual nodes in a graph by learning from their features and neighboring nodes. For example, in social networks, the task could be to classify users as either bots or humans based on their interaction patterns. [15, 25–28, 78].
- **Graph classification** is about predicting a label for an entire graph by aggregating node features and graph-level information. For example, in chemistry, it could be used to classify molecules as toxic or non-toxic based on their molecular structure [16, 29–31].

GNNs rely on two fundamental components for each of these tasks: graph filtering for node classification and graph pooling for graph classification. These components play a crucial role in GNNs to learn meaningful representations of graph-structured data. Graph filtering enables information to propagate across node neighborhoods and allows each node's representation to be updated by its neighbors. This process helps capture structural patterns within the graph and enhances their ability to represent relationships within the network which ultimately leads to more informative node embeddings.

On the other hand, graph pooling serves as a mechanism for reducing the size of the graph while retaining its most important structural components. Pooling techniques achieve this by either merging nodes into supernodes or selectively retaining the most

relevant nodes, which improves computational efficiency and helps the model generalize better. By reducing the complexity of large-scale graphs, pooling approach enables GNNs to handle graph-level tasks, such as graph classification, where the goal is to derive meaningful insights from entire graph structures rather than individual nodes.

Together, graph filtering for propagating information across connected nodes, and graph pooling for selectively condensing key features are the two fundamental operations that enable GNNs to win the contest in node and graph classification tasks. The following sections explore these two components in detail, discussing their mechanisms, and advancements in recent research.

## 2.3 Graph Filtering

Graph filtering is a fundamental component of GNNs and plays a key role in how information is propagated across a graph. Broadly, it can be classified into two main types: spectral-based filters and spatial-based filters. These approaches define different methodologies for constructing graph filters, each with its own advantages and challenges in capturing structural dependencies and node relationships.

### 2.3.1 Spectral Graph Filtering

The spectral graph filter is based on spectral graph theory, a mathematical framework that analyzes graphs through the eigenvalues and eigenvectors of their adjacency or Laplacian matrices [79]. This approach provides a frequency-based perspective on graph-structured data and enables more global transformations.

The foundation of spectral graph filtering lies in the Graph Fourier Transform (GFT), which decomposes a graph signal into its spectral components [80]. Given a signal $\mathbf{f} \in \mathbf{R}^N$ defined on the nodes of a graph, the GFT computes its representation in the frequency domain using the eigenvectors of the graph Laplacian. This transformation allows the signal to be expressed in terms of different frequency modes, and provides insight into how information propagates across the graph.

**Graph Fourier Transform (GFT).** The Graph Fourier Transform of a signal $\mathbf{f} \in \mathbf{R}^N$ defined on a graph $G$ is defined as follows:

$$\hat{\mathbf{f}} = \mathbf{U}^\top \mathbf{f} \tag{2.3}$$

where $\mathbf{U}$ is computed of the eigen-vectors of the Laplacian of graph $G$, and $\hat{\mathbf{f}}$ is the coefficients of graph's Fourier.

To modify a graph signal, a graph filter is applied in the spectral domain by modulating the Fourier coefficients of $\mathbf{f}$. This process enables selective enhancement or suppression of specific frequency components and allows for precise control over the information flow within the network. Low-frequency components capture smooth, global structures, while high-frequency components highlight localized variations. The ability to manipulate these spectral properties makes spectral graph filtering a powerful tool for tasks such as node classification and graph classification.

Despite its strong theoretical foundation, spectral filtering often suffers from computational limitations, particularly in large graphs where eigenvector calculations are costly. To address this, various approximation techniques, such as Chebyshev polynomials and localized spectral filters, have been developed to improve scalability while preserving the effectiveness of spectral-based filtering [72].

In order to modulate the frequency components of a graph signal $\mathbf{f} \in \mathbf{R}^N$, a filter is applied to its Fourier coefficients. This filtering process is defined as $\hat{g}(\Lambda) \cdot \mathbf{U}^\top \mathbf{f}$ where:

- $\mathbf{U} \in \mathbf{R}^{N \times N}$ is the matrix of eigenvectors of the graph Laplacian (or another graph-related operator),
- $\Lambda \in \mathbf{R}^{N \times N}$ is a diagonal matrix whose entries are the corresponding eigenvalues.

The function $\hat{g}(\cdot)$ is applied element-wise to the diagonal entries of $\Lambda$ and effectively modulates each frequency component of the graph signal. To illustrate, the matrix $\Lambda$ is defined as:

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_N \end{pmatrix} \tag{2.4}$$

and after applying the function $\hat{g}(\cdot)$, we obtain:

$$\hat{g}(\Lambda) = \begin{pmatrix} \hat{g}(\lambda_1) & 0 & \cdots & 0 \\ 0 & \hat{g}(\lambda_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{g}(\lambda_N) \end{pmatrix} \tag{2.5}$$

This operation allows for precise control over the frequency content of the graph signal. By choosing an appropriate function $\hat{g}(\cdot)$, one can design filters that, for example, reducing high-frequency noise (low-pass filtering) or emphasize rapid variations (high-pass filtering). Once the Fourier coefficients have been modulated, the inverse graph Fourier transform is typically applied to convert the filtered signal back into the vertex domain. This reconstruction is given by:

$$\mathbf{f}_{\text{filtered}} = \mathbf{U}\,\hat{g}(\Lambda)\,\mathbf{U}^{\top}\mathbf{f} \tag{2.6}$$

Thus, the overall process consists of the following steps:

1. Transform the signal $\mathbf{f}$ into the spectral domain using the GFT: $\hat{\mathbf{f}} = \mathbf{U}^{\top}\mathbf{f}$.
2. Modulate the spectral coefficients using the filter function: $\hat{g}(\Lambda)\,\hat{\mathbf{f}}$.
3. Transform the modulated signal back to the vertex domain to obtain the filtered signal $\mathbf{f}_{\text{filtered}}$.

Figure 2.2 illustrate the process of spectral filtering.

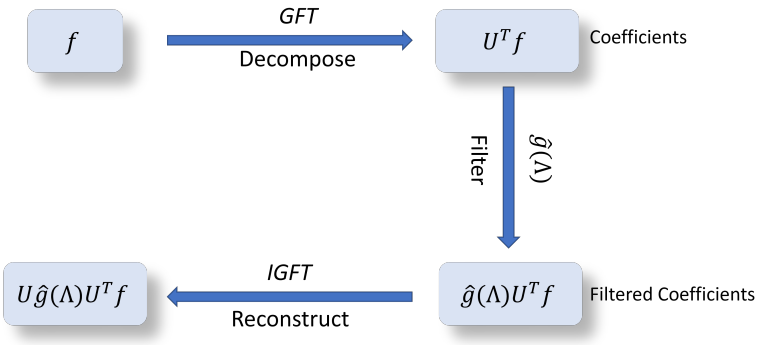



FIGURE 2.2: The process of spectral filtering.

**Designing the Graph Filter Function $\hat{g}(\Lambda)$.** Now, the key question in spectral graph filtering is how to design the filter function $\hat{g}(\Lambda)$ to effectively modulate the frequency components of a graph signal. The choice of $\hat{g}(\Lambda)$ determines the nature of the filtering operation, such as whether it acts as a low-pass, high-pass, or band-pass filter.

Bruna et al. [81] proposed an approach that provides a complete flexibility in designing graph filters by allowing them to be entirely learned from data. In this formulation, the filter function is parameterized as a diagonal matrix:

$$\hat{g}(\Lambda) = \begin{pmatrix} \mathbf{w}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{w}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{w}_N \end{pmatrix} \tag{2.7}$$

where each $\mathbf{w}_i$ is a learnable parameter corresponding to a specific eigenvalue $\lambda_i$ of the graph Laplacian. This approach provides complete freedom in defining how different frequency components should be modulated.

However, despite its flexibility, this method has a major computational drawback: it requires an explicit eigenvalue decomposition of the Laplacian matrix, which is an $\mathcal{O}(N^3)$ operation. This high computational cost makes it impractical for large graphs, where $N$ (the number of nodes) can be very large.

**Polynomial Approximation for Efficient Filtering** To overcome the computational burden of explicit eigen decomposition, Defferrard et al. [72] proposed using polynomial approximations of the filter function $\hat{g}(\lambda)$. Instead of learning individual parameters for each eigenvalue, the function is approximated as a truncated polynomial expansion in terms of the Laplacian matrix:

$$\hat{g}(\lambda) \approx \sum_{k=0}^{K} \mathbf{w}_k \lambda^k \tag{2.8}$$

By using this polynomial approximation, the spectral filtering operation can be rewritten as:

$$\mathbf{U} \cdot \hat{g}(\Lambda) \cdot \mathbf{U}^\top \mathbf{f} = \sum_{k=0}^{K} \mathbf{w}_k \mathbf{L}^k \mathbf{f} \tag{2.9}$$

Here, $\mathbf{L}$ is the normalized graph Laplacian, and the filter function is now expressed as a polynomial of the Laplacian, which eliminates the need for explicit eigen decomposition. Instead of computing the eigenvectors of $\mathbf{L}$, we directly apply powers of $\mathbf{L}$ to the signal, which can be computed efficiently using matrix multiplications.

Using polynomials to approximate graph filters offers several advantages, including computational efficiency, since the filtering operation now scales linearly with the number of nodes, making it significantly more efficient than eigen decomposition.

The problem with the spectral filtering approach, despite its simplicity, is that the polynomial expansion of the equation $\sum_{k=0}^{\infty} \mathbf{w}_k \mathbf{x}^k$ is inherently unstable due to the non-orthogonality of the standard monomial basis $\{x^k\}$. This lack of orthogonality results in poor numerical conditioning, making the filtering process highly sensitive to small variations in eigenvalues. Consequently, applying high-degree polynomials can lead to overfitting, and inefficient computations, particularly for large graphs.

**Chebyshev Polynomials: A More Stable Basis** To address these stability issues, Defferrard et al. [72] proposed the use of Chebyshev polynomials, which form an orthogonal basis over the interval $[-1, 1]$. Unlike the standard monomials, Chebyshev polynomials provide a well-conditioned approximation space, which reduce numerical instability and improving computational efficiency.

The Chebyshev polynomials are defined recursively as follows:

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), \quad \text{for } k \geq 2 \tag{2.10}$$

This recurrence relation allows for efficient computation without requiring explicit matrix diagonalization. Using Chebyshev polynomials, the spectral filtering operation can be rewritten in terms of a truncated Chebyshev expansion:

$$\mathbf{U}\hat{g}(\Lambda)\mathbf{U}^T\mathbf{f} = \sum_{k=0}^{K} \mathbf{w}_k T_k(\mathbf{L})\mathbf{f} \tag{2.11}$$

Where, $T_k(\mathbf{L})$ is the Chebyshev polynomial of order $k$ applied to the normalized Laplacian matrix $\mathbf{L}$, and $\mathbf{w}_k$ are the learnable parameters that define the spectral filter.

By adopting the Chebyshev polynomial approximation, spectral graph filtering benefits from several key improvements:

- **Numerical Stability:** The orthogonality of Chebyshev polynomials mitigates the instability issues present in standard polynomial expansions, and ensures smoother and more reliable filter learning.
- **Computational Efficiency:** The filtering operation can be computed recursively, eliminating the need for explicit eigen decomposition. This reduces the complexity from $\mathcal{O}(N^3)$, and makes it scalable for large graphs.
- **Localized Filtering:** The approximation depends only on $K$-hop neighbors, meaning the operation remains localized in the node space. This is particularly useful for tasks requiring local feature extraction.

### 2.3.2 Spatial Graph Filtering

The use of Chebyshev polynomials in spectral graph filtering plays a crucial role in improving the scalability of GNNs. By reducing computational overhead and enhancing numerical stability, this approach enables the application of spectral-based GNNs to large-scale real-world graphs.

To further simplify the Chebyshev polynomial approximation, Kipf et al. [25] proposed the Graph Convolutional Network (GCN), which effectively reduces the complexity of the spectral filter by setting $K = 1$. This truncation of the ChebNet leads to a simplified filter defined as:

$$\hat{g}(\Lambda) = \mathbf{w}_0 + \mathbf{w}_1(\Lambda - I) \tag{2.12}$$

where $\Lambda$ represents the diagonal matrix of eigenvalues, and $I$ is the identity matrix.

By imposing the constraint $\mathbf{W} = \mathbf{w}_0 = -\mathbf{w}_1$, the filter expression simplifies further to:

$$\hat{g}(\Lambda) = \mathbf{W}(2I - \Lambda) \tag{2.13}$$

This formulation allows the spectral convolution to be efficiently implemented without explicitly computing eigenvectors. With an additional regularization step, the spectral

filtering operation can be re-expressed in the vertex domain as:

$$U\hat{g}(\Lambda)U^T\mathbf{f} = \mathbf{W}\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\right)\mathbf{f} \tag{2.14}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + I$ is the adjacency matrix augmented with self-loops, and $\tilde{\mathbf{D}}$ is the corresponding degree matrix of $\tilde{\mathbf{A}}$. This regularization incorporates self-connections and ensures that each node's own features are included during aggregation.

Matrix $\mathbf{W}$ contains the filter weights, which serve to weigh and combine the features from neighboring nodes. Essentially, $\mathbf{W}$ can be conceptualized as a set of coefficients that determines the importance of each neighbor's features when updating a node's representation. The aggregation process thereby enables the GCN to integrate information across the graph. This iterative message-passing mechanism ensures that information propagates through multiple layers and ultimately captures both local and higher-order dependencies in the graph.

From a spatial perspective, the GCN [25] intuitively performs neighborhood aggregation, where each node gathers information from its local neighborhood to update its representation (see Figure 2.3). In this formulation, each node aggregates the feature representations of its immediate neighbors, as well as its own, before applying a linear transformation using the learnable weight matrix $\mathbf{W}$. This process ensures that each node iteratively refines its representation based on the structural information encoded in the graph. This is part of a broader framework known as **message passing**.

The **message-passing** technique is a core component of GCN architectures, where a node's embedding, or latent vector representation, is recursively updated by aggregating information from its neighboring nodes. This iterative process enables each node to gather and refine information from its local neighborhood and thereby enhancing its representation [82–84].

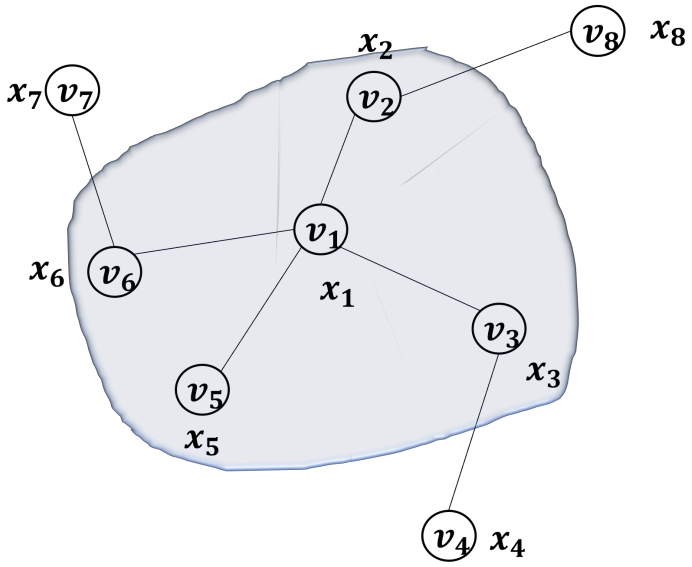



FIGURE 2.3: Overview of neighborhood aggregation, where a node updates its representation by incorporating information from its adjacent nodes. $x_i$ is the corresponding feature matrix (signal vector in spectral filtering) of node $v_i$

Message passing generalizes this idea by defining a two-step process:

- **Message Aggregation:** Each node collects and combines feature representations from its neighbors based on a predefined aggregation function, ensuring that local graph structure influences the node's updated representation.
- **Feature Update:** The aggregated information is transformed through a learnable function, often involving a weight matrix and a non-linear activation, to refine node embeddings iteratively.

This iterative approach enables nodes to incorporate information from progressively larger neighborhoods as more layers are stacked. Over multiple iterations, nodes gain a richer understanding of their local and global graph information. Fundamentally, the message passing in the GCN performs a weighted aggregation of neighborhood features followed by a feature transformation. The update rule for the $(k+1)$-th layer of a GCN is given by:

$$\mathbf{H}_i^{(k+1)} = \sigma\left(\sum_{v_j \in \mathcal{N}(v_i)\cup\{v_i\}} C[i,j]\mathbf{H}_j^{(k)}\mathbf{W}^{(k)}\right), \quad \forall v_i \in V \tag{2.15}$$

where

$$C = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}} \tag{2.16}$$

Here, $V$ is the set of nodes; $\mathcal{N}(v_i)$ is the set of neighbors of node $v_i$; $\mathbf{H}_i^{(k)}$ represents the embedding matrix of node $v_i$ at layer $k$, $\mathbf{W}^{(k)}$ is the trainable weight matrix, and $\sigma(\cdot)$ is a non-linear activation function such as ReLU. The matrix $C$ normalizes the adjacency matrix $\widetilde{\mathbf{A}}$ (which includes self-loops) using the degree matrix $\widetilde{\mathbf{D}}$ to ensure stable gradient propagation and prevent numerical instabilities.

GCN is almost on top of the list of algorithms for GNNs architectures. However, it has a notable drawback. The aggregation function is applied to every node connected to the target node which poses a significant challenge for large networks in many problems. A straightforward solution to this issue was proposed in 2017 by Hamilton et al. [28], who introduced the GraphSAGE model. Instead of aggregating information from the entire set of neighboring nodes, GraphSAGE selects a random sample of nodes (for instance, using a uniform distribution, $\mathcal{N}(v_i) \rightarrow \mathcal{N}_s(v_i)$) to perform the aggregation. This approach reduces the computational burden and makes the model more scalable for large and dense garphs.

Hence the $k+1$-th layer of GraphSAGE

$$\mathbf{H}_{\mathcal{N}_s(v_i)}^{(k+1)} = AGG\left(\left\{\mathbf{H}_i^{(k)}, v_j \in \mathcal{N}_s(v_i)\right\}\right) \tag{2.17}$$

$$\mathbf{H}_i^{(k+1)} = \sigma\left(\mathbf{W}\left[\mathbf{H}_i^{(k)}, \mathbf{H}_{\mathcal{N}_s(v_i)}^{(k+1)}\right]\right) \tag{2.18}$$

Where $\mathcal{N}_s(v_i)$ is a randomly selected neighbor of target node $v_i$.

Another model that has gained significant attention is the Graph Attention Network [27]. Graph Attention Network enhances the performance of earlier techniques by incorporating an attention mechanism that dynamically assigns learnable attention weights to each neighboring node. Specifically, each node is assigned a unique attention coefficient, $\alpha_{ij}$ in equation 2.19, which quantifies the importance of its neighbor $v_j$ in the aggregation process. This adaptive weighting allows the model to prioritize more relevant neighbors.

$$\alpha_{ij} = \frac{\exp\left(\sigma\left(\vec{\mathbf{a}}^T\left[\mathbf{W}\vec{H}_i \| \mathbf{W}\vec{H}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\sigma\left(\vec{\mathbf{a}}^T\left[\mathbf{W}\vec{H}_i \| \mathbf{W}\vec{H}_k\right]\right)\right)} \tag{2.19}$$

Where $\sigma$ is $LeakyReLU$ function, $\vec{\mathbf{a}}$ is the learnable attention vector. Figure 2.4 illustrates the GAT model updating prcess.

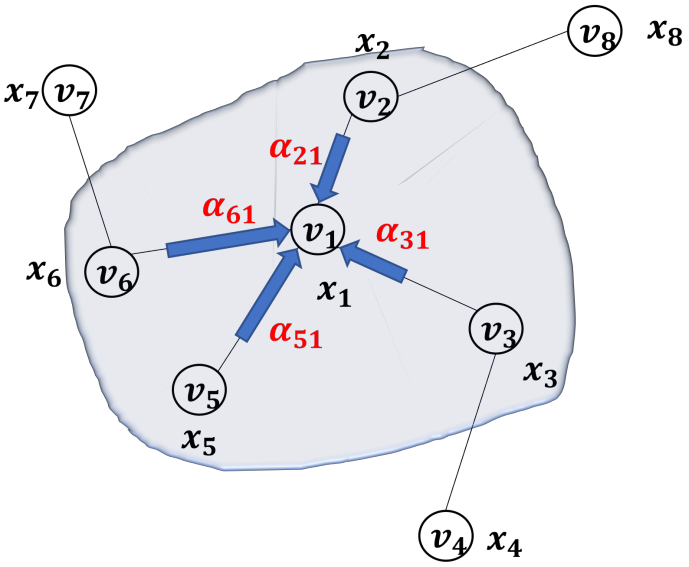



FIGURE 2.4: Graph Attention Network

We explored the concept of graph filters and various approaches that have been used to design them for effective information propagation in GNNs. While well-known GNNs models use graph filters to aggregate neighborhood information, they often face different limitations, such as over-smoothing, scalability, and so on, which makes it challenging to apply these models to large-scale graphs. To overcome these drawbacks, numerous advanced GNNs architectures have been proposed, by incorporating techniques such as residual connections [85, 86], attention mechanisms [87, 88], spectral decompositions [89–91], efficient sampling strategies [92, 93], and etc. These innovations aim to enhance model expressiveness, improving computational efficiency, and enable GNNs to scale to massive real-world graphs while maintaining meaningful node representations. As research in this field continues to evolve, new methodologies are being introduced to further optimize graph learning paradigms and address emerging challenges in complex networked data.

## 2.4 Graph Pooling

In graph classification task, handling large graph datasets efficiently requires techniques that reduce their size while preserving essential structural and feature information. Graph pooling is one such approach, designed to create hierarchical representations by progressively coarsening the graph.

Graph pooling is analogous to downsampling in convolutional neural networks (CNNs). The primary responsibility of the graph pooling layer is to decrease the size of the graph, which serves as the input for the subsequent layers (see Fig. 2.5) [94, 95]. By effectively reducing the graph's complexity, graph pooling enables the model to process large graphs more efficiently.

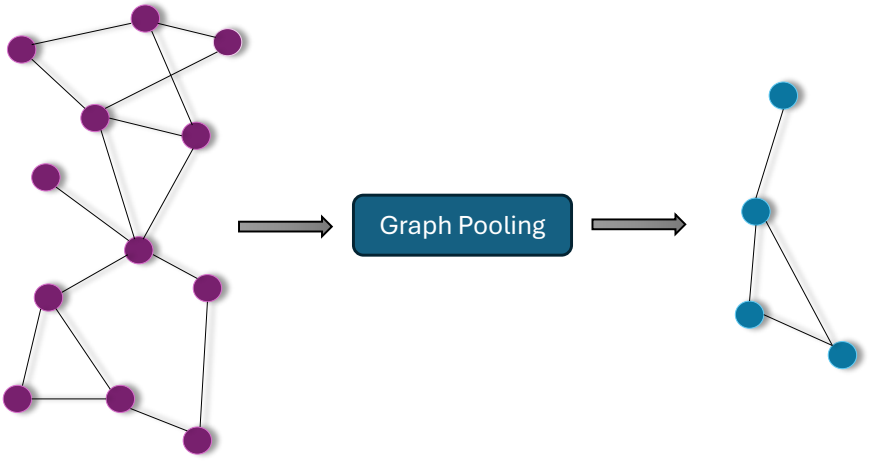



FIGURE 2.5: Graph Pooling operation reduce the size of the graph

Generally, graph pooling techniques are classified into two main classes.

1. **Global Pooling**. This method reduces the entire graph into a single representation by aggregating all node features into a fixed-size vector. It is commonly used for graph-level tasks such as classification and regression. For instance, Unlike standard pooling methods that simply aggregate individual node features—such as by taking the average or maximum of node features, Second-Order method [96] goes a step further by considering the interactions and correlations between nodes within the graph. Instead of treating nodes independently, second-order pooling captures how nodes relate to each other, whether through direct connections or more complex structural patterns. The SortPool method [97] ranks nodes based on scores derived from the transformed features of those nodes. This approach uses a form of feature transformation that allows for effective node ranking. On the other hand, the Global-Attention mechanism [98] employs an attention-based approach to selectively aggregate the features of nodes. By doing so, it enhances the overall representation of the graph, and ensures that more significant node features are emphasized in the final graph representation. Meanwhile, GMT [99]

approaches the pooling problem by framing it as a multiset encoding task, and incorporate auxiliary information to improve the representation. It utilizes multi-head attention to better capture the intricate interactions between nodes within the graph.

Global pooling methods continue to be favored in many applications due to their simplicity and computational efficiency, which make them attractive for use in graph processing tasks. However, while global pooling techniques are widely used, they do face limitations, especially when applied to large and complex graphs. In such cases, the pooling process can sometimes oversimplify the graph, failing to capture its intricate structural details. This single-step reduction approach can lead to a loss of critical information that may be important for certain tasks.

2. **Hierarchical Pooling**. This approach progressively coarsens the graph by iteratively selecting and merging nodes while preserving structural and feature information at multiple levels. It is particularly useful for learning hierarchical representations in large graphs [100, 101]. This approach mainly is devided into two categories: cluster-based pooling and node dropping pooling.

    1. **Cluster-Based Pooling** methods aggregate nodes into groups according to their similarities and structural connections. Rather than selecting or removing nodes, these techniques create clusters of closely related nodes and substitute them with a single representative node. This approach maintains significant graph structures while greatly reducing computational demands. For instance, [100] learns soft cluster assignments for nodes at each GNN layer, mapping nodes to clusters, which then form the coarsened input for subsequent layers. This method enables end-to-end training. StructPool method [102] captures high-order structural relationships between nodes through conditional random fields and effectively learns graph representations by considering the dependencies among nodes. CliquePool method [103] coarsens graphs by aggregating maximal cliques, which aim to limit node dispersal by assigning each node to a single clique-pool. K-Plex method [104] uses concepts from graph theory, specifically k-plexes, to create non-parametric pooling operations. By focusing on dense subgraphs where each node is connected to at least a certain number of others, it effectively captures structural

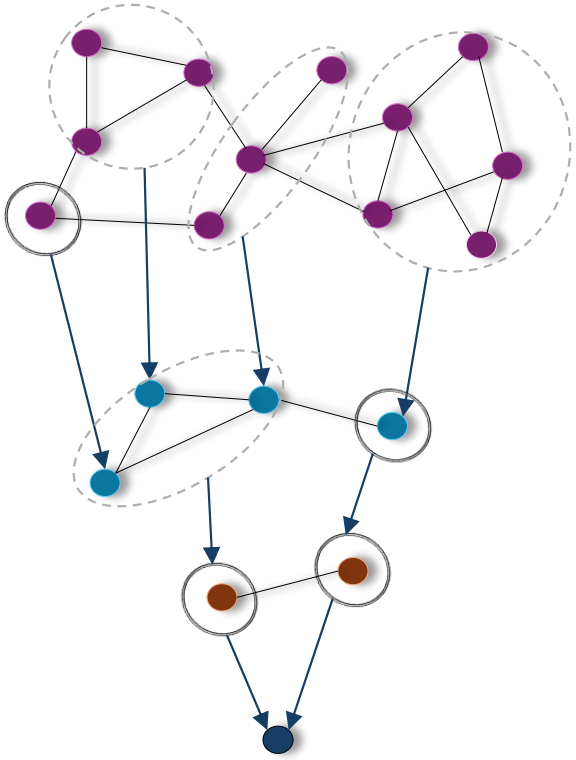


FIGURE 2.6: Clustering-based pooling methods where nodes are grouped into clusters based on their similarities and structural relationships.

information. Haar pooling model [38] is based on compressive Haar transforms, which apply a sequence of clustering operations to reduce the graph's size while preserving structural information. EigenPooling [105] integrates graph convolutional networks with a pooling operator derived from the graph Fourier transform. These examples are the notable examples of such pooling strategies, which adaptively group nodes based on both feature and structural resemblances. The procedure for cluster-based pooling of an original graph is depicted in Figure 2.6.

Beside the practical applications in real-world problems, however, these methods face several limitations, including over-smoothing of node representations, and the necessity to predefine the number of clusters, which limits flexibility. Additionally, node-clustering approaches often require auxiliary networks to learn node-cluster mappings, which introduce complexity and potential overfitting. Furthermore, the computational intensity of clustering algorithms can be a significant drawback, especially when dealing with large graphs.

2. **Node Dropping Pooling** methods discard less important nodes to reduce the graph size while retaining key structural and feature information. These methods apply scoring mechanisms that assign unique importance or relevance values to each node across the graph. After ranking the nodes based on these scores, the $top - k$ nodes with the highest values are selected for further processing. This approach efficiently reduces the graph's size while retaining the most significant nodes ans subgraphs. This method has been successfully applied in various studies to boost the efficiency of graph processing algorithms. For instance, SAGPool [2] employs a self-attention mechanism to assign importance scores to nodes and adaptively selects nodes based on their learned scores. SSPool method [1] employs a Siamese network architecture to maximize mutual information between graph representations. By using graph infomax pooling, it learns informative and discriminative features. Gpool method [106] selects a subset of nodes based on their importance, which is determined by their scalar projection values onto a learnable projection vector.

   The node dropping methods similarly face some limitation. For instance, the process of discarding nodes can lead to the loss of valuable structural and feature information. Moreover, relying solely on node significance scores for selection may overlook the diversity inherent in node features and graph structures. However, in contrast to cluster-based methods, node dropping is computationally more efficient, as it directly removes nodes rather than generating clusters, which makes it faster and less resource-intensive.

   Figure 2.7 illustrates the entire process of node drop pooling approach.

In the following sections, we provide a more detailed description of all the datasets used in our experiments. Additionally, we provide an explanation of the evaluation metrics employed to assess models performance to ensure clarity on how effectiveness was measured across different tasks.

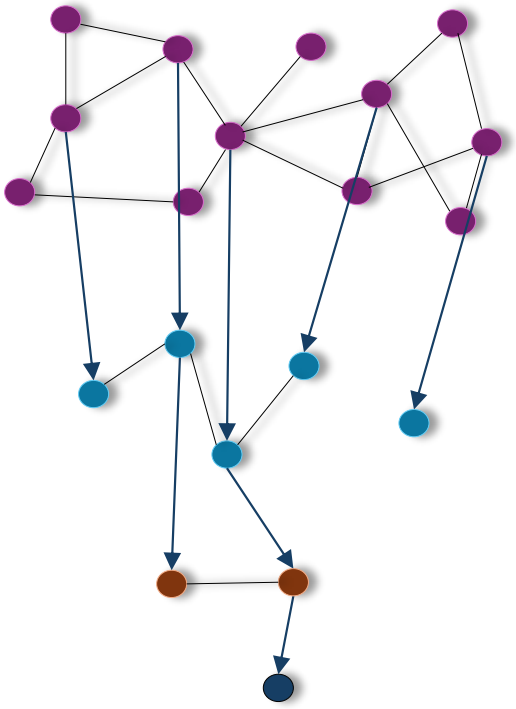


FIGURE 2.7: An illustration of Node drop pooling approach, where ranking mechanisms, such as learned importance scores or attention weights is used to determine which nodes should be removed

## 2.5 Data

For all our model experiments, we employed the following publicly available datasets. These datasets serve as standard benchmarks in graph learning research and cover a diverse range of domains, including citation networks, e-commerce co-purchase graphs, and large-scale academic graphs. Each dataset provides a unique structure and challenge, enabling a comprehensive evaluation of our models. All datasets used in this study can be accessed online [3].

### 2.5.1 Node Classification Datasets

- **Cora.** Cora is a real-world bibliographic dataset that represents documents and the citation links between them in the field of computer science [107]. The documents in this dataset are categorized into seven classes: Case-Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule

Learning, and Theory. Each class represents a distinct area of research, providing a rich and diverse set of examples for evaluating the performance of GNNs in classifying nodes based on their features and relationships.

- **CiteSeer.** CiteSeer is another citation network dataset that encompasses scientific papers in computer science [107]. The papers in CiteSeer are categorized into six research areas: Agents, Artificial Intelligence (AI), Databases (DB), Information Retrieval (IR), Machine Learning (ML), and Human-Computer Interaction (HCI). This dataset serves as an additional benchmark for assessing the ability of GNNs to accurately classify and manage documents across various specialized fields within computer science.

- **PubMed.** Pubmed is a citation graph consisting of papers related to diabetes research [107]. The papers in this dataset are divided into three categories, making it another valuable resource for evaluating the effectiveness of GNNs in handling domain-specific classification tasks.

- **DBLP.** The DBLP dataset is a comprehensive bibliographic dataset that captures the co-authorship relationships among researchers in the field of computer science. It is widely used for node classification, link prediction, and community detection tasks. The dataset includes 17,716 nodes, with each node representing an individual researcher. Edges represent co-authorship relationships, indicating that two researchers have co-authored one or more papers [108].

- **LastFM Asia.** LastFM Asia is a network of social friendships among Asian users of the music streaming service LastFM [109]. The classes in this dataset represent the home countries of the users, providing a unique challenge for GNNs to classify nodes based on social and geographical attributes.

- **Cora-ML.** The Cora-ML dataset is similar to the Cora dataset but differs in the number of nodes and edges, as detailed in Table 3.1.

- **Wiki** Wiki dataset represents a network of Wikipedia pages, where each node corresponds to a Wikipedia page, and edges represent the hyperlinks between them. The nodes are labeled based on the categories or topics of the Wikipedia pages, which are used as the target classes for node classification [110, 111].

- **BlogCatalog.** BlogCatalog is a widely used dataset for node classification tasks. It represents a network of social relationships between bloggers on the BlogCatalog website. In this dataset, each node corresponds to a blogger, and edges represent the social interactions between them. Nodes are labeled with categories that indicate the topics of the blogs, which are used as the target classes for node classification [110, 111].

- **Co-CS** Co-CS (Computer Science) dataset represents a co-authorship network of computer science researchers. The dataset contains nodes corresponding to individual researchers, and edges represent co-authorship relationships between them. The nodes are labeled based on the research topics or fields of the researchers, which serve as the target classes for node classification [112].

- **Co-phy.** The Co-Phy (Physics) dataset represents a co-authorship network of physics researchers. The dataset contains nodes corresponding to individual researchers, and edges represent co-authorship relationships between them. The nodes are labeled based on the research topics or fields of the researchers, which serve as the target classes for node classification [112].

- **OGBN-arxiv**. The OGBN-arxiv dataset is a citation network where each node represents an academic paper from arXiv, and edges denote citation relationships between these papers. Each paper is associated with a feature vector derived from word embeddings of its title and abstract. The primary task involves classifying papers into their respective research areas which makes it a widely used benchmark for evaluating graph-based learning models [113].

- **OGBN-product**. The OGBN-product dataset is an Amazon co-purchase network, where nodes represent products, and edges indicate that two products are frequently bought together. The task involves node classification, where each product is assigned a category based on its type [113].

- **OGBN-paper100M**. The OGBN-paper100M dataset is a large-sclae academic citation network consisting of over 100 million papers. Nodes represent research papers, and edges denote citation relationships. The primary goal is to classify papers into various research fields, making it one of the largest and most challenging benchmarks for graph-based learning [113].

### 2.5.2 Graph Classification Data

- **DD (D&D)** [114] is a dataset consisting of protein structure graphs, where nodes represent amino acids and edges indicate spatial proximity. It is widely used for protein classification tasks to help in analyzing structural properties and functional similarities between different proteins. The dataset is particularly useful for studying how protein structures relate to their biological functions.

- **PROTEINS** [115] is another protein structure dataset, similar to DD (D&D), where graphs model the spatial interactions of amino acids. However, it focuses on classifying proteins based on their enzymatic and non-enzymatic properties. This dataset is commonly utilized in bioinformatics research for tasks involving protein function prediction and structural analysis.

- **NCI109** [116] is a dataset of chemical compound graphs, where nodes correspond to atoms and edges represent chemical bonds. The main objective is to predict whether a given compound exhibits anti-cancer activity. It serves as a benchmark in computational drug discovery, aiding in the identification of potential chemotherapeutic agents.

- **NCI1** [116] is another chemical compound dataset, structurally similar to NCI109. It is used for the same cancer-related activity prediction task but provides a different set of molecular structures for evaluation. Both NCI1 and NCI109 are widely employed in cheminformatics to develop machine learning models for molecular property prediction.

- **Mutagenicity** [117] contains molecular graphs designed for mutagenicity prediction, where nodes denote atoms and edges represent chemical bonds. The goal is to determine whether a compound is mutagenic, which is crucial for assessing potential health risks associated with chemical substances.

- **COLLAB** [118] comprises graphs representing academic collaboration networks, where nodes correspond to researchers, and edges indicate co-authorship relationships. The classification task involves identifying different research fields based on network structures, making it useful for studying scholarly collaborations.

- **IMDB-B** [118] consists of ego-networks extracted from the Internet Movie Database (IMDB), where nodes represent actors, and edges signify co-appearances in the

same movie. The dataset is used for binary classification of movies into two distinct genres, enabling research into social network analysis and film categorization.

- **IMDB-M** [118] is an extension of IMDB-B, where movies are categorized into one of three genres instead of two. This dataset provides a more complex classification task by capturing richer social interactions among actors based on their collaborations in different film genres.

## 2.6 Evaluation Metrics

To comprehensively assess model performance, we utilize two key classification metrics: Accuracy and F1-score. Accuracy provides a straightforward measure of overall correctness, while the F1-score balances precision and recall, making it particularly useful for imbalanced datasets. By employing both metrics, we ensure a more robust evaluation of model effectiveness.

### 2.6.1 Accuracy.

The Accuracy score is a widely used metric for classification tasks, representing the proportion of correctly classified instances in relation to the total dataset. It is computed as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + TN + FP} \tag{2.20}$$

where $TP$, $TN$, $FN$, and $FP$ denote the number of true positives, true negatives, false negatives, and false positives, respectively. A higher accuracy score indicates better classification performance by reflecting the proportion of correctly predicted labels. However, accuracy alone may not be sufficient when dealing with class imbalances, as it does not differentiate between precision and recall.

### 2.6.2 F1-score.

To address limitations of Accuracy, we also evaluate models using the F1-score, which balances Precision and Recall to provide a more informative performance measure. The F1-score is the harmonic mean of precision and recall and is defined as:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.21}$$

where Precision and Recall are computed as:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN} \tag{2.22}$$

A higher F1-score indicates a better balance between precision and recall, making it particularly useful in scenarios where misclassification costs are uneven. Unlike Accuracy, the F1-score ensures that both positive and negative predictions are appropriately considered, providing a more reliable measure of model performance in cases where class distributions are skewed.

# Chapter 3

# Multi-Scale Node Neighborhood Aggregation Via Graph Coarsening and Transformer

*Graph Convolution Networks (GCNs) have shown superiority in many applications entailing classification and prediction tasks. Vanilla GCNs are based on updating a node embedding locally with directly connected neighbors and not considering distant nodes. To address this issue, a straightforward solution is to increase the number of GCN layers, letting messages (i.e., information) from distant nodes to be propagated to a target node after a few iterations. Such an approach, however, suffers from resembling of node representations. In this chapter, we propose a model named NACFormer that incorporates distant node information based on the Transformer model and graph coarsening. NACFormer jointly learns the embeddings produced from multi-head attention blocks and from a GCN model trained on a smaller graph. Further, we train the model on a range of coarsened graphs, which not only increases model accuracy but also enhances its adaptability across varying graph structures. By experiments on real-world datasets for node classification tasks, we demonstrate that NACFormer better captures the information of distant nodes, thus producing node embeddings of higher quality that improve node classification accuracy.*

This chapter has been accepted at the IJCNN 2025 conference:

- Ghanbari, M., Bagloee, SA., Qi, J., and Sarvi, M. Multi-Scale Node Neighborhood Aggregation Via Graph Coarsening and Transformer. In International Joint Conference on Neural Networks (IJCNN), 1-8, 2025.

## 3.1 Introduction

Graph Neural Networks (GNN) are a class of deep learning models that have attracted significant attention in recent years [25, 27]. They have a wide range of applications [119], due to their strong empirical performance and the prevalence of graph-structured data. Among different GNNs, Graph Convolutional Networks (GCN) are one of the most popular models [25]. The key step in GCNs is massage passing, where a node's embedding (i.e., a latent vector representation) is iteratively updated based on aggregating the information of the neighboring nodes.

Although GCNs have shown strong performance on tasks such as node classification and link prediction, the local embedding update mechanism of vanilla GCNs overlooks the dependencies of distant nodes. Unlike Convolution Neural Networks (CNN), stacking more GCN layers does not solve the problem, while such an approach leads to a situation where every node's embedding resembles those of the other nodes [46].

Meanwhile, the Transformer model, which was originally proposed for sequential data [120], has shown remarkable performance in a variety of domains such as language modeling, image processing, and time series prediction [121, 122]. The Transformer model has also found its way into the GNN architecture, resulting in a novel approach known as the graph-transformer [123–125]. The emergence of graph-transformer architectures represents a significant leap forward in addressing the inherent challenges of capturing dependencies between distant nodes. By incorporating self-attention, graph-transformer models can attend to relevant information from different nodes in the graph. To this end, a recent study [126] provides empirical evidence showing the potential benefits of global attention mechanisms. The study demonstrates that incorporating global attention can improve the performance of models by allowing them to attend to all parts of the input graph.

While graph-transformers are the current state of the art, there are other approaches such as graph coarsening (Fig. 3.1). Graph coarsening provides a solution that derives the node embeddings on a set of smaller graphs that are similar to the original graph [127]. As an interesting bonus, the graph coarsening strategy can also act as a regularizer and lead to a better generalization performance than the vanilla GCNs [128]. For example, by looking closer at the APPNP model [129] (similar results with other

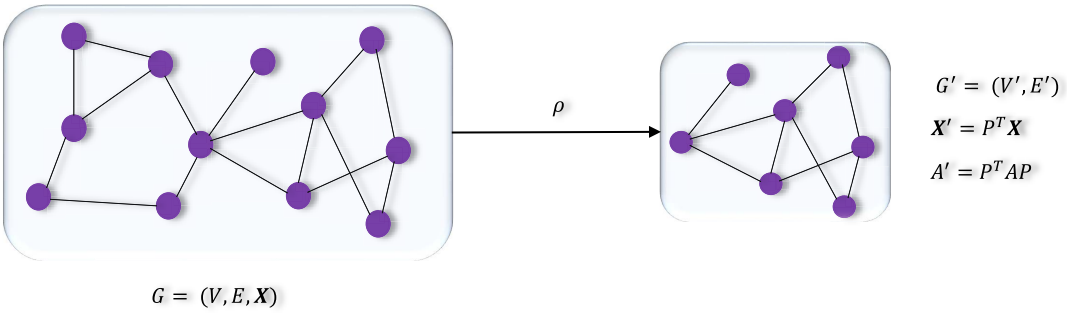



FIGURE 3.1: Graph coarsening technique to produce a smaller graph $G^{'}$

models were obtained in our experiments) over the coarsened graph ($\rho = 0.8$) in Table 3.7, we see that its accuracy over the Cora dataset increased by 2% (from 87.15% to 89.06%) comparing with the same model while trained on the full graph in Table 3.3.

We observe that combining graph-transformers with graph coarsening helps enhance model accuracy. In this chapter, we present Node **N**eighborhood **A**ggregation Via Graph **C**oarsening and Trans**former** (NACFormer). NACFormer takes the idea of Multi-Head Attention (MHA) (the building block of Transformer), together with the APPNP model [129] trained on a coarsened graph, to enhance node representation learning. Our goal is to design a model that learns both myopic and hyperopic interactive information (i.e., near and distant nodes). NACFormer is built upon three main components: a graph coarsening algorithm to produce a coarsened graph that preserves the spectral properties similar to the original graph; an MHA block to capture long-range dependencies; and a GNN-based model to learn the embeddings.

Moreover, we present a generalized model called NACFormer$_{MS}$ that can gather nodes' information from various resolutions over a range of coarsened graphs. Overall, our main contributions in this work are:

1. We propose a model named NACFormer with a new neighborhood aggregation method, based on Transformer and graph coarsening.

2. We further generalize the NACFormer model to enhance the node embedding quality by capturing the information within different graph coarsening scales. This creates an advanced model named NACFormer$_{MS}$.

3. Our experiments on nine datasets demonstrate the effectiveness of the proposed models, showcasing their superiority over state-of-the-art (SOTA) graph-transformer and non-transformer-based competitors under different settings (full-supervised and semi-supervised).

## 3.2 Related Work

Representation learning is a process to automatically extract meaningful features from data without manual feature engineering [130]. Graph representation learning, a specialized area within representation learning, focuses on learning informative representations of graph-structured data and capturing structural and relational information.

A good number of methods have been presented for graph representation learning in recent years. Deep-Walk [131] produces a path formed by a random walk over the connected nodes. Node2Vec [132] extracts random walks using DFS and BFS algorithms. These paths are then treated as sentences, and the word2vec embedding learning technique is applied to compute node embeddings. LINE [133] learns embeddings by preserving first and second-order proximities. SDNE [134] learns by mapping input data into a highly non-linear latent space to preserve the network structure. Despite the effectiveness of these methods, their node embedding generation is locally focused, ignoring distant nodes.

The emergence of GCN-based methods led to significant advancements in graph representation learning. However, these models limit the massage passing – a crucial part of their structure – to the local neighborhood. A few works have turned to hierarchical message passing, where a graph is reduced in size, and the embeddings produced can better incorporate the information of the distant nodes when they are converted to nearer neighbors in a coarsened graph. Examples are HGCN [135], HGNet [136] and HC-GNN [37]. In addition to these models, a few works have studied the graph-transformer models to capture features of distant nodes. For example, Gophormer [137] improves model performance using a Node2Seq module to sample ego-graphs as the input of Transformers which serves as a data augmentation technique. SAN [123] utilizes a learned positional encoding based on the Laplacian spectrum, enabling a fully connected Transformer model that excels in graph tasks. Graphormer [124] is based on the

Transformer architecture and encodes the structural information of a graph with the model.

In this work, we bring both graph coarsening and the Transformer model together and design a model that can enhance the quality of node embeddings generated by a GCN-based architecture. Our model can leverage both feature information and topological structures of the graph to enhance the quality of the learned embeddings.

## 3.3 Preliminary

### 3.3.1 Problem Statement

Let $G = (V, E, \mathbf{X})$ be a graph with $N = |V|$ vertices where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of vertices, $E \subseteq V \times V$ is the set of edges, and the $i$-th row of $\mathbf{X} \in \mathbf{R}^{N \times f}$ is the feature vector of node $v_i$ with $f$ dimensions. We use $\mathbf{A} \in \{0, 1\}^{N \times N}$ to denote the adjacency matrix of graph $G$, where $\mathbf{A}_{ij} = 1$ if $(v_i, v_j) \in E$, otherwise $\mathbf{A}_{ij} = 0$.

Although our model can be plugged into other learning problems such as link prediction, we use node classification as an example application where the aim is to predict the class labels of nodes. For supervised node classification with $l$ classes, we use $Y \in \{0, 1\}^{N \times l}$ to represent the label of nodes where $Y_i$ is the corresponding one-hot indicator vector if $v_i$ is labeled, otherwise $Y_i$ is a vector of zeros. To this end, a GCN-based model operates on graph $G$ to produce a node embedding $\mathbf{H} \in \mathbf{R}^{N \times d}$ for every node, where $d$ is the embedding dimensionality. The node embeddings are then passed through the convolution layer to convert embeddings into class predictions. For the learning process, we minimize the negative log-likelihood loss function $\ell(Y, \hat{Y})$.

### 3.3.2 Graph Convolutional Networks

Graph Convolutional Networks [25] are formulated as follows:

$$\mathbf{H}^{(k)} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(k-1)} \mathbf{W}^{(l)} \tag{3.1}$$

where $\mathbf{W}^{(k)} \in \mathbf{R}^{f \times d}$ is a learnable matrix (recall that $d$ is the node embedding dimensionality); $\tilde{\mathbf{D}}$ is the diagonal matrix of $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ with $\mathbf{I}_N$ is an identity matrix; $\mathbf{H}^{(k)} \in \mathbf{R}^{N \times d}$ is computed after $k$ steps of the GCN layer and $\mathbf{H}^{(0)} = \mathbf{X}$.

Reference [129] introduces a propagation strategy based on personalized PageRank, combining ideas from neural networks and the PageRank algorithm [138] to achieve better generalization and robustness in node classification tasks. To efficiently compute the personalized PageRank scores during the process, a fast approximation of its propagation scheme is called APPNP. The APPNP model incorporates skip connections to enhance the model's expressiveness. The propagation process of APPNP is written as follows:

$$\mathbf{Z}^0 = \mathbf{H} = F(\mathbf{X}, \mathcal{W}) \tag{3.2}$$

Where $f$ is a neural network and $W$ is the weight matrix.

$$\mathbf{Z}^{K+1} = (1-\alpha)\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{Z}^k + \alpha\mathbf{H} \tag{3.3}$$

where $\alpha \in (0, 1]$.

### 3.3.3 Transformer

The Transformer model [120] is fundamentally based on the multi-head attention mechanism, which was introduced to effectively model dependencies across input sequences [139]. Unlike traditional sequence-processing architectures, such as recurrent neural networks [140] or long short-term memory networks[141], the Transformer eliminates recurrence and relies entirely on self-attention to capture global relationships between elements in a sequence.

At the core of the Transformer is the Scaled Dot-Product Attention, which computes attention scores by measuring the similarity between query and key representations. The attention function is formulated as:

$$Att(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_q}}\right)\mathbf{V} \tag{3.4}$$

Here $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are the query, key, and value matrices:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_k, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_v \tag{3.5}$$

where $\mathbf{W}_q, \mathbf{W}_k \in \mathbf{R}^{f \times d_q}$ and $\mathbf{W}_v \in \mathbf{R}^{f \times d_v}$ are learnable matrices. Multi-head attention is a concatenation of multiple feed-forward neural networks to linearly project $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ for $h$ times (or attention heads):

$$MultiHeadAtt(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [Att_1 \| \dots \| Att_h]\mathbf{W}^0 \tag{3.6}$$

where $\mathbf{W}^0 \in \mathbf{R}^{hd_v \times d}$, $d_q = d_v = \frac{d}{h}$, and $\|$ means concatenation.

### 3.3.4 Graph Coarsening

Graph coarsening is an approach to reduce the size of a graph while keeping crucial characteristics [142]. Given a graph $G$, the coarsened graph is denoted by $G^{'} = (V^{'}, E^{'}, \mathbf{X}^{'})$, with $N^{'} = |V^{'}|$ as the number of nodes in $G^{'}$. The coarsened graph $G^{'}$ is obtained by computing a partition

$$\mathcal{P} = \{C_1, C_2, \dots, C_{N^{'}}\} \tag{3.7}$$

from the original graph's nodes and treating each cluster $C_i$ as a super node, using a user-defined reduction ratio defined as

$$\rho = \frac{N^{'}}{N} \tag{3.8}$$

.

In matrix format, a partitioning is represented by $\hat{P} \in \{0,1\}^{N \times N^{'}}$

$$\hat{P}_{ij} = \begin{cases} 1, & \text{if node } v_i \text{ belongs to cluster } C_j \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

The normalized version of $\hat{P}$ can be defined by

$$P = \hat{P}\mathcal{C}^{-1/2} \tag{3.10}$$

where the diagonal matrix $\mathcal{C}$ represents the sizes of each cluster:

$$\mathcal{C} = \text{diag}(|C_1|, |C_2|, \ldots, |C_{N'}|) \tag{3.11}$$

and

$$P_{ij} = \frac{1}{\sqrt{|C_j|}}, \quad \text{if } v_i \in C_j, \quad \text{otherwise } P_{ij} = 0 \tag{3.12}$$

Using this transformation, the adjacency matrix and feature matrix of the coarsened graph are obtained through:

$$\mathbf{A}^{'} = P^T \mathbf{A} P, \quad \mathbf{X}^{'} = P^T \mathbf{X} \tag{3.13}$$

Additionally, the labels of super nodes in the coarsened graph are computed by propagating the original node labels $Y$ as:

$$Y^{'} = P^T Y \tag{3.14}$$

where the dominant class within each cluster is assigned as the label of the corresponding super node.

## 3.4 Proposed Model

By taking a closer look at the attention mechanism:

$$Att = \mathcal{A}\mathbf{V} = \mathcal{A}\mathbf{X}\mathbf{W}_v \tag{3.15}$$

where

$$\mathcal{A} = softmax(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_q}}) \tag{3.16}$$

it can be found that its structure resembles that of a GCN with a fixed adjacency matrix:

$$\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W} \tag{3.17}$$

where multi-head attention works with attention weight matrix, $\mathcal{A}$, which is calculated by the query and key matrices. We design our model, NACFormer, to take advantage of distant nodes to enhance node representation learning, since the attention matrix, being a similarity matrix quantifies the relevance of each node to the others in the graph.

### 3.4.1 NACFormer Structure

The overview of NACFormer is presented in Fig. 3.2 and the main steps of the model is summarized as follows:

1. Apply a coarsening algorithm on graph $G$ to compute $G^{'}$, its corresponding adjacency matrix $\mathbf{A}^{'}$, feature matrix $\mathbf{X}^{'}$ and labels $Y^{'}$.
2. Train NACFormer on $G^{'} = (V^{'}, E^{'}, \mathbf{X}^{'})$ and save the parameters.
3. Use the trained model on graph $G$ to make predictions.

Given a graph $G$, we first run a coarsening algorithm with a given coarsening ratio, $\rho$, to build a coarsened graph $G^{'} = (V^{'}, E^{'}, \mathbf{X}^{'})$. We use the variation neighborhoods algorithm [142] which aims to simplify a graph by combining the neighborhoods of certain nodes into super nodes in order to minimize a spectral distance between the coarsened graph and the original graph. This process is repeated iteratively until the desired coarsening ratio is achieved. The embedding learning process then will be run on $G^{'}$, where the feature matrix, $\mathbf{X}^{'}$, is fed into a multi-head attention block to produce the embeddings $\mathbf{H}^{'}_{T} \in \mathbf{R}^{N^{'} \times d}$ (detailed in Section 3.3.4). Meanwhile, in order to allow the APPNP model to learn from the produced embeddings, we need to link the generated embeddings together for training in a joint way. To this end, we combine $\mathbf{H}^{'}_{T}$ with the neural network function in Equation (3.2). We first pass $\mathbf{X}^{'}$ through the first layer of a two-layer neural network approximation to obtain $\mathbf{H}^{'} = \mathbf{X}^{'}\mathbf{W}_1$, and then replace the generated function $F = \mathbf{H} = (\mathbf{H}^{'})\mathbf{W}_2$ with:

$$\mathbf{H} = ([\mathbf{H}^{'} \| \mathbf{H}^{'}_{T}])\mathbf{W}_2 \tag{3.18}$$

Here, $\mathbf{W}_2 \in \mathbf{R}^{2d \times l}$, $\mathbf{H} \in \mathbf{R}^{N' \times l}$, and $\|$ is the concatenation operator. Finally, the generated embeddings $\mathbf{H}$ are fed into the convolution layer to convert the embeddings into class predictions: $\hat{Y}' = \mathbf{Z}' = (1-\alpha)\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{\frac{1}{2}}\mathbf{H} + \alpha\mathbf{H}$, where $\mathbf{Z}' \in \mathbf{R}^{N' \times l}$. Hence, our method is able to jointly learn the near and distant neighbor information for each node without increasing the number of layers. The model training target is to minimize the loss function $\ell(\mathbf{Z}', Y')$ to find the optimum parameters of NACFormer$_{G'}(W)$.

At the final step, the trained parameters of the model on $G'$, NACFormer$_{G'}(W)$, is ready for evaluation on the original graph. The weight matrix $W$ has dimensions $f \times d$, where $f$ represents the dimensionality of the features and $d$ represents the dimensionality of the node embeddings. This dimensionality of $W$ remains consistent regardless of the size of the graph. Since the weight matrix $W$ is independent of the graph size, it can be applied to the original graph directly. The model parameters learned from the smaller graph $G'$ are transferable and can be effectively used for evaluation on the original graph $G$.

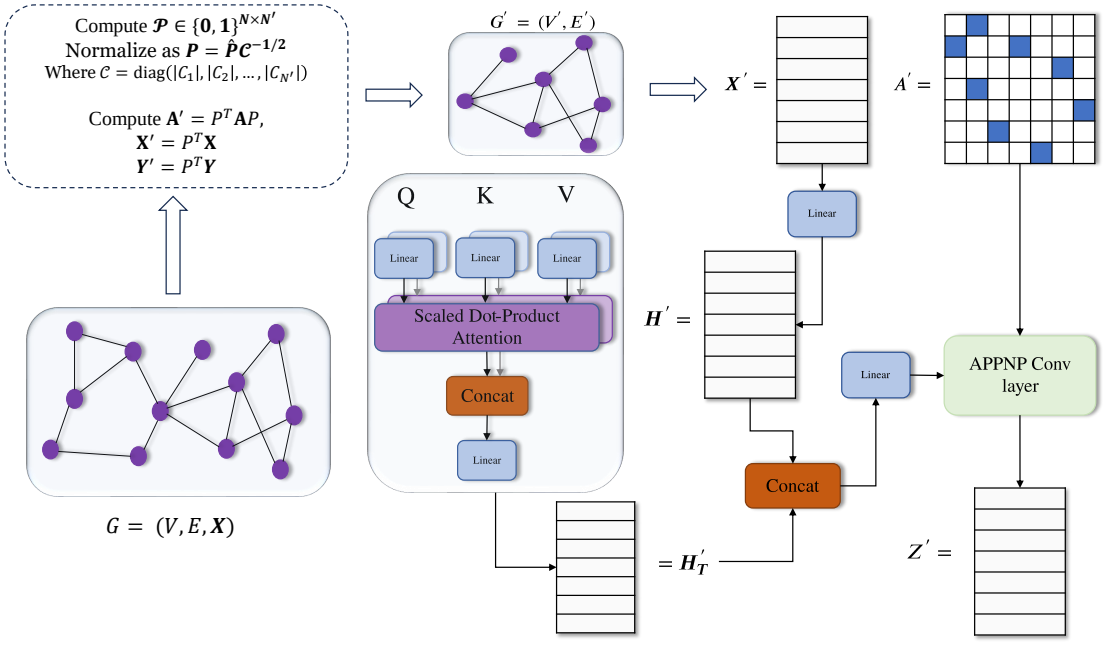



FIGURE 3.2: Structure of NACFormer. The input graph $G$ is first processed by a coarsening algorithm to identify key structural nodes and subgraphs while computing the corresponding adjacency, feature, and label matrices. The resulting coarsened graph is then passed through a multi-head attention block and an APPNP convolution layer, to jointly generate the node embeddings. Finally, the learned embeddings are utilized to classify the nodes.

### 3.4.2 Multi-Scale NACFormer Structure

NACFormer is only trained on a coarsened graph $G^{'}$. We take one step further for better quality of node embeddings by training NACFormer in a multi-granularity manner. We train the model over a series of coarsened graphs using different coarsening ratios, $\{\rho_1, \rho_2, \ldots, \rho_n\}$.

We summarize the steps of the Multi-Scale NACFormer, NACFormer$_{MS}$, as follows:

1. A coarsening algorithm is applied on graph $G$ to compute a series of coarsened graphs $\{G_1, G_2, \ldots, G_n\}$, including their corresponding adjacency $\{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n\}$, feature matrices $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n\}$ and class labels $\{Y_1, Y_2, \ldots, Y_n\}$.

2. The NACFormer model is trained on all produced graphs, $\{G_i\}_{i=1}^{n}$, where $G_i = (V_i, E_i, \mathbf{X}_i)$, with the model's parameters being saved only if the validation result improves upon the previous iteration.

3. The trained model is evaluated on the original graph $G$ to make predictions.

These steps are illustrated in Fig. 3.3, with the detailed model training steps presented in Algorithm 1.

### 3.4.3 Model Training Time.

The time costs of NACFormer include:

1. Constructing a coarsened graph using the neighborhood-based coarsening algorithm [142] takes time linear to the number of nodes, $\mathcal{O}(|V|)$. Note that graph coarsening can be efficiently pre-computed.

2. The coarsened graph $G^{'}$ has $|V^{'}| = (1-\rho)|V|$ vertices and the number of edges in $G^{'}$ is upper bounded by $\min\{|E|, (1-\rho)|V|^2\}$. This implies that training on $G^{'}$ has a worst-case computational time cost that is $(1-\rho)$ times the cost of the original GCN. Note that the number of edges is smaller than this upper bound and the computational savings are higher in practice.

**Algorithm 1:** NACFormer$_{MS}$ training

**Input:** Adjacency matrix $\mathbf{A}$, node feature matrix $\mathbf{X}$, node labels $\mathbf{Y}$, teleport probability $\alpha$, Max-Epochs, number of propagation steps $K$ **(1)**
**Output:** Trained model parameters $\mathbf{W}$ **(2)**
Preprocess adjacency matrix $\mathbf{A}$ to obtain normalized matrix **(3)**
Initialize the model parameters $\mathbf{W}$ **(4)**
**for** $\rho_1$ *to* $\rho_n$ **do** **(5)**
  Compute the coarsened graph $G^{'} = (V^{'}, E^{'})$ **(6)**
  Compute the coarsened adjacency $\mathbf{A}^{'}$ **(7)**
  Compute the coarsened feature matrix $\mathbf{X}^{'}$ **(8)**
  Compute the coarsened node labels $Y^{'}$ **(9)**
  **while** *The Max-Epochs is not reached* **do** **(10)**
    Load model parameters $\mathbf{W}$ **(11)**
    Compute the first layer of $F(\mathbf{X}^{'}, \mathbf{W})$ (Equation (3.2)) to obtain $\mathbf{H}^{'}$ **(12)**
    Compute multi-head attention to obtain $\mathbf{H}^{'}_{T}$ **(13)**
    Concatenate the embeddings and pass to the second layer to obtain $\mathbf{H}$ **(14)**
    Compute the APPNP convolution layer: **(15)**
    **for** $k = 0$ ***to*** $K$ **do** **(16)**
      $\mathbf{Z}^{k+1} = (1-\alpha)\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{Z}^{k} + \alpha\mathbf{H}$ **(17)**
    **end** **(18)**
  **end** **(19)**
  Save the model parameters $\mathbf{W}$ **(20)**
**end** **(21)**

3. The time complexity of multi-head attention is quadratic to the number of nodes in the coarsened graph, $\mathcal{O}(|V'|^2)$. To overcome the quadratic time issue, we exploit the Performer model [143], which uses only linear (as opposed to quadratic) space and time complexity. This implies that the space and time complexity of training on $G^{'}$ is $\mathcal{O}(|V'|)$.

## 3.5 Experiments

To comprehensively evaluate the performance of NACFormer and NACFormer$_{MS}$, we conduct extensive experiments on real-world datasets, focusing on multi-label node classification tasks.

### 3.5.1 Datasets

We use nine real-world datasets, as summarized in Table 3.1.

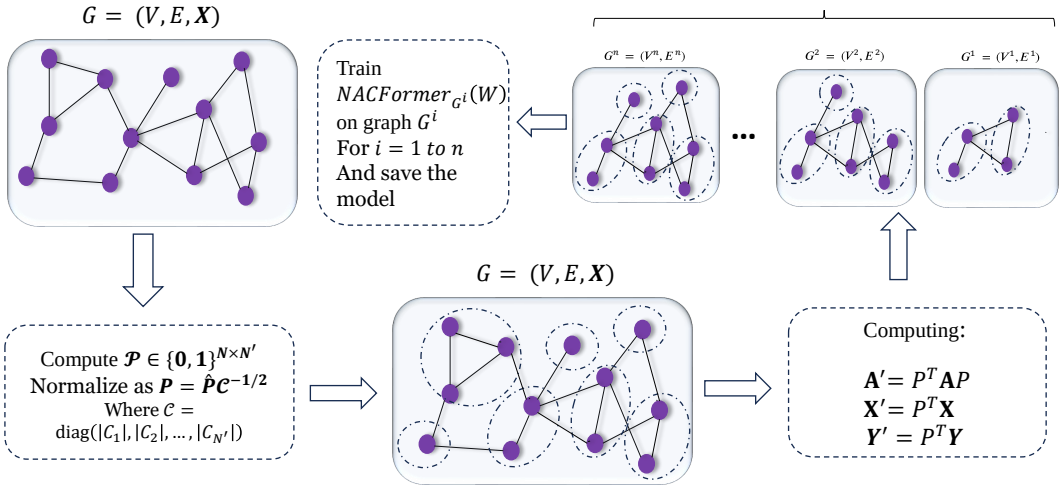



FIGURE 3.3: Structure of Multi-Scale NACFormer. NACFormer$_{MS}$ is trained on a range of coarsened graphs $G^1, G^2, \cdots, G^n$ produced by coarsening ratios $\{\rho_1, \rho_2, \cdots, \rho_n\}$. A NACFormer model is trained on every coarsened graph, with the model's parameters being saved only if the validation accuracy outperforms that of the previous iteration. The trained model is evaluated on the original graph $G$ to make predictions.

TABLE 3.1: Summary of datasets used in our experiment

| Dataset | # Vertices | # Edges | # Classes | # Features |
|---|---|---|---|---|
| Cora [107] | 2,708 | 10,556 | 7 | 1,433 |
| CiteSeer [107] | 3,327 | 9,104 | 6 | 3,703 |
| Pubmed [107] | 19,717 | 88,648 | 3 | 500 |
| DBLP [108] | 17,716 | 105,734 | 4 | 1,639 |
| LastFMAsia [109] | 7,624 | 55,612 | 18 | 128 |
| Cora-ML [108] | 2,995 | 16,316 | 7 | 2,879 |
| Wiki [111] | 2,405 | 17,981 | 17 | 4,973 |
| BlogCatalog [111] | 5,196 | 343,486 | 6 | 8,189 |
| OGB-arxiv [113] | 169,343 | 1,166,243 | 40 | 128 |

### 3.5.2 Competitors

We compare NACFormer and NACFormer$_{MS}$ against GCN [25], GraphSAGE [28], GAT [27], APPNP [129], SGC [144], JKNet [145], ChebNetII [73], [125], SAN [123], Graphormer [124], Gophormer [137], ANS-GT [146], and PC-Conv [147].

### 3.5.3 Parameter Settings

Table 3.2 summarizes the hyperparameters for our models. In addition to these parameters, we also analyzed the maximum number of heads $h$ (see Fig. 3.4), and the coarsening

ratio $\rho$ (see Fig. 3.5). The results are detailed in the subsequent sections.

We train the model using the Adam optimizer [148] and the negative log-likelihood loss. A fixed dropout rate of 0.5 and the eLU activation function is applied between the layers. The training is terminated if the validation accuracy does not improve for 10 consecutive epochs. The same setting is applied to NACFormer$_{MS}$ with the only difference being that it trains over the coarsened graphs produced by coarsening ratios $\{\rho_1 = 0.1, \rho_2 = 0.2, \ldots, \rho_9 = 0.9\}$.

TABLE 3.2: The grid search space for the hyperparameters.

| Hyperparameter | Values |
|---|---|
| Learning rate | $1e-3, 1e-4, 1e-5$ |
| Hidden size | 64, 128, 512 |
| Epochs | 80, 200 |
| Attention heads | 4, 8 |
| Weight decay | $1e-2, 1e-3$ |
| K | 10, 20 |

For the APPNP baseline model in Table 3.7, the setting follows those of NACFormer (for fair comparison). It consists of two linear layers followed by an APPNP convolutional layer. Also, the dropout rate is 0.5 and the ReLU activation function is applied between the layers.

In all of our implementations, we use GNN libraries PyTorch Geometric[1] [3]. To construct the partitions used in the graph hierarchy, we apply the variation neighborhoods[2] algorithm [142]. For performance evaluation, we compare models based on their accuracy on the test set. Since different splits of the data (train/validation/test) may lead to a substantially different ranking of models [112], for a fair comparison, we evaluate the models over different splits as follows:

- Full-supervised Split. Under the full-supervised split setting, For all datasets, we randomly split the nodes into 60%, 20%, and 20% for training, validation, and testing.
- Semi-supervised Split. Under the semi-supervised split setting, we randomly split the nodes into 2.5%, 2.5%, and 95% for training, validation, and testing.

[1] https://pytorch-geometric.readthedocs.io/en/latest/
[2] https://github.com/loukasa/graph-coarsening

All models are trained and tested on an in-house GPU cluster. The experiment was executed on a single NVIDIA A100 GPU, with 100 GB of memory allocated for the task.

### 3.5.4 Results

#### 3.5.4.1 Overall Performance Results

Table 3.3 presents the node classification accuracy of NACFormer and NACFormer$_{MS}$ against the baseline models over the full-supervised split that is commonly used in the literature. In addition, Table 3.4 presents the accuracy of the models on the semi-supervised split. The best results are highlighted in bold. The accuracy values for the baseline models are harvested from the literature [137]. According to the tables, our models ranked at the top for most cases, demonstrating the superior performance of our models when aggregating information from a range of different scales. Furthermore, our models exhibited competitive results when compared to other models on the full-supervised split of Pubmed.

TABLE 3.3: Node classification accuracy on the full-supervised split. The best results are highlighted in bold.

| Model | Cora | Citeseer | Pubmed | DBLP |
|---|---|---|---|---|
| GCN [25] | 87.33±0.38 | 79.43±0.26 | 84.86±0.19 | 83.62±0.13 |
| GAT [27] | 86.29±0.53 | 80.13±0.62 | 84.40±0.05 | 84.19±0.19 |
| GraphSAGE [28] | 86.90±0.84 | 79.23±0.53 | 86.19±0.18 | 84.73±0.28 |
| SGC [144] | 86.58±0.26 | 76.23±0.29 | 83.52±0.10 | 81.14±0.44 |
| APPNP [129] | 87.15±0.43 | 79.33±0.35 | 87.04±0.17 | 84.40±0.17 |
| JKNet [145] | 87.70±0.65 | 78.43±0.31 | 87.64±0.26 | 84.57±0.28 |
| ChebNetII [73] | 88.71±0.93 | 80.53±0.79 | 88.93±0.29 | 85.20±0.15 |
| GT-full [125] | 63.40±0.94 | 58.75±1.06 | 77.29±0.50 | 78.15±0.41 |
| GT-sparse [125] | 71.84±0.62 | 67.38±0.76 | 82.11±0.39 | 81.04±0.27 |
| SAN [123] | 74.02±1.01 | 70.64±0.97 | 86.22±0.43 | 83.11±0.32 |
| Graphormer [124] | 72.85±0.76 | 66.21±0.83 | 82.76±0.24 | 80.93±0.39 |
| Gophormer [137] | 87.85±0.10 | 80.23±0.09 | 89.40±0.14 | 85.20±0.20 |
| ANS-GT [146] | 88.60±0.45 | 80.25±0.39 | 89.56±0.55 | 85.12±0.17 |
| PC-Conv [147] | 90.02±0.62 | 81.76±0.78 | **91.30±0.38** | 85.78±0.51 |
| **NACFormer (0.7)** | 87.71±0.42 | 76.53±1.22 | 86.92±0.46 | 85.43±0.56 |
| **NACFormer (0.8)** | 89.43±1.28 | 77.75±1.29 | 86.97±0.63 | 85.05±0.19 |
| **NACFormer$_{MS}$** | **92.18±1.57** | **82.83±0.62** | 87.62±0.44 | **86.63±0.56** |

NACFormer$_{MS}$ performed remarkably well on Cora, Citeseer, and DBLP under the full-supervised split. However, it achieved lower accuracy on Pubmed. In contrast, it followed the opposite trend under the semi-supervised split by attaining higher accuracy

TABLE 3.4: Accuracy of models on the semi-supervised split. The best results are highlighted in bold.

| Method/Dataset | Cora | Citeseer | Pubmed | dblp | LastFMAsia | Cora_ML | Wiki | BlogCatalog |
|---|---|---|---|---|---|---|---|---|
| GCN | 75.84±0.57 | 68.53±0.82 | 81.27±0.41 | 80.43±0.18 | 70.47±0.91 | 76.26±0.68 | 55.14±1.40 | 48.05±0.77 |
| GAT | 76.73±0.63 | 65.89±0.45 | 82.12±0.002 | 79.77±0.46 | 73.61±0.48 | 78.04±0.65 | 54.35±2.00 | 36.23±2.32 |
| SGC | 74.26±0.48 | 68.27±0.54 | 81.79±0.36 | 79.18±0.14 | 72.51±0.78 | 79.44±0.22 | 50.34±1.58 | 50.69±1.26 |
| ARMA | 69.02±0.73 | 63.62±0.23 | 80.99±0.08 | 75.69±0.22 | 71.07±1.40 | 70.94±2.43 | 53.57±0.47 | 72.13±4.69 |
| APPNP | 79.89±0.77 | 68.75±0.43 | 82.69±0.28 | 81.77±0.16 | 76.44±0.85 | 80.89±0.36 | 56.56±1.47 | 58.76±2.37 |
| GraphSage | 70.04±0.26 | 68.88±0.08 | 77.38±0.11 | 77.83±0.41 | 75.31±0.21 | 68.25±1.47 | 48.09±0.69 | 76.91±0.15 |
| ChebNetII | 82.42±0.64 | 69.89±1.21 | 79.51±1.03 | 80.58±0.97 | 76.71±0.55 | 71.74±6.91 | 46.90±2.53 | 73.63±0.75 |
| **NACFormer (0.8)** | 80.59±1.13 | 67.96±1.49 | 84.25±0.53 | **82.84±0.42** | 79.29±0.83 | 82.10±0.56 | 59.50±1.40 | 76.17±2.71 |
| **NACFormer$_{MS}$** | **83.13±2.66** | **70.01±2.07** | **84.82±0.34** | 82.81±0.33 | **82.58±0.31** | **85.55±1.13** | **68.25±0.50** | **85.98±4.28** |

on the Pubmed dataset as well. This result reinforces the key conclusion of an earlier study [112], which highlights the potential impact of data splits on model performance rankings. The observed discrepancies serve as a reminder that it is critical to carefully consider the choice of data split to ensure the validity of the results obtained.

In most cases, while NACFormer's performance was comparable to the existing models, NACFormer$_{MS}$ outperformed them, highlighting the effectiveness of learning node features at multiple graph coarsening scales. By leveraging the multi-scale approach, NACFormer$_{MS}$ was able to produce embeddings of higher quality, leading to better node classification accuracy across a range of datasets.

#### 3.5.4.2 Results Over Large Graphs

Table 3.5 presents a comparative analysis of the performance of APPNP and NACFormer on the OGB-ARXIV dataset across different graph coarsening ratios. The results indicate that, for the OGB-ARXIV dataset, NACFormer consistently outperforms APPNP, which demonstrates its robustness and effectiveness in handling varying coarsening ratios.

TABLE 3.5: Performance comparison between NACFormer and APPNP on the OGB-ARXIV dataset. The number of attention heads of NACFormer is 8 and the accuracy is averaged over 5 runs.

| $\rho$ | OGB-ARXIV | |
|---|---|---|
| | APPNP | **NACFormer** |
| 0.01 | 55.58±7.18 | **57.88±2.78** |
| 0.1 | 62.52±0.16 | **63.29±1.11** |
| 0.5 | **66.78±0.78** | 66.72±0.85 |
| 0.7 | 67.19±0.11 | **67.30±0.13** |

#### 3.5.4.3 Significance of Coarsened Graph

We further conducted a comparison between our model trained on a coarsened graph $G'$ and when it is trained on the full graph $G$ to shed light on the significance of the presence and absence of graph coarsening on our model's performance. Table 3.6 presents the node classification accuracy results. This comparison offers insights into the efficacy of different training strategies with coarsened graph configurations yielding better classification accuracy.

TABLE 3.6: Node classification accuracy of our model with and without Coarsening.

| Dataset/Method | No coarsening | $\rho = 0.8$ | NACFormer$_{MS}$ |
|---|---|---|---|
| **Cora** | 78.75±1.50 | 80.59±1.13 | **83.13±2.66** |
| **Citeseer** | 67.15±0.92 | 67.96±1.49 | **70.01±2.07** |
| **Pubmed** | 83.08±0.48 | 84.25±0.53 | **84.82±0.34** |
| **DBLP** | 82.02±0.29 | **82.84±0.42** | 82.81±0.33 |
| **LastFMAsia** | 79.06±0.43 | 79.29±0.83 | **82.58±0.31** |
| **Cora_ML** | 80.93±0.22 | 82.10±0.56 | **85.55±1.13** |
| **Wiki** | 59.05±2.30 | 59.50±1.40 | **68.25±0.50** |
| **BlogCatalog** | 81.2±3.47 | 76.17±2.71 | **85.98±4.28** |

#### 3.5.4.4 Significance of Multi-head Attention

To verify the effectiveness of the multi-head attention mechanism in enhancing the quality of embeddings by incorporating the information of distant nodes in a graph, we compare NACFormer against APPNP on both types of dataset splits and with different graph coarsening ratios $\rho = 0.4$ and $\rho = 0.8$. For a fair comparison, here, APPNP has a similar setting to that of NACFormer. Table 3.7 presents the results. As can be seen in most cases, NACFormer outperforms in terms of test accuracy.

TABLE 3.7: Comparison of NACFormer and APPNP over all datasets on graphs with different coarsening ratios. The number of attention heads of NACFormer is 8 and the accuracies are averaged over 5 runs.

| Model\Data | Split | Ratio | Cora | Citeseer | Pubmed | DBLP |
|---|---|---|---|---|---|---|
| APPNP | Full-supervised | $\rho = 0.8$ | 89.06±0.93 | 76.05±1.84 | 86.94±0.45 | **85.30±0.40** |
| **NACFormer** | | | **89.43±1.28** | **77.75±1.29** | **86.97±0.63** | 85.05±0.19 |
| APPNP | | $\rho = 0.4$ | **88.07±1.02** | 76.17±1.66 | **86.63±0.17** | 83.87±0.57 |
| **NACFormer** | | | 86.97±0.66 | **76.50±0.76** | 86.18±0.33 | **84.31±0.66** |
| APPNP | Semi-supervised | $\rho = 0.8$ | 80.24±1.82 | 67.84±1.31 | **84.38±0.39** | 82.20±0.47 |
| **NACFormer** | | | **80.59±1.13** | **67.96±1.49** | 84.25±0.53 | **82.84±0.42** |
| APPNP | | $\rho = 0.4$ | 79.04±1.94 | 66.03±1.72 | 83.70±0.55 | 81.95±0.70 |
| **NACFormer** | | | **80.71±2.24** | **67.16±0.54** | **83.81±0.87** | **82.36±0.44** |

Here, the conclusion is clear since the only difference between these two models is the presence and absence of the attention mechanism. Although the APPNP model is capable of capturing the impact of distant nodes by itself [129], it may still struggle with capturing complex, non-linear relationships between nodes that cannot be easily modeled. In this case, multi-head attention is more effective for the learning task.

#### 3.5.4.5 Analysis Over Number of Attention Heads

Next, we study the efficacy of NACFormer with a different number of attention heads. We evaluate the performance of the proposed model over the node classification task for $h = 1, 2, 4, 8$, and 16 as different numbers of attention heads. As Fig. 3.4 shows, in most cases, increasing the number of heads (beyond 8) does not necessarily result in improved accuracy. Regarding the Performer [143], the use of the FAVOR+ algorithm allows the model to achieve linear complexity. This means that increasing the number of heads may not necessarily increase the computational time and this is because FAVOR+ reduces the number of computations required for each attention head.

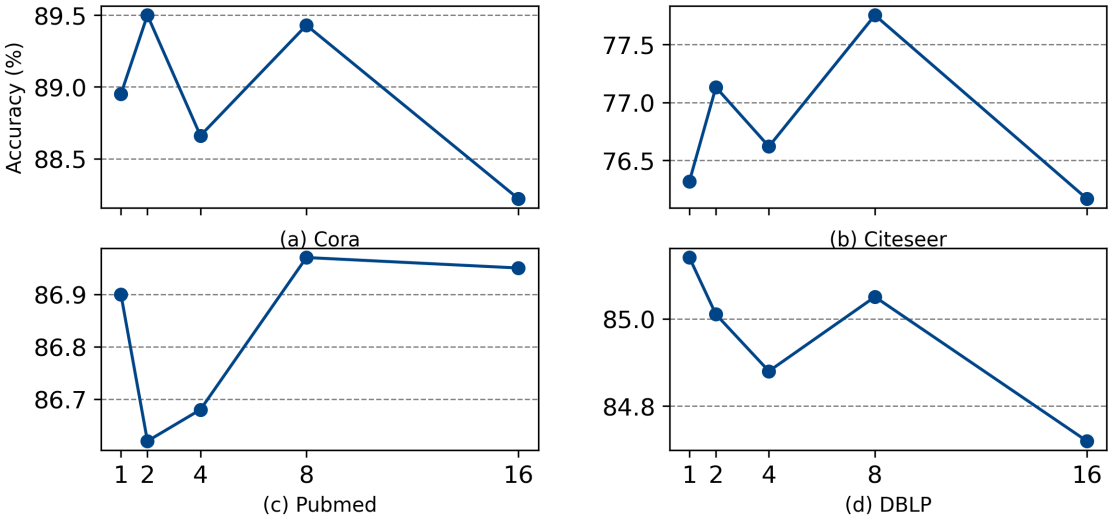



FIGURE 3.4: Accuracy of NACFormer on different datasets when varying the number of attention heads ($h = 1, 2, 4, 8, 16$). The coarsening ratio $\rho = 0.8$ is used.

#### 3.5.4.6 Analysis Over the Ratio

We also explore the effectiveness of NACFormer on different graph coarsening ratio values, by varying $\rho$ from 0.1 to 0.9 across different datasets. As shown in Fig. 3.5, in

almost all cases, NACFormer learned on the coarsened graphs with higher ratios yields better performance likely due to the increased proportion of graph structures captured in the coarser representations.

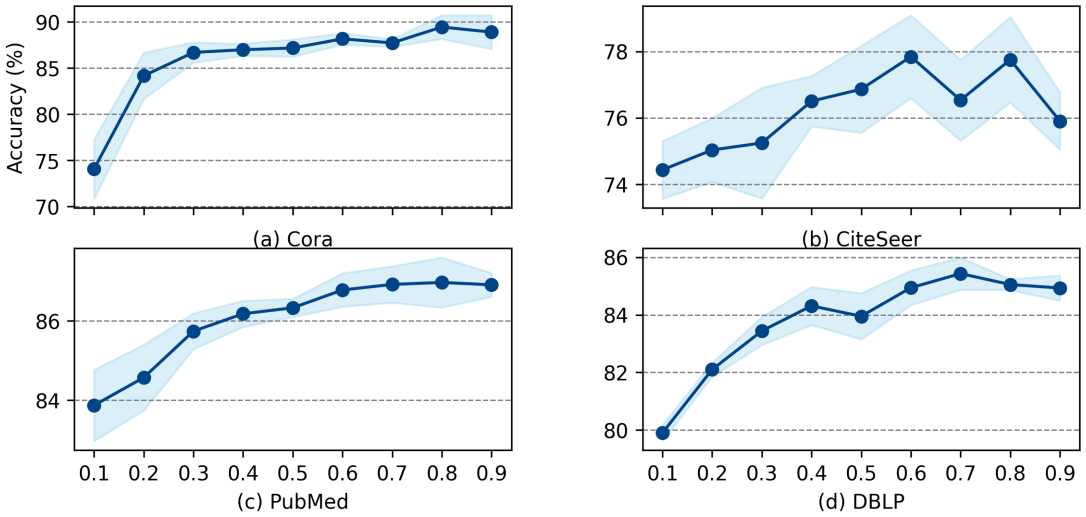



FIGURE 3.5: Accuracy of NACFormer with different graph coarsening ratios, from $\rho = 0.1$ to $\rho = 0.9$. The number of attention heads is set to 8.

#### 3.5.4.7 Analysis Over the Teleport Probability

Here, we provide an analysis of the effect of the teleport probability parameter, $\alpha$. Table 3.8 illustrates the relationship between $\alpha$ and the performance of NACFormer. As shown, $\alpha = 0.1$ yields the highest accuracy across all cases since it ensures stable incorporation of both graph structure and node features. Additionally, it is aligned with the default setting used in prior studies [128, 129]. In all our experiments, we set the teleport probability $\alpha$ to 0.1.

TABLE 3.8: NACFormer performance over datasets when $\alpha$ is varied. The number of attention heads is fixed at 8 and the graph coarsening ratio is 0.8.

| **Data**\$\alpha$ | $10^{-3}$ | $10^{-2}$ | $10^{-1}$ | 1 |
|---|---|---|---|---|
| **Cora** | 87.27 +/- 1.19 | 86.64 +/- 1.25 | **89.43 +/- 1.28** | 76.84 +/- 0.92 |
| **CiteSeer** | 75.99 +/- 1.23 | 74.88 +/- 1.50 | **77.75 +/- 1.29** | 72.01 +/- 1.18 |
| **Pubmed** | 82.81 +/- 0.55 | 83.38 +/- 0.44 | **86.97 +/- 0.63** | 86.65 +/- 0.60 |
| **DBLP** | 83.48 +/- 0.27 | 84.64 +/- 0.44 | **85.05 +/- 0.19** | 75.84 +/- 0.28 |

#### 3.5.4.8 Model Running Time

Fig. 3.6 illustrates the time of our model using the Performer attention technique (to compute the multi-head attention block) and the standard attention technique for a complete run (including training, validation and testing). The computational time of the standard attention technique is proportional to the number of attention heads, number of nodes, and embedding dimension. In contrast, the Performer technique is designed to be more computationally efficient, using the FAVOR+ algorithm to achieve linear complexity with respect to the number of nodes, independent of the number of attention heads. This makes Performer a better choice when dealing with a larger graph or a higher number of attention heads. From Fig. 3.6, we observe that the standard attention technique requires more time to run as the number of nodes increases, e.g., the Pubmed dataset. Moreover, it is worth noting that the density of the dataset plays a significant role in the computational time of the Performer model. The DBLP dataset, being denser with a higher number of edges between nodes, requires more computation time compared to the Pubmed dataset. This highlights the impact of density on the running time of Performer.

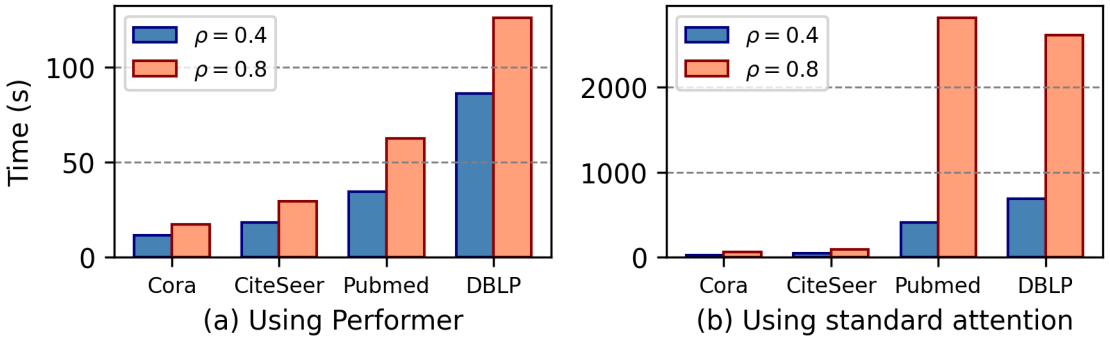



FIGURE 3.6: Running time of NACFormer model on different ratios by training through different techniques: (a) Performer and (b) standard.

## 3.6 Conclusion

We proposed a novel representation learning approach for graph convolutional networks that aggregates the impact of both near and distant nodes, which differs from the vanilla GCN that focuses on aggregating embeddings from a local neighborhood. Our proposed model, NACFormer, and its advanced version, NACFormer$_{MS}$, learn the information of

distant nodes via multi-head attention and that of nearby nodes by the adjacency matrix both applied to coarsened graphs for better generalization. By explicitly modeling both local and global dependencies, our framework achieves a balanced representation that retains structural information across different graph scales. Extensive experiments on real-world datasets show that our models outperform SOTA transformer-based and non-transformer-based models regarding node classification accuracy using the node embeddings learned. However, the reliance on coarsened graphs introduces a dependency on the coarsening strategy, which could affect performance in scenarios with highly irregular or noisy graphs. Future work could explore adaptive coarsening techniques to refine performance further.

# Chapter 4

# Feature-Aware Unsupervised Detection of Important Nodes in Graphs

*Identifying important nodes in a graph is a crucial task in many applications such as biological networks, transportation networks, and social networks. The main difficulty in this task is the lack of ground truth. Existing methods mainly focus on the structure of the graphs and ignore the node features or operate in a supervised manner. In this chapter, we design a trainable model named FadiGNN that incorporates node feature information as well as topology structure in an unsupervised manner to score the nodes within a graph. It consists of Graph Convolutional Networks (GCNs) and adapted personalized PageRank, along with a loss function that ensures the convergence of the model. To show the effectiveness of FadiGNN, experiments are conducted on two different application settings; node classification and active learning. The experiments are conducted on different real-world datasets, including large graph data. The results on node classification shows that FadiGNN outperforms the state-of-the-art models by an average of 7.62% in terms of accuracy.*

This chapter is published at the ADMA 2024 conference:

- Ghanbari, M., Bagloee, SA., Qi, J., and Sarvi, M. Feature-Aware Unsupervised Detection of Important Nodes in Graphs. In International Conference on Advanced Data Mining and Applications, 98-113, 2024.

## 4.1 Introduction

The significance of graph-structured data has increased in domains such as transportation networks [149], social networks [150], and molecular structure [151], emphasizing their adaptability in understanding complex relationships in today's information-rich landscape [40].

A key task in such graphs is identifying critical nodes that are of particular importance. For instance, in drug design, the identification of crucial nodes within protein-protein interaction networks is beneficial. This is due to the fact that critical nodes represent a set of proteins whose removal would disrupt fundamental interactions, thus contributing to the neutralization of potentially harmful organisms such as bacteria or viruses [17]. In road networks, the failure of a few nodes has a cascading effect on the performance of other nodes. This phenomenon reduces the overall efficiency of the transportation system. In particular, congestion at key intersections might lead to the collapse of an extensive road network [152]. Recognizing the importance of nodes within the road network is essential where it potentially helps in reducing congestion, strengthening the network against failures, and ensuring smoother transportation operations.

Existing methods for the problem can be divided into traditional and learning-based ones. Traditional techniques use heuristic measures such as node degree, betweenness centrality, eigenvector centrality, etc., to evaluate the importance of nodes within a graph. Among the traditional techniques, PageRank [138] holds a distinctive position, originally developed for ranking web pages. PageRank is a link analysis algorithm that assigns each node in a graph a numerical weight, emphasizing the importance of nodes based on their connections and the importance of nodes linking to them. PageRank assigns a score to each node by using a Markov chain framework, which models the iterative transition of a random walker moving between nodes based on the graph structure, to obtain a unique converged ranking result [153]. However, the primary limitation of these traditional techniques is that they depend only on the graph structures and overlook the node feature information. This reliance on the graph topology neglects the crucial details within the individual nodes and limits the evaluation of node importance, and consequently leads to a decrease in the accuracy of the ranking outcomes. For instance, in a social network, a new member may have few structural connections, but their unique attributes could make them highly influential in specific discussions or activities.

A ranking technique focusing solely on the graph structure potentially underestimates the importance of such nodes.

Learning-based techniques, in contrast, use learning algorithms to predict node importance. These techniques offer flexibility and adaptability to complex network structures. For instance, in Graph Convolutional Networks (GCN) [25], a key step is message passing, where a node's embedding undergoes iterative updates by aggregating information from neighboring nodes' features. This mechanism allows GCNs to effectively capture complex relationships within graph data and can exploit both the structure of the graph and the nodes features to make better predictions. Several prior studies have used supervised learning methods [17, 32, 154]. However, the practicality of deploying such techniques becomes a challenge since the graphs in these contexts are often unlabeled [47].

To overcome this issue, we propose to incorporate the GCN model to utilize the trainable framework for leaning the topology and feature information of nodes along with a personalized PageRank algorithm to keep the unsupervised learning procedure. This integration exploits the merits of both techniques while addressing their limitations in overlooking node feature information and the need for labeled data.

**FadiGNN**, a novel model based on two components: adapted personalized PageRank [129] that considers node feature information and GCN that allows for the gradual learning of node representations. Together, the model is a combination of learning-based approaches and unsupervised ranking methods that enhance personalized PageRank scores of the nodes in a graph.

The main contributions of this work are:

1. We introduce an unsupervised model named FadiGNN based on GNNs to compute the personalized PageRank scores of the nodes in a graph.

2. Our model incorporates both the graph structure and the node features, offering an accurate node importance ranking.

3. We show the effectiveness of our model in two learning tasks; node classification and active learning. The model's performance is validated on widely used datasets including large graph data, showcasing its superiority over state-of-the-art models.

## 4.2 Related Work

Methodologies for evaluating node importance within a graph can generally be divided into two categories: traditional techniques and learning-based techniques [155].

*Traditional techniques* use heuristic metrics to identify key nodes within a graph. A class of techniques widely employed is the centrality measures, including node degree, betweenness, closeness, Eigenvector, Harmonic, and PageRank [32, 138, 156]. These measures offer valuable insights into the importance of nodes based on connectivity, proximity, and influence, etc. Nevertheless, most of these measures are computationally expensive and may become challenging and impractical over large-scale graphs. Over the years, studies have sought to address this issue by proposing modified versions of these measures. Xiang et al. [157] proposed an efficient algorithm based on progressive sampling and shortest-path approximation. The algorithm creates network centroids using the adjacency information entropy of the nodes and initially estimates the shortest paths. Xu et al. [158] proposed an important node detection method based on node adjacency information entropy by considering the weights and directions of the edges in graphs. Bowater et al. [159] extended the adapted PageRank algorithm by modifying the jumping probabilities so that the random walker jumps to nearby intersections rather than distant ones.

An advantage of these techniques is their applicability in the unsupervised setting, reducing costs associated with labeling real-world datasets. However, a significant drawback is that node features are ignored.

*Learning-based techniques* turn to machine learning models to learn the important nodes in a graph. For example, Fan et al. [154] turn the task into a learning problem, by employing an encoder-decoder framework. The encoder represents nodes with embedding vectors based on the network structure, and the decoder transforms these into scalars, indicating the relative betweenness centrality rank. A ranking loss is used to train the model to identify the score of nodes based on their betweenness centrality. Maurya et al. [160] proposed a GNN-based model for approximating betweenness and closeness centrality. The model employs a selective feature aggregation scheme, separately considering incoming and outgoing paths for estimating the number of shortest paths through the node. Huang et al. [149] proposed a deep learning-based model by learning

spatio-temporal dependencies among roads. Utilizing a temporal graph attention model, this model captures traffic interactions and evaluates link influence on the entire road network, allowing the identification of critical links based on their importance ranking.

These learning-based models offer a notable advantage as they can leverage deep learning architectures to effectively capture both node feature and topology structure information. However, a significant drawback lies in their reliance on a supervised setting. These models can be initially trained on smaller graphs in a supervised manner, and then be transferred and applied to larger-scale evaluations, providing scalability and efficiency. For instance, the model proposed by Fan et al. [154] is able to assign betweenness centrality scores to nodes for larger graphs, by initially training on small-scale graphs.

To address the gaps in the literature, we propose FadiGNN, an unsupervised GCN-based model. Note that in a recent work, the CUL model [161] employs the eigenvector centrality technique based on the power iteration algorithm [162] in conjunction with the GCN. Compared with CUL, our model excels in two distinct aspects: architecture and experimental results. We enhanced the personalized PageRank algorithm by integrating it with the GCN model for node ranking, resulting in a more advanced variant. This extension is more flexible and accurate since it takes into account the personalized preferences rather than the traditional power iteration algorithm which computes global importance. Additionally, when compared to the CUL model, our model demonstrated superior performance.

## 4.3 Preliminaries

### 4.3.1 Problem Statement

Let $G = (V, E, \mathbf{X})$ be a graph with $N = |V|$ vertices where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of vertices, $E \subseteq V \times V$ is the set of edges, and the $i$-th row of $\mathbf{X} \in \mathbf{R}^{N \times f}$ is the feature vector of node $v_i$ with $f$ dimensions. We use $\mathbf{A} \in \{0, 1\}^{N \times N}$ to denote the adjacency matrix of graph $G$, where $\mathbf{A}_{ij} = 1$ if $(v_i, v_j) \in E$, otherwise $\mathbf{A}_{ij} = 0$.

We aim to generate a score vector $\mathbf{Z} \in \mathbf{R}^{N \times 1}$, where $z_i$ corresponds to the importance of the $i$-th node of graph $G$, based on feature matrix $\mathbf{X}$ and the adjacency matrix $\mathbf{A}$ in a learnable and unsupervised way. Intuitively, the importance of a node is measured

similarly to the personalized PageRank algorithm by following a random path through a graph starting from a chosen node and assigning higher scores to nodes that are more connected to the starting node. We aim to iteratively learn the feature information as well through the GCN model.

### 4.3.2 Graph Convolutional Networks

Graph Convolutional Networks (GCNs) [25] is a pivotal model in graph-based deep learning. Its core learning equation is defined as:

$$\mathbf{H}^{(k)} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(k-1)}\mathbf{W}^{(k)} \tag{4.1}$$

where $\mathbf{W}^{(k)} \in \mathbf{R}^{f\times d}$ is a learnable matrix ($d$ is the node embedding dimensionality); $\tilde{\mathbf{D}}$ is the diagonal matrix of $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ with $\mathbf{I}_N$ being an identity matrix; $\mathbf{H}^{(k)} \in \mathbf{R}^{N\times d}$ is computed after $k$ steps of the GCN layer and $\mathbf{H}^{(0)} = \mathbf{X}$.

This equation describes the propagation rule of the information across graph nodes through successive layers in a GCN. It summarizes how each layer aggregates information from neighboring nodes, weighted by the learnable matrix $\mathbf{W}$, and updates the node embeddings accordingly.

## 4.4 Proposed Model

Personalized PageRank is an unsupervised algorithm that ranks nodes within a graph based on their proximity to a starting node. It traverses the graph along random paths and assigns higher scores to nodes closely connected to the starting node. While it is strong in the identification of important nodes within a graph, a significant limitation is the graph feature ignorance. Our model is inspired by the adapted personalized PageRank [129], which is a solution to this issue. As we aim to embed it into a trainable framework, we integrate GCN into our model. This learning process empowers FadiGNN to gradually exploit the feature and structural information of nodes and allows it to achieve better accuracy in the node ranking process.

In Section 4.4.1, we will detail the idea of FadiGNN. In Section 4.4.2, we present the module structure. Figure 4.1 illustrates the model structure and components.

### 4.4.1 Feature-aware Personalized PageRank

The PageRank algorithm [138] evaluates the importance of web pages (nodes) based on the structure of a graph, considering how nodes are linked to each other. The basic idea is that nodes with more incoming links from other important pages are deemed more important themselves. The original PageRank equation, $\pi_{pr} = \tilde{\mathbf{A}}_{rw}\pi_{pr}$, uses a matrix $\tilde{\mathbf{A}}_{rw} = \mathbf{A}\mathbf{D}^{-1}$, where $\pi_{pr}$ is the pagerank score. Personalized PageRank as a special version of PageRank that is modified to focus on a specific starting point [138]. For node $x$, personalized PageRank uses $\pi_{ppr}(i_x) = (1-\alpha)\hat{\tilde{\mathbf{A}}}\pi_{ppr}(i_x) + \alpha i_x$, where $\alpha$ (a scalar between 0 and 1) represents the likelihood of restarting from the chosen node and $i_x$ is a one-hot indicator vector. By using unit matrix $\mathbf{I}_n$ instead of $i_x$, the fully personalized PageRank matrix is as follows:

$$\mathbf{\Pi}_{ppr} = \alpha(\mathbf{I}_n - (1-\alpha)\hat{\tilde{\mathbf{A}}})^{-1} \tag{4.2}$$

Here, the arrays of matrix $\mathbf{\Pi}_{ppr}$ specifies the influence scores of nodes on each other. Due to the matrix symmetry characteristics, $\mathbf{\Pi}_{ppr}^{(xy)} = \mathbf{\Pi}_{ppr}^{(yx)}$.

To utilize the influence scores, Gasteiger et al. [129] use the PageRank matrix to achieve better generalization by adding feature information to compute the embedding $\mathbf{Z}_{PPNP} = softmax(\mathbf{\Pi}_{ppr}\mathbf{H})$, where $\mathbf{H} = F(\mathbf{X}, \mathcal{W})$. To efficiently compute $\mathbf{\Pi}_{ppr}\mathbf{H}$, they proposed a propagation scheme called APPNP. The APPNP model incorporates skip connections to enhance the model's expressiveness, which is written as follows:

$$\mathbf{Z}^{(0)} = \mathbf{H} = F(\mathbf{X}, \mathcal{W}) \tag{4.3}$$

$$\mathbf{Z}^{(K+1)} = (1-\alpha)\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{Z}^{(k)} + \alpha\mathbf{H} \tag{4.4}$$

Here, $F$ is a function approximation, $\mathcal{W} \in \mathbf{R}^{f\times d}$ is the weight matrix and $\alpha \in (0, 1]$.

Note that the matrix $\mathbf{\Pi}_{ppr}$ always exists. Indeed $\mathbf{\Pi}_{ppr}$ exists iff the determinant $det(\hat{\tilde{\mathbf{A}}} - \frac{1}{1-\alpha}\mathbf{I}_n) \neq 0$. Since $\alpha \in (0, 1]$, then $\frac{1}{1-\alpha} > 1$. Further, since the matrix $\hat{\tilde{\mathbf{A}}}$ has the same

eigenvalues as $\tilde{\mathbf{A}}_{rw}$, and the largest eigenvalue of $\tilde{\mathbf{A}}_{rw}$ is 1, $\frac{1}{1-\alpha}$ cannot be an eigenvalue. Thus, $\mathbf{\Pi}_{ppr}$ always exists.

Also, note that the approximation method always converges to the $\mathbf{\Pi}_{ppr}\mathbf{H}$. By expanding Equation 4.4, then

$$\mathbf{Z}^{(k)} = \left( (1-\alpha)^k (\hat{\tilde{\mathbf{A}}})^k + \alpha \sum_{i=0}^{k-1} (1-\alpha)^i (\hat{\tilde{\mathbf{A}}})^i \right) \mathbf{H} \tag{4.5}$$

If we take the limit $k \to \infty$, the first term of the summation tends to be 0 (since $\alpha \in (0, 1]$), and the second term becomes a geometric series. Since $\alpha \in (0, 1]$ and also $\hat{\tilde{\mathbf{A}}}$ is symmetrically normalized, the series converges and therefore $\det(\hat{\tilde{\boldsymbol{A}}}) \leq 1$, thus:

$$\mathbf{Z}^{(\infty)} = \alpha \left( \mathbf{I}_n - (1-\alpha)\hat{\tilde{\mathbf{A}}} \right)^{-1} \mathbf{H} \tag{4.6}$$

that is the equation to calculate $\mathbf{\Pi}_{ppr}\mathbf{H}$.

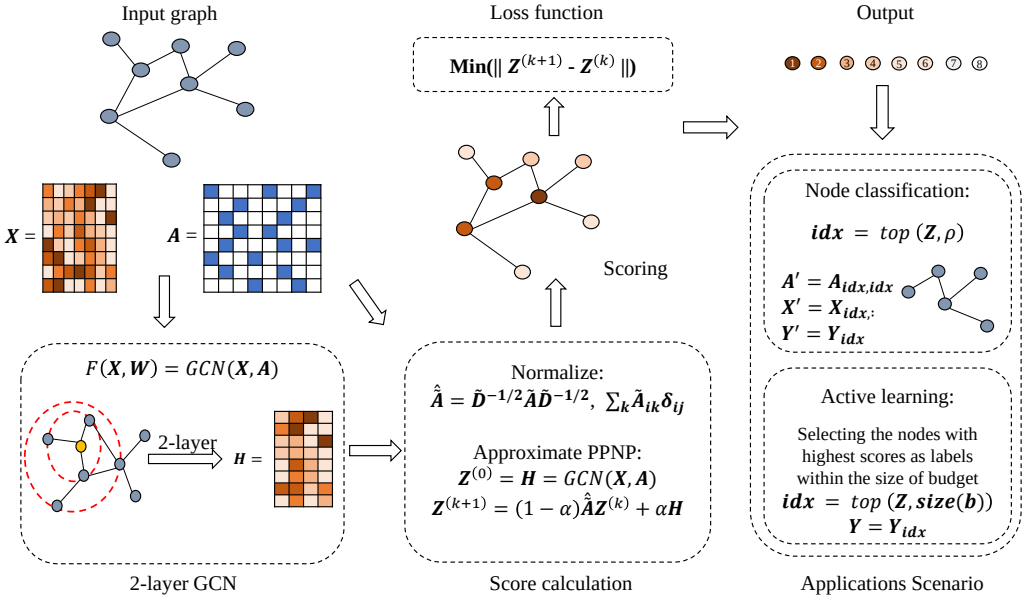



FIGURE 4.1: Structure of our model FadiGNN. The model starts by applying a 2-layer GCN model on the adjacency matrix $\boldsymbol{A}$ and feature matrix $\mathbf{X}$ to compute the initial node embeddings. The generated embeddings are then used in the adapted personalized PageRank algorithm for score initialization. In the end, the loss function is used to reduce the difference between consecutive outputs for the learning process.

### 4.4.2 Model Architecture

In our model architecture, we rely on the approximation strategy outlined in Section 4.4.1. This approximation strategy was originally designed for node classification tasks. Yet, as we seek to extend its utility to evaluate node importance in a graph, and also given the iterative nature of this approximation, ensuring convergence requires defining an appropriate loss function. Therefore, we define the loss function as:

$$\ell(\mathbf{Z}^{(k+1)}, \mathbf{Z}^{(k)}) = ||\mathbf{Z}^{(k+1)} - \mathbf{Z}^{(k)}||_2 \tag{4.7}$$

where $\mathbf{Z}^{(k+1)}$ and $\mathbf{Z}^{(k)}$ are the outputs at epochs $k+1$ and $k$, respectively. This loss function measures the difference between every two epochs and leads the iterative process toward the convergence.

**Algorithm 2:** FadiGNN algorithm

**Input:** Adjacency matrix $\mathbf{A}$, node features $\mathbf{X}$, number of layers $L$, learnable weight matrices $\mathbf{W}^{(1)}, \cdots \mathbf{W}^{(L)}$, $\mathcal{W}$, and MaxIter (1)
**Parameter:** teleport probability $\alpha$, GNN parameters (2)
**Output:** Score vector $\mathbf{Z}$ (3)
Initialize $\mathbf{Z}^{(0)} = \mathbf{H}^{(0)} = F(\mathbf{X}, \mathcal{W})$ (4)
Compute $\hat{\tilde{\mathbf{A}}} = \hat{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \hat{\mathbf{D}}^{-\frac{1}{2}}$ (5)
**for** $l = 1$ *to* $L$ **do** (6)
  $\mathbf{H}^{(l)} = \hat{\tilde{\mathbf{A}}} \mathbf{H}^{(l-1)} \mathbf{W}^{(l)}$ (7)
  $\mathbf{H}^{(l)} = \text{ReLU}(\mathbf{H}^{(l)})$ (8)
  **for** $k = 1$ *to* *MaxIter* **do** (9)
    **return** $\mathbf{Z}^{(k)} = \mathbf{H}^{(L)}$ (10)
    $\mathbf{Z}^{(k+1)} = (1-\alpha) \hat{\tilde{\mathbf{A}}} \mathbf{Z}^{(k)} + \alpha \mathbf{H}^{(0)}$ (11)
  **end** (12)
  Loss $||\mathbf{Z}^{(k+1)} - \mathbf{Z}^{(k)}||_2$ (13)
**end** (14)
**return** $\mathbf{Z} = \mathbf{Z}^{(MaxIter)}$ (15)

It is noteworthy that as $\ell(\mathbf{Z}^{(k+1)}, \mathbf{Z}^{(k)})$ approaches zero, incorporating additional regularization (e.g., $\mathcal{L}_2$ regularization) can be introduced to prevent the loss function from reaching zero. The detailed training procedure of our model is described in Algorithm 2. Additionally, as mentioned earlier, function $F(\mathbf{X}, \mathcal{W})$ specified in Equation 4.3 is a function approximation that produces the embeddings. Various techniques can be used for this function, including GNNs. This step is followed by the iterative process outlined in Equation 4.4.

### 4.4.3 Training Time Cost Analysis.

The key part of our model is the computation of $\mathbf{H} = F(\mathbf{X}, \mathcal{W})$, where different function approximation can be employed. This leads to varying computational complexities. For instance, a 2-layer GCN computational time is of $\mathcal{O}(N^2)$ where $N$ is the number of nodes. Experimentally, its computational time per epoch was measured over graphs of different sizes as follows: Cora (2,708 nodes and 10,556 edges, 0.0027s), CiteSeer (3,327 nodes and 9,104 edges, 0.0027s), Pubmed (19,717 nodes and 88,648 edges, 0.0036s), CS (18,333 nodes and 163,788 edges, 0.0086s), and Physics (34,493 nodes and 495,924 edges, 0.017s). By incorporating newer methods that offer lower complexity, the overall complexity of our approach decreases accordingly. This reduction in complexity enhances the scalability to handling larger graphs.

## 4.5 Experiments

To validate the effectiveness of our scoring model, we test on two application scenarios: multi-class node classification and active learning.

### 4.5.1 Node Classification

For node classification, we initially create a condensed graph $G^{'} = (V^{'}, E^{'}, \mathbf{X}^{'})$ by selecting top-ranked nodes computed by our Algorithm 2. This experiment is based on the premise that the condensed graph $G^{'}$ accurately represents the most pivotal structural details and effectively determines the most informative nodes while enhancing computational efficiency [128]. Subsequently, we employ different GNNs models, among which GCN is included, trained on $G^{'}$. Using the trained model, we proceed to assign labels to nodes within the original graph $G$. The detailed procedure of the node classification experiment is described in Algorithm 3.

Note that the weight matrix $\mathbf{W}$ in GCN is of size $f \times d$. This dimensionality of $\mathbf{W}$ remains consistent regardless of the size of the graph. Since $\mathbf{W}$ is independent of the graph size, it can be applied to the original graph without any compatibility issues. Parameters learned from the smaller graph $G^{'}$ are transferable and can be efficiently employed for evaluating the original graph $G$.

**Algorithm 3:** Algorithm for the node classification

**Input:** Original graph $G = (V, E, \mathbf{X})$, initial labels $Y$, ratio $\rho$, and a GNN model **(1)**

Apply a ranking method (e.g. FadiGNN) to compute a score vector $\mathbf{Z}$ for nodes $V$

Compute $idx = \text{top-rank}(\mathbf{Z}, \rho)$ **(2)**

Compute adjacency matrix, feature matrix and labels: $\mathbf{A}^{'} = \mathbf{A}_{idx,idx}$, $\mathbf{X}^{'} = \mathbf{X}_{idx,:}$, **(3)**

$\mathbf{Y}^{'} = \mathbf{Y}_{idx}$

Learn weight matrix, $W^*$, to minimize $\ell(GCN_{G'}(W^*), Y^{'})$, such as negative log-likelihood loss **(4)**

**Output:** Node labels for the original graph $G$ **(5)**

#### 4.5.1.1 Datasets.

We use seven real-world datasets in our experiments, as summarized in Table 4.1.

#### 4.5.1.2 Competitors.

For the node classification task, we compare FadiGNN against three traditional graph algorithms including Personalized PageRank [138], EigenVector Centrality [163], and K-center [164], and also with three state-of-the-art graph coarsening methods including FGC [165], GCOND [166], and SCAL [128]. Graph coarsening methods simplify graphs by creating a smaller yet representative subgraph of an input graph while preserving essential connections. In addition, we compare against the CUL model [161] discussed in Section 4.2.

#### 4.5.1.3 Parameter Settings.

The default model to approximate $F(\mathbf{X}, \mathcal{W})$ is GCN (unless stated otherwise) which is trained with the Adam optimizer, employing a learning rate of 0.01 and applying $L_2$ regularization set at 0.0005 on the weights. We use an embedding size selected from $\{64, 128\}$ alongside a consistent dropout rate of 0.5 and the ReLU activation function between layers. The maximum number of training epochs is selected from $\{200, 400\}$ and we fine-tune the teleport probability, $\alpha$, within 0.85, and 0.95.

For the node classification task, we employ a 2-layer GCN model with the embedding dimension equal to 64 following a previous work [112]. We train the model using ADAM optimizer with a learning rate selected from $\{0.005, 0.001, 0.01\}$ and $L_2$ regularization of 0.0005 on the weights. The negative log-likelihood loss is the loss function. We use a

TABLE 4.1: Summary of datasets used in our experiment. All the datasets are available in Pytorch geometric library [3].

| Name | # Vertices | # Edges | # Classes | # Features |
|---|---|---|---|---|
| Cora | 2,708 | 10,556 | 7 | 1,433 |
| CiteSeer | 3,327 | 9,104 | 6 | 3,703 |
| Pubmed | 19,717 | 88,648 | 3 | 500 |
| Co-CS | 18,333 | 163,788 | 15 | 6,805 |
| Co-phy | 34,493 | 495,924 | 5 | 8,415 |
| CoraML | 2,995 | 16,316 | 7 | 2,879 |
| OGB-arxiv | 169,343 | 1,166,243 | 40 | 128 |

fixed dropout rate of 0.5 and the ReLU activation function is applied between the layers. The maximum number of training epochs is selected from $\{400, 1000, 2000, 3000\}$ and the training is terminated if the validation accuracy does not improve for 50 consecutive epochs. The reduction ratio $\rho$ is set accordingly and mentioned in tables and figures. In all of our implementations, we use GNN libraries PyTorch Geometric[1] [3]. Our implementation is available on Github[2].

#### 4.5.1.4 Results.

Table 4.2 shows the node classification accuracy of the model on five real datasets. The best results are highlighted in boldface. The node classification accuracy values of existing graph coarsening methods are taken from the literature [165]. Our model FadiGNN outperformed the other methods in most cases, highlighting its superior capability in identifying important nodes. Even when trained on the smaller graph $G^{'}$, our model showed commendable accuracy. This means that the important and informative nodes are preserved in the smaller graph $G^{'}$ due to higher ranks gained through our model.

*Impact of underlying neural networks.* For a better assessment of our model, here we perform an ablation study over different models to approximate $F(\mathbf{X}, \mathcal{W})$, such as feed-forward networks (FNN), GCN [25], Sage [28], SGC [144], and GAT [27]. The results presented in Table 4.3 shows that GCN performing better in generating high-quality scores.

*Comparing with learning on the full graphs.* Table 4.4 shows the node classification accuracy of our model compared to the personalized PageRank method, all trained on the graph $G^{'}$ with $\rho = 0.05$ and evaluated on the complete graph $G$. In particular,

[1] https://pytorch-geometric.readthedocs.io/en/latest/
[2] https://github.com/ghimohammadr/FadiGNN

TABLE 4.2: Node classification accuracy of models on different datasets. The best results are highlighted in boldface. *Despite adjustments to parameters such as the number of epochs and learning rates, etc, there was no enhancement in the result of K-center values for the Co-phy dataset.

| Dataset | \ρ | Coarsening | | | | Traditional | | | **FadiGNN** | $\#G$ | $\#G'$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | GCOND | SCAL | FGC | K-center | Centrality | PPR | CUL | | Nodes | Nodes |
| Cora | 0.3 | 81.56±0.6 | 79.42±1.71 | 85.79±0.24 | 84.00±1.35 | 71.06±0.52 | 84.74±0.17 | 86.23±2.27 | **86.69±1.13** | 2,708 | 812 |
| | 0.1 | 81.37±0.40 | 71.38±3.62 | 81.46±0.79 | 72.16±3.96 | 25.15±1.75 | 77.05±0.62 | 80.07±2.27 | **82.05±0.30** | | 270 |
| | 0.05 | 79.93±0.44 | 55.32±7.03 | **80.01±0.51** | 58.12±4.50 | 16.00±0.17 | 73.75±1.51 | 71.79±2.45 | 74.24±3.20 | | 135 |
| Citeseer | 0.3 | 72.43±0.94 | 68.87±1.37 | 74.64±1.37 | 71.24±1.20 | 64.08±0.86 | 74.13±0.78 | 71.04±1.37 | **76.52±1.73** | 3,327 | 998 |
| | 0.1 | 70.46±0.47 | 71.38±3.62 | 73.36±0.53 | 66.37±1.23 | 29.85±2.17 | 67.56±1.76 | 66.99±1.65 | **73.63±1.23** | | 332 |
| | 0.05 | 64.03±2.4 | 55.32±7.03 | **71.02±0.96** | 64.28±4.09 | 20.40±0.14 | 59.10±1.12 | 60.84±3.03 | 67.46±0.97 | | 166 |
| Co-phy | 0.05 | 93.05±0.26 | 73.09±7.41 | 94.27±0.25 | 94.24±0.71 | 50.51±0.00* | 94.01±0.17 | 94.95±0.31 | **95.61±0.18** | 34,493 | 1724 |
| | 0.03 | 92.81±0.31 | 63.65±9.65 | 94.02±0.20 | 93.82±0.67 | 50.51±0.00* | 93.56±0.36 | 94.66±0.42 | **95.33±0.30** | | 1034 |
| | 0.01 | 92.79±0.4 | 31.08±2.65 | 93.08±0.22 | 92.32±1.70 | 50.51±0.00* | 90.36±0.67 | 93.06±0.56 | **93.60±0.27** | | 344 |
| Pubmed | 0.05 | 78.16±0.30 | 72.82±2.62 | 80.73±0.44 | 79.42±0.98 | 43.01±0.15 | 83.66±0.80 | 84.41±0.77 | **85.27±0.98** | 19,717 | 985 |
| | 0.03 | 78.04±0.47 | 70.24±2.63 | 79.91±0.30 | 76.22±0.94 | 42.37±0.77 | 81.31±0.23 | 83.28±0.67 | **83.65±0.69** | | 591 |
| | 0.01 | 77.2±0.20 | 54.49±10.5 | 78.42±0.43 | 75.13±2.95 | 43.08±1.18 | 76.90±0.60 | 77.23±0.89 | **80.49±0.41** | | 197 |
| Co-CS | 0.05 | 86.29±0.63 | 34.45±10.0 | 89.60±0.39 | 86.00±6.65 | 74.73±0.58 | 88.24±0.27 | 88.04±0.82 | **91.49±0.24** | 18,333 | 916 |
| | 0.03 | 86.32±0.45 | 26.06±9.29 | 88.29±0.79 | 80.20±12.96 | 63.50±0.21 | 84.87±0.57 | 84.42±0.92 | **88.66±0.22** | | 549 |
| | 0.01 | 84.01±0.02 | 14.42±8.5 | **86.37±1.36** | 67.08±22.35 | 40.56±1.03 | 79.04±0.29 | 67.55±0.15 | 83.28±0.57 | | 183 |

TABLE 4.3: The performance of our model through different algorithms as function approximation $F(\mathbf{X}, \mathcal{W})$. Models are trained on $\rho = 0.05$ of the graph.

| Dataset | Sage | FFN | GCN | SGC | GAT |
|---|---|---|---|---|---|
| Cora | 70.40±1.12 | 68.21±2.95 | 74.24±3.20 | 70.11±0.97 | 63.88±2.11 |
| Citeseer | 62.93±0.81 | 62.75±0.79 | 67.46±0.97 | 60.48±0.55 | 57.01±1.52 |
| Co-phy | 94.53±0.62 | 94.79±0.26 | 95.61±0.18 | 94.30±0.46 | 94.29±0.20 |
| Pubmed | 82.97±0.62 | 81.95±0.96 | 85.27±0.98 | 83.37±0.31 | 84.48±0.83 |
| Co-cs | 89.91±0.34 | 89.37±0.73 | 91.49±0.24 | 91.43±0.34 | 91.00±0.54 |

TABLE 4.4: The table shows how well our model performs compared to the original Graph (No Coarsening). Additionally, the results indicate that the accuracy of our model trained on $\rho = 0.05$ of the graph is much better than the PageRank model.

| Dataset | $G'$ (PPR) | $G'$ (**FadiGNN**) | Full graph $G$ |
|---|---|---|---|
| Cora | 73.75±1.51 | 74.24±3.20 | 89.50±1.23 |
| Citeseer | 59.10±1.12 | 67.46±0.97 | 79.10±1.68 |
| Co-phy | 94.01±0.17 | 95.61±0.18 | 96.22±0.72 |
| Pubmed | 83.66±0.80 | 85.27±0.98 | 88.89±0.59 |
| Co-cs | 88.24±0.27 | 91.49±0.24 | 92.83±0.11 |

our model achieves better node classification accuracy due to the integration of feature information $\mathbf{X}'$. Results show the importance to use both the structure of the graph and its feature information to make the model more accurate.

Additionally, the last column in the table shows the performance of the GCN model when trained on the original graph $G$ rather than graph $G'$.

*Result on large dataset OGB-arxiv.* Table 4.5 presents the results of Eigenvector centrality, PPR, FadiGNN, and CUL [161] on the large dataset, OGB-arxiv. This dataset represents the citation network of CS papers from ARXIV indexed by MAG. Each node is a paper with a feature vector, and each directed edge indicates a citation.

TABLE 4.5: Accuracy of models on OGB-arxiv dataset.

| Model/$\rho$ | 0.01 | 0.03 | 0.05 |
|---|---|---|---|
| Centrality | 19.47±1.25 | 22.34±1.31 | 25.31±0.68 |
| PPR | 15.48±0.14 | 27.07±1.42 | 31.45±0.58 |
| CUL | 52.75±1.52 | 59.78±0.23 | 59.77±0.25 |
| **FadiGNN** | **53.83±1.16** | **59.87±0.28** | **59.91±0.23** |

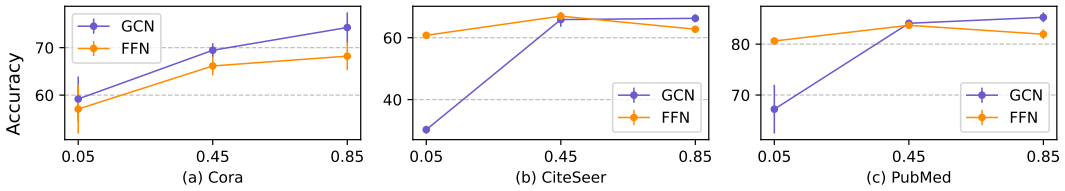



FIGURE 4.2: The influence of teleport probability on model accuracy is examined across three datasets. The GCN model is used for node classification. The plots represent the GCN and FFN as function approximation $F(\mathbf{X}, \mathcal{W})$, which are assessed for their performance under varying teleport probability.

*Impact of the teleport probability.* In this set of experiments, we explore the impact of the teleport probability parameter, denoted as $\alpha$, on model accuracy. As depicted in Figure 4.2, the node classification accuracy of both GCN and FNN models is investigated for Cora, Citeseer, and Pubmed datasets by varying the value of $\alpha$. The trend indicates the superior performance of GCN over FNN, with accuracy consistently improving as $\alpha$ increases. This observation highlights that as the value of $\alpha$ increases, there is a greater inclination of the model to rely on the features of the data rather than purely structural characteristics. This also highlights the potential impact of the feature information during the learning process.

### 4.5.2 Active Learning

In situations where data labeling is an expensive process, active learning operates by selecting the most valuable and informative samples for labeling. This approach proves especially advantageous where there are resource constraints and allows machine learning models to achieve high performance with a smaller labeled dataset. Here, the objective is to demonstrate the proficiency of our model in ranking important nodes for labeling. To this end, we follow the structure of the SPA model [167]. The SPA model combines community detection utilizing the SCAN algorithm with the PageRank scoring approach to enhance the effectiveness and informativeness of sample selection. The model prioritizes nodes that are informative and central in structure.

**Algorithm 4:** SPA+FadiGNN for active learning

**Input:** Adjacency matrix $\mathbf{A}$, damping factor $\alpha$, labeling budget $b$; **(1)**
Initialize $S = \emptyset$ **(2)**
Convert matrix $A$ to graph $G$ **(3)**
Implement SCAN algorithm to detect communities $\{C_1, C_2, \ldots, C_k\}$ **(4)**
**for** *each community* $C_i$ *in* $\{C_1, C_2, \ldots, C_k\}$ **do** **(5)**
  Run FadiGNN to calculate scores for all nodes in $C_i$ with damping factor $\alpha$ **(6)**
  Find node in $C_i$ with the highest score **(7)**
  Add the node to $S$ **(8)**
**end** **(9)**
**if** $|S| < b$ **then** **(10)**
  Calculate scores for all nodes in $G \backslash S$ **(11)**
  Sort nodes in $G \backslash S$ by scores in descending order **(12)**
  Add nodes from the sorted list to $S$ until $|S| = b$ **(13)**
**end** **(14)**
**return** $S$ to be labeled **(15)**

TABLE 4.6: Overview of the Macro-F1 performance of the SPA model for active learning employing the GCN architecture. The provided numerical values represent the average Macro-F1 score with the most outstanding score highlighted in bold.

| Model\Data | Citeseer | | | CoraML | | |
|---|---|---|---|---|---|---|
| | 20 | 40 | 80 | 20 | 40 | 80 |
| Random | 28.4 ±12.6 | 37.6±6.7 | 48.9±5.8 | 48.07±6.20 | 64.53±5.17 | 73.93±3.98 |
| SPA + PageRank | 46.3±0.5 | 57.1±0.1 | 60.8±2.4 | 54.41±0.95 | 72.84±1.39 | 79.44±0.27 |
| SPA + **FadiGNN** | **50.31±4.67** | **59.23±0.82** | **62.98±0.72** | **56.81±2.45** | **74±1.62** | **79.87±0.24** |

Here, to evaluate the effectivenessv our model, we replace the simple PageRank with our scoring model, FadiGNN. The detailed procedure of the active learning approach is described in Algorithm 4. Additionally, To ensure a fair comparison, we adopted the same structural framework and parameter values as used in [167] including the clustering threshold $\epsilon = 0.5$ and minimum neighbors $\mu = 3$. Table 4.6 represents the results of different approaches for the Citeseer and CoraML datasets. We evaluated the model's performance for different budgets including 20, 40, and 80. In all cases, the Random method demonstrates significantly lower F1 scores compared to the other models that highlights the effectiveness of emphasizing on the importance nodes. All methods show an overall trend of enhanced F1 scores as the budget increases. While the SPA model demostrates a solid performance in utilizing PageRank to identify importance nodes, however, FadiGNN enhances the models capacity and consistently achieves the highest scores across all cases.

## 4.6 Conclusion

We presented FadiGNN, a GCN-based model designed to evaluate node importance in a graph by integrating both node features and topological structure in an unsupervised manner. FadiGNN begins by processing a graph's adjacency matrix and feature matrix with a GCN model to compute the embedding for nodes. Subsequently, the node embeddings are employed in the adapted personalized PageRank algorithm to produce the output vector. Finally, the output vector used to compute the loss function and determine the nodes scores. FadiGNN is validated on real-world datasets with two application scenarios: node classification and active learning. The results on node classification shows that FadiGNN outperforms the state-of-the-art models by an average of 7.62% in terms of accuracy across the datasets.

A significant advantage of our framework lies in its flexibility, as it can be easily adapted to various centrality techniques, such as KATZ centrality, total communicability centrality, and others. However, a major drawback compared to traditional methods is the added complexity from the training process. Future work could explore potential solutions to reduce the complexity. For instance, incorporating faster approximation methods such as scalable GNNs, to provide a more efficient alternative that preserves the framework's adaptability while improving performance in larger datasets. Another direction is the potential for extending the proposed approach to other types of graphs, such as knowledge graphs, which have been the focus of a few recent studies [168, 169].

# Chapter 5

# Centrality-aware Hierarchical Graph Pooling

*Graph pooling is the backbone of graph classification that condense graph information into a compact form. Current pooling methods often treat all node information irrespective of their importance limits their ability in capturing either key graph substructures or node information. To this end, we propose CentralityPool, a hierarchical pooling method for sieving the most important nodes via centrality techniques and conduct experiments over graph classification and node classification task. We employ four centrality techniques: Personalized PageRank, KATZ, Total Communicability, and Eigenvector centrality to identify key nodes. Due to time-consuming, we incorporate the efficient approximation strategies to reduce computation complexity. Hence, at comparable computation time with SOTA methods, CentralityPool improves graph classification accuracy by up to 4% by average across eight benchmark datasets and achieves competitive node classification results across large-scale benchmark datasets.*

This chapter has been submitted and is currently under its second revision (minor revision) on April 20, 2025, at the IEEE Transactions on Network Science and Engineering.

- Ghanbari, M., Bagloee, SA., Qi, J., and Sarvi, M. CentralityPool: Centrality-aware Hierarchical Graph Pooling. IEEE Transactions on Network Sciences and Engineering, 1-13, 2025.

## 5.1 Introduction

Graph Neural Networks (GNNs) [14, 25, 27, 28, 40] are a type of deep learning model that has gained considerable attention in recent years due to their strong performance and the widespread presence of graph-structured data. These models have been applied across various domains such as transportation [170], recommender systems [171, 172], natural language processing [173], and computer vision [174], reporting impressive empirical results. A common task in GNNs is graph-level representation learning, which maps graphs into low-dimensional spaces and enables more effective processing of downstream tasks. As a key component, graph pooling [2, 100, 175–177] is a technique for graph-level representation learning. By aggregating and summarizing node features, graph pooling creates compact, informative representations that facilitate more efficient processing and analysis. This technique plays a vital role in improving the performance of downstream tasks, such as graph classification, by reducing the dimensionality and capturing important structural information from the original graph.

Graph pooling techniques can generally be categorized into two main classes: global pooling [178] and hierarchical pooling [175]. In global pooling, the graph representation is generated in a single step. In contrast, hierarchical pooling coarsens the graph in different stages (See Fig. 5.1). Node clustering pooling [100, 179] and node drop pooling [2] are the two main methods of hierarchical pooling. In node clustering, the nodes are grouped into clusters and new nodes are generated based on these clusters; in node drop pooling, a subset of nodes from the original graph is selected based on some importance scores. A critical aspect of node drop pooling is how nodes are scored based on their importance.

In this chapter, we study hierarchical graph pooling and propose a centrality-aware pooling layer named *CentralityPool*, which exploits centrality measures in an end-to-end fashion. Centrality techniques effectively identify the most influential nodes within a graph. Using them to guide graph pooling ensures that key structural information is preserved during pooling. By focusing on the central nodes, centrality-aware pooling layers enhances the quality of graph representations and consequently leads to improved performance. Additionally, centrality measures provide a more interpretable and adaptable approach to pooling, making them suitable for various types of graphs.

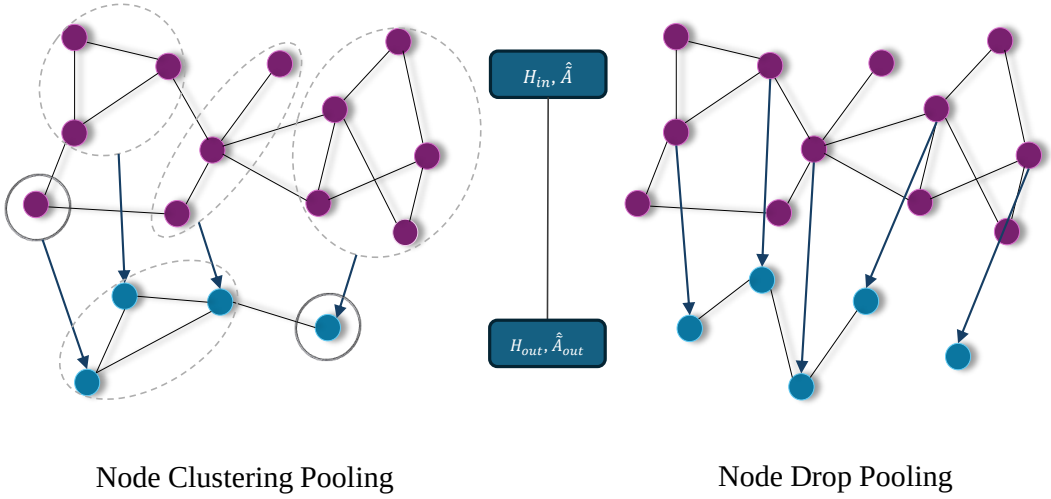



FIGURE 5.1: Node clustering pooling and node drop pooling are the two main methods of hierarchical pooling. Node clustering pooling operates by grouping nodes into clusters based on their similarities or structural properties. Node drop pooling, on the other hand, focuses on selecting a subset of nodes from the original graph based on the node importance scores.

Our goal is to design a framework that can be integrated with a range of centrality measures while considering both graph structural information and node feature information. To overcome the computational challenges typically associated with centrality techniques, we employ their respective approximation methods. The approximations significantly reduce complexity, making our solution more practical and scalable. We consider the following well-known centrality measures:

- Personalized PageRank centrality [138], which identifies nodes that are important because they are linked to other important nodes. A damping factor is used to balance between following links and randomly jumping to other nodes, which allows for focusing on local as well as global node importance.
- KATZ centrality [180], which accounts for both directly and indirectly connected nodes, with longer paths being discounted more which provides a measure of influence.
- Total Communicability centrality [181], which measures how well a node can communicate with the rest of the graph, considering all possible paths where longer paths contributing less to the score.
- Eigenvector centrality [182], which ranks nodes based on their connections to other high-score nodes, capturing the recursive nature of influence within the network.

Additionally, we found that centrality measures can also be used as convolution layers in message passing component of GNNs, due to their structural similarity in the form of an adjacency matrix. Therefore, we also provide experiments over node classification task on the public benchmarks.

The main contributions of this work are:

1. We propose CentralityPool, a novel hierarchical graph pooling framework based on centrality techniques to select the important nodes.

2. We integrate CentralityPool with four well-known centrality measures: personalized PageRank, KATZ centrality, total communicability, and eigenvector centrality. We exploit the approximation properties of these centrality measures to significantly reduce the complexity of the proposed methods.

3. We conduct extensive experiments on both graph classification (eight datasets) and node classification (six datasets) tasks across common benchmarks. The results demonstrate the effectiveness of the CentralityPool and highlight its superiority over state-of-the-art pooling and convolution layer competitors.

The remainder of the paper is organized as follows. We first cover related work. Then, we give the problem definition and present our framework. After that, we discuss the experimental results for graph classification and multi-class node classification tasks and finally, conclude the paper.

## 5.2 Related Work

In recent years, a wide variety of graph pooling methods have emerged, aiming to improve the efficiency and effectiveness of GNNs by reducing graph size while preserving crucial information. These methods can be broadly categorized into two main types: *global pooling* and *hierarchical pooling*. Hierarchical pooling itself can be further subdivided into two subcategories: node clustering pooling and node drop pooling. Each category offers unique approaches to graph pooling, with specific advantages and limitations. In the following subsections, we provide a detailed review for each of these categories.

### 5.2.1 Global Pooling

Global pooling methods aim to generate a graph representation in a single, holistic step. In these methods, the entire graph is typically processed to produce a fixed-size vector representation, which serves as the final embedding representation of the graph. This method is straightforward and efficient, especially for small to medium-sized graphs, as it pools the graph into a smaller graph in one go. For example, SortPool [97] ranks nodes by their scores, which are derived from the nodes' transformed features. GlobalAttention [98] employs an attention mechanism to selectively aggregate node features and enhances the graph representation. GMT [99] formulates the pooling problem as a multiset encoding with auxiliary information. It applies multi-head attention to capture interactions between nodes. Global pooling can struggle to capture structural details, particularly in large and complex graphs, where the reduction process may oversimplify the graph's inherent intricacies. Despite these challenges, global pooling remains a popular choice due to its simplicity and computational efficiency.

### 5.2.2 Node Clustering Pooling

Node clustering pooling operates by grouping nodes into clusters based on their similarities or structural properties. This method effectively reduces the graph's complexity by treating each cluster as a single node in a coarsened graph and therefore, preserving the local structural information while simplifying the overall graph representation. Node clustering pooling is particularly advantageous for capturing the hierarchical nature of graphs, as it allows the model to iteratively pool the graph while retaining key features. Various methods have been proposed within this subcategory, each offering different strategies for cluster formation and aggregation. DiffPool [100] is a differentiable graph pooling method that uses GNNs to learn the assignment matrices and cluster the similar nodes into a fixed number of clusters. StructPool [102] integrates conditional random fields into the graph pooling problem. SEP [183], as inspired by structural entropy, designs a global optimization algorithm to generate the cluster assignment matrices for pooling. A general limitation of this class of pooling methods is their inefficiency due to high time and storage requirements. This inefficiency arises from the need to compute a dense cluster assignment matrix, which can be resource-intensive, especially with larger graphs [99].

### 5.2.3 Node Drop Pooling

Node drop pooling, focuses on selecting a subset of nodes from the original graph based on the node importance scores. Generally, a score generator function is used for scoring nodes and subsequently a node selector function, usually the $\text{Top}_k$ function, is applied to generate the coarsened graph. This method is more straightforward than node clustering pooling in terms of graph reduction, as it directly removes less important nodes. An advantage is the computational efficiency. TopKPool [106] uses a trainable projection vector to generate importance scores for node selection. SAGPool [2] uses GCNs models to assign scores, adapting to the graph's structure. SSPool [1] identifies the most informative subset of nodes by estimating the mutual information between the coarsened graph and an augmented version, ensuring that the pooled graph retains essential features.

An issue with the node drop pooling method is that it requires careful consideration in the node selection process to ensure that the resulting graph representation remains informative and effective.

To this end, we present and investigate hierarchical pooling based on centrality techniques. We design pooling layers that uses centrality measures to effectively identify the most influential subset of nodes within a graph and ensures that key structural information is preserved during pooling. Our method is able to capture multi-scale information by coarsening the graph, due to the hierarchical structure. Moreover, as a node drop pooling technique, it is efficient. Based on the results of our experiments on graph classification, incorporating centrality-aware pooling layers has demonstrated the potential to significantly enhance the quality of graph representations, which are able to effectively capture important structural information within the graph.

In addition, since centrality techniques can be interpreted as a variant of the adjacency matrix which enables them to function as convolution layers, we implemented these methods and provided experiments on the node classification task on six common datasets (including OGB large datasets). The results showed a competitive performance on different datasets.

## 5.3 Preliminary

### 5.3.1 Problem Statement

Let $G = (V, E, \mathbf{X})$ be a graph with $N = |V|$ vertices where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of vertices, $E \subseteq V \times V$ is the set of edges, and the $i$-th row of $\mathbf{X} \in \mathbf{R}^{N \times f}$ is the feature vector of node $v_i$ with $f$ dimensions. We use $\mathbf{A} \in \{0, 1\}^{N \times N}$ to denote the adjacency matrix of graph $G$, where $\mathbf{A}_{ij} = 1$ if $(v_i, v_j) \in E$, otherwise $\mathbf{A}_{ij} = 0$.

While the primary focus of our method is on graph classification, we also extended our experiments to include node classification to demonstrate the broader applicability of our proposed techniques. In graph classification, the objective is to train a model that can accurately predict the class label for a graph, whereas in node classification, the focus is on predicting the class labels for individual nodes within a graph.

In what follows, we briefly describe Graph Convolutional Networks, graph centrality computation techniques that serve as the basis of our proposed framework.

### 5.3.2 Graph Convolutional Networks

Graph Convolutional Networks (GCNs) are a class of neural networks designed to operate directly on graph-structured data. They extend the concept of convolutional operations from grid-based data to graph data. The core idea of GCNs is to aggregate information from a node's neighbors to compute a new node representation. This is achieved through a layer-wise propagation mechanism, where each node updates its representation based on its neighbors' features information. The GCNs layer is formulated as follows:

$$\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W} \tag{5.1}$$

Here, $\mathbf{W} \in \mathbf{R}^{f \times d}$ is a learnable weight matrix, where $d$ represents the node embedding dimensionality. The term $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ denotes the adjacency matrix of the graph with added self-loops, where $\mathbf{I}_N$ is the identity matrix, ensuring each node also considers its own features during the aggregation process. $\tilde{\mathbf{D}}$ is the diagonal degree matrix associated with $\tilde{\mathbf{A}}$, used to normalize the adjacency matrix symmetrically. The output of the GCN

layer is an $N \times d$ matrix where each row corresponds to the updated features of a node. The raw node feature matrix $\mathbf{X}$ is the input to the network.

This approach allows GCNs to effectively capture both node features and graph structure, making them useful for tasks such as node classification and graph classification.

### 5.3.3 Iterative Centralities

In the following subsections, we explain four well-known centrality measures, including an in-depth analysis of their approaches and their approximation techniques.

**Total Communicability Centrality (TC)** Total Communicability Centrality [182] is based on the idea that a node is important if it can communicate with many other nodes, not just directly but also through indirect connections. This centrality measure considers all possible walks in the graph, weighted by their length.

The total communicability equation is as follow:

$$EXP(\hat{\tilde{\boldsymbol{A}}}) = e^{\hat{\tilde{\boldsymbol{A}}}} \tag{5.2}$$

where $e$ denotes the exponent function.

Due to the inherent difficulty involved in calculating $e^{\hat{\tilde{\boldsymbol{A}}}}$ directly, we take advantage of its fast approximation. The approximation strategy significantly reduces computational complexity while trying to maintain the accuracy. The detailed equation is as follows:

$$e^{\hat{\tilde{\boldsymbol{A}}}} = \sum_{k=0}^{\infty} \frac{\hat{\tilde{\boldsymbol{A}}}^k}{k!} \tag{5.3}$$

Here, a walk is a sequence of edges that connects a series of nodes, and $\hat{\tilde{\boldsymbol{A}}}^k$ represents all possible walks of length $k$. Longer walks contribute less to the centrality score because they are less direct. This is reflected in the series expansion, where each term $\frac{\hat{\tilde{\boldsymbol{A}}}^k}{k!}$ in the summation accounts for walks of length $k$, and the factorial in the denominator reduces the impact of longer walks.

**Personalized PageRank Centrality (PPR)** PageRank centrality [129, 138] is based on the idea that a node is important if it is linked to other important nodes. This concept was originally developed by Google to rank web pages in their search engine. The intuition is that a web page linked by many other pages (especially those that are themselves important) is likely to be important. The PageRank matrix can be written as follows:

$$\Pi = \alpha \left( \boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}} \right)^{-1} \tag{5.4}$$

Where $\Pi$ is the PageRank matrix, $\boldsymbol{I}_n$ is the identity matrix, and $\alpha$ is the damping factor. Again, due to the difficulty in calculating the inverse matrix, we utilize a fast approximation method to simplify the process and reduce the computational complexity. The approximation incorporates skip connections to enhance the model's expressiveness, which is written as follows:

$$\pi^{k+1} = (1-\alpha)\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\pi^k + \alpha\pi^0 \tag{5.5}$$

Where $\alpha \in (0, 1]$. The damping factor $\alpha$ controls how much of the importance a node passes to its neighbors. The process continues until it reaches the maximum iteration number, $K$, where the distribution of importance no longer changes significantly between iterations.

**Eigenvector Centrality (EV)** Eigenvector centrality [182] measures the importance of a node based on the importance of its neighbors. This technique is particularly useful in networks where connections to important nodes are more valuable than connections to many nodes. In other words, a node's importance is proportional to the sum of the importance of its neighbors. To this end, eigenvector centrality can be expressed by the following equation:

$$\hat{\tilde{\boldsymbol{A}}}\mathbf{e} = \mathbf{e} \tag{5.6}$$

In practical terms, we solve:

$$\hat{\tilde{\boldsymbol{A}}}\mathbf{e} = \lambda\mathbf{e} \tag{5.7}$$

Where $\mathbf{e}$ is eigenvector of matrix $\hat{\tilde{\boldsymbol{A}}}$. This equation states that the centrality vector $\mathbf{e}$ is an eigenvector associated with the largest eigenvalue $\lambda$.

To reduce the computational complexity and approximately find the principal eigenvector (and thus the centrality scores), the power method is commonly used [184]. This iterative method starts with an initial vector and repeatedly multiplies it by the matrix $\hat{\tilde{\boldsymbol{A}}}$ and normalizes the result each time until convergence to the principal eigenvector. The power iteration equation can be written as follows:

$$\mathbf{e}^{k+1} = \frac{\hat{\tilde{\boldsymbol{A}}}\mathbf{e}^k}{\left\|\hat{\tilde{\boldsymbol{A}}}\mathbf{e}^k\right\|} \tag{5.8}$$

This iterative process ensures that it converges to the principal eigenvector that represents the most important nodes in the network.

**KATZ Centrality (KATZ)** KATZ centrality [180] is similar to the total communicability centrality. It also give lower weight to longer walks between nodes. However, its weighting scheme is different. It provides a measure of influence by accounting for both the number of direct and indirect connections a node has, with the indirect connections weighted less.

The KATZ centrality equation can be represented in matrix form as follows:

$$\boldsymbol{K} = (\boldsymbol{I}_n - \beta\hat{\tilde{\boldsymbol{A}}})^{-1}\mathbf{1} \tag{5.9}$$

Where $\boldsymbol{I}_n$ is the identity matrix and $\mathbf{1}$ is a column vector of ones. The factor $\beta$ must be chosen carefully (it must be less than the reciprocal of the largest eigenvalue of $\hat{\tilde{\boldsymbol{A}}}$) to ensure that the series converges. It controls how much influence longer paths have on the centrality score.

Similarly, we employ a fast approximation strategy to compute the KATZ matrix as follows:

$$KATZ = \sum_{k=0}^{\infty}(\beta\hat{\tilde{\boldsymbol{A}}})^k \tag{5.10}$$

KATZ centrality considers all paths from one node to another, but it discounts longer paths more heavily by multiplying them by a small factor $\beta^k$. This ensures that direct

neighbors contribute more to a node's centrality than nodes connected through longer chains.

## 5.4 Proposed Model

In a standard hierarchical pooling procedure (illustrated in Fig. 5.2), a GCN-based model processes the entire graph $G$ for all nodes to produce node embeddings $\mathbf{H}_{\text{in}} \in \mathbf{R}^{N \times d}$ where $d$ is the node embedding dimensionality. It then processed by a pooling layer, and a subset of nodes are selected to produce a new coarsened adjacency matrix $\mathbf{H}_{\text{out}}$ , and feature matrix $\hat{\hat{\boldsymbol{A}}}_{\text{out}}$ (Fig. 5.1). After repeated layers (usually, three layers), the computed embeddings of each layer are aggregated and fed into a multi layers perception (MLP) to be transformed into class predictions. During the learning process, the model optimizes its parameters by minimizing the negative log-likelihood loss function $\ell(y, \hat{y})$.

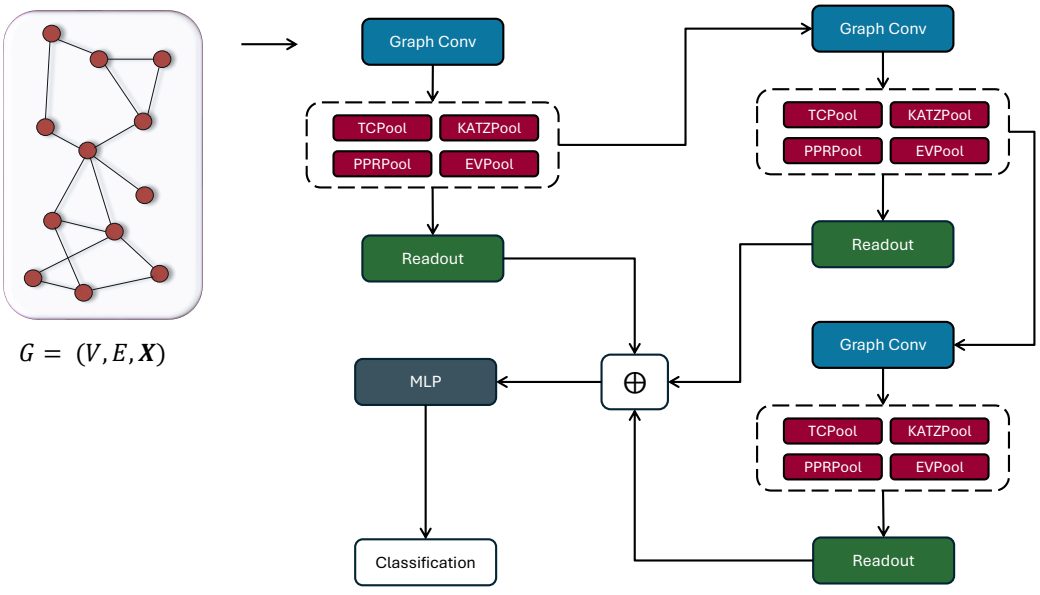



FIGURE 5.2: Hierarchical graph pooling. Following [1, 2], we employ a hierarchical classification pooling approach with three GCN layers, each followed by a pooling layer (TCPool, KATZPool, PPRPool, or EVPool). The input graph is processed by a GCN layer, followed by a pooling operation that reduces its size. The pooled graph is then passed to the next GCN layer, and after each pooling step, a readout layer aggregates node features into a fixed-size graph representation. After summation over all readout outputs, a Multi-Layer Perceptron (MLP) is applied for final graph classification.

In this work, we introduce CentralityPool which uses a set of different well-known centrality techniques [50, 185] as pooling layers to identify and select the most important nodes within a graph. These techniques evaluate various aspects of a node's position in the graph such as its connectivity, influence, or ability to facilitate communication, to assess node importance within a network, each offering distinct advantages. PageRank centrality [138] evaluates nodes based on the principle that connections to other important nodes enhance a node's importance. It focuses on the overall structure of the network, and considering both the quantity and quality of edges in a graph. Total Communicability Centrality [181] evaluates a node's score across the entire network by considering contributions from all possible paths connecting that node to others. This technique provides a comprehensive measure of a node's reachability and connectivity within complex networks, which makes it valuable to understand the importance of nodes in diverse network structures. KATZ Centrality [180] balances the importance of nodes by incorporating both their connections and their potential influence within the network. It provides a more refined understanding of a node's overall impact. Eigenvector Centrality [182], on the other hand, emphasizes on the importance of being connected to other influential nodes by providing a more localized view of importance based on the influence of a node's neighbors.

Although these techniques all share the common objective of assessing node importance within a network, they each approach this task from distinct perspectives, which profoundly impacts their applications and the insights they provide in network analysis.

### 5.4.1 Pooling Layer Architecture

The proposed approximation strategies can now be employed during pooling process to assign scores to nodes within a graph. Let $\mathbf{H}_{\text{in}}$ denote the output of the graph convolution layer (Fig. 5.2), which serves as the node embeddings input to the pooling layer. The approximation process begins by initializing the input matrix $\mathbf{H}$ through a two-layer neural network, formulated as follows:

$$\mathbf{Z}^0 = \mathbf{H} = F(\mathbf{H}_{\text{in}}, \mathcal{W}) \tag{5.11}$$

Here, $F$ represents a two-layer neural network function, while $\mathcal{W}$ denotes the associated weight matrix. The matrix $\mathbf{H}$ acts as the initial embedding matrix $\mathbf{Z}^0$, obtained by applying $F$ to the input embeddings $\mathbf{H}_{\text{in}}$.

Following this initialization step the iterative methods discussed are employed to score the node based on both the structural and feature information. Each of the centrality's approximations step can be expressed as follows:

Following this initialization step, iterative methods are applied. Each step in the centrality approximation process can be expressed as follows:

- TCPool:

$$\mathbf{Z}^{k+1} = \left( \sum_{k=0}^{K} \frac{\hat{\hat{\boldsymbol{A}}}^k}{k!} \right) \boldsymbol{H} \tag{5.12}$$

  The series always converges to the exact solution, given by $\mathbf{Z}^{(\infty)} = e^{\hat{\hat{\boldsymbol{A}}}} \boldsymbol{H}$, based on the Taylor series expansion. However, in practice, we iterate the process up to a predefined number of steps, denoted as $\mathbf{Z}^K$.

- PPRPool:

$$\mathbf{Z}^{k+1} = (1-\alpha)\hat{\hat{\boldsymbol{A}}}\mathbf{Z}^k + \alpha\mathbf{H} \tag{5.13}$$

  This iterative method converges to the exact solution $\mathbf{Z}^{(\infty)} = \boldsymbol{\Pi}_{ppr}\mathbf{H}$. Expanding the equation, we get:

$$\mathbf{Z}^k = \left( (1-\alpha)^k (\hat{\hat{\mathbf{A}}})^k + \alpha \sum_{i=0}^{k-1} (1-\alpha)^i (\hat{\hat{\mathbf{A}}})^i \right) \mathbf{H} \tag{5.14}$$

  As $k \to \infty$, the first term vanishes (since $\alpha \in (0, 1]$), and the second term forms a convergent geometric series.

- EVPool:

$$\mathbf{Z}^{k+1} = \frac{\hat{\hat{\boldsymbol{A}}}\mathbf{Z}^k}{\left\| \hat{\hat{\boldsymbol{A}}}\mathbf{Z}^k \right\|} \tag{5.15}$$

  By the Power method [184], the equation converges to the eigenvector of the largest eigenvalue.

- KATZPool:

$$\mathbf{Z}^{k+1} = \sum_{k=0}^{K} (\beta \hat{\hat{\boldsymbol{A}}})^k \boldsymbol{H} \tag{5.16}$$

  Note that the series always converge to the exact $\mathbf{Z}^{(\infty)} = (\boldsymbol{I}_n - \beta \hat{\hat{\boldsymbol{A}}})^{-1} \mathbf{1} \boldsymbol{H}$ based on the Neumann series expansion. However, in practice, we iterate the process up to a predefined number of steps, denoted as $\mathbf{Z}^K$.

Building on the previous steps, we use a predefined hyperparameter $\rho$ as the pooling ratio to control the number of nodes to preserve after pooling. Specifically, we select the top $\lceil \rho N \rceil$ nodes based on their values in $\mathbf{Z}^K$. This ensures that only the most important nodes and their connections, according to the learned embeddings, are preserved for subsequent processing.

$$\text{idx} = \text{top} - \text{rank}(\mathbf{Z}^K, \lceil \rho N \rceil) \tag{5.17}$$

$$\mathbf{Z}_{\text{mask}} = \mathbf{Z}^K_{\text{idx}} \tag{5.18}$$

Here, $\text{top} - \text{rank}(...)$ is a function that returns the indices of the top node values, $idx$ is an indexing operation and $\mathbf{Z}_{\text{mask}}$ is the selected nodes' embeddigs.

Based on $idx$, we can derive the coarsened adjacency matrix and the embedding matrix as follows:

$$\hat{\hat{\boldsymbol{A}}}_{\text{out}} = \hat{\hat{\boldsymbol{A}}}_{\text{idx, idx}} \tag{5.19}$$

$$\mathbf{H}_{\text{out}} = \mathbf{H}_{idx,:} \odot \mathbf{Z}_{\text{mask}} \tag{5.20}$$

where $\mathbf{H}_{\text{idx},:}$ is the row-wise indexed embedding matrix, $\odot$ is the broadcasted element-wise product, and $A_{\text{idx, idx}}$ is the row-wise and column-wise indexed adjacency matrix. $\mathbf{H}_{\text{out}}$ and $\hat{\hat{\boldsymbol{A}}}_{\text{out}}$ are the new embedding matrix and the corresponding adjacency matrix, respectively.

For clarity and ease of understanding, Algorithm 5 provides a detailed description of the process using the KATZPool method. The structure illustrated for KATZPool can be similarly adapted for other centrality-aware layers (TCPool, EVPool, and PPRPool).

**Algorithm 5:** KATZPool algorithm

**Input:** Adjacency matrix $\mathbf{A}$, Node input matrix $\mathbf{H}_{\text{in}}$, Pooling ratio $\rho$, Parameter $\beta$, Max iteration $K$ **(1)**

**Output:** $\mathbf{H}_{\text{out}}$ and $\hat{\tilde{\boldsymbol{A}}}_{\text{out}}$ **(2)**

Normalize adjacency matrix $\hat{\tilde{\boldsymbol{A}}} \leftarrow \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ **(3)**

Initialize the parameters $\mathcal{W}$ **(4)**

Initialize input using a 2-layer neural networks $\mathbf{H} \leftarrow F(\mathbf{H}_{\text{in}}, \mathcal{W})$ **(5)**

Set the initial matrix $\mathbf{Z}^0 \leftarrow \boldsymbol{H}$ **(6)**

Set the accumulation matrix $\mathbf{R} \leftarrow \mathbf{Z}^0$ **(7)**

Define matrix for power computation $\mathbf{M} \leftarrow \beta\hat{\tilde{\boldsymbol{A}}}$ **(8)**

**for** *1 to* $K$ **do** **(9)**

  $\mathbf{R} \leftarrow \mathbf{R} + \mathbf{Z}^k$ **(10)**

  $\mathbf{Z}^{k+1} \leftarrow \mathbf{M} \cdot \mathbf{Z}^k$ **(11)**

  $k \leftarrow k + 1$ **(12)**

**end** **(13)**

$\text{idx} \leftarrow \text{top} - \text{rank}(\mathbf{Z}^K, \lceil \rho N \rceil)$ **(14)**

$\mathbf{Z}_{\text{mask}} \leftarrow \mathbf{Z}^K_{\text{idx}}$ **(15)**

$\hat{\tilde{\boldsymbol{A}}}_{\text{out}} \leftarrow \hat{\tilde{\boldsymbol{A}}}_{\text{idx, idx}}$ **(16)**

$\mathbf{H}_{\text{out}} \leftarrow \mathbf{H}_{idx,:} \odot \mathbf{Z}_{\text{mask}}$ **(17)**

**Return** $\mathbf{H}_{\text{out}}$ , $\hat{\tilde{\boldsymbol{A}}}_{\text{out}}$ **(18)**

### 5.4.2 Training Complexity.

Here, we discuss the complexity of various centrality-aware pooling layers. Directly calculating the full matrices for TCPool, PPRPool, KATZPool, and EVPool is computationally inefficient and leads to a complexity of $\mathcal{O}(n^3)$. Instead, the corresponding explained approximations are employed to enhance the complexity. For PPRPool, approximations reduce its time complexity to $\mathcal{O}(n^2)$, and space complexity to $\mathcal{O}(n)$ as they avoid storing large matrices [129]. Likewise, KATZPool, TCPool, and EVPool, achieving similar efficiency. By localizing the centrality computation via only considering nearby nodes within a certain radius (i.e., short paths only) rather than the entire graph, linear computational complexity $\mathcal{O}(n)$ is also achievable. These approximations enable the practical application of centrality measures to large graphs by significantly reducing memory requirements.

TABLE 5.1: Statistics of data sets used for graph classification.

| Name | Number of graphs | Avg. # of Nodes per Graph | Avg. # of Edges per Graph | Number of Classes |
|---|---|---|---|---|
| DD | 1,178 | 284.3 | 715.66 | 2 |
| PROTEINS | 1,113 | 39.1 | 72.82 | 2 |
| NCI1 | 4,110 | 29.9 | 32.30 | 2 |
| NCI109 | 4,127 | 29.7 | 32.13 | 2 |
| Mutagenicity | 4337 | 30.32 | 30.77 | 2 |
| IMDB-B | 1000 | 19.77 | 96.53 | 2 |
| IMDB-M | 1500 | 13.00 | 65.94 | 3 |
| COLLAB | 5000 | 74.49 | 2457.78 | 3 |

## 5.5 Experiments

We evaluate the performance of our centrality-aware pooling layers by experiments on real-world datasets through multi-class graph classification and node classification.

### 5.5.1 Graph Classification

We evaluate two distinct architectures: hierarchical pooling and global pooling approaches. Figures 5.2 and 5.3 illustrate the hierarchical and global pooling architectures, respectively.

#### 5.5.1.1 Datasets

We conduct experiments on eight widely used graph-level benchmark datasets [186], including biological networks, molecular networks, and social networks:

- **DD (D&D)** and **PROTEINS** [114, 115] contain protein structure graphs where nodes represent amino acids and edges indicate spatial proximity. These datasets are used for protein structure classification.
- **NCI109** and **NCI1** [116] contain chemical compound graphs where nodes are atoms and edges are bonds. The task is to predict a compound's activity against cancer.
- **Mutagenicity** [117] contains Molecular graphs for predicting whether a compound is mutagenic, with nodes as atoms and edges as chemical bonds.
- **COLLAB** [118] contains graphs representing academic collaboration networks where nodes are researchers and edges indicate co-authorship. The task is to classify research fields.

- **IMDB-B** [118] contains Ego-networks of actors with nodes representing actors and edges representing co-appearances.
- **IMDB-M** [118] is similar to IMDB-B but classifies movies into one of three genres to reflect more complex social interactions.

The statistics of these datasets are summarized in Table 5.1.

#### 5.5.1.2 Competitors

We compare the performance of our pooling layers including TCPool, PPRPool, KATZPool, and EVPool with nine state-of-the-art pooling methods including SET2SET [178], SORTPOOL [97], DIFFPOOL [100], TOPK [106], SAGPOOL [2], ASAP [175], VIPool [187], CGIPool[188], and SSPool [1].

#### 5.5.1.3 Train/Validation/Test Split

Following previous works [1], [188], we randomly split the data into train/validation/test subsets, where 80% is used for training, 10% for validation, and the remaining 10% for testing. To ensure fairness, we use the hierarchical classification model of the 3-layer GCN each followed by a pooling and readout layers as illustrated in Fig. 5.2. For the global pooling structure, we adhere to the methodology described in [2], which involves applying three consecutive GCN layers followed by a global pooling layer. This architecture is depicted in Figure 5.3.

#### 5.5.1.4 Parameter Settings

Table 5.2 summarizes the hyperparameters for our framework. In addition to these parameters, we also conducted analyses on the maximum number of iterations $K$ (see Fig. 5.5), and the pooling ratio $\rho$ (global version, refer to Fig. 5.6). These analyses are detailed in the subsequent sections.

We utilize the PyTorch Geometric library [3] for our GNNs implementation[1]. To evaluate performance, we compare the models based on their accuracy on the test set. All models

[1]https://pytorch-geometric.readthedocs.io/en/latest/

TABLE 5.2: The grid search space for the hyperparameters. The pooling ratio 0.75 is used only for the hierarchical pooling architecture.

| Hyperparameter | Range |
|---|---|
| Learning rate | $1e-3, 1e-4, 1e-5$ |
| Hidden size | 128, 512, 1024 |
| Weight decay | $1e-2, 1e-3$ |
| Pooling ratio $\rho$ | 0.75 |

TABLE 5.3: Performance comparison of different pooling layers across various datasets. The best results are highlighted in bold and the second-best in underlined text.

| Method | Biochemical Domain | | Molecular Domain | | | Social Domain | | |
|---|---|---|---|---|---|---|---|---|
| | PROTEINS | DD | NCI1 | NCI109 | Mutagenicity | IMDB-B | IMDB-M | COLLAB |
| # Graphs | 1,113 | 1,178 | 4,110 | 4,127 | 4337 | 1000 | 1500 | 5000 |
| # Nodes (Avg.) | 39.1 | 284.3 | 29.9 | 29.7 | 30.32 | 19.77 | 13.00 | 7449 |
| # Edges (Avg.) | 72.82 | 715.66 | 32.30 | 32.13 | 30.77 | 96.53 | 65.94 | 2547.78 |
| # Classes | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 |
| SET2SET [178] | 71.46±2.17 | 71.94±0.56 | 74.82±0.85 | 74.12±1.31 | 77.69±0.55 | 70.98±0.78 | 48.64±0.46 | 77.62±0.38 |
| SORTPOOL [97] | 70.89±2.23 | 75.58±0.72 | 74.98±0.96 | 74.43±1.26 | 77.95±0.67 | 70.82±0.66 | 48.71±0.58 | 77.85±0.51 |
| DIFFPOOL [100] | 71.26±2.66 | 77.56±0.41 | 77.12±0.98 | 76.43±1.42 | 80.44±0.82 | 73.20±0.85 | 50.57±0.83 | 78.78±0.72 |
| TOPK [106] | 72.61±2.23 | 73.63±0.55 | 76.22±1.14 | 75.26±1.35 | 79.14±0.76 | 71.68±0.87 | 49.32±0.72 | 77.98±0.63 |
| SAGPOOL [2] | 73.16±2.31 | 74.72±0.82 | 76.84±1.21 | 75.98±1.47 | 79.18±0.82 | 72.20±0.91 | 49.92±0.69 | 78.56±0.77 |
| ASAP [175] | 73.86±2.52 | 76.58±1.04 | 77.49±1.35 | 76.33±1.58 | 80.12±0.89 | 72.74±0.93 | 50.28±0.61 | 78.95±0.69 |
| VIPool [187] | 73.65±2.66 | 76.89±1.69 | 77.49±1.81 | 76.73±1.62 | 80.19±1.02 | 73.37±1.41 | 50.49±0.96 | 78.87±1.23 |
| CGIPool[188] | 74.10±2.31 | 76.30±2.65 | 78.62±1.04 | 77.94±1.37 | 80.65±0.79 | 72.40±0.87 | 51.45±0.65 | 80.30±0.69 |
| SSPool [1] | **85.18±1.34** | 77.06±2.22 | 79.61±1.03 | 80.14±1.40 | 80.92±1.09 | 74.10±2.62 | 51.73±1.40 | 81.08±0.79 |
| **TCPool** | 78.92±0.67 | **80.50±0.98** | **79.81±0.41** | **81.69±0.93** | 81.47±1.20 | 74.80±0.40 | 52.53±0.27 | 80.80±0.18 |
| **EVPool** | 80.89±0.44 | 63.70±1.24 | 57.08±1.77 | 55.80±5.59 | 75.59±3.11 | 69.60±1.36 | 51.20±0.78 | 73.20±1.21 |
| **PPRPool** | 79.46±0.56 | 77.48±0.34 | 74.65±1.35 | 76.04±1.24 | 80.09±0.68 | 75.00±2.68 | **54.00±0.73** | **82.40±0.13** |
| **KATZPool** | 84.29±0.91 | 77.14±0.34 | 78.35±0.63 | 81.21±0.49 | **83.49±0.64** | **76.80±0.40** | 53.47±0.27 | 81.20±0.63 |

are trained and tested on an in-house GPU cluster. The experiment was executed on a single NVIDIA A100 GPU partition, with 70.96 GB of memory allocated for the task.

#### 5.5.1.5 Results

**Overall Performance Results** Table 5.3 shows the graph classification accuracy of our methods compared to the state-of-the-art methods. The table highlights the best-performing results in bold and the second-best in underlined text. The results indicate that our models consistently achieve the highest accuracy in most cases which highlights their superior performance. This demonstrates the effectiveness of centrality-aware pooling layers in enhancing accuracy. Based on the results, centrality-aware techniques have a significant impact on enhancing the quality of graph embeddings by effectively scoring and prioritizing nodes based on their importance within the graph. By focusing on the most important nodes, these techniques ensure that the final embeddings are capable of capturing the critical structural and feature information of the graphs.

Among the centrality-aware pooling layers, TCPool, PPRPool, and KATZPool consistently achieved strong and competitive performance. KATZPool demonstrates consistently strong results, especially in the biochemical (average: 80.72) and molecular (average: 81.02). TCPool also performs well, particularly in the molecular domain (average: 80.99), and PPRPool notably excels in the social domain (average: 70.47). In contrast, EVPool did not perform as well and showed a noticeable divergence in results. One reason for this is that EVPool's approximation mechanism does not accumulate previous scores at each step. In contrast, the accumulation property present in the approximations of the other layers—through the summation of prior scores—significantly enhances their accuracy in identifying important nodes. The results show the varying strengths of centrality measures when applied to graph pooling and in preserving key graph features.

#### 5.5.1.6 Global Pooling

Here, we conducted a global pooling version of our model. The architecture consists of a 3-layer GCN followed by a pooling (TCPool, KATZPool, PPRPool, or EVPool) layer. Following the pooling function, a readout layer is used, and its output is fed into an MLP classifier for graph classification. The architecture of global pooling is illustrated in 5.3. Additionally, Algorithm 6 provides a step-by-step graph classification task using the global pooling method.

**Algorithm 6:** Graph Classification algorithm using global pooling

**Input:** Graph $G = (V, E)$ with node features $\mathbf{X}$, adjacency matrix $\hat{\tilde{\boldsymbol{A}}}$ **(1)**
**Output:** Class label $\hat{y}$ **(2)**
$\mathbf{H} \leftarrow \mathbf{X}$ %Initialize node features **(3)**
$\mathbf{H}_1 \leftarrow \text{GCNLayer}(\mathbf{H}, \hat{\tilde{\boldsymbol{A}}})$ %Apply GCN Layer **(4)**
$\mathbf{H}_2 \leftarrow \text{GCNLayer}(\mathbf{H}_1, \hat{\tilde{\boldsymbol{A}}})$ %Apply GCN Layer **(5)**
$\mathbf{H}_3 \leftarrow \text{GCNLayer}(\mathbf{H}_2, \hat{\tilde{\boldsymbol{A}}})$ %Apply GCN Layer **(6)**
$\mathbf{H} \leftarrow \text{Concat}(\mathbf{H}_1 || \mathbf{H}_2 || \mathbf{H}_3)$ %Concatenate **(7)**
$\mathbf{H} \leftarrow \text{PoolingLayer}(\mathbf{H})$ %Apply Pooling Layer such as KATZPool (5) **(8)**
$\mathbf{H} \leftarrow \text{Readout}(\mathbf{H})$ %Global readout **(9)**
$\hat{y} \leftarrow \text{MLP}(\mathbf{H})$ %Classify using a fully connected layer **(10)**
**RETURN** $\hat{y}$ **(11)**

Figure 5.4 illustrates the accuracy of the models across both hierarchical and global classification structures. The results indicate that the global pooling architecture not

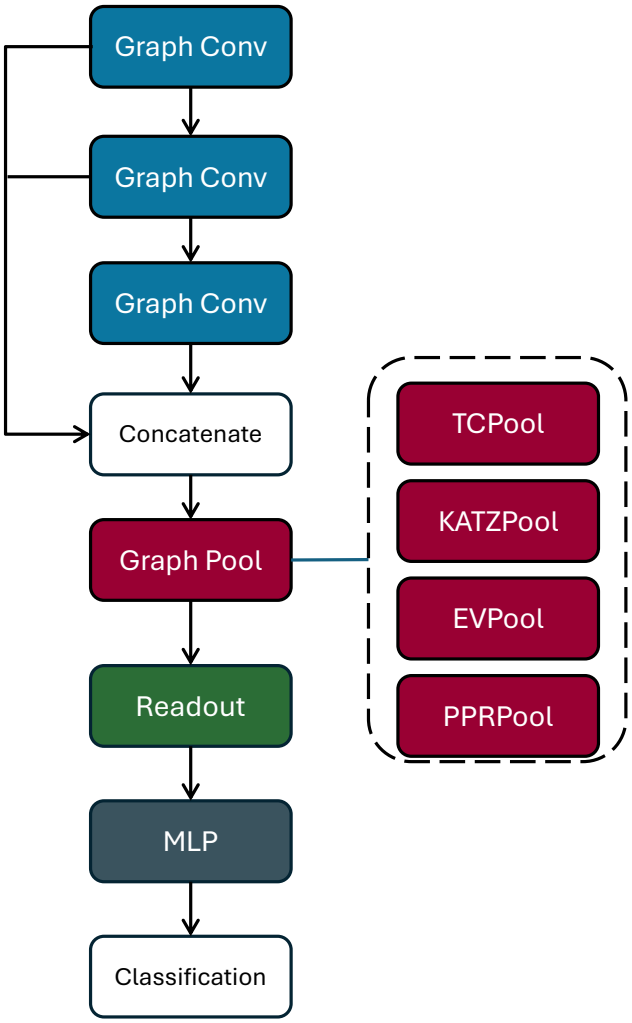



FIGURE 5.3: Global pooling structure. Following [2], we employ a global classification model featuring a 3-layer GCN followed by a one-step pooling approach. The pooling layer can be one of TCPool, KATZPool, PPRPool, or EVPool. After a 3-layer GCN, the outputs are concatenated to form a general representation. A pooling layer is applied and a readout layer is used to aggregate node features into a fixed-size representation. Ultimately, a Multi-Layer Perceptron (MLP) layer is used to classify the graphs.

only demonstrates competitive performance across all datasets but also outperforms the hierarchical structure in several instances. Moreover, the global pooling approach has lower computational complexity compared to the hierarchical method (Fig. 5.7).

**Impact of $K$** We investigate the impact of varying the number of iterative steps $K \in \{5, 10, 20, 40, 60, 80, 100\}$ within the approximation strategy. This analysis is conducted on two benchmark datasets: DD and Proteins. Fig. 5.5 illustrates the accuracies of

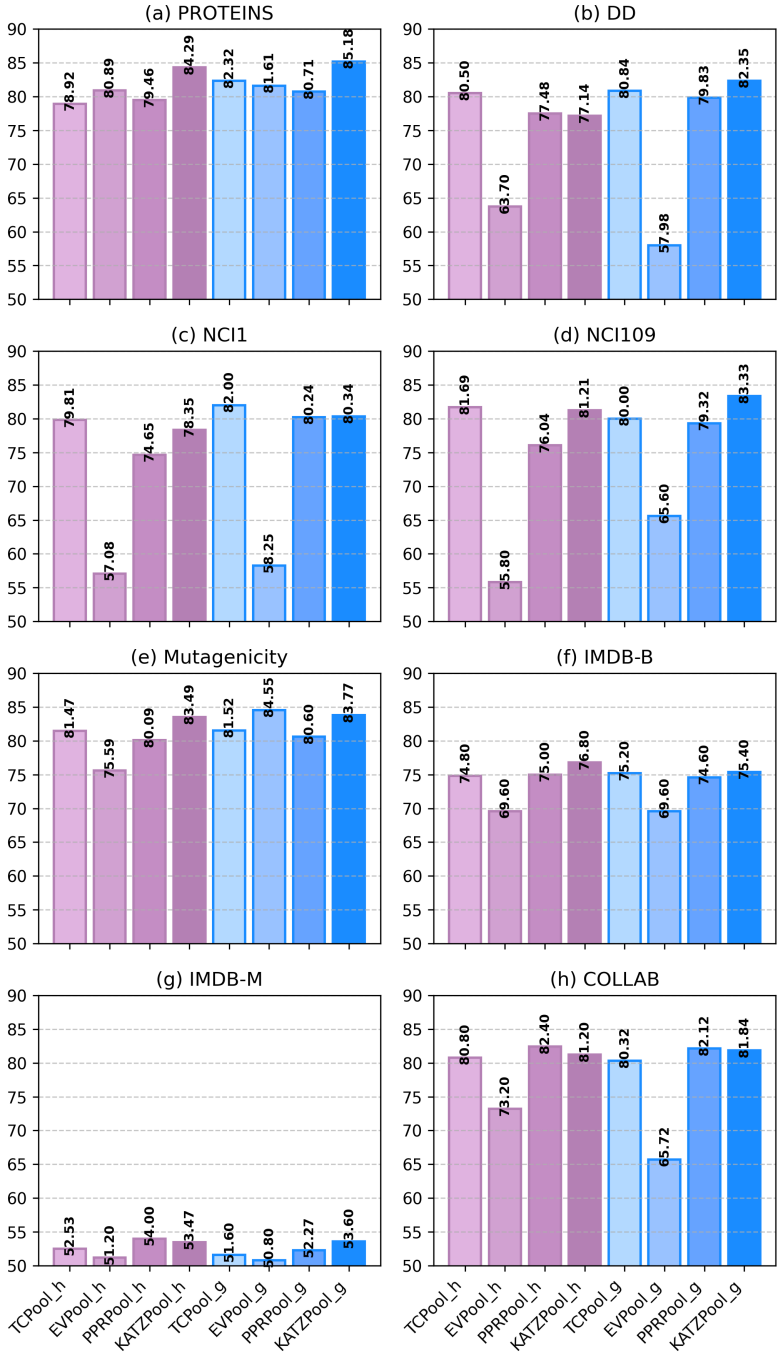



FIGURE 5.4: The result for the graph classification of all pooling layers over hierarchical and global structures. $\text{Pool}_g$ denotes the global pooling and $\text{Pool}_h$ denotes the hierarchical pooling approaches.

different pooling models. In general, KATZPool maintains high accuracy with fewer iterations which reflects its higher capacity in approximating node importance. TCPool and PPRPool show moderate improvement from higher $K$ values. EVPool exhibits stable performance and maintains an accuracy of around 80% across the PROTEINS dataset. However, compared to other layers, EVPool still underperforms on the DD

dataset.

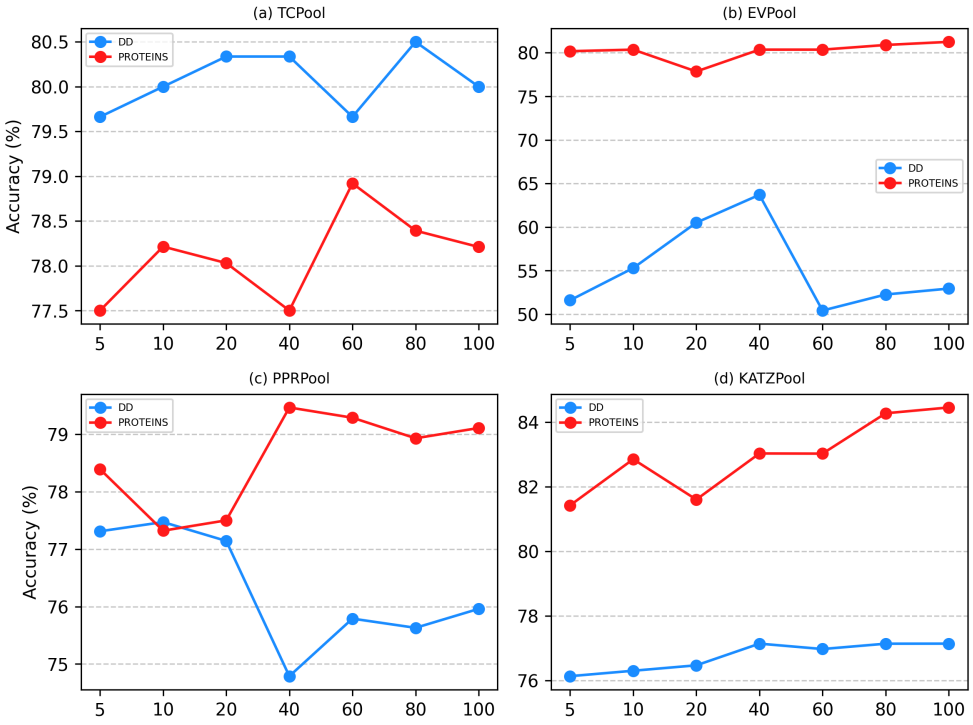



FIGURE 5.5: Impact of $K$ parameter in the layers approximation. The results are reported for the DD and PROTEINS datasets.

**Impact of Pooling Ratio** We further examine the impact of the pooling ratio $\rho \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ on model performance across different layers, using the global pooling architecture. Figure 5.6 presents the accuracy results for various models. Overall, the KATZPool outperforms the other layers, particularly at moderate pooling ratios, likely due to its ability to effectively capture node importance across multiple connections. PPRPool shows steady improvement as the pooling ratio increases which demonstrates its ability to boost accuracy. Similarly, TCPool's performance on the PROTEINS dataset improves as the pooling ratio increases, while EVPool significantly underperforms, likely due to its non-accumulative update mechanism which makes it less suitable.

### 5.5.2 Semi-supervised Node Classification

We observe that within the GCN equation, $\hat{\tilde{\boldsymbol{A}}}\mathbf{X}\mathbf{W}$, the adjacency matrix $\hat{\tilde{\boldsymbol{A}}}$ can be replaced with various matrices of its forms. For instance, Wu et al. [144] proposes

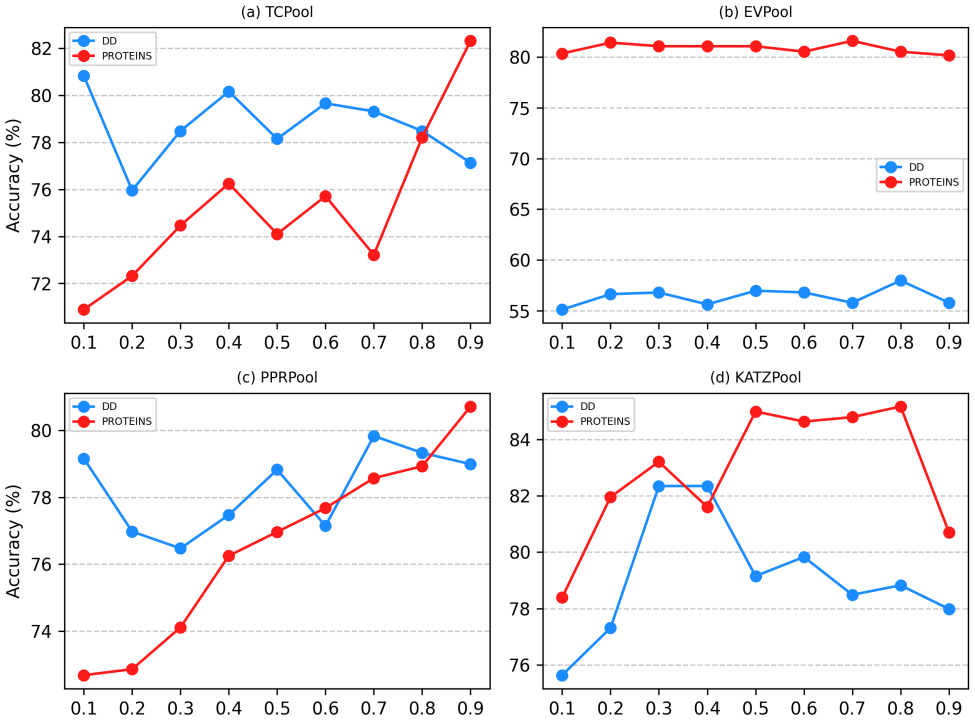



FIGURE 5.6: Impact of pooling ratio, $\rho$. The results are reported for the DD and PROTEINS datasets.

to replace the adjacency matrix with its higher power to capture information from long-distance nodes. By raising the adjacency matrix to higher powers, their method enables the model to aggregate information not only from connected neighbors but also from nodes that are far away in the graph. This substitution indicates that the developed centrality-aware layers are applicable to the convolution layers. Note that, the Personalized PageRank algorithm has been employed as a convolution layer in recent studies [129, 189]. Thus, We do not repeat the comparison with it again due to its substantial similarity.

For node-level tasks, we utilize three well-established citation:

- **Cora** is a real-world bibliographic dataset featuring academic papers and their citation relationships within the field of computer science [107]. The dataset includes documents classified into seven distinct categories.
- **CiteSeer** is a citation network consisting of computer science papers, which are organized into six distinct categories [107].

- **PubMed** is a citation graph focusing on research papers [107], with the papers categorized into three specific classes.

The statistics of these datasets are summarized in Table 5.4.

TABLE 5.4: Statistics of data sets used for node classification.

| Name | Number of Vertices | Number of Edges | Number of Classes | Number of Features |
|---|---|---|---|---|
| Cora | 2,708 | 10,556 | 7 | 1,433 |
| CiteSeer | 3,327 | 9,104 | 6 | 3,703 |
| Pubmed | 19,717 | 88,648 | 3 | 500 |
| OGB-arxiv | 169,343 | 1,166,243 | 40 | 128 |
| OGB-products | 2,449,029 | 61,859,140 | 47 | 100 |
| OGB-paper100M | 111,059,956 | 1,615,685,87 | 172 | 128 |

Since different splits of the data (train/validation/test) may lead to a substantially different ranking of models [112], for a fair comparison, we evaluate the models over different splits as follows:

- Fix Split. For all datasets, we randomly split 20 nodes per class for training, 500 nodes for validation, and 1,000 nodes for testing.
- Semi-supervised Split. We randomly split the nodes into 48%, 32%, and 20% for training, validation, and testing.

For node classification, we compare the performance of TCConv, KATZConv, and EV-Conv with state-of-the-art convolution layers, including MLP, GCN [25], ChebNet [72], APPNP [129], ARMA [190], GPRGNN [189], BernNet [191], ChebNetII [73], ANS-GT [146], GATv2 [192], DET [193], NAGphormer [194], Gapformer [195], LRGNN [196] and PCNet [147]. We train the models using Adam optimizer [148] with a learning rate $lr$ selected from $\{1e-2, 1e-3, 1e-4, 1e-5\}$ and $L_2 \in \{1e-3, 1e-4\}$. The loss function is the negative log-likelihood loss. The embedding dimensionality $d$ is selected from $\{64, 128, 256\}$; maximum number of iterations $K$ from $\{10, 20\}$; maximum number of epochs from $\{80, 150, 400\}$; and training is terminated if the validation accuracy does not improve for $\{10, 20\}$ consecutive epochs. A fixed dropout rate of 0.5 and the ReLU activation function is applied between the layers.

Table 5.5 presents the node classification accuracy of convolution layers over all datasets within different data splits. It shows that TCPool and KATZConv perform on par with the existing models in most cases, and outperform the other models on the CiteSeer

dataset. TCPool and KATZConv are able to produce high-quality embeddings that lead to superior performance across a range of datasets. However, similar to graph classification, EVConv's performance is below the others due to its non-accumulative update mechanism.

TABLE 5.5: Node classification accuracy of convolution layers on different datasets. The best results are highlighted in bold and the second-best in underlined text.

| Model\Data | Split | Cora | CiteSeer | PubMed |
|---|---|---|---|---|
| # Nodes | | 2708 | 3327 | 19717 |
| # Features | | 1,433 | 3,703 | 500 |
| MLP [25] | Fixed | 57.17±1.34 | 56.75±1.55 | 70.52±0.27 |
| GCN [25] | | 79.19±1.37 | 69.71±1.32 | 78.81±0.24 |
| ChebNet [72] | | 78.08±0.86 | 67.87±1.49 | 73.96±0.31 |
| ARMA [190] | | 79.14±1.07 | 69.35±1.44 | 78.31±0.22 |
| APPNP [129] | | 82.39±0.68 | 69.79±0.92 | 79.97±0.28 |
| GPRGNN [189] | | 82.37±0.91 | 69.22±1.27 | 79.28±0.33 |
| BernNet [191] | | 82.17±0.86 | 69.44±0.97 | 79.48±0.41 |
| ChebNetII [73] | | 82.42±0.64 | 69.89±1.21 | 79.51±1.03 |
| PCNet [147] | | **82.81±0.50** | 69.92±0.70 | **80.01±0.88** |
| **TCConv** | | 80.78±0.57 | 70.70±0.33 | 78.70±0.22 |
| **KATZConv** | | 82.51±0.64 | **71.18±0.17** | 78.85±0.31 |
| **EVConv** | | 76.19±0.72 | 68.33±1.48 | 75.81±1.76 |
| ANS-GT [146] | Semi-Supervised | 86.71±1.45 | 74.57±1.51 | 89.76±0.46 |
| GATv2 [192] | | 87.25±0.89 | 75.72±1.30 | 85.75±0.55 |
| DET [193] | | 86.30±1.41 | 75.37±1.41 | 86.28±0.44 |
| NAGphormer [194] | | 85.77±1.35 | 73.69±1.48 | 87.87±0.33 |
| Gapformer [195] | | 87.37±0.76 | 76.21±1.47 | 88.98±0.46 |
| LRGNN [196] | | 88.33±0.89 | 77.53±1.31 | **90.24±0.64** |
| PCNet [147] | | 88.41±0.66 | 77.50±1.06 | 89.51±0.28 |
| **TCConv** | | **88.77±1.14** | **77.58±0.69** | 88.40±0.35 |
| **KATZConv** | | 88.28±1.15 | 76.48±1.80 | 88.24±0.31 |
| **EVConv** | | 83.35±1.14 | 74.04±1.31 | 82.29±0.54 |

TABLE 5.6: Node classification accuracy of all convolution layers on large OGB datasets.

| **Model** | **OGB-arxiv** | **OGB-products** | **OGB-papers100m** |
|---|---|---|---|
| PPRConv | 68.07±0.52 | 75.41±0.08 | 59.24±0.50 |
| KATZConv | 68.55±0.13 | 74.29±1.10 | 54.35±0.70 |
| TCConv | 65.37±0.32 | 67.34±0.12 | 51.40±0.24 |
| EVConv | 62.12±0.85 | 63.24±1.12 | 35.16±0.23 |

**Performance on Large Dataset OGB** Table 5.6 shows the result of node classification accuracy for different convolution layers on the large datasets OGB-Arxiv (169,343 nodes, 1,166,243 edges), OGB-products (2,449,029 nodes, 61,859,140 edges) and OGB-papers100m (111,059,956 nodes and 1,615,685,87 edges) [113]. PPRConv layer achieved

the highest accuracy on OGB-Products and OGB-papers100m, while KATZConv outperformed APPNP on OGB-arxiv. TCConv showed moderate performance, while EVConv had the lowest accuracy, likely due to its approximation mechanism, which does not accumulate scores from previous iterations at each step.

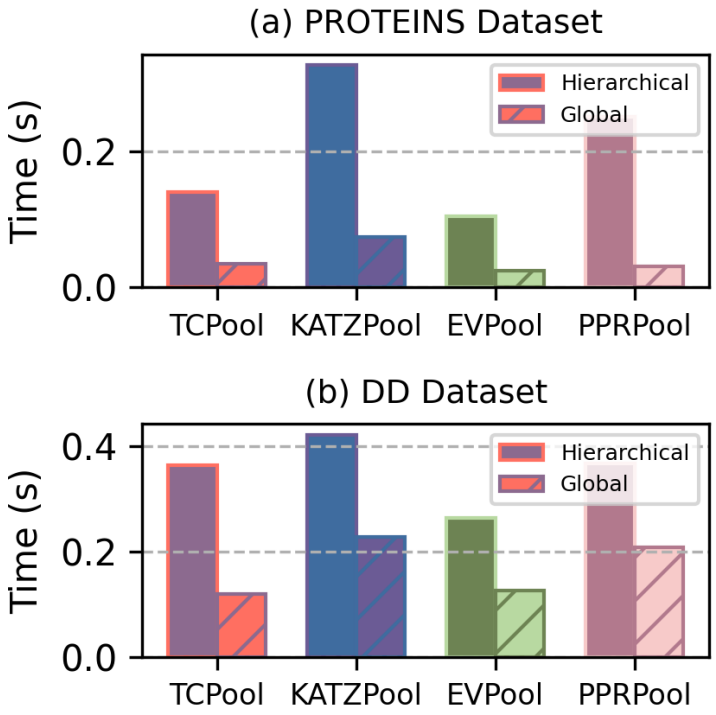



FIGURE 5.7: The time consumption of different pooling layers per epoch for the hierarchical and global approaches in the graph classification task.

**Time and Memory Usage** Fig. 5.7 illustrates the computational time per epoch for different pooling layers on the PROTEINS and DD datasets, using both hierarchical and global pooling approaches. The results highlight differences in computational efficiency among the pooling techniques. The global pooling approach generally require less time. When using the hierarchical approach, the KATZPool layer intrigued the highest computational cost.

Additionally, we analyzed the average time and memory usage of the convolutional layers in the node classification scenario across multiple datasets, including one of the large-scale datasets in the literature, OGB-papers100M [113]. Table 5.7 presents the results. The GAT model [27] has higher computational costs, both in terms of running time and memory consumption, compared to other approaches. Even on smaller datasets

TABLE 5.7: Average time and memory usage in Seconds/MB per epoch for the node classification task.

| Model/Data | Cora | CiteSeer | PubMed | OGB-papers100m |
|---|---|---|---|---|
| GAT [27] | 0.0039/151.68 | 0.0049/191.10 | 0.0063/965.48 | 279.43/12376.83 |
| PPRConv | 0.0032/50.85 | 0.0033/125.00 | 0.0033/111.74 | 259.83/1334.81 |
| KATZConv | 0.0042/50.84 | 0.0043/124.97 | 0.0049/111.84 | 292.13/1543.14 |
| TCConv | 0.0029/50.85 | 0.0029/125.00 | 0.0029/111.57 | 234.03/1324.22 |
| EVConv | 0.0042/50.85 | 0.0043/125.00 | 0.0043/113.77 | 243.33/2257.81 |

such as Cora, CiteSeer, and PubMed, GAT remains the most memory-intensive model, which makes it less practical for large-scale applications. In contrast, the proposed layers show notable efficiency, requiring substantially less memory while achieving faster training times in most cases. This efficiency highlights their scalability for handling larger datasets, where computational efficiency is a crucial factor.

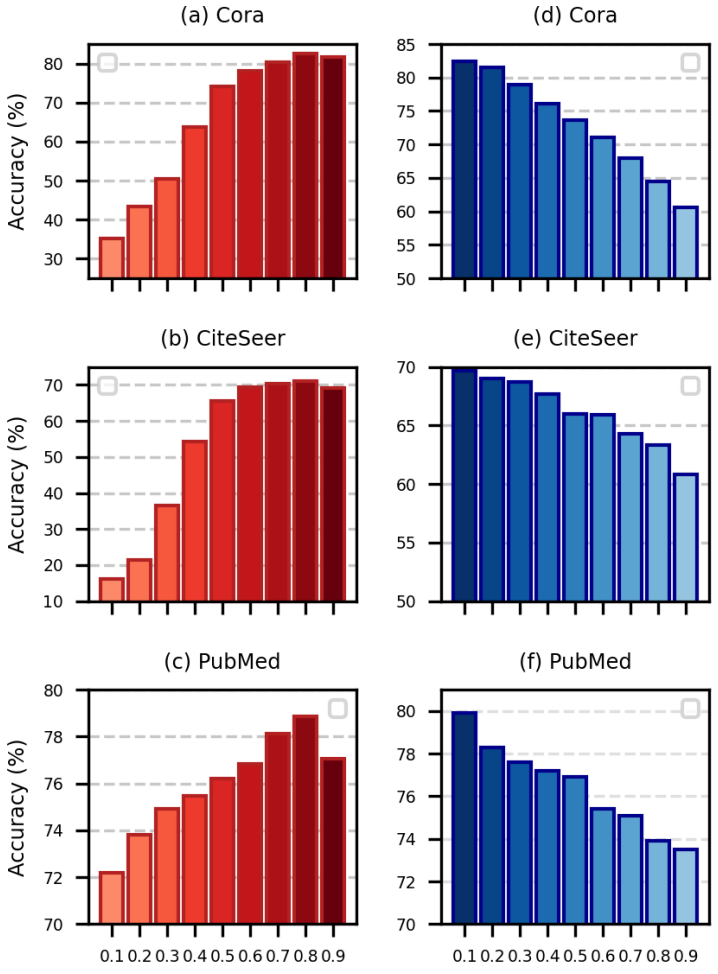



FIGURE 5.8: Impact of $\beta$ and $\alpha$ parameters in KATZConv (red) and PPRConv (blue) layers on the node classification accuracy of Cora, Citeseer and PubMed datasets.

**Impact of Teleport Parameters** To show the impact of the teleport parameters, $\beta$ and $\alpha$, on the performance of the KATZConv and PPRConv layers, Fig. 5.8 presents model accuracy results as $\alpha$ varies from 0.1 to 0.9. The results indicate that larger values of $\beta$ benefit the KATZConv layer (red), whereas lower values enhance the PPRConv layer (blue), leading to more effective embeddings. This improvement can be attributed to the increased involvement of adjacency matrices with different powers in the approximation strategy. However, a decline in accuracy at $\alpha = 0.9$ suggests that equally weighting different paths (i.e., powers of the adjacency matrix) may not be optimal.

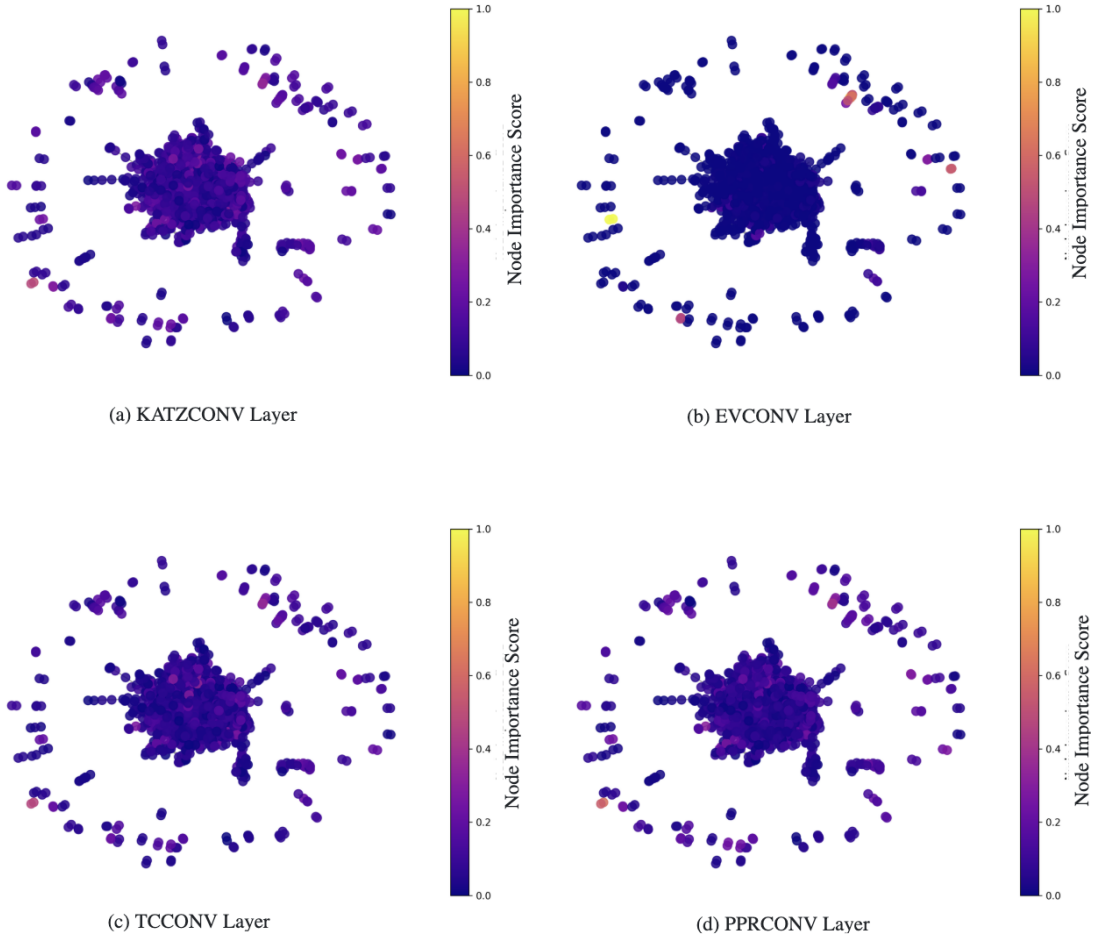
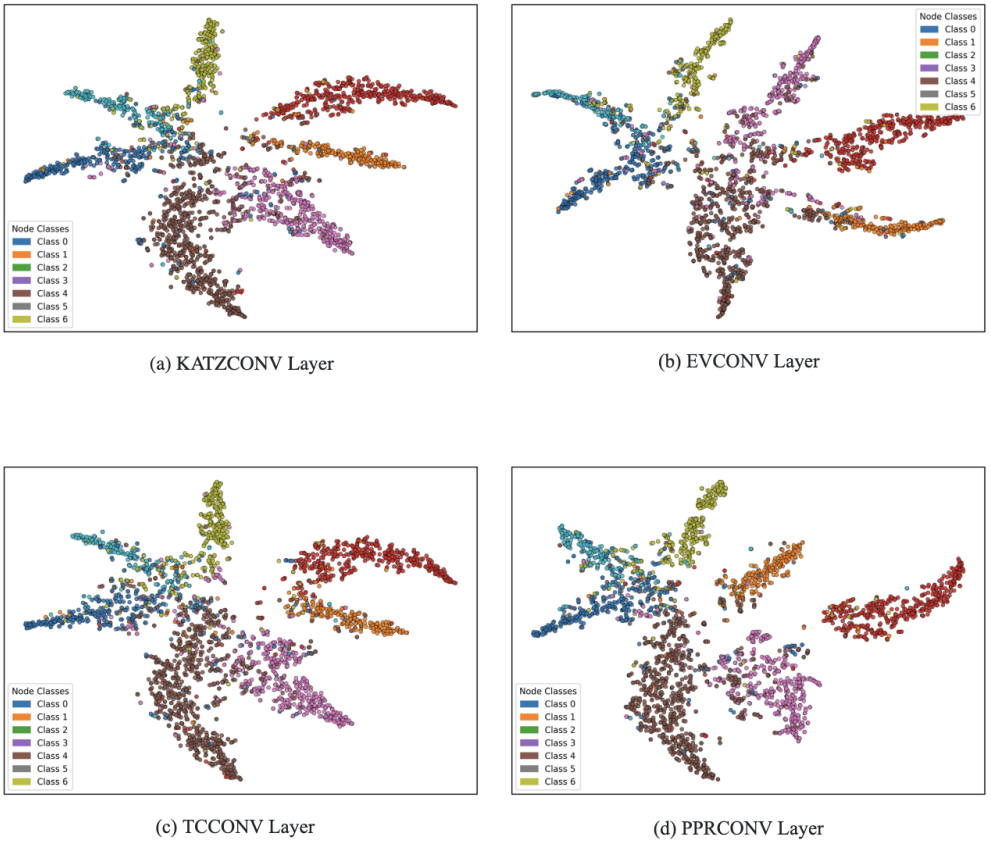



FIGURE 5.9: The t-SNE derived from different learned embeddings on the Cora dataset.

## 5.6 Discussion and Future Work

Figures 5.9 and 5.10 illustrates node selection across convolution layers over the Cora dataset. PPRConv, KATZConv, and TCConv select similar nodes, which reflect their comparable performance, while EVConv selects different nodes, likely explaining its




(a) KATZCONV Layer

(b) EVCONV Layer

(c) TCCONV Layer

(d) PPRCONV Layer

FIGURE 5.10: The graph visualization with node importance scores derived from different learned embeddings on the Cora dataset. Darker colors indicate lower importance.

lower effectiveness due to the lack of an accumulative mechanism. This limitation causes local node features to fade over iterations which leads to suboptimal performance, as global information is crucial in the node and graph classification tasks. To address this, future work could introduce a residual or personalization mechanism, such as skip connections or teleportation terms, into the EVPool (e.g., EVConv) approximation approach. This approach offers a theoretical way to address the limitations of traditional eigenvector propagation and would preserve EV's global structure without sacrificing scalability and enhance its effectiveness in graph representation learning. Another future work, since we are focusing on individual analysis of centrality techniques, could be the development of a learnable weighting mechanism for their dynamic integration and the optimization of the propagation depth.

## 5.7 Conclusion

In this chapter, we introduced CentralityPool, a novel graph pooling framework that exploits the power of centrality techniques as pooling layers to enhance graph-level classification tasks. CentralityPool is a set of layers built upon four well-known centrality techniques: Personalized PageRank, Total Communicability, KATZ centrality, and Eigenvector centrality. Each of these techniques offers a unique approach to identifying the most important nodes in a graph to ensure that the critical structural information is preserved during the pooling process. In addition to the hierarchical-based pooling methods, we explored the global pooling architecture and provided an analysis of their performance across different datasets. By extensive experiments on real-world datasets, we show that our models outperform state-of-the-art pooling layers in terms of graph classification. Furthermore, we extended the application of centrality techniques by incorporating them as convolution layers in node classification tasks. The results on node classification showed competitive accuracies against state-of-the-art node-level models. Overall, centrality techniques offer a powerful method that can be applied to a wide range of graph-based tasks, from classification to node-level analysis.

# Chapter 6

# Conclusion and Future Directions

## 6.1 Conclusions

In conclusion, node influence analysis plays a crucial role in various applications of GNNs, particularly in tasks such as node classification and graph classification. Node classification heavily relies on the process of message passing, where node embeddings are updated through the propagation of information across the graph. This process is inherently tied to the analysis of node influence, as the impact of neighboring nodes on a target node's embedding determines the accuracy of classification results. Similarly, graph classification depends on effective graph pooling methods, where the selection and aggregation of key nodes during the hierarchical pooling process significantly impact the preservation of critical structural information.

In this thesis, we have introduced three innovative frameworks that push the boundaries of GCN in addressing these challenges. The first framework, NACFormer, and its advanced version, NACFormer$_{MS}$, enhance the aggregation of both local and distant node information through multi-head attention and adjacency matrix-based mechanisms. This approach allows for more accurate node representations by balancing the influence of both nearby and distant nodes. Secondly, FadiGNN integrates both node features and graph topology to evaluate node importance in an unsupervised manner by using the personalized PageRank algorithm to enhance node ranking. Finally, we introduced CentralityPool, a graph pooling framework that uses centrality techniques as pooling layers, preserving essential structural information during the graph-level classification process.

Our key contributions are reiterated as follows.

### 6.1.1 Multi-Scale Node Neighborhood Aggregation Via Graph Coarsening and Transformer

In Chapter 3, we presented a multi-scale node neighborhood aggregation to incorporating distant node information, NACFormer. This model enhances GCN by incorporating a multi-head attention mechanism from Transformer by allowing it to intelligently aggregate information from both local and distant nodes. This approach mitigates the oversmoothing problem and ensures that node features remain distinctive even after multiple propagation steps. Unlike GCNs with fixed neighborhood aggregation due to

the limitations, NACFormer finds most important and even distant nodes during message passing and improves information flow across the graph.

Additionally, NACFormer utilizes graph coarsening to reduce the complexity of graphs while preserving essential structures. This approach make the computations more efficient for long-range information propagation, and improves generalization across diverse graph structures. The advanced version of NACFormer, NACFormer$_{ms}$, train on a set of coarsened graphs derived from the original graph at varying reduction ratios. This multi-scale training approach enhances the model's generalization ability and leads to a more enhanced embeddings and improved accuracy in graph-based tasks.

Through extensive experiments across the node classification task, NACFormer$_{ms}$ consistently outperforms state-of-the-art models by effectively learning features from both local and distant nodes across nine different datasets, including large-scale graphs. To ensure a fair comparison, we used two different data splits and benchmarked NACFormer$_{ms}$ against both transformer-based and non-transformer-based models from the existing literature. In both data splits, NACFormer$_{ms}$ achieved superior performance, demonstrating a clear advantage in terms of accuracy and its ability to generalize across diverse graph structures.

### 6.1.2 Feature-Aware Unsupervised Detection of Important Nodes in Graphs

In Chapter 4, we addressed the challenge of identifying important nodes within a graph by proposing FadiGNN, a feature-aware Personalized PageRank model. This hybrid approach integrates GCN with centrality technique, particularly Personalized PageRank, to enhance node scoring by using both node features and graph structures.

key advantage of FadiGNN is its ability to perform node ranking in an unsupervised manner by overcoming the common challenge of limited labeled datasets. By employing a well-designed loss function, the model ensures stable convergence while effectively learning node importance without relying on manual annotations. This unsupervised learning approach makes FadiGNN highly adaptable across different graph structures, which allows it to generalize well across various datasets and applications. Furthermore,

by integrating both feature-based and topology-based information, the model refines node importance scores beyond what traditional centrality measures can achieve.

To further enhance computational efficiency, FadiGNN incorporates approximation strategies for centrality measures to enables it to scale effectively to large graphs without compromising accuracy. These approximations preserve the effectiveness of centrality measures while significantly reducing the computational burden and make the model suitable for real-world applications where scalability is critical. Unlike traditional centrality-based methods, which rely solely on structural properties, FadiGNN's learning-based approach ensures greater flexibility, robustness, and adaptability across diverse datasets and application domains.

Through extensive experiments across two distinct application scenarios, FadiGNN consistently outperforms traditional and state-of-the-art models in both node classification and active learning on seven different datasets including large graphs. By effectively learning both node features and graph structures, it achieves superior accuracy in ranking influential nodes. For node classification, the original graph is compressed into a smaller version at different ratios, where a distinct model is trained on. The model is then evaluated on the original graph to assess how effectively the nodes are ranked and selected. For active learning, varying labeling budgets are used during training to analyze the model's performance under different supervision levels.

The results highlight FadiGNN's adaptability to accurately ranking key nodes. This demonstrates its strong potential for real-world applications where identifying influential nodes is crucial.

### 6.1.3 Centrality-aware Hierarchical Graph Pooling

In Chapter 5, we presented centrality-aware hierarchical graph pooling, CnetralityPool, to investigate the centrality measures in the graph pooling process.

CentralityPool introduces a novel approach to graph pooling by employing four widely recognized centrality techniques including Personalized PageRank, Total Communicability, Katz Centrality, and Eigenvector Centrality—individually to guide node selection. The model aims to find the most important nodes and subgraphs and contribute them to the final representation to improve classification accuracy. CentralityPool extends its

flexibility by offering results for both hierarchical and global pooling mechanisms. Hierarchical pooling focuses on capturing local node relationships at multiple scales, while global pooling captures overall graph-level information.

In addition to pooling, another huge contribution of this work is the incorporation of these centrality measures directly into convolution layers (since the matrix representaion of each of these techniques are in the form of and adjacency matrix) to enhance node classification by integrating structural awareness into feature aggregation.

Additionally, since the centrality matrices can be computationally expensive to calculate, approximation strategies are employed to reduce both computational time and memory usage. These strategies significantly improve efficiency and enables the model to handle large-scale datasets with billions of nodes and edges without sacrificing performance. By optimizing the computational process, these approximations make it feasible to apply CentralityPool to complex, real-world graph data while maintaining high accuracy and scalability.

To evaluate the models, we conducted extensive experiments across two application scenarios: graph classification and node classification tasks. For the graph classification task, the developed pooling layers—PPRPool, TCPool, KATZPool, and EVPool—were tested on eight graph datasets. The results demonstrated the superiority of these pooling methods, particularly PPRPool, TCPool, and KATZPool, in effectively capturing key structural information during the pooling process.

For the node classification task, the developed convolution layers—PPRConv, TCConv, KATZConv, and EVConv—were applied to six datasets, including three large-scale datasets, such as OGBN-paper100M, which contains over one hundred million nodes and one billion edges. The results across two different data splits consistently showed superior performance compared to state-of-the-art models. Furthermore, the scalability of these models on large-scale datasets highlights their practicality for real-world graph applications by balancing efficiency and accuracy.

## 6.2 Application and Future Work

### 6.2.1 Application

Node influence analysis has broad applicability across various domains where data can be structured as graphs. This section outlines three primary applications including transportation networks, social networks, and biological systems.

#### 6.2.1.1 Transportation Networks

One of the primary applications of node influence analysis is in transportation networks. Transportation networks, such as road and traffic systems, can be naturally represented as graphs, where nodes correspond to critical locations (e.g., intersections, road segments, or transit hubs), and edges define connectivity between them. In such networks, examining the influence of the nodes is essential to identify key roads and intersections that significantly impact overall traffic flow, congestion levels, and network resilience [197]. For instance, in a city's road network, certain highways or major intersections serve as critical nodes that facilitate the majority of traffic flow. If a high-impact intersection experiences a failure, such as an accident or road closure, it can trigger congestion that spreads throughout the network, which leads to delays across multiple regions. Once the influential roads or intersections are identified, traffic authorities can implement appropriate strategies to mitigate cascading failures and maintain network efficiency.

#### 6.2.1.2 Social Networks

In social networks, nodes typically represent individuals or organizations, while edges represent relationships, communications, or interactions. Analysing node influence is crucial for understanding how information, behavior, or influence propagates through the network. This has practical applications in viral marketing, where companies aim to identify and target key influencers who can maximize product adoption through word-of-mouth [198]. Similarly, in misinformation detection, influential nodes can serve as early indicators or blockers of false content spread [199].

#### 6.2.1.3 Biological Networks

Biological networks, such as protein–protein interaction (PPI) networks, gene regulatory networks, and metabolic pathways, inherently rely on graph-based representations. In these networks, nodes represent biological entities (e.g., proteins, genes), and edges represent interactions or regulatory relationships. Identifying influential nodes can reveal essential proteins or genes that control critical biological processes [200]. This is particularly useful in drug discovery, where targeting highly influential nodes may yield more effective therapeutic outcomes. In disease pathway analysis, detecting key regulatory genes can improve the understanding of disease mechanisms, and allow for earlier diagnosis and more personalized treatment strategies.

In conclusion, this thesis has laid a solid foundation for advanced node ranking, graph pooling and graph filtering, and demonstrated the potential of integrating GNNs with centrality techniques.

### 6.2.2 Future Direction

Each of the proposed models, NACFormer, FadiGNN and CentralityPool, has demonstrated superior performance across a diverse range of real-world datasets, by achieving state-of-the-art results in different application scenarios including node classification and graph classification tasks.

However, despite their empirical success, several limitations and open challenges remain. These limitations not only highlight the boundaries of current methodologies but also pave the way for promising future research directions. In what follows, we provide a detailed discussion of these limitations, along with potential avenues for addressing them in future work.

- **NACFormer: Multi-Scale Node Neighborhood Aggregation Via Graph Coarsening and Transformer**: Since the model integrates graph coarsening to refine the input graph and construct a condensed representation, its performance becomes inherently tied to the effectiveness of the underlying coarsening strategy. While coarsening reduces the computational training time due to the smaller graph and helps the model focus on more meaningful structural patterns by reducing the

influence of noisy or insignificant details in the original graph, it also introduces a critical dependency that can hinder the model's ability to generalize—particularly in scenarios involving highly irregular, sparse, or noisy graphs. In such cases, an inappropriate coarsening strategy may result in the loss of critical structural information or the distortion of essential topological patterns, ultimately undermining the model's ability to accurately represent the original graph.

The choice of the coarsening algorithm can significantly influence how well the hierarchical structure of the graph is preserved, which in turn affects the tasks such as node classification. As a result, there is a need for more flexible and data-aware coarsening strategies that can adapt to the intrinsic properties of the graph.

To address this limitation, future work could focus on designing adaptive coarsening methods. This could include learning-based approaches where the coarsening mechanism is integrated into the end-to-end training pipeline and allows the model to jointly optimize structural reduction and task performance. Such advancements have the potential to enhance the applicability across a broader range of graph domains.

- **FadiGNN: Feature-Aware Unsupervised Detection of Important Nodes in Graphs**: A key limitation of FadiGNN, in comparison to traditional methods, is the increased complexity associated with its training process. The increased complexity arises from the model's dependence on sophisticated graph-based learning techniques, which, although effective, require substantial computational power and time. Consequently, this can create significant challenges, especially when handling large datasets or networks with a high volume of nodes and edges. The training time for the model can grow nonlinearly as the size of the graph increases, which limits its applicability in real-time applications or scenarios involving massive datasets.

  To address this issue, future research could focus on reducing training complexity by investigating faster approximation techniques. For example, scalable GNNs architectures, such as those that employ graph sampling or attention mechanisms, could be explored to optimize computational efficiency while preserving the model's ability to learn meaningful representations.

Additionally, extending FadiGNN's applicability to other types of graphs, such as knowledge graphs or temporal graphs, presents an exciting opportunity. Knowledge graphs, with their rich semantic structure and complex relationships between entities, could benefit from FadiGNN's ability to capture intricate node dependencies. This extension could open up new application areas, such as recommendation systems. Exploring these directions not only broadens the scope of FadiGNN but also presents an opportunity to refine its capabilities and further enhance its performance across a variety of domains.

Furthermore, FadiGNN is built upon Personalized PageRank centrality technique, which plays a crucial role in capturing the importance of nodes based on personalized diffusion patterns. While PPR centrality has proven to be highly effective in identifying influential nodes in a variety of graph-based tasks, it is only one of many centrality measures available in the literature. A promising direction for future research could involve adapting alternative centrality techniques, such as Betweenness centrality, Closeness centrality, or KATZ centrality, into the FadiGNN framework to enhance flexibility across a broader range of graphs.

- **CentralityPool: Centrality-aware Hierarchical Graph Poolings**: A key limitation of this work is the absence of an accumulative mechanism within the EVPool (e.g., EVConv) layer. Without such a mechanism, the influence of local node features diminishes progressively over multiple iterations, which leads to a loss of valuable information. This is particularly problematic since global information plays a critical role in tasks such as node and graph classification, where the ability to capture both local and global structural patterns is essential for accurate predictions. As the model lacks a mechanism to preserve this context, the performance of the EVPool layer becomes suboptimal, particularly for graphs with complex, hierarchical structures or long-range dependencies.

  To address this issue, future research could explore the introduction of a residual or personalization mechanism into the EVPool (e.g., EVConv) approximation approach. For instance, incorporating skip connections, which allow information to flow directly across layers, or teleportation terms, which reintroduce initial information at each iteration, could help mitigate the loss of important node features over time. These mechanisms are able to preserve the global structure encoded in the eigenvectors approximation algorithm and the model can efficiently handle

large graphs without sacrificing performance. This adaptation would allow the model to more effectively balance local and global information, thereby enhancing its overall predictive accuracy.

Additionally, a promising direction for future work—since our analysis has primarily focused on individual centrality techniques—could be the development of a learnable weighting framework. Such a framework would enable the dynamic integration and optimization of the propagation depth for each of the four centrality layers introduced. By allowing the model to learn how best to combine the contributions of different centrality measures at varying depths, this approach could lead to improved task performance across a broader range of graph-based tasks, and consequently enhance the flexibility and effectiveness of the framework.

# Bibliography


[1] Zhi-Peng Li, Hai-Long Su, Xiao-Bo Zhu, Valeriya Gribova, Vladimir Fedorovich Filaretov, and De-Shuang Huang. SSPool: A simple siamese framework for graph infomax pooling. *IEEE Transactions on Network Science and Engineering*, 11(1): 463–470, 2024.

[2] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pages 3734–3743, 2019.

[3] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[4] Afzal Badshah, Ali Daud, Riad Alharbey, Ameen Banjar, Amal Bukhari, and Bader Alshemaimri. Big data applications: Overview, challenges and future. *Artificial Intelligence Review*, 57(11):290, 2024.

[5] Fitsum Gebreegziabher and Ripon Patgiri. The major challenges of big graph and their solutions: A review. *Advances in Computers*, 128:399–421, 2023.

[6] Xingwang Zhao, Jiye Liang, and Jie Wang. A community detection algorithm based on graph compression for large-scale social networks. *Information Sciences*, 551:358–372, 2021.

[7] Rui Li, Xin Yuan, Mohsen Radfar, Peter Marendy, Wei Ni, Terrence J O'Brien, and Pablo M Casillas-Espinosa. Graph signal processing, graph neural network and graph learning on biological data: a systematic review. *IEEE Reviews in Biomedical Engineering*, 16:109–135, 2021.

[8] Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence*, 2(2): 109–127, 2021.

[9] Yujie Mo, Liang Peng, Jie Xu, Xiaoshuang Shi, and Xiaofeng Zhu. Simple unsupervised graph representation learning. In *AAAI conference on artificial intelligence*, volume 36, pages 7797–7805, 2022.

[10] Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. Graph representation learning: a survey. *APSIPA Transactions on Signal and Information Processing*, 9, 2020.

[11] Shima Khoshraftar and Aijun An. A survey on graph representation learning methods. *ACM Transactions on Intelligent Systems and Technology*, 15(1):1–55, 2024.

[12] Yu Zhou, Haixia Zheng, Xin Huang, Shufeng Hao, Dengao Li, and Jumin Zhao. Graph neural networks: Taxonomy, advances, and trends. *ACM Transactions on Intelligent Systems and Technology*, 13(1):1–54, 2022.

[13] Xin Zheng, Yi Wang, Yixin Liu, Ming Li, Miao Zhang, Di Jin, Philip S Yu, and Shirui Pan. Graph neural networks for graphs with heterophily: A survey. *arXiv preprint arXiv:2202.07082*, 2022.

[14] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

[15] Shunxin Xiao, Shiping Wang, Yuanfei Dai, and Wenzhong Guo. Graph neural networks in node classification: survey and evaluation. *Machine Vision and Applications*, 33(1):4, 2022.

[16] Jie Liu, Jiamou Liu, Kaiqi Zhao, Yanni Tang, and Wu Chen. Tp-gnn: Continuous dynamic graph neural network for graph classification. In *International Conference on Data Engineering*, pages 2848–2861, 2024.

[17] Sai Munikoti, Laya Das, and Balasubramaniam Natarajan. Scalable graph neural network-based framework for identifying critical nodes and links in complex networks. *Neurocomputing*, 468:211–221, 2022.

[18] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[19] Lei Cai, Jundong Li, Jie Wang, and Shuiwang Ji. Line graph neural networks for link prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5103–5113, 2021.

[20] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *Advances in Neural Information Processing Systems*, volume 34, pages 29476–29490, 2021.

[21] Kun Zhan, Changqing Zhang, Junpeng Guan, and Junsheng Wang. Graph learning for multiview clustering. *IEEE Transactions on Cybernetics*, 48(10):2887–2895, 2017.

[22] Suyuan Liu, Qing Liao, Siwei Wang, Xinwang Liu, and En Zhu. Robust and consistent anchor graph learning for multi-view clustering. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[23] Xunxun Wu, Chang-Dong Wang, Jia-Qi Lin, Wu-Dong Xi, and S Yu Philip. Motif-based contrastive learning for community detection. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

[24] Xunlian Wu, Wanying Lu, Yining Quan, Qiguang Miao, and Peng Gang Sun. Deep dual graph attention auto-encoder for community detection. *Expert Systems with Applications*, 238:122182, 2024.

[25] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[26] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 9061–9073, 2021.

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.

[28] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[29] Xiaobin Hong, Wenzhong Li, Chaoqun Wang, Mingkai Lin, and Sanglu Lu. Label attentive distillation for gnn-based graph classification. In *AAAI Conference on Artificial Intelligence*, volume 38, pages 8499–8507, 2024.

[30] Yu Xie, Yanfeng Liang, Maoguo Gong, A Kai Qin, Yew-Soon Ong, and Tiantian He. Semisupervised graph neural networks for graph classification. *IEEE Transactions on Cybernetics*, 53(10):6222–6235, 2022.

[31] Yu Xie, Shengze Lv, Yuhua Qian, Chao Wen, and Jiye Liang. Active and semi-supervised graph neural networks for graph classification. *IEEE Transactions on Big Data*, 8(4):920–932, 2022.

[32] Matheus RF Mendonça, André MS Barreto, and Artur Ziviani. Approximating network centrality measures using node embedding and machine learning. *IEEE Transactions on Network Science and Engineering*, 8(1):220–230, 2020.

[33] Yonghua Zhu, Lei Feng, Zhenyun Deng, Yang Chen, Robert Amor, and Michael Witbrock. Robust node classification on graph data with graph and label noise. In *AAAI conference on artificial intelligence*, volume 38, pages 17220–17227, 2024.

[34] Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Classic gnns are strong baselines: Reassessing gnns for node classification. *Advances in Neural Information Processing Systems*, 37:97650–97669, 2024.

[35] Youfa Liu and Bo Du. Frequency domain-oriented complex graph neural networks for graph classification. *IEEE Transactions on Neural Networks and Learning Systems*, 36(2):2733–2746, 2025.

[36] Zixiao Wang and Jicong Fan. Graph classification via reference distribution learning: theory and practice. *Advances in Neural Information Processing Systems*, 37: 137698–137740, 2024.

[37] Zhiqiang Zhong, Cheng-Te Li, and Jun Pang. Hierarchical message-passing graph neural networks. *Data Mining and Knowledge Discovery*, 37(1):381–408, 2023.

[38] Yu Guang Wang, Ming Li, Zheng Ma, Guido Montufar, Xiaosheng Zhuang, and Yanan Fan. Haar graph pooling. In *International conference on machine learning*, pages 9952–9962, 2020.

[39] Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar, and Muhan Zhang. How powerful are k-hop message passing graph neural networks. In *Advances in Neural Information Processing Systems*, volume 35, pages 4776–4790, 2022.

[40] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[41] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, et al. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. *ACM Transactions on Recommender Systems*, 1(1):1–51, 2023.

[42] Gabriele Corso, Hannes Stark, Stefanie Jegelka, Tommi Jaakkola, and Regina Barzilay. Graph neural networks. *Nature Reviews Methods Primers*, 4(1):17, 2024.

[43] Fan Liang, Cheng Qian, Wei Yu, David Griffith, and Nada Golmie. Survey of graph neural networks and applications. *Wireless Communications and Mobile Computing*, 2022(1):9261537, 2022.

[44] Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications*, 207:117921, 2022.

[45] Wenbing Huang, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Tackling over-smoothing for general graph convolutional networks. *arXiv preprint arXiv:2008.09864*, 2020.

[46] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

[47] Jian Chen, Wei Wang, Keping Yu, Xiping Hu, Ming Cai, and Mohsen Guizani. Node connection strength matrix-based graph convolution network for traffic flow prediction. *IEEE Transactions on Vehicular Technology*, 2023.

[48] Carlo Abate and Filippo Maria Bianchi. Maxcutpool: differentiable feature-aware maxcut for pooling in graph neural networks. In *International Conference on Learning Representations*, 2025.

[49] Chuang Liu, Yibing Zhan, Baosheng Yu, Liu Liu, Bo Du, Wenbin Hu, and Tongliang Liu. On exploring node-feature and graph-structure diversities for node drop graph pooling. *Neural Networks*, 167:559–571, 2023.

[50] Akrati Saxena and Sudarshan Iyengar. Centrality measures in complex networks: A survey. *arXiv preprint arXiv:2011.07190*, 2020.

[51] Francis Bloch, Matthew O Jackson, and Pietro Tebaldi. Centrality measures in networks. *Social Choice and Welfare*, 61(2):413–453, 2023.

[52] Jonathan L Gross, Jay Yellen, and Mark Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018.

[53] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory*. Springer Publishing Company, Incorporated, 2008.

[54] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice Hall Upper Saddle River, 2001.

[55] Tianming Zhang, Yunjun Gao, Jie Zhao, Lu Chen, Lu Jin, Zhengyi Yang, Bin Cao, and Jing Fan. Efficient exact and approximate betweenness centrality computation for temporal graphs. In *ACM Web Conference*, pages 2395–2406, 2024.

[56] Mingkai Lin, Wenzhong Li, Lynda J Song, Cam-Tu Nguyen, Xiaoliang Wang, and Sanglu Lu. Sake: Estimating katz centrality based on sampling for large-scale social networks. *ACM Transactions on Knowledge Discovery from Data*, 15(4): 1–21, 2021.

[57] Qing Xu, Lizhu Sun, and Changjiang Bu. The two-steps eigenvector centrality in complex networks. *Chaos, Solitons & Fractals*, 173:113753, 2023.

[58] Junlong Zhang and Yu Luo. Degree centrality, betweenness centrality, and closeness centrality in social network. In *International Conference on Modelling, Simulation and Applied Mathematics*, pages 300–303, 2017.

[59] Warih Maharani, Alfian Akbar Gozali, et al. Degree centrality and eigenvector centrality in twitter. In *International Conference on Telecommunication Systems Services and Applications*, pages 1–5, 2014.

[60] Giorgio Roffo and Simone Melzi. Ranking to learn: Feature ranking and selection via eigenvector centrality. In *New Frontiers in Mining Complex Patterns: 5th International Workshop, NFMCP, Held in Conjunction with ECML-PKDD*, pages 19–35, 2017.

[61] Kousik Das, Sovan Samanta, and Madhumangal Pal. Study on centrality measures in social networks: a survey. *Social Network Analysis and Mining*, 8:1–11, 2018.

[62] Ernesto Estrada and Naomichi Hatano. Communicability graph and community structures in complex networks. *Applied Mathematics and Computation*, 214(2): 500–511, 2009.

[63] Khadidja Henni, Neila Mezghani, and Charles Gouin-Vallerand. Unsupervised graph-based feature selection via subspace and pagerank centrality. *Expert Systems with Applications*, 114:46–53, 2018.

[64] Mingji Yang, Hanzhi Wang, Zhewei Wei, Sibo Wang, and Ji-Rong Wen. Efficient algorithms for personalized pagerank computation: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[65] Xiao Li, Li Sun, Mengjie Ling, and Yan Peng. A survey of graph neural network based recommendation in social networks. *Neurocomputing*, 549:126441, 2023.

[66] Zhiwei Guo and Heng Wang. A deep graph neural network-based mechanism for social recommendations. *IEEE Transactions on Industrial Informatics*, 17(4): 2776–2783, 2020.

[67] Roshni Iyer, Yewen Wang, Wei Wang, and Yizhou Sun. Non-euclidean mixture model for social network embedding. In *Advances in Neural Information Processing Systems*, volume 37, pages 111464–111488, 2025.

[68] Ruoxi Sun, Hanjun Dai, and Adams Wei Yu. Does gnn pretraining help molecular representation? In *Advances in Neural Information Processing Systems*, volume 35, pages 12096–12109, 2022.

[69] Yuyang Wang, Zijie Li, and Amir Barati Farimani. Graph neural networks for molecules. In *Machine Learning in Molecular Sciences*, pages 21–66. 2023.

[70] Tianyu Liu, Yuge Wang, Rex Ying, and Hongyu Zhao. Muse-gnn: Learning unified gene representation from multimodal biological graph data. In *Advances in neural information processing systems*, volume 36, pages 24661–24677, 2023.

[71] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. Graph neural networks. In *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 27–37. 2022.

[72] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[73] Mingguo He, Zhewei Wei, and Ji-Rong Wen. Convolutional neural networks on graphs with chebyshev approximation, revisited. In *Advances in Neural Information Processing Systems*, 2022.

[74] Yuankai Luo, Lei Shi, Mufan Xu, Yuwen Ji, Fengli Xiao, Chunming Hu, and Zhiguang Shan. Impact-oriented contextual scholar profiling using self-citation graphs. In *Conference on Knowledge Discovery and Data Mining*, pages 4572–4583, 2023.

[75] Saeed Rahmani, Asiye Baghbani, Nizar Bouguila, and Zachary Patterson. Graph neural networks for intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 24(8):8846–8885, 2023.

[76] Moshe Eliasof, Eldad Haber, and Eran Treister. Feature transportation improves graph neural networks. In *AAAI Conference on Artificial Intelligence*, volume 38, pages 11874–11882, 2024.

[77] Hourun Li, Yusheng Zhao, Zhengyang Mao, Yifang Qin, Zhiping Xiao, Jiaqi Feng, Yiyang Gu, Wei Ju, Xiao Luo, and Ming Zhang. Graph neural networks in intelligent transportation systems: Advances, applications and trends. *arXiv preprint arXiv:2401.00713*, 2024.

[78] Meng Liu, Zhengyang Wang, and Shuiwang Ji. Non-local graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):10270–10276, 2022.

[79] Daniel Spielman. Spectral graph theory. *Combinatorial Scientific Computing*, 18 (18), 2012.

[80] Joao Domingos and José MF Moura. Graph fourier transform: A stable approximation. *IEEE Transactions on Signal Processing*, 68:4422–4437, 2020.

[81] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and deep locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

[82] Zhiqian Chen, Fanglan Chen, Lei Zhang, Taoran Ji, Kaiqun Fu, Liang Zhao, Feng Chen, Lingfei Wu, Charu Aggarwal, and Chang-Tien Lu. Bridging the gap between spatial and spectral domains: A unified framework for graph neural networks. *ACM Comput. Surv.*, 56(5), 2023.

[83] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Message passing neural networks. In *Machine learning meets quantum physics*, pages 199–214. 2020.

[84] Li Zhang, Dan Xu, Anurag Arnab, and Philip HS Torr. Dynamic graph message passing networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3726–3735, 2020.

[85] Xiaorui Liu, Jiayuan Ding, Wei Jin, Han Xu, Yao Ma, Zitao Liu, and Jiliang Tang. Graph neural networks with adaptive residual. In *Advances in Neural Information Processing Systems*, volume 34, pages 9720–9733, 2021.

[86] Lecheng Zheng, Dongqi Fu, Ross Maciejewski, and Jingrui He. Drgnn: Deep residual graph neural network with contrastive learning. *Transactions on Machine Learning Research*, 2024.

[87] Chengcheng Sun, Chenhao Li, Xiang Lin, Tianji Zheng, Fanrong Meng, Xiaobin Rui, and Zhixiao Wang. Attention-based graph neural networks: a survey. *Artificial Intelligence Review*, 56(Suppl 2):2263–2310, 2023.

[88] Shuo Zhang and Lei Xie. Improving attention mechanism in graph neural networks via cardinality preservation. In *International Joint Conference on Artificial Intelligence*, volume 2020, page 1395, 2020.

[89] Simon Markus Geisler, Arthur Kosmala, Daniel Herbst, and Stephan Günnemann. Spatio-spectral graph neural networks. In *Advances in Neural Information Processing Systems*, volume 37, pages 49022–49080, 2025.

[90] Deyu Bo, Chuan Shi, Lele Wang, and Renjie Liao. Specformer: Spectral graph neural networks meet transformers. *arXiv preprint arXiv:2303.01028*, 2023.

[91] Deyu Bo, Xiao Wang, Yang Liu, Yuan Fang, Yawen Li, and Chuan Shi. A survey on spectral graph neural networks. *arXiv preprint arXiv:2302.05631*, 2023.

[92] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. Sampling methods for efficient training of graph convolutional networks: A survey. *IEEE/CAA Journal of Automatica Sinica*, 9(2):205–234, 2021.

[93] Zhen Song, Yu Gu, Tianyi Li, Qing Sun, Yanfeng Zhang, Christian S Jensen, and Ge Yu. Adgnn: towards scalable gnn training with aggregation-difference aware sampling. *ACM on Management of Data*, 1(4):1–26, 2023.

[94] Yao Ma and Jiliang Tang. *Deep learning on graphs*. Cambridge University Press, 2021.

[95] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in Neural Information Processing Systems*, 28, 2015.

[96] Zhengyang Wang and Shuiwang Ji. Second-order pooling for graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(6): 6870–6880, 2020.

[97] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI conference on artificial intelligence*, volume 32, 2018.

[98] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[99] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. *arXiv preprint arXiv:2102.11533*, 2021.

[100] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[101] Matheus Cavalcante, Samuel Riedel, Antonio Pullini, and Luca Benini. Mempool: A shared-l1 memory many-core cluster with a low-latency interconnect. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 701–706, 2021.

[102] Hao Yuan and Shuiwang Ji. Structpool: Structured graph pooling via conditional random fields. In *International Conference on Learning Representations*, 2020.

[103] Enxhell Luzhnica, Ben Day, and Pietro Lio. Clique pooling for graph classification. *arXiv preprint arXiv:1904.00374*, 2019.

[104] Davide Bacciu, Alessio Conte, Roberto Grossi, Francesco Landolfi, and Andrea Marino. K-plex cover pooling for graph neural networks. *Data Mining and Knowledge Discovery*, 35(5):2200–2220, 2021.

[105] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In *International Conference on Knowledge Discovery & Data Mining*, pages 723–731, 2019.

[106] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *International Conference on Machine Learning*, pages 2083–2092, 2019.

[107] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, pages 40–48, 2016.

[108] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

[109] Benedek Rozemberczki and Rik Sarkar. Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models. In *Conference on Information and Knowledge Management*, pages 1325–1334, 2020.

[110] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Juncheng Liu, and Sourav S Bhowmick. Scaling attributed network embedding to massive graphs. *VLDB Endowment*, 14(1):37–49, 2020.

[111] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Sourav S Bhowmick, and Juncheng Liu. PANE: scalable and effective attributed network embedding. *The VLDB Journal*, 32(6):1237–1262, 2023.

[112] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[113] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, pages 22118–22133, 2020.

[114] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

[115] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[116] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14:347–375, 2008.

[117] Jeroen Kazius, Ross McGuire, and Roberta Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of Medicinal Chemistry*, 48(1): 312–320, 2005.

[118] Pinar Yanardag and SVN Vishwanathan. A structural smoothing framework for robust graph comparison. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

[119] Rui Liu, Pengwei Xing, Zichao Deng, Anran Li, Cuntai Guan, and Han Yu. Federated graph neural networks: Overview, techniques, and challenges. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–17, 2024.

[120] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, page 5998–6008, 2017.

[121] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-Human Language Technologies*, pages 4171–4186, 2019.

[122] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. In *International Speech Communication Association*, 2020.

[123] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In *Advances in Neural Information Processing Systems*, 2021.

[124] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Advances in Neural Information Processing Systems*, pages 28877–28888, 2021.

[125] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[126] Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to graph transformers. *Transactions on Machine Learning Research*, 2024.

[127] Taha Atahan Akyildiz, Amro Alabsi Aljundi, and Kamer Kaya. Understanding coarsening for embedding large-scale graphs. In *IEEE International Conference on Big Data*, pages 2937–2946, 2020.

[128] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. In *International Conference on Knowledge Discovery & Data Mining*, pages 675–684, 2021.

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

[130] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[131] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *International Conference on Knowledge Discovery and Data Mining*, pages 701–710, 2014.

[132] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *International Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.

[133] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *International Conference on World Wide Web*, pages 1067–1077, 2015.

[134] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234, 2016.

[135] Kan Guo, Yongli Hu, Yanfeng Sun, Sean Qian, Junbin Gao, and Baocai Yin. Hierarchical graph convolution network for traffic forecasting. In *AAAI Conference on Artificial Intelligence*, volume 35, pages 151–159, 2021.

[136] Ladislav Rampášek and Guy Wolf. Hierarchical graph neural nets can capture long-range interactions. In *International Workshop on Machine Learning for Signal Processing*, pages 1–6, 2021.

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

[138] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[139] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[140] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[141] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[142] Andreas Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

[143] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

[144] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pages 6861–6871, 2019.

[145] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462, 2018.

[146] Zaixi Zhang, Qi Liu, Qingyong Hu, and Chee-Kong Lee. Hierarchical graph transformer with adaptive node sampling. In *Advances in Neural Information Processing Systems*, 2022.

[147] Bingheng Li, Erlin Pan, and Zhao Kang. Pc-conv: Unifying homophily and heterophily with two-fold filtering. In *AAAI Conference on Artificial Intelligence*, volume 38, pages 13437–13445, 2024.

[148] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[149] Xinlong Huang, Simon Hu, Wei Wang, Ioannis Kaparias, Shaopeng Zhong, Xiaoxiang Na, Michael GH Bell, and Der-Horng Lee. Identifying critical links in urban transportation networks based on spatio-temporal dependency learning. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[150] Sho Tsugawa and Kohei Watabe. Identifying influential brokers on social media from social network structure. In *International AAAI Conference on Web and Social Media*, volume 17, pages 842–853, 2023.

[151] Lei Xiao, Shuangyan Wang, and Gang Mei. Efficient parallel algorithm for detecting influential nodes in large biological networks on the graphics processing unit. *Future Generation Computer Systems*, 106:1–13, 2020.

[152] Li Daqing, Jiang Yinan, Kang Rui, and Shlomo Havlin. Spatial correlation analysis of cascading failures: congestions and blackouts. *Scientific reports*, 4(1):5381, 2014.

[153] Chin-Chi Hsu, Yi-An Lai, Wen-Hao Chen, Ming-Han Feng, and Shou-De Lin. Unsupervised ranking using graph structures and node attributes. In *International Conference on Web Search and Data Mining*, pages 771–779, 2017.

[154] Changjun Fan, Li Zeng, Yuhui Ding, Muhao Chen, Yizhou Sun, and Zhong Liu. Learning to identify high betweenness centrality nodes from scratch: A novel graph neural network approach. In *International Conference on Information and Knowledge Management*, pages 559–568, 2019.

[155] Min Zhang, Xiaojuan Wang, Lei Jin, Mei Song, and Ziyang Li. A new approach for evaluating node importance in complex networks via deep learning methods. *Neurocomputing*, 497:13–27, 2022.

[156] Sebastiano Vigna and Paolo Boldi. Axioms for centrality. *Internet Mathematics*, 10(3):222–262, 2014.

[157] Nan Xiang, Qilin Wang, and Mingwei You. Estimation and update of betweenness centrality with progressive algorithm and shortest paths approximation. *Scientific Reports*, 13(1):17110, 2023.

[158] Xiang Xu, Cheng Zhu, Qingyong Wang, Xianqiang Zhu, and Yun Zhou. Identifying vital nodes in complex networks by adjacency information entropy. *Scientific reports*, 10(1):2691, 2020.

[159] David Bowater and Emmanuel Stefanakis. Extending the adapted pagerank algorithm centrality model for urban street networks using non-local random walks. *Applied Mathematics and Computation*, 446:127888, 2023.

[160] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. Graph neural networks for fast node ranking approximation. *ACM Transactions on Knowledge Discovery from Data*, 15(5):1–32, 2021.

[161] Appan Rakaraddi and Mahardhika Pratama. Unsupervised learning for identifying high eigenvector centrality nodes: A graph neural network approach. In *International Conference on Big Data*, pages 4945–4954, 2021.

[162] F. Lin and W. W. Cohen. Power iteration clustering. In *International Conference on Machine Learning*, volume 10, page 655–662, 2010.

[163] Felipe Grando and Luis C Lamb. Estimating complex networks centrality via neural networks and machine learning. In *International Joint Conference on Neural Networks*, pages 1–8, 2015.

[164] MohammadHossein Bateni, Hossein Esfandiari, Manuela Fischer, and Vahab Mirrokni. Extreme k-center clustering. In *AAAI Conference on Artificial Intelligence*, volume 35, pages 3941–3949, 2021.

[165] Manoj Kumar, Anurag Sharma, Shashwat Saxena, and Sandeep Kumar. Featured graph coarsening with similarity guarantees. In *International Conference on Machine Learning*, pages 17953–17975, 2023.

[166] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. *arXiv preprint arXiv:2110.07580*, 2021.

[167] Ricky Maulana Fajri, Yulong Pei, Lu Yin, and Mykola Pechenizkiy. A structural-clustering based active learning for graph neural networks. In *International Symposium on Intelligent Data Analysis*, pages 28–40, 2024.

[168] Han Huang, Leilei Sun, Bowen Du, Chuanren Liu, Weifeng Lv, and Hui Xiong. Representation learning on knowledge graphs for node importance estimation. In *Conference on Knowledge Discovery & Data Mining*, pages 646–655, 2021.

[169] Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. Estimating node importance in knowledge graphs using graph neural networks. In *SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 596–606, 2019.

[170] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. In *Advances in neural information processing systems*, volume 33, pages 17804–17815, 2020.

[171] Chuhan Wu, Fangzhao Wu, Yongfeng Huang, and Xing Xie. User-as-graph: User modeling with heterogeneous graph pooling for news recommendation. In *International Joint Conference on Artificial Intelligence*, pages 1624–1630, 2021.

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

[173] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *AAAI conference on artificial intelligence*, pages 7370–7377, 2019.

[174] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[175] Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. In *AAAI conference on artificial intelligence*, volume 34, pages 5470–5477, 2020.

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

[177] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 35(2):2708–2718, 2022.

[178] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[179] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning*, pages 874–883, 2020.

[180] Robert A Hanneman and Mark Riddle. Introduction to social network methods, 2005.

[181] Michele Benzi and Christine Klymko. Total communicability as a centrality measure. *Journal of Complex Networks*, 1(2):124–149, 2013.

[182] Francesco Tudisco, Francesca Arrigo, and Antoine Gautier. Node and layer eigenvector centralities for multiplex networks. *SIAM Journal on Applied Mathematics*, 78(2):853–876, 2018.

[183] Junran Wu, Xueyuan Chen, Ke Xu, and Shangzhe Li. Structural entropy guided graph hierarchical pooling. In *International conference on machine learning*, pages 24017–24030, 2022.

[184] Chunfeng Cui and Liqun Qi. A power method for computing the dominant eigenvalue of a dual quaternion hermitian matrix. *Journal of Scientific Computing*, 100 (1):21, 2024.

[185] Pasquale De Meo, Mark Levene, Fabrizio Messina, and Alessandro Provetti. A general centrality framework-based on node navigability. *IEEE Transactions on Knowledge and Data Engineering*, 32(11):2088–2100, 2019.

[186] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[187] Maosen Li, Siheng Chen, Ya Zhang, and Ivor Tsang. Graph cross networks with vertex infomax pooling. In *Advances in Neural Information Processing Systems*, volume 33, pages 14093–14105, 2020.

[188] Yunsheng Pang, Yunxiang Zhao, and Dongsheng Li. Graph pooling via coarsened graph infomax. In *SIGIR Conference on Research and Development in Information Retrieval*, pages 2177–2181, 2021.

[189] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020.

[190] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3496–3507, 2022.

[191] Mingguo He, Zhewei Wei, Hongteng Xu, et al. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. *Advances in Neural Information Processing Systems*, 34:14239–14251, 2021.

[192] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022.

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

[194] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. NAGphormer: A tokenized graph transformer for node classification in large graphs. In *International Conference on Learning Representations*, 2023.

[195] Chuang Liu, Yibing Zhan, Xueqi Ma, Liang Ding, Dapeng Tao, Jia Wu, and Wenbin Hu. Gapformer: Graph transformer with graph pooling for node classification. In *International Joint Conference on Artificial Intelligence*, pages 2196–2205, 2023.

[196] Langzhang Liang, Xiangjing Hu, Zenglin Xu, Zixing Song, and Irwin King. Predicting global label relationship matrix for graph neural networks under heterophily. In *Advances in Neural Information Processing Systems*, pages 10909–10921, 2023.

[197] Xinlong Huang, Jian Chen, Ming Cai, Wei Wang, and Xiping Hu. Traffic node importance evaluation based on clustering in represented transportation networks. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):16622–16631, 2022.

[198] Chen Ling, Junji Jiang, Junxiang Wang, My T Thai, Renhao Xue, James Song, Meikang Qiu, and Liang Zhao. Deep graph representation learning and optimization for influence maximization. In *International Conference on Machine Learning*, pages 21350–21361, 2023.

[199] Huyen Trang Phan, Ngoc Thanh Nguyen, and Dosam Hwang. Fake news detection: A survey of graph neural network methods. *Applied Soft Computing*, 139:110235, 2023.

[200] Houwang Zhang, Zhenan Feng, and Chong Wu. A non-local graph neural network for identification of essential proteins. In *International Joint Conference on Neural Networks*, pages 1–8, 2022.